Week 7

AP Computer Science A

Homework bundle for this week

本周作业合集

exercises.lol

Contents 目录

2. Selection and Iteration

Starter Quiz · 课前小测

AP Computer Science A Name: _____ Score: _____

1. Choosing which code to run based on a condition is called...
   - ☐ selection
   - ☐ repetition
   - ☐ sequence
   - ☐ an error
2. Which operator tests whether two values are equal?
   - ☐ ==
   - ☐ =
   - ☐ !=
   - ☐ <=
3. In an if-else statement, how many of the two branches run?
   - ☐ exactly one
   - ☐ both
   - ☐ neither
   - ☐ it depends on braces
4. In an else-if ladder, which condition's block runs?
   - ☐ the first one that is true
   - ☐ the last one that is true
   - ☐ all true ones
   - ☐ the shortest one
5. a && b is true when...
   - ☐ both a and b are true
   - ☐ at least one is true
   - ☐ neither is true
   - ☐ a is false
6. Outer loop runs 3 times, inner runs 3 times each. How many times does the inner body run?

   _____

7. With int age = 18;, the expression age > 18 is true.

   ☐ True ☐ False

8. int i=1, sum=0; while (i<=5){ sum+=i; i++; } What is sum at the end?

   _____

9. How many times does the body of for (int i = 0; i < 5; i++) run?

   _____

10. In nested loops, the inner loop runs fully for every single pass of the outer loop.

    ☐ True ☐ False

AP Computer Science A

1. **selection**

   Selection (if) picks a path; repetition (loops) repeats a block.

2. **==**

   == tests equality; = assigns.

3. **exactly one**

   Exactly one branch runs based on the condition.

4. **the first one that is true**

   Java takes the first true condition top to bottom.

5. **both a and b are true**

   && (AND) needs both operands true.

6. **9**

   $3 \times 3 = 9$.

7. **False**

   18 is not greater than 18; age >= 18 would be true.

8. **15**

   1+2+3+4+5 = 15.

9. **5**

   i takes 0,1,2,3,4 → 5 passes.

10. **True**

    That's why the inner body runs outer × inner times.

AP COMPUTER SCIENCE A • EXERCISE SHEET

2.1 Algorithms with Selection and Repetition

Name: _____ Class: _____ Date: _____ **Total: 9 marks**

KEY WORDS repetition 重复 • selection 选择 • sequencing 顺序

Objective

Build the skills to answer exam questions on **algorithms with selection and repetition**.

You must be able to:

- express an algorithm using **sequencing** 顺序, **selection** 选择, and **repetition** 重复
- explain how selection lets an algorithm make choices
- explain how repetition lets an algorithm repeat steps

1 Worked examples

Study these first. Each one shows the method for a question type used later.

■ Three building blocks

Every algorithm is built from:

- **Sequencing** — running steps in order.
- **Selection** — choosing between paths (an `if` statement).
- **Repetition** — repeating steps (a loop).

■ Example

"For each score, if it is above 50, print it" uses **repetition** (for each) plus **selection** (if above 50).

2 Practice

2.1 Name the three building blocks of an algorithm. [2]

2.2 State what selection lets an algorithm do. [1]

2.3 State what repetition lets an algorithm do. [1]

3 Exam-style questions

3.1 Choosing between paths in an algorithm is [1]

- **A** sequencing
- **B** selection
- **C** repetition
- **D** casting

3.2 Repeating a block of steps is [1]

- **A** sequencing
- **B** selection
- **C** repetition
- **D** overloading

3.3 An algorithm checks each of 10 scores and prints those above 50.

(a) Name the building block that repeats the check. [1]

(b) Name the building block that decides "above 50". [1]

(c) Name the building block that runs steps in order. [1]

Solutions

2.1 sequencing, selection, repetition.

2.2 it lets the algorithm choose between different paths based on a condition.

2.3 it lets the algorithm repeat a block of steps.

3.1 B.

3.2 C.

3.3 (a) repetition. (b) selection. (c) sequencing.

1. The `APCalendar` class contains methods used to calculate information about a calendar. You will write two methods of the class.

```
public class APCalendar
{

   /** Returns true if year is a leap year and false otherwise. */
   private static boolean isLeapYear(int year)
   {  /* implementation not shown */  }

   /** Returns the number of leap years between year1 and year2, inclusive.
    *  Precondition: 0 <= year1 <= year2
    */
   public static int numberOfLeapYears(int year1, int year2)
   {  /* to be implemented in part (a) */  }

   /** Returns the value representing the day of the week for the first day of year,
    *  where 0 denotes Sunday, 1 denotes Monday, ..., and 6 denotes Saturday.
    */
   private static int firstDayOfYear(int year)
   {  /* implementation not shown */  }

   /** Returns n, where month, day, and year specify the nth day of the year.
    *  Returns 1 for January 1 (month = 1, day = 1) of any year.
    *  Precondition: The date represented by month, day, year is a valid date.
    */
   private static int dayOfYear(int month, int day, int year)
   {  /* implementation not shown */  }

   /** Returns the value representing the day of the week for the given date
    *  (month, day, year), where 0 denotes Sunday, 1 denotes Monday, ...,
    *  and 6 denotes Saturday.
    *  Precondition: The date represented by month, day, year is a valid date.
    */
   public static int dayOfWeek(int month, int day, int year)
   {  /* to be implemented in part (b) */  }

   // There may be instance variables, constructors, and other methods not shown.

}
```

(a) Write the static method `numberOfLeapYears`, which returns the number of leap years between `year1` and `year2`, inclusive.

In order to calculate this value, a helper method is provided for you.

- `isLeapYear(year)` returns `true` if `year` is a leap year and `false` otherwise.

Complete method `numberOfLeapYears` below. You must use `isLeapYear` appropriately to receive full credit.

```
/** Returns the number of leap years between year1 and year2, inclusive.
 *  Precondition: 0 <= year1 <= year2
 */
public static int numberOfLeapYears(int year1, int year2)
```

(b) Write the static method `dayOfWeek`, which returns the integer value representing the day of the week for the given date (`month`, `day`, `year`), where `0` denotes Sunday, `1` denotes Monday, ..., and `6` denotes Saturday. For example, 2019 began on a Tuesday, and January 5 is the fifth day of 2019. As a result, January 5, 2019, fell on a Saturday, and the method call `dayOfWeek(1, 5, 2019)` returns `6`.

As another example, January 10 is the tenth day of 2019. As a result, January 10, 2019, fell on a Thursday, and the method call `dayOfWeek(1, 10, 2019)` returns `4`.

In order to calculate this value, two helper methods are provided for you.

- `firstDayOfYear(year)` returns the integer value representing the day of the week for the first day of `year`, where `0` denotes Sunday, `1` denotes Monday, ..., and `6` denotes Saturday. For example, since 2019 began on a Tuesday, `firstDayOfYear(2019)` returns `2`.
- `dayOfYear(month, day, year)` returns *n*, where `month`, `day`, and `year` specify the *n*th day of the year. For the first day of the year, January 1 (`month = 1, day = 1`), the value `1` is returned. This method accounts for whether `year` is a leap year. For example, `dayOfYear(3, 1, 2017)` returns `60`, since 2017 is not a leap year, while `dayOfYear(3, 1, 2016)` returns `61`, since 2016 is a leap year.

Class information for this question

```
public class APCalendar

private static boolean isLeapYear(int year)
public static int numberOfLeapYears(int year1, int year2)
private static int firstDayOfYear(int year)
private static int dayOfYear(int month, int day, int year)
public static int dayOfWeek(int month, int day, int year)
```

Complete method `dayOfWeek` below. You must use `firstDayOfYear` and `dayOfYear` appropriately to receive full credit.

```
/** Returns the value representing the day of the week for the given date
 *  (month, day, year), where 0 denotes Sunday, 1 denotes Monday, ...,
 *  and 6 denotes Saturday.
 *  Precondition: The date represented by month, day, year is a valid date.
 */
public static int dayOfWeek(int month, int day, int year)
```

AP COMPUTER SCIENCE A • EXERCISE SHEET

2.2 Boolean Expressions

Name: _____ Class: _____ Date: _____ **Total: 9 marks**

KEY WORDS boolean expression 布尔表达式 • relational operators 关系运算符

Objective

Build the skills to answer exam questions on **Boolean expressions**.

You must be able to:

- write a **Boolean expression** 布尔表达式 that evaluates to `true` or `false`
- use the **relational operators** 关系运算符 `<`, `<=`, `>`, `>=`, `==`, `!=`
- understand that `==` tests equality while `=` performs assignment

1 Worked examples

Study these first. Each one shows the method for a question type used later.

■ Boolean expressions

A Boolean expression evaluates to `true` or `false`, built with **relational operators**:

- `5 == 5` is `true` (equality)
- `3 != 4` is `true` (not equal)
- `7 >= 7` is `true`

■ == vs =

`==` **tests** equality; `=` **assigns** a value. Confusing them is a common bug.

2 Practice

2.1 State what a Boolean expression evaluates to. [1]

2.2 State the difference between `==` and `=`. [2]

2.3 Evaluate `7 >= 7`. [1]

---

3 Exam-style questions

3.1 Which operator tests equality? [1]

- **A** `=`
- **B** `==`
- **C** `!=`
- **D** `<=`

3.2 `4 != 4` evaluates to [1]

- **A** `true`
- **B** `false`
- **C** `4`
- **D** an error

3.3 Given `int x = 6, y = 9;`:

(a) evaluate `x < y`. [1]

(b) evaluate `x == y`. [1]

(c) evaluate `x != y`. [1]

Solutions

2.1 either `true` or `false`.

2.2 `==` tests whether two values are equal; `=` assigns a value to a variable.

2.3 `true`.

3.1 B.

3.2 B.

3.3 (a) `true`. (b) `false`. (c) `true`.

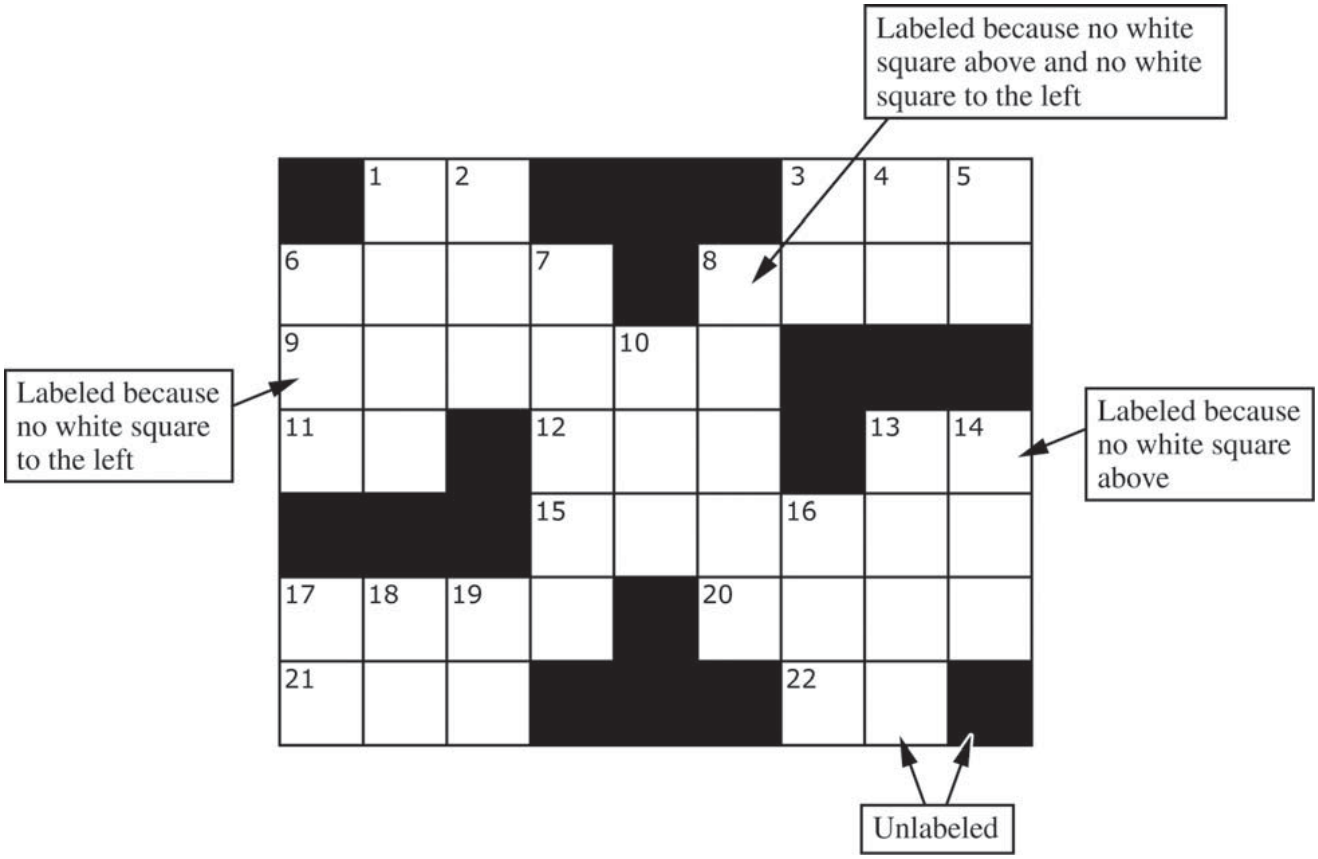
3. A crossword puzzle grid is a two-dimensional rectangular array of black and white squares. Some of the white squares are labeled with a positive number according to the *crossword labeling rule*.

   The crossword labeling rule identifies squares to be labeled with a positive number as follows.

   A square is labeled with a positive number if and only if
   - the square is white and
   - the square does not have a white square immediately above it, or it does not have a white square immediately to its left, or both.

   The squares identified by these criteria are labeled with consecutive numbers in row-major order, starting at 1.

   The following diagram shows a crossword puzzle grid and the labeling of the squares according to the crossword labeling rule.



This question uses two classes, a `Square` class that represents an individual square in the puzzle and a `Crossword` class that represents a crossword puzzle grid. A partial declaration of the `Square` class is shown below.

```
public class Square
{
    /** Constructs one square of a crossword puzzle grid.
     *  Postcondition:
     *    - The square is black if and only if isBlack is true.
     *    - The square has number num.
     */
    public Square(boolean isBlack, int num)
    {  /* implementation not shown */  }

    // There may be instance variables, constructors, and methods that are not shown.
}
```

A partial declaration of the `Crossword` class is shown below. You will implement one method and the constructor in the `Crossword` class.

```
public class Crossword
{
    /** Each element is a Square object with a color (black or white) and a number.
     *  puzzle[r][c] represents the square in row r, column c.
     *  There is at least one row in the puzzle.
     */
    private Square[][] puzzle;

    /** Constructs a crossword puzzle grid.
     *  Precondition: There is at least one row in blackSquares.
     *  Postcondition:
     *    - The crossword puzzle grid has the same dimensions as blackSquares.
     *    - The Square object at row r, column c in the crossword puzzle grid is black
     *      if and only if blackSquares[r][c] is true.
     *    - The squares in the puzzle are labeled according to the crossword labeling rule.
     */
    public Crossword(boolean[][] blackSquares)
    {  /* to be implemented in part (b) */  }

    /** Returns true if the square at row r, column c should be labeled with a positive number;
     *          false otherwise.
     *  The square at row r, column c is black if and only if blackSquares[r][c] is true.
     *  Precondition: r and c are valid indexes in blackSquares.
     */
    private boolean toBeLabeled(int r, int c, boolean[][] blackSquares)
    {  /* to be implemented in part (a) */  }

    // There may be instance variables, constructors, and methods that are not shown.
}
```

Part (a) begins on page 14.

(a) Write the `Crossword` method `toBeLabeled`. The method returns `true` if the square indexed by row `r`, column `c` in a crossword puzzle grid should be labeled with a positive number according to the crossword labeling rule; otherwise it returns `false`. The parameter `blackSquares` indicates which squares in the crossword puzzle grid are black.

Class information for this question

```
public class Square

public Square(boolean isBlack, int num)

public class Crossword

private Square[][] puzzle

public Crossword(boolean[][] blackSquares)
private boolean toBeLabeled(int r, int c, boolean[][] blackSquares)
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Complete method `toBeLabeled` below.

```
/** Returns true if the square at row r, column c should be labeled with a positive number;
 *          false otherwise.
 *  The square at row r, column c is black if and only if blackSquares[r][c] is true.
 *  Precondition: r and c are valid indexes in blackSquares.
 */
private boolean toBeLabeled(int r, int c, boolean[][] blackSquares)
```

Part (b) begins on page 16.

(b) Write the `Crossword` constructor. The constructor should initialize the crossword puzzle grid to have the same dimensions as the parameter `blackSquares`. Each element of the puzzle grid should be initialized with a reference to a `Square` object with the appropriate color and number. The number is positive if the square is labeled and 0 if the square is not labeled.

Class information for this question

```
public class Square

public Square(boolean isBlack, int num)

public class Crossword

private Square[][] puzzle

public Crossword(boolean[][] blackSquares)
private boolean toBeLabeled(int r, int c, boolean[][] blackSquares)
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Assume that `toBeLabeled` works as specified, regardless of what you wrote in part (a). You must use `toBeLabeled` appropriately to receive full credit.

Complete the `Crossword` constructor below.

```
/** Constructs a crossword puzzle grid.
 *  Precondition: There is at least one row in blackSquares.
 *  Postcondition:
 *    – The crossword puzzle grid has the same dimensions as blackSquares.
 *    – The Square object at row r, column c in the crossword puzzle grid is black
 *      if and only if blackSquares[r][c] is true.
 *    – The squares in the puzzle are labeled according to the crossword labeling rule.
 */
public Crossword(boolean[][] blackSquares)
```

AP COMPUTER SCIENCE A • EXERCISE SHEET

2.3 if Statements

Name: _____ Class: _____ Date: _____ **Total: 8 marks**

KEY WORDS if statement 条件语句 • flow of control 控制流

Objective

Build the skills to answer exam questions on **if statements**.

You must be able to:

- write an **if statement** 条件语句 that runs code only when a condition is `true`
- add an **else** clause for when the condition is `false`
- understand how a conditional changes the **flow of control** 控制流

1 Worked examples

Study these first. Each one shows the method for a question type used later.

■ if and else

```
if (score >= 60) {
    System.out.println("pass");
} else {
    System.out.println("fail");
}
```

The `if` block runs only when the condition is `true`; the `else` block runs when it is `false`.

■ Flow of control

A conditional changes **which statements run**, letting the program make decisions.

2 Practice

2.1 State what an if statement does. [1]

2.2 State what an else clause does. [1]

2.3 For `if (n > 0) print "positive"; else print "not positive";`, state the output when `n` is -3. [1]

---

3 Exam-style questions

3.1 An if statement runs its block when the condition is [1]

- **A** `false`
- **B** `true`
- **C** `null`
- **D** `0`

3.2 An else clause runs when the condition is [1]

- **A** `true`
- **B** `false`
- **C** missing
- **D** positive

3.3 `if (t >= 100) print "boiling"; else print "not boiling";`.

(a) State the output when `t` is 100. [1]

(b) State the output when `t` is 50. [1]

(c) Name what an if statement changes. [1]

Solutions

2.1 it runs a block of code only when its condition is `true`.

2.2 it runs alternative code when the condition is `false`.

2.3 `not positive`.

3.1 B.

3.2 B.

3.3 (a) `boiling`. (b) `not boiling`. (c) the flow of control.

2. The `Bottle` class, which you will write, represents a bottle that contains liquid.

`Bottle` objects are created by calls to a constructor with a `double` parameter that represents the bottle's capacity, in milliliters (mL), which is the maximum amount of liquid in the bottle. Assume that this value will be greater than or equal to 0. When `Bottle` objects are constructed, each is filled to its capacity.

The `Bottle` class contains an `updateAmount` method, which updates the amount of liquid in the bottle. This method has a `double` parameter that represents the amount of liquid, in mL, to be removed from the bottle, with a precondition that the parameter is always greater than zero and less than or equal to the amount of liquid remaining in the bottle. If updating the amount of liquid in the bottle would cause it to be less than 25% of the capacity, the bottle is filled and reset to the capacity. The `updateAmount` method returns a `double` that represents the amount remaining in the bottle, in mL, after the amount has been updated.

The following table contains a sample code execution sequence and the corresponding results. The code execution sequence appears in a class other than `Bottle`.

Statement	Return Value (blank if no value)	Explanation
`double amt;`		
`Bottle water = new Bottle(1000.0);`		A `Bottle` object named `water` is constructed with a capacity of 1,000 mL of liquid.
`amt = water.updateAmount(400.0);`	`600.0`	400 mL are used, so 600 mL remain in the bottle.
`amt = water.updateAmount(100.0);`	`500.0`	100 mL are used, so 500 mL remain in the bottle.
`amt = water.updateAmount(300.0);`	`1000.0`	300 mL are used, so 200 mL remain in the bottle. Since this amount is less than 250 mL (25% of the capacity of 1,000 mL), the bottle is immediately filled and reset to the initial amount of 1,000 mL.
`Bottle shampoo = new Bottle(40.0);`		A `Bottle` object named `shampoo` is constructed with a capacity of 40 mL of liquid.
`amt = shampoo.updateAmount(30.0);`	`10.0`	30 mL are used, so 10 mL remain in the bottle.
`amt = shampoo.updateAmount(1.0);`	`40.0`	1 mL is used, so 9 mL remain in the bottle. Since this amount is less than 10 mL (25% of the capacity of 40 mL), the bottle is immediately filled and reset to the initial amount of 40 mL.

Write the complete `Bottle` class. Your implementation must meet all specifications and conform to the examples shown in the table.

1. This question simulates birds or possibly a bear eating at a bird feeder. The following `Feeder` class contains information about how much food is in the bird feeder and simulates how much food is eaten. You will write two methods of the `Feeder` class.

```
public class Feeder
{
   /**
    *  The amount of food, in grams, currently in the bird feeder; initialized in the constructor and
    *  always greater than or equal to zero
    */
   private int currentFood;

   /**
    *  Simulates one day with numBirds birds or possibly a bear at the bird feeder,
    *  as described in part (a)
    *  Precondition: numBirds > 0
    */
   public void simulateOneDay(int numBirds)
   {  /* to be implemented in part (a) */  }

   /**
    *  Returns the number of days birds or a bear found food to eat at the feeder in this simulation,
    *  as described in part (b)
    *  Preconditions: numBirds > 0, numDays > 0
    */
   public int simulateManyDays(int numBirds, int numDays)
   {  /* to be implemented in part (b) */  }

   // There may be instance variables, constructors, or methods that are not shown.
}
```

GO ON TO THE NEXT PAGE.

(a) Write the `simulateOneDay` method, which simulates `numBirds` birds or possibly a bear at the feeder for one day. The method determines the amount of food taken from the feeder on this day and updates the `currentFood` instance variable. The simulation accounts for normal conditions, which occur 95% of the time, and abnormal conditions, which occur 5% of the time.

Under normal conditions, the simulation assumes that on any given day, only birds visit the feeder and that each bird at the feeder consumes the same amount of food. This standard amount consumed is between 10 and 50 grams of food, inclusive, in 1-gram increments. That is, on any given day, each bird might eat 10, 11, . . . , 49, or 50 grams of food. The amount of food eaten by each bird on a given day is randomly generated and each integer from 10 to 50, inclusive, has an equal chance of being chosen.

For example, a run of the simulation might predict that for a certain day under normal conditions, each bird coming to the feeder will eat 11 grams of food. If 10 birds come to the feeder on that day, then a total of 110 grams of food will be consumed.

If the simulated food consumed is greater than the amount of food in the feeder, the birds empty the feeder and the amount of food in the feeder at the end of the day is zero.

Under abnormal conditions, a bear empties the feeder and the amount of food in the feeder at the end of the day is zero.

The following examples show possible results of three calls to `simulateOneDay`.

- Example 1: If the feeder initially contains 500 grams of food, the call `simulateOneDay(12)` could result in 12 birds eating 20 grams of food each, leaving 260 grams of food in the feeder.
- Example 2: If the feeder initially contains 1,000 grams of food, the call `simulateOneDay(22)` could result in a bear eating all the food, leaving 0 grams of food in the feeder.
- Example 3: If the feeder initially contains 100 grams of food, the call `simulateOneDay(5)` could result in 5 birds attempting to eat 30 grams of food each. Since the feeder initially contains less than 150 grams of food, the feeder is emptied, leaving 0 grams of food in the feeder.

GO ON TO THE NEXT PAGE.

Complete the `simulateOneDay` method.

```
/**
 *  Simulates one day with numBirds birds or possibly a bear at the bird feeder,
 *  as described in part (a)
 *  Precondition: numBirds > 0
 */
public void simulateOneDay(int numBirds)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class Feeder

private int currentFood

public void simulateOneDay(int numBirds)
public int simulateManyDays(int numBirds, int numDays)
```

GO ON TO THE NEXT PAGE.

(b) Write the `simulateManyDays` method. The method uses `simulateOneDay` to simulate `numBirds` birds or a bear coming to the feeder on at most `numDays` consecutive days. The simulation returns the number of days that birds or a bear found food at the feeder.

Consider the following examples.

Value of `currentFood` and Method Call	**Possible Outcomes and Resulting Return Value**

`currentFood: 2400`
 	
`simulateManyDays(10, 4)`
	Day 1: `simulateOneDay` leaves 2100 grams of food in the feeder.
	Day 2: `simulateOneDay` leaves 1650 grams of food in the feeder.
	Day 3: `simulateOneDay` leaves 1500 grams of food in the feeder.
	Day 4: `simulateOneDay` leaves 1260 grams of food in the feeder.
	
	The simulation returns `4` because, on all four days of the simulation, birds or a bear found food at the feeder. The instance variable `currentFood` has the value `1260`.

`currentFood: 250`
 	
`simulateManyDays(10, 5)`
	Day 1: `simulateOneDay` leaves 150 grams of food in the feeder.
	Day 2: `simulateOneDay` leaves 0 grams of food in the feeder.
	
	The simulation returns `2` because, on two of the five simulated days, birds or a bear found food at the feeder. The instance variable `currentFood` has the value `0`.

`currentFood: 0`
 	
`simulateManyDays(5, 10)`
	The simulation returns `0` because no food was found at the feeder on any day. The instance variable `currentFood` has the value `0`.

GO ON TO THE NEXT PAGE.

Complete the `simulateManyDays` method. Assume that `simulateOneDay` works as intended, regardless of what you wrote in part (a). You must use `simulateOneDay` appropriately in order to receive full credit.

```
/**
 *  Returns the number of days birds or a bear found food to eat at the feeder in this simulation,
 *  as described in part (b)
 *  Preconditions: numBirds > 0, numDays > 0
 */
public int simulateManyDays(int numBirds, int numDays)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class Feeder

private int currentFood

public void simulateOneDay(int numBirds)
public int simulateManyDays(int numBirds, int numDays)
```

1. This question involves the `WordMatch` class, which stores a secret string and provides methods that compare other strings to the secret string. You will write two methods in the `WordMatch` class.

```
public class WordMatch
{
    /** The secret string. */
    private String secret;

    /** Constructs a WordMatch object with the given secret string of lowercase letters. */
    public WordMatch(String word)
    {
        /* implementation not shown */
    }

    /** Returns a score for guess, as described in part (a).
     *  Precondition: 0 < guess.length() <= secret.length()
     */
    public int scoreGuess(String guess)
    {  /* to be implemented in part (a) */  }

    /** Returns the better of two guesses, as determined by scoreGuess and the rules for a
     *  tie-breaker that are described in part (b).
     *  Precondition: guess1 and guess2 contain all lowercase letters.
     *                guess1 is not the same as guess2.
     */
    public String findBetterGuess(String guess1, String guess2)
    {  /* to be implemented in part (b) */  }
}
```

(a) Write the `WordMatch` method `scoreGuess`. To determine the score to be returned, `scoreGuess` finds the number of times that `guess` occurs as a substring of `secret` and then multiplies that number by the square of the length of `guess`. Occurrences of `guess` may overlap within `secret`.

Assume that the length of `guess` is less than or equal to the length of `secret` and that `guess` is not an empty string.

The following examples show declarations of a `WordMatch` object. The tables show the outcomes of some possible calls to the `scoreGuess` method.

```
WordMatch game = new WordMatch("mississippi");
```

Value of `guess`	**Number of Substring Occurrences**	**Score Calculation**: (**Number of Substring Occurrences**) x (**Square of the Length of** `guess`)	**Return Value of** `game.scoreGuess(guess)`
`"i"`	4	`4 * 1 * 1 = 4`	4
`"iss"`	2	`2 * 3 * 3 = 18`	18
`"issipp"`	1	`1 * 6 * 6 = 36`	36
`"mississippi"`	1	`1 * 11 * 11 = 121`	121

```
WordMatch game = new WordMatch("aaaabb");
```

Value of `guess`	**Number of Substring Occurrences**	**Score Calculation**: (**Number of Substring Occurrences**) x (**Square of the Length of** `guess`)	**Return Value of** `game.scoreGuess(guess)`
`"a"`	4	`4 * 1 * 1 = 4`	4
`"aa"`	3	`3 * 2 * 2 = 12`	12
`"aaa"`	2	`2 * 3 * 3 = 18`	18
`"aabb"`	1	`1 * 4 * 4 = 16`	16
`"c"`	0	`0 * 1 * 1 = 0`	0

Complete the `scoreGuess` method.

```
/** Returns a score for guess, as described in part (a).
 *  Precondition: 0 < guess.length() <= secret.length()
 */
public int scoreGuess(String guess)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class WordMatch

private String secret

public WordMatch(String word)
public int scoreGuess(String guess)
public String findBetterGuess(String guess1, String guess2)
```

(b) Write the `WordMatch` method `findBetterGuess`, which returns the better guess of its two `String` parameters, `guess1` and `guess2`. If the `scoreGuess` method returns different values for `guess1` and `guess2`, then the guess with the higher score is returned. If the `scoreGuess` method returns the same value for `guess1` and `guess2`, then the alphabetically greater guess is returned.

The following example shows a declaration of a `WordMatch` object and the outcomes of some possible calls to the `scoreGuess` and `findBetterGuess` methods.

```
WordMatch game = new WordMatch("concatenation");
```

Method Call	Return Value	Explanation
`game.scoreGuess("ten");`	`9`	`1 * 3 * 3`
`game.scoreGuess("nation");`	`36`	`1 * 6 * 6`
`game.findBetterGuess("ten", "nation");`	`"nation"`	Since `scoreGuess` returns `36` for `"nation"` and `9` for `"ten"`, the guess with the greater score, `"nation"`, is returned.
`game.scoreGuess("con");`	`9`	`1 * 3 * 3`
`game.scoreGuess("cat");`	`9`	`1 * 3 * 3`
`game.findBetterGuess("con", "cat");`	`"con"`	Since `scoreGuess` returns `9` for both `"con"` and `"cat"`, the alphabetically greater guess, `"con"`, is returned.

Complete method `findBetterGuess`.

Assume that `scoreGuess` works as specified, regardless of what you wrote in part (a). You must use `scoreGuess` appropriately to receive full credit.

```
/** Returns the better of two guesses, as determined by scoreGuess and the rules for a
 *  tie-breaker that are described in part (b).
 *  Precondition: guess1 and guess2 contain all lowercase letters.
 *                guess1 is not the same as guess2.
 */
public String findBetterGuess(String guess1, String guess2)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class WordMatch

private String secret

public WordMatch(String word)
public int scoreGuess(String guess)
public String findBetterGuess(String guess1, String guess2)
```

AP COMPUTER SCIENCE A • EXERCISE SHEET

2.4 Nested if Statements

Name: _____ Class: _____ Date: _____ **Total: 8 marks**

KEY WORDS if statement 条件语句

Objective

Build the skills to answer exam questions on **nested if statements**.

You must be able to:

- write a **nested if** 嵌套条件 by placing one `if` inside another
- use an **else-if chain** to choose among several mutually exclusive cases
- understand that only one branch of an else-if chain runs for each input

1 Worked examples

Study these first. Each one shows the method for a question type used later.

■ Nested ifs and else-if chains

A **nested if** places one `if` inside another. An **else-if chain** tests conditions in order and runs the **first** one that is true:

```
if (s >= 90)      grade = "A";
else if (s >= 80) grade = "B";
else              grade = "C";
```

For any single input, **exactly one** branch runs.

■ Tracing a value

Trace `s = 85` through the chain: is `s >= 90`? No. Is `s >= 80`? Yes, so `grade = "B"` and the rest is skipped. Order matters – put the **strictest** test first, or a looser test will catch the value early.

2 Practice

2.1 State what a nested if is. [1]

2.2 State how many branches of an else-if chain run for one input. [1]

2.3 For the grade example above, state the result when `s` is 82. [1]

3 Exam-style questions

3.1 An else-if chain lets you choose among [1]

- **A** one case
- **B** exactly two cases
- **C** several mutually exclusive cases
- **D** no cases

3.2 For one input, the number of branches of an else-if chain that run is [1]

- **A** zero
- **B** one
- **C** two
- **D** all of them

3.3 `if (m >= 50) { if (m >= 75) r = "distinction"; else r = "pass"; } else r = "fail";`.

(a) State `r` when `m` is 90. [1]

(b) State `r` when `m` is 55. [1]

(c) State `r` when `m` is 30. [1]

Solutions

2.1 an if statement placed inside another if statement.

2.2 one.

2.3 `B`.

3.1 C.

3.2 B.

3.3 (a) `distinction`. (b) `pass`. (c) `fail`.

AP COMPUTER SCIENCE A • EXERCISE SHEET

2.5 Compound Boolean Expressions

Name: _____ Class: _____ Date: _____ **Total: 8 marks**

KEY WORDS short-circuit evaluation 短路求值 • logical operators 逻辑运算符 • boolean expression 布尔表达式

Objective

Build the skills to answer exam questions on **compound Boolean expressions**.

You must be able to:

- combine conditions with **AND** (`&&`) and **OR** (`||`)
- use **NOT** (`!`) to reverse a Boolean value
- understand **short-circuit evaluation** 短路求值

1 Worked examples

Study these first. Each one shows the method for a question type used later.

■ Logical operators

- `&&` (**AND**) is `true` only when **both** sides are true.
- `||` (**OR**) is `true` when **either** side is true.
- `!` (**NOT**) reverses the value: `!true` is `false`.

■ Short-circuit evaluation

`&&` stops as soon as the left side is `false`; `||` stops as soon as the left side is `true` — the right side is not even checked.

2 Practice

2.1 State what the `&&` operator requires in order to be `true`. [1]

2.2 Evaluate `(5 > 3) || (2 > 8)`. [1]

2.3 State what short-circuit evaluation means. [1]

---

3 Exam-style questions

3.1 The logical AND operator in Java is [1]

- **A** `&`
- **B** `&&`
- **C** `AND`
- **D** `+`

3.2 `!(true)` evaluates to [1]

- **A** `true`
- **B** `false`
- **C** `1`
- **D** an error

3.3 Given `int a = 4, b = 0;`:

(a) evaluate `(a > 0) && (b > 0)`. [1]

(b) evaluate `(a > 0) || (b > 0)`. [1]

(c) State what the `!` operator does. [1]

Solutions

2.1 both of its sides must be `true`.

2.2 `true` (the left side is true).

2.3 `&&` and `||` stop evaluating as soon as the result is known.

3.1 B.

3.2 B.

3.3 (a) `false`. (b) `true`. (c) it reverses the Boolean value.

AP COMPUTER SCIENCE A • EXERCISE SHEET

2.6 Comparing Boolean Expressions

Name: _____ Class: _____ Date: _____ **Total: 8 marks**

KEY WORDS equivalent 等价 • de morgan's laws 德摩根定律 • boolean expression 布尔表达式

Objective

Build the skills to answer exam questions on **comparing Boolean expressions**.

You must be able to:

- determine whether two Boolean expressions are **equivalent** 等价
- apply **De Morgan's Laws** 德摩根定律 to rewrite `!(a && b)` and `!(a || b)`
- simplify a compound Boolean expression

1 Worked examples

Study these first. Each one shows the method for a question type used later.

■ Equivalent expressions

Two Boolean expressions are **equivalent** if they always give the **same** result for every input.

■ De Morgan's Laws

$$!(a \;\&\&\; b) = !a \;||\; !b, \qquad !(a \;||\; b) = !a \;\&\&\; !b.$$

■ Simplifying

`!(x > 5)` is simply `x <= 5`.

2 Practice

2.1 State what it means for two Boolean expressions to be equivalent. [1]

2.2 Rewrite `!(a && b)` using De Morgan's Law. [1]

2.3 Rewrite `!(x > 5)` as a simpler comparison. [1]

---

3 Exam-style questions

3.1 `!(a || b)` is equivalent to [1]

- **A** `!a || !b`
- **B** `!a && !b`
- **C** `a && b`
- **D** `a || b`

3.2 Two Boolean expressions are equivalent if they [1]

- **A** look the same
- **B** always give the same result
- **C** are both `true`
- **D** both use `&&`

3.3 Simplify or rewrite the following.

(a) Rewrite `!(p && q)`. [1]

(b) Rewrite `!(x >= 10)` as a comparison. [1]

(c) State whether `!(true)` equals `false`. [1]

Solutions

2.1 they give the same result for every possible input.

2.2 `!a || !b`.

2.3 `x <= 5`.

3.1 B.

3.2 B.

3.3 (a) `!p || !q`. (b) `x < 10`. (c) yes.

AP COMPUTER SCIENCE A • EXERCISE SHEET

2.7 while Loops

Name: _____ Class: _____ Date: _____ **Total: 9 marks**

KEY WORDS while loop 当型循环 • loop control variable 循环控制变量 • infinite loop 无限循环

Objective

Build the skills to answer exam questions on **while loops**.

You must be able to:

- write a **while loop** 当型循环 that repeats while its condition is `true`
- understand that the condition is tested **before** each pass
- update a **loop control variable** 循环控制变量 so the loop makes progress

1 Worked examples

Study these first. Each one shows the method for a question type used later.

■ while loop

```
int i = 0, sum = 0;
while (i < 3) {   // tested before each pass
    sum += i;
    i++;          // update, or the loop never ends
}
```

■ Progress toward stopping

The **loop control variable** must change so the condition eventually becomes `false`; otherwise the loop runs forever (an infinite loop).

2 Practice

2.1 State when a while loop tests its condition. [1]

2.2 State what happens if the loop control variable is never updated. [1]

2.3 Trace: `int i = 0, sum = 0; while (i < 3) { sum += i; i++; }`. State the final `sum`. [2]

3 Exam-style questions

3.1 A while loop tests its condition [1]

- **A** after each pass
- **B** before each pass
- **C** never
- **D** only once at the end

3.2 A loop whose control variable never changes may become [1]

- **A** a for loop
- **B** an infinite loop
- **C** a method
- **D** a cast

3.3 `int n = 1; while (n <= 4) { System.out.print(n); n++; }`.

(a) State the output. [1]

(b) State the final value of `n`. [1]

(c) State when the condition is checked. [1]

Solutions

2.1 before each pass through the loop.

2.2 the loop never stops (an infinite loop).

2.3 $0 + 1 + 2 = 3$.

3.1 B.

3.2 B.

3.3 (a) `1234`. (b) `5`. (c) before each pass.

1. This question simulates birds or possibly a bear eating at a bird feeder. The following `Feeder` class contains information about how much food is in the bird feeder and simulates how much food is eaten. You will write two methods of the `Feeder` class.

```
public class Feeder
{
   /**
    *  The amount of food, in grams, currently in the bird feeder; initialized in the constructor and
    *  always greater than or equal to zero
    */
   private int currentFood;

   /**
    *  Simulates one day with numBirds birds or possibly a bear at the bird feeder,
    *  as described in part (a)
    *  Precondition: numBirds > 0
    */
   public void simulateOneDay(int numBirds)
   {  /* to be implemented in part (a) */  }

   /**
    *  Returns the number of days birds or a bear found food to eat at the feeder in this simulation,
    *  as described in part (b)
    *  Preconditions: numBirds > 0, numDays > 0
    */
   public int simulateManyDays(int numBirds, int numDays)
   {  /* to be implemented in part (b) */  }

   // There may be instance variables, constructors, or methods that are not shown.
}
```

GO ON TO THE NEXT PAGE.

(a) Write the `simulateOneDay` method, which simulates `numBirds` birds or possibly a bear at the feeder for one day. The method determines the amount of food taken from the feeder on this day and updates the `currentFood` instance variable. The simulation accounts for normal conditions, which occur 95% of the time, and abnormal conditions, which occur 5% of the time.

Under normal conditions, the simulation assumes that on any given day, only birds visit the feeder and that each bird at the feeder consumes the same amount of food. This standard amount consumed is between 10 and 50 grams of food, inclusive, in 1-gram increments. That is, on any given day, each bird might eat 10, 11, . . . , 49, or 50 grams of food. The amount of food eaten by each bird on a given day is randomly generated and each integer from 10 to 50, inclusive, has an equal chance of being chosen.

For example, a run of the simulation might predict that for a certain day under normal conditions, each bird coming to the feeder will eat 11 grams of food. If 10 birds come to the feeder on that day, then a total of 110 grams of food will be consumed.

If the simulated food consumed is greater than the amount of food in the feeder, the birds empty the feeder and the amount of food in the feeder at the end of the day is zero.

Under abnormal conditions, a bear empties the feeder and the amount of food in the feeder at the end of the day is zero.

The following examples show possible results of three calls to `simulateOneDay`.

- Example 1: If the feeder initially contains 500 grams of food, the call `simulateOneDay(12)` could result in 12 birds eating 20 grams of food each, leaving 260 grams of food in the feeder.
- Example 2: If the feeder initially contains 1,000 grams of food, the call `simulateOneDay(22)` could result in a bear eating all the food, leaving 0 grams of food in the feeder.
- Example 3: If the feeder initially contains 100 grams of food, the call `simulateOneDay(5)` could result in 5 birds attempting to eat 30 grams of food each. Since the feeder initially contains less than 150 grams of food, the feeder is emptied, leaving 0 grams of food in the feeder.

GO ON TO THE NEXT PAGE.

Complete the `simulateOneDay` method.

```
/**
 *  Simulates one day with numBirds birds or possibly a bear at the bird feeder,
 *  as described in part (a)
 *  Precondition: numBirds > 0
 */
public void simulateOneDay(int numBirds)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class Feeder

private int currentFood

public void simulateOneDay(int numBirds)
public int simulateManyDays(int numBirds, int numDays)
```

GO ON TO THE NEXT PAGE.

(b) Write the `simulateManyDays` method. The method uses `simulateOneDay` to simulate `numBirds` birds or a bear coming to the feeder on at most `numDays` consecutive days. The simulation returns the number of days that birds or a bear found food at the feeder.

Consider the following examples.

Value of `currentFood` and Method Call	**Possible Outcomes and Resulting Return Value**

`currentFood: 2400`
 	
`simulateManyDays(10, 4)`
	Day 1: `simulateOneDay` leaves 2100 grams of food in the feeder.
	Day 2: `simulateOneDay` leaves 1650 grams of food in the feeder.
	Day 3: `simulateOneDay` leaves 1500 grams of food in the feeder.
	Day 4: `simulateOneDay` leaves 1260 grams of food in the feeder.
	
	The simulation returns `4` because, on all four days of the simulation, birds or a bear found food at the feeder. The instance variable `currentFood` has the value `1260`.

`currentFood: 250`
 	
`simulateManyDays(10, 5)`
	Day 1: `simulateOneDay` leaves 150 grams of food in the feeder.
	Day 2: `simulateOneDay` leaves 0 grams of food in the feeder.
	
	The simulation returns `2` because, on two of the five simulated days, birds or a bear found food at the feeder. The instance variable `currentFood` has the value `0`.

`currentFood: 0`
 	
`simulateManyDays(5, 10)`
	The simulation returns `0` because no food was found at the feeder on any day. The instance variable `currentFood` has the value `0`.

GO ON TO THE NEXT PAGE.

Complete the `simulateManyDays` method. Assume that `simulateOneDay` works as intended, regardless of what you wrote in part (a). You must use `simulateOneDay` appropriately in order to receive full credit.

```
/**
 *  Returns the number of days birds or a bear found food to eat at the feeder in this simulation,
 *  as described in part (b)
 *  Preconditions: numBirds > 0, numDays > 0
 */
public int simulateManyDays(int numBirds, int numDays)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class Feeder

private int currentFood

public void simulateOneDay(int numBirds)
public int simulateManyDays(int numBirds, int numDays)
```

4. This question involves a path through a two-dimensional (2D) array of integers, where the path is based on the values of elements in the array. When an element of the 2D array is accessed, the first index is used to specify the row and the second index is used to specify the column. The following `Location` class represents a row and column position in the 2D array.

```
public class Location
{
   private int theRow;
   private int theCol;

   public Location(int r, int c)
   {
      theRow = r;
      theCol = c;
   }

   public int getRow()
   {  return theRow;  }

   public int getCol()
   {  return theCol;  }
}
```

The following `GridPath` class contains the 2D array and methods to use to determine a path through the array. You will write two methods of the `GridPath` class.

```
public class GridPath
{
    /** Initialized in the constructor with distinct values that never change */
    private int[][] grid;

    /**
     *  Returns the Location representing a neighbor of the grid element at row and col,
     *  as described in part (a)
     *  Preconditions: row is a valid row index and col is a valid column index in grid.
     *       row and col do not specify the element in the last row and last column of grid.
     */
    public Location getNextLoc(int row, int col)
    {  /* to be implemented in part (a) */  }

    /**
     *  Computes and returns the sum of all values on a path through grid, as described in
     *  part (b)
     *  Preconditions: row is a valid row index and col is a valid column index in grid.
     *       row and col do not specify the element in the last row and last column of grid.
     */
    public int sumPath(int row, int col)
    {  /* to be implemented in part (b) */  }

    // There may be instance variables, constructors, and methods that are not shown.
}
```

(a) Write the `getNextLoc` method, which returns a `Location` object that represents the smaller of two neighbors of the `grid` element at `row` and `col`, according to the following rules.

- The two neighbors that are considered are the element below the given element and the element to the right of the given element, if they exist.
- If both neighbors exist, the `Location` of the neighbor with the smaller value is returned. Two neighbors will always have different values.
- If only one neighbor exists, the `Location` of the existing neighbor is returned.

For example, assume that `grid` contains the following values.

	0	1	2	3	4
0	12	3	4	13	5
1	11	21	2	14	16
2	7	8	9	15	0
3	10	17	20	19	1
4	18	22	30	25	6

The following table shows some sample calls to `getNextLoc`.

Method Call	Explanation
`getNextLoc(0, 0)`	Returns the neighbor to the right (the `Location` representing the element at row 0 and column 1), since `3 < 11`
`getNextLoc(1, 3)`	Returns the neighbor below (the `Location` representing the element at row 2 and column 3), since `15 < 16`
`getNextLoc(2, 4)`	Returns the neighbor below (the `Location` representing the element at row 3 and column 4), since the given element has no neighbor to the right
`getNextLoc(4, 3)`	Returns the neighbor to the right (the `Location` representing the element at row 4 and column 4), since the given element has no neighbor below

In the example, the `getNextLoc` method will never be called with row `4` and column `4`, as those values would violate the precondition of the method.

Complete the `getNextLoc` method.

```
/**
 *  Returns the Location representing a neighbor of the grid element at row and col,
 *  as described in part (a)
 *  Preconditions: row is a valid row index and col is a valid column index in grid.
 *       row and col do not specify the element in the last row and last column of grid.
 */
public Location getNextLoc(int row, int col)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class Location

private int theRow
private int theCol

public Location(int r, int c)
public int getRow()
public int getCol()

public class GridPath

private int[][] grid

public Location getNextLoc(int row, int col)
public int sumPath(int row, int col)
```

(b) Write the `sumPath` method, which returns the sum of all values on a path in `grid`. The path begins with the element at `row` and `col` and is determined by successive calls to `getNextLoc`. The path ends when the element in the last row and the last column of `grid` is reached.

For example, consider the following contents of `grid`. The shaded elements of `grid` represent the values on the path that results from the method call `sumPath(1, 1)`. The method call returns `19` because 3 + 2 + 9 + 4 + 0 + 1 = 19.

	0	1	2	3	4
0	12	30	40	25	5
1	11	3	22	15	43
2	7	2	9	4	0
3	8	33	18	6	1

Write the `sumPath` method. Assume `getNextLoc` works as intended, regardless of what you wrote in part (a). You must use `getNextLoc` appropriately in order to receive full credit.

```
/**
 * Computes and returns the sum of all values on a path through grid, as described in
 * part (b)
 * Preconditions: row is a valid row index and col is a valid column index in grid.
 *      row and col do not specify the element in the last row and last column of grid.
 */
public int sumPath(int row, int col)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class Location

private int theRow
private int theCol

public Location(int r, int c)
public int getRow()
public int getCol()

public class GridPath

private int[][] grid

public Location getNextLoc(int row, int col)
public int sumPath(int row, int col)
```

STOP

END OF EXAM

1. This question involves reasoning about a simulation of a frog hopping in a straight line. The frog attempts to hop to a goal within a specified number of hops. The simulation is encapsulated in the following `FrogSimulation` class. You will write two of the methods in this class.

```
public class FrogSimulation
{
    /** Distance, in inches, from the starting position to the goal. */
    private int goalDistance;

    /** Maximum number of hops allowed to reach the goal. */
    private int maxHops;


    /** Constructs a FrogSimulation where dist is the distance, in inches, from the starting
     *  position to the goal, and numHops is the maximum number of hops allowed to reach the goal.
     *  Precondition: dist > 0; numHops > 0
     */
    public FrogSimulation(int dist, int numHops)
    {
       goalDistance = dist;
       maxHops = numHops;
    }


    /** Returns an integer representing the distance, in inches, to be moved when the frog hops.
     */
    private int hopDistance()
    {  /* implementation not shown */  }


    /** Simulates a frog attempting to reach the goal as described in part (a).
     *  Returns true if the frog successfully reached or passed the goal during the simulation;
     *          false otherwise.
     */
    public boolean simulate()
    {  /* to be implemented in part (a) */  }


    /** Runs num simulations and returns the proportion of simulations in which the frog
     *  successfully reached or passed the goal.
     *  Precondition: num > 0
     */
    public double runSimulations(int num)
    {  /* to be implemented in part (b) */  }
```

(a) Write the `simulate` method, which simulates the frog attempting to hop in a straight line to a goal from the frog's starting position of 0 within a maximum number of hops. The method returns `true` if the frog successfully reached the goal within the maximum number of hops; otherwise, the method returns `false`.

The `FrogSimulation` class provides a method called `hopDistance` that returns an integer representing the distance (positive or negative) to be moved when the frog hops. A positive distance represents a move toward the goal. A negative distance represents a move away from the goal. The returned distance may vary from call to call. Each time the frog hops, its position is adjusted by the value returned by a call to the `hopDistance` method.

The frog hops until one of the following conditions becomes true:

- The frog has reached or passed the goal.
- The frog has reached a negative position.
- The frog has taken the maximum number of hops without reaching the goal.

The following example shows a declaration of a `FrogSimulation` object for which the goal distance is 24 inches and the maximum number of hops is 5. The table shows some possible outcomes of calling the `simulate` method.

```
FrogSimulation sim = new FrogSimulation(24, 5);
```

	Values returned by `hopDistance()`	Final position of frog	Return value of `sim.simulate()`
Example 1	`5, 7, -2, 8, 6`	`24`	`true`
Example 2	`6, 7, 6, 6`	`25`	`true`
Example 3	`6, -6, 31`	`31`	`true`
Example 4	`4, 2, -8`	`-2`	`false`
Example 5	`5, 4, 2, 4, 3`	`18`	`false`

Class information for this question

```
public class FrogSimulation

private int goalDistance
private int maxHops

private int hopDistance()
public boolean simulate()
public double runSimulations(int num)
```

Complete method `simulate` below. You must use `hopDistance` appropriately to receive full credit.

```
/** Simulates a frog attempting to reach the goal as described in part (a).
 *  Returns true if the frog successfully reached or passed the goal during the simulation;
```

(b) Write the `runSimulations` method, which performs a given number of simulations and returns the proportion of simulations in which the frog successfully reached or passed the goal. For example, if the parameter passed to `runSimulations` is `400`, and 100 of the 400 `simulate` method calls returned `true`, then the `runSimulations` method should return `0.25`.

Complete method `runSimulations` below. Assume that `simulate` works as specified, regardless of what you wrote in part (a). You must use `simulate` appropriately to receive full credit.

```
/** Runs num simulations and returns the proportion of simulations in which the frog
 *  successfully reached or passed the goal.
```

AP COMPUTER SCIENCE A • EXERCISE SHEET

2.8 for Loops

Name: _____ Class: _____ Date: _____ **Total: 9 marks**

KEY WORDS for loop 计数循环 • while loop 循环

Objective

Build the skills to answer exam questions on **for loops**.

You must be able to:

- write a **for loop** 计数循环 with an initialization, a condition, and an update
- understand how a for loop packs the setup, test, and change into one line
- rewrite a while loop as an equivalent for loop

1 Worked examples

Study these first. Each one shows the method for a question type used later.

for loop

```
for (int i = 0; i < 5; i++) {
    System.out.print(i);   // prints 01234
}
```

The header has three parts: **initialization** (`int i = 0`), **condition** (`i < 5`), and **update** (`i++`).

Equivalent while

The same logic could be written as a while loop; the for loop just gathers the setup, test, and change in one place.

2 Practice

2.1 Name the three parts of a for loop header. [2]

2.2 State how many times `for (int i = 0; i < 5; i++)` runs its body. [1]

2.3 State one advantage of a for loop over a while loop. [1]

3 Exam-style questions

3.1 The three parts of a for loop header are initialization, condition, and [1]

- **A** body
- **B** update
- **C** return
- **D** cast

3.2 `for (int i = 1; i <= 3; i++)` runs its body [1]

- **A** 2
- **B** 3
- **C** 4
- **D** infinitely many

times.

3.3 `for (int k = 0; k < 4; k++) System.out.print(k);`.

(a) State the output. [1]

(b) State how many times the body runs. [1]

(c) State the value of `k` that first stops the loop. [1]

Solutions

2.1 initialization, condition, update.

2.2 5 times.

2.3 it keeps the setup, test, and change together in one line.

3.1 B.

3.2 B.

3.3 (a) `0123`. (b) 4 times. (c) `k = 4`.

1. This question involves dog walkers who are paid through a dog-walking company to take dogs for one-hour walks. A dog-walking company has a varying number of dogs that need to be walked during each hour of the day. A dog-walking company is represented by the following `DogWalkCompany` class.

```
public class DogWalkCompany
{
  /**
   * Returns the number of dogs, always greater than 0, that are available
   * for a walk during the time specified by hour
   * Precondition: 0 <= hour <= 23
   */
  public int numAvailableDogs(int hour)
  { /* implementation not shown */ }

  /**
   * Decreases, by numberDogsWalked, the number of dogs available for a walk
   * during the time specified by hour
   * Preconditions: 0 <= hour <= 23
   *                numberDogsWalked > 0
   */
  public void updateDogs(int hour, int numberDogsWalked)
  { /* implementation not shown */ }

  /* There may be instance variables, constructors,
     and methods that are not shown. */
}
```

A dog walker is associated with a dog-walking company and is represented by the `DogWalker` class. You will write two methods of the `DogWalker` class.

```
public class DogWalker
{
  /** The maximum number of dogs this walker can walk simultaneously
      per hour */
  private int maxDogs;

  /** The dog-walking company this dog walker is associated with */
  private DogWalkCompany company;

  /**
   * Assigns max to maxDogs and comp to company
   * Precondition: max > 0
   */
  public DogWalker(int max, DogWalkCompany comp)
  { /* implementation not shown */ }

  /**
   * Takes at least one dog for a walk during the time specified by
   * hour, as described in part (a)
   * Preconditions: 0 <= hour <= 23
   *                maxDogs > 0
   */
  public int walkDogs(int hour)
  { /* to be implemented in part (a) */ }
```

```
  /**
   * Performs an entire dog-walking shift and returns the amount
   * earned, in dollars, as described in part (b)
   * Preconditions: 0 <= startHour <= endHour <= 23
   *                maxDogs > 0
   */
  public int dogWalkShift(int startHour, int endHour)
  { /* to be implemented in part (b) */ }

  /* There may be instance variables, constructors,
     and methods that are not shown. */
}
```

A. Write the `walkDogs` method, which updates and returns the number of dogs this dog walker walks during the time specified by `hour`. Values of `hour` range from `0` to `23`, inclusive.

A helper method, `numAvailableDogs`, has been provided in the `DogWalkCompany` class. The method returns the number of dogs available to be taken for a walk at a given hour. The dog walker will always walk as many dogs as the dog-walking company has available to walk, as long as the available number of dogs is not greater than the maximum number of dogs that the dog walker can handle, represented by the value of `maxDogs`.

Another helper method, `updateDogs`, has also been provided in the `DogWalkCompany` class. So that multiple dog walkers do not sign up to walk the same dogs, the `walkDogs` method should use `updateDogs` to update the dog-walking company with the number of dogs this dog walker will walk during the given hour. The parameters of the `updateDogs` method indicate how many dogs this dog walker will walk during the time specified by `hour`.

For example, if the dog-walking company has 10 dogs that need to be walked at the given hour but the dog walker's maximum is 4, the `updateDogs` method should be used to indicate that this dog walker will walk 4 of the 10 dogs during the given hour. As another example, if the dog-walking company has 3 dogs that need to be walked at the given hour and the dog walker's maximum is 4, the `updateDogs` method should be used to indicate that this dog walker will walk all 3 of the available dogs during the given hour.

The `walkDogs` method should return the number of dogs to be walked by this dog walker during the time specified by `hour`.

Complete method `walkDogs`. You must use `numAvailableDogs` and `updateDogs` appropriately to receive full credit.

```
/**
 * Takes at least one dog for a walk during the time specified by
 * hour, as described in part (a)
 * Preconditions: 0 <= hour <= 23
 *                maxDogs > 0
 */
public int walkDogs(int hour)
```

B. Write the `dogWalkShift` method, which performs a dog-walking shift of the hours in the range `startHour` to `endHour`, inclusive, and returns the total amount earned. For example, a dog-walking shift from `14` to `16` is composed of three one-hour dog walks starting at hours `14`, `15`, and `16`.

For each hour, the base pay is $5 per dog walked plus a bonus of $3 if at least one of the following is true.

- `maxDogs` dogs are walked
- the walk occurs between the peak hours of 9 and 17, inclusive

The following table shows an example of the calculated amount earned by walking dogs from hour 7 through hour 10, inclusive.

Hour	Maximum Number of Dogs	Number of Dogs Walked	Amount Earned, in Dollars
7	3	3	3 × 5 + 3 = 18
8	3	2	2 × 5 = 10
9	3	2	2 × 5 + 3 = 13
10	3	3	3 × 5 + 3 = 18
Total			59

Complete method `dogWalkShift`. Assume that `walkDogs` works as specified, regardless of what you wrote in part (a). You must use `walkDogs` appropriately to earn full credit.

```
/**
 * Performs an entire dog-walking shift and returns the amount
 * earned, in dollars, as described in part (b)
 * Preconditions: 0 <= startHour <= endHour <= 23
 *                maxDogs > 0
 */
public int dogWalkShift(int startHour, int endHour)
```

1. This question involves simulation of the play and scoring of a single-player video game. In the game, a player attempts to complete three levels. A level in the game is represented by the `Level` class.

```
public class Level
{
   /** Returns true if the player reached the goal on this level and returns false otherwise */
   public boolean goalReached()
   {  /* implementation not shown */  }

   /** Returns the number of points (a positive integer) recorded for this level */
   public int getPoints()
   {  /* implementation not shown */  }

   // There may be instance variables, constructors, and methods that are not shown.
}
```

GO ON TO THE NEXT PAGE.

Play of the game is represented by the `Game` class. You will write two methods of the `Game` class.

```
public class Game
{
   private Level levelOne;
   private Level levelTwo;
   private Level levelThree;

   /** Postcondition: All instance variables have been initialized. */
   public Game()
   {  /* implementation not shown */  }

   /** Returns true if this game is a bonus game and returns false otherwise */
   public boolean isBonus()
   {  /* implementation not shown */  }

   /** Simulates the play of this Game (consisting of three levels) and updates all relevant
    *  game data
    */
   public void play()
   {  /* implementation not shown */  }

   /** Returns the score earned in the most recently played game, as described in part (a) */
   public int getScore()
   {  /* to be implemented in part (a) */  }

   /** Simulates the play of num games and returns the highest score earned, as
    *  described in part (b)
    *  Precondition: num > 0
    */
   public int playManyTimes(int num)
   {  /* to be implemented in part (b) */  }

   // There may be instance variables, constructors, and methods that are not shown.
}
```

GO ON TO THE NEXT PAGE.

(a) Write the `getScore` method, which returns the score for the most recently played game. Each game consists of three levels. The score for the game is computed using the following helper methods.

- The `isBonus` method of the `Game` class returns `true` if this is a bonus game and returns `false` otherwise.
- The `goalReached` method of the `Level` class returns `true` if the goal has been reached on a particular level and returns `false` otherwise.
- The `getPoints` method of the `Level` class returns the number of points recorded on a particular level. Whether or not recorded points are earned (included in the game score) depends on the rules of the game, which follow.

The score for the game is computed according to the following rules.

- Level one points are earned only if the level one goal is reached. Level two points are earned only if both the level one and level two goals are reached. Level three points are earned only if the goals of all three levels are reached.
- The score for the game is the sum of the points earned for levels one, two, and three.
- If the game is a bonus game, the score for the game is tripled.

GO ON TO THE NEXT PAGE.

The following table shows some examples of game score calculations.

	Level One Results	Level Two Results	Level Three Results	`isBonus` Return Value	Score Calculation
`goalReached` **Return Value:** `getPoints` **Return Value:**	`true` `200`	`true` `100`	`true` `500`	`true`	(200 + 100 + 500) × 3 = 2,400 The recorded points for levels one, two, and three are earned because the goals were reached in all three levels. The earned points are multiplied by 3 because `isBonus` returns `true`.
`goalReached` **Return Value:** `getPoints` **Return Value:**	`true` `200`	`true` `100`	`false` `500`	`false`	200 + 100 = 300 The recorded points for level one and level two are earned because the goal was reached in levels one and two. The recorded points for level three are not earned because the goal was not reached in level three.
`goalReached` **Return Value:** `getPoints` **Return Value:**	`true` `200`	`false` `100`	`true` `500`	`true`	200 × 3 = 600 The recorded points for only level one are earned because the goal was not reached in level two. The earned points are multiplied by 3 because `isBonus` returns `true`.
`goalReached` **Return Value:** `getPoints` **Return Value:**	`false` `200`	`true` `100`	`true` `500`	`false`	0 Because the goal in level one was not reached, no points are earned for any level.

Complete the `getScore` method.

```
/** Returns the score earned in the most recently played game, as described in part (a) */
public int getScore()
```

Begin your response at the top of a new page in the Free Response booklet and fill in the appropriate circle indicating the question number.
If there are multiple parts to this question, write the part letter with your response.

GO ON TO THE NEXT PAGE.

(b) Write the `playManyTimes` method, which simulates the play of `num` games and returns the highest game score earned. For example, if the four plays of the game that are simulated as a result of the method call `playManyTimes(4)` earn scores of `75`, `50`, `90`, and `20`, then the method should return `90`.

Play of the game is simulated by calling the helper method `play`. Note that if `play` is called only one time followed by multiple consecutive calls to `getScore`, each call to `getScore` will return the score earned in the single simulated play of the game.

Complete the `playManyTimes` method. Assume that `getScore` works as intended, regardless of what you wrote in part (a). You must call `play` and `getScore` appropriately in order to receive full credit.

```
/** Simulates the play of num games and returns the highest score earned, as
 *  described in part (b)
 *  Precondition: num > 0
 */
public int playManyTimes(int num)
```

Begin your response at the top of a new page in the Free Response booklet and fill in the appropriate circle indicating the question number.
If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class Level

public boolean goalReached()
public int getPoints()

public class Game

private Level levelOne
private Level levelTwo
private Level levelThree

public Game()
public boolean isBonus()
public void play()
public int getScore()
public int playManyTimes(int num)
```

GO ON TO THE NEXT PAGE.

1. The `APCalendar` class contains methods used to calculate information about a calendar. You will write two methods of the class.

```
public class APCalendar
{

   /** Returns true if year is a leap year and false otherwise. */
   private static boolean isLeapYear(int year)
   {  /* implementation not shown */  }

   /** Returns the number of leap years between year1 and year2, inclusive.
    *  Precondition: 0 <= year1 <= year2
    */
   public static int numberOfLeapYears(int year1, int year2)
   {  /* to be implemented in part (a) */  }

   /** Returns the value representing the day of the week for the first day of year,
    *  where 0 denotes Sunday, 1 denotes Monday, ..., and 6 denotes Saturday.
    */
   private static int firstDayOfYear(int year)
   {  /* implementation not shown */  }

   /** Returns n, where month, day, and year specify the nth day of the year.
    *  Returns 1 for January 1 (month = 1, day = 1) of any year.
    *  Precondition: The date represented by month, day, year is a valid date.
    */
   private static int dayOfYear(int month, int day, int year)
   {  /* implementation not shown */  }

   /** Returns the value representing the day of the week for the given date
    *  (month, day, year), where 0 denotes Sunday, 1 denotes Monday, ...,
    *  and 6 denotes Saturday.
    *  Precondition: The date represented by month, day, year is a valid date.
    */
   public static int dayOfWeek(int month, int day, int year)
   {  /* to be implemented in part (b) */  }

   // There may be instance variables, constructors, and other methods not shown.

}
```

(a) Write the static method `numberOfLeapYears`, which returns the number of leap years between `year1` and `year2`, inclusive.

In order to calculate this value, a helper method is provided for you.

- `isLeapYear(year)` returns `true` if `year` is a leap year and `false` otherwise.

Complete method `numberOfLeapYears` below. You must use `isLeapYear` appropriately to receive full credit.

```
/** Returns the number of leap years between year1 and year2, inclusive.
 *  Precondition: 0 <= year1 <= year2
 */
public static int numberOfLeapYears(int year1, int year2)
```

(b) Write the static method `dayOfWeek`, which returns the integer value representing the day of the week for the given date (`month`, `day`, `year`), where `0` denotes Sunday, `1` denotes Monday, ..., and `6` denotes Saturday. For example, 2019 began on a Tuesday, and January 5 is the fifth day of 2019. As a result, January 5, 2019, fell on a Saturday, and the method call `dayOfWeek(1, 5, 2019)` returns `6`.

As another example, January 10 is the tenth day of 2019. As a result, January 10, 2019, fell on a Thursday, and the method call `dayOfWeek(1, 10, 2019)` returns `4`.

In order to calculate this value, two helper methods are provided for you.

- `firstDayOfYear(year)` returns the integer value representing the day of the week for the first day of `year`, where `0` denotes Sunday, `1` denotes Monday, ..., and `6` denotes Saturday. For example, since 2019 began on a Tuesday, `firstDayOfYear(2019)` returns `2`.
- `dayOfYear(month, day, year)` returns *n*, where `month`, `day`, and `year` specify the *n*th day of the year. For the first day of the year, January 1 (`month = 1, day = 1`), the value `1` is returned. This method accounts for whether `year` is a leap year. For example, `dayOfYear(3, 1, 2017)` returns `60`, since 2017 is not a leap year, while `dayOfYear(3, 1, 2016)` returns `61`, since 2016 is a leap year.

Class information for this question

```
public class APCalendar

private static boolean isLeapYear(int year)
public static int numberOfLeapYears(int year1, int year2)
private static int firstDayOfYear(int year)
private static int dayOfYear(int month, int day, int year)
public static int dayOfWeek(int month, int day, int year)
```

Complete method `dayOfWeek` below. You must use `firstDayOfYear` and `dayOfYear` appropriately to receive full credit.

```
/** Returns the value representing the day of the week for the given date
 *  (month, day, year), where 0 denotes Sunday, 1 denotes Monday, ...,
 *  and 6 denotes Saturday.
 *  Precondition: The date represented by month, day, year is a valid date.
 */
public static int dayOfWeek(int month, int day, int year)
```

1. This question involves reasoning about a simulation of a frog hopping in a straight line. The frog attempts to hop to a goal within a specified number of hops. The simulation is encapsulated in the following `FrogSimulation` class. You will write two of the methods in this class.

```
public class FrogSimulation
{
    /** Distance, in inches, from the starting position to the goal. */
    private int goalDistance;

    /** Maximum number of hops allowed to reach the goal. */
    private int maxHops;


    /** Constructs a FrogSimulation where dist is the distance, in inches, from the starting
     *  position to the goal, and numHops is the maximum number of hops allowed to reach the goal.
     *  Precondition: dist > 0; numHops > 0
     */
    public FrogSimulation(int dist, int numHops)
    {
       goalDistance = dist;
       maxHops = numHops;
    }


    /** Returns an integer representing the distance, in inches, to be moved when the frog hops.
     */
    private int hopDistance()
    {  /* implementation not shown */  }


    /** Simulates a frog attempting to reach the goal as described in part (a).
     *  Returns true if the frog successfully reached or passed the goal during the simulation;
     *          false otherwise.
     */
    public boolean simulate()
    {  /* to be implemented in part (a) */  }


    /** Runs num simulations and returns the proportion of simulations in which the frog
     *  successfully reached or passed the goal.
     *  Precondition: num > 0
     */
    public double runSimulations(int num)
    {  /* to be implemented in part (b) */  }
```

(a) Write the `simulate` method, which simulates the frog attempting to hop in a straight line to a goal from the frog's starting position of 0 within a maximum number of hops. The method returns `true` if the frog successfully reached the goal within the maximum number of hops; otherwise, the method returns `false`.

The `FrogSimulation` class provides a method called `hopDistance` that returns an integer representing the distance (positive or negative) to be moved when the frog hops. A positive distance represents a move toward the goal. A negative distance represents a move away from the goal. The returned distance may vary from call to call. Each time the frog hops, its position is adjusted by the value returned by a call to the `hopDistance` method.

The frog hops until one of the following conditions becomes true:

- The frog has reached or passed the goal.
- The frog has reached a negative position.
- The frog has taken the maximum number of hops without reaching the goal.

The following example shows a declaration of a `FrogSimulation` object for which the goal distance is 24 inches and the maximum number of hops is 5. The table shows some possible outcomes of calling the `simulate` method.

```
FrogSimulation sim = new FrogSimulation(24, 5);
```

	Values returned by `hopDistance()`	Final position of frog	Return value of `sim.simulate()`
Example 1	`5, 7, -2, 8, 6`	`24`	`true`
Example 2	`6, 7, 6, 6`	`25`	`true`
Example 3	`6, -6, 31`	`31`	`true`
Example 4	`4, 2, -8`	`-2`	`false`
Example 5	`5, 4, 2, 4, 3`	`18`	`false`

Class information for this question

```
public class FrogSimulation

private int goalDistance
private int maxHops

private int hopDistance()
public boolean simulate()
public double runSimulations(int num)
```

Complete method `simulate` below. You must use `hopDistance` appropriately to receive full credit.

```
/** Simulates a frog attempting to reach the goal as described in part (a).
 *  Returns true if the frog successfully reached or passed the goal during the simulation;
```

(b) Write the `runSimulations` method, which performs a given number of simulations and returns the proportion of simulations in which the frog successfully reached or passed the goal. For example, if the parameter passed to `runSimulations` is `400`, and 100 of the 400 `simulate` method calls returned `true`, then the `runSimulations` method should return `0.25`.

Complete method `runSimulations` below. Assume that `simulate` works as specified, regardless of what you wrote in part (a). You must use `simulate` appropriately to receive full credit.

```
/** Runs num simulations and returns the proportion of simulations in which the frog
 *  successfully reached or passed the goal.
```

AP COMPUTER SCIENCE A • EXERCISE SHEET

2.9 Implementing Selection and Iteration Algorithms

Name: _____ Class: _____ Date: _____ **Total: 9 marks**

KEY WORDS running total 累加 • iteration 迭代 • selection 选择

Objective

Build the skills to answer exam questions on **implementing selection and iteration algorithms**.

You must be able to:

- combine **selection** and **iteration** to build a complete algorithm
- use a loop with an `if` inside to **count** or **filter** values
- accumulate a **running total** 累加 or track a **maximum**/minimum

1 Worked examples

Study these first. Each one shows the method for a question type used later.

■ Loop with an if inside

To **count** values meeting a condition, loop over them and increment a counter inside an `if`:

```
int count = 0;
for (int x : nums) {
    if (x % 2 == 0) count++;   // count evens
}
```

■ Accumulate and track

A **running total** adds each value as the loop runs. To find a **maximum**, keep a "best so far" variable and update it whenever a larger value appears.

2 Practice

2.1 State how to count values meeting a condition inside a loop. [1]

2.2 Describe how to find the maximum of a list of numbers. [2]

2.3 State what a “running total” is. [1]

3 Exam-style questions

3.1 To count how many scores are above 50, you use a loop with [1]

- **A** no condition
- **B** an `if` inside it
- **C** a cast
- **D** a constructor

3.2 A variable that adds each value as a loop runs is a [1]

- **A** counter of items
- **B** running total
- **C** loop control variable
- **D** parameter

3.3 A loop goes through `{4, 9, 2, 7}` tracking the largest so far (starting with the first value).

(a) State the maximum so far after seeing 4 and 9. [1]

(b) State the final maximum. [1]

(c) Name the technique of adding all the values together. [1]

Solutions

2.1 loop over the values and increment a counter inside an `if` that tests the condition.

2.2 keep a "maximum so far" variable, and update it whenever the current value is larger.

2.3 a variable that accumulates the sum of values as the loop runs.

3.1 B.

3.2 B.

3.3 (a) 9. (b) 9. (c) accumulating a running total.

1. This question involves the `Account` class, which is used to represent user accounts for a website. The `Account` class contains a helper method, `isAvailable`, which determines whether a username is available. You will write a constructor and a method in the `Account` class.

```
public class Account
{
   private String username; // To be initialized in part (a)

   /**
    *  Creates a username based on the parameter requestedName. If the
    *  username is unavailable, appends successive integers, beginning
    *  with 1, to requestedName until an available username is found,
    *  as described in part (a).
    */
   public Account(String requestedName)
   { /* to be implemented in part (a) */ }

   /**
    *  Returns true if the parameter str is an available username;
    *  returns false otherwise.
    */
   public static boolean isAvailable(String str)
   { /* implementation not shown */ }

   /**
    *  Returns a shortened version of username with each hyphen ("-")
    *  and the character before it removed, as described in part (b)
    *  Preconditions: username does not start or end with a hyphen.
    *                 username does not contain consecutive hyphens.
    *                 username.length() >= 2
    *  Postcondition: username is unchanged.
    */
   public String getShortenedName()
   { /* to be implemented in part (b) */ }

   /* There may be instance variables, constructors, and methods
      that are not shown. */
}
```

The following information applies to part A.

In part A, you will write the `Account` constructor, which creates a username based on the parameter `requestedName`.

A helper method, `isAvailable`, has been provided to determine whether a username is available. The `isAvailable` method returns `true` if its parameter is an available username and returns `false` otherwise.

If `requestedName` is an available username, the `Account` constructor will assign `requestedName` to the instance variable `username`. If not, the constructor will try different variations of `requestedName` until an available username is found and assigned to the instance variable `username`. The variations consist of `requestedName` followed by an integer, as in the following example.

- If `requestedName` is `"Luis-Cruz"` and this username is not available, then the constructor will check the availability of `"Luis-Cruz1"`, `"Luis-Cruz2"`, `"Luis-Cruz3"`, and so on, until an available username is found. The first available username found is assigned to `username`.
- If `requestedName` is `"PSmith"` and this username is available, then `"PSmith"` is assigned to `username`.

The following information applies to part B.

In part B, you will write the `getShortenedName` method, which returns a shortened version of the instance variable `username` with each hyphen (`"-"`) and the character immediately before it removed. If no hyphens appear in `username`, the value of `username` is returned.

For example, if `username` is `"Amy-Marie-Lin"`, `getShortenedName` should return `"AmMariLin"`.

As another example, if `username` is `"SammyB3"`, `getShortenedName` should return `"SammyB3"`.

Part A

Complete the `Account` constructor. You must use `isAvailable` appropriately in order to receive full credit.

```
/**
 *  Creates a username based on the parameter requestedName. If the
 *  username is unavailable, appends successive integers, beginning
 *  with 1, to requestedName until an available username is found,
 *  as described in part (a).
 */
public Account(String requestedName)
```

Part B

Complete method `getShortenedName`.

```
/**
 *  Returns a shortened version of username with each hyphen ("-")
 *  and the character before it removed, as described in part (b)
 *  Preconditions: username does not start or end with a hyphen.
 *                 username does not contain consecutive hyphens.
 *                 username.length() >= 2
 *  Postcondition: username is unchanged.
 */
public String getShortenedName()
```

1. This question involves dog walkers who are paid through a dog-walking company to take dogs for one-hour walks. A dog-walking company has a varying number of dogs that need to be walked during each hour of the day. A dog-walking company is represented by the following `DogWalkCompany` class.

```
public class DogWalkCompany
{
  /**
   * Returns the number of dogs, always greater than 0, that are available
   * for a walk during the time specified by hour
   * Precondition: 0 <= hour <= 23
   */
  public int numAvailableDogs(int hour)
  { /* implementation not shown */ }

  /**
   * Decreases, by numberDogsWalked, the number of dogs available for a walk
   * during the time specified by hour
   * Preconditions: 0 <= hour <= 23
   *                numberDogsWalked > 0
   */
  public void updateDogs(int hour, int numberDogsWalked)
  { /* implementation not shown */ }

  /* There may be instance variables, constructors,
     and methods that are not shown. */
}
```

A dog walker is associated with a dog-walking company and is represented by the `DogWalker` class. You will write two methods of the `DogWalker` class.

```
public class DogWalker
{
  /** The maximum number of dogs this walker can walk simultaneously
      per hour */
  private int maxDogs;

  /** The dog-walking company this dog walker is associated with */
  private DogWalkCompany company;

  /**
   * Assigns max to maxDogs and comp to company
   * Precondition: max > 0
   */
  public DogWalker(int max, DogWalkCompany comp)
  { /* implementation not shown */ }

  /**
   * Takes at least one dog for a walk during the time specified by
   * hour, as described in part (a)
   * Preconditions: 0 <= hour <= 23
   *                maxDogs > 0
   */
  public int walkDogs(int hour)
  { /* to be implemented in part (a) */ }
```

```
  /**
   * Performs an entire dog-walking shift and returns the amount
   * earned, in dollars, as described in part (b)
   * Preconditions: 0 <= startHour <= endHour <= 23
   *                maxDogs > 0
   */
  public int dogWalkShift(int startHour, int endHour)
  { /* to be implemented in part (b) */ }

  /* There may be instance variables, constructors,
     and methods that are not shown. */
}
```

A. Write the `walkDogs` method, which updates and returns the number of dogs this dog walker walks during the time specified by `hour`. Values of `hour` range from `0` to `23`, inclusive.

A helper method, `numAvailableDogs`, has been provided in the `DogWalkCompany` class. The method returns the number of dogs available to be taken for a walk at a given hour. The dog walker will always walk as many dogs as the dog-walking company has available to walk, as long as the available number of dogs is not greater than the maximum number of dogs that the dog walker can handle, represented by the value of `maxDogs`.

Another helper method, `updateDogs`, has also been provided in the `DogWalkCompany` class. So that multiple dog walkers do not sign up to walk the same dogs, the `walkDogs` method should use `updateDogs` to update the dog-walking company with the number of dogs this dog walker will walk during the given hour. The parameters of the `updateDogs` method indicate how many dogs this dog walker will walk during the time specified by `hour`.

For example, if the dog-walking company has 10 dogs that need to be walked at the given hour but the dog walker's maximum is 4, the `updateDogs` method should be used to indicate that this dog walker will walk 4 of the 10 dogs during the given hour. As another example, if the dog-walking company has 3 dogs that need to be walked at the given hour and the dog walker's maximum is 4, the `updateDogs` method should be used to indicate that this dog walker will walk all 3 of the available dogs during the given hour.

The `walkDogs` method should return the number of dogs to be walked by this dog walker during the time specified by `hour`.

Complete method `walkDogs`. You must use `numAvailableDogs` and `updateDogs` appropriately to receive full credit.

```
/**
 * Takes at least one dog for a walk during the time specified by
 * hour, as described in part (a)
 * Preconditions: 0 <= hour <= 23
 *                maxDogs > 0
 */
public int walkDogs(int hour)
```

B. Write the `dogWalkShift` method, which performs a dog-walking shift of the hours in the range `startHour` to `endHour`, inclusive, and returns the total amount earned. For example, a dog-walking shift from `14` to `16` is composed of three one-hour dog walks starting at hours `14`, `15`, and `16`.

For each hour, the base pay is $5 per dog walked plus a bonus of $3 if at least one of the following is true.

- `maxDogs` dogs are walked
- the walk occurs between the peak hours of 9 and 17, inclusive

The following table shows an example of the calculated amount earned by walking dogs from hour 7 through hour 10, inclusive.

Hour	Maximum Number of Dogs	Number of Dogs Walked	Amount Earned, in Dollars
7	3	3	3 × 5 + 3 = 18
8	3	2	2 × 5 = 10
9	3	2	2 × 5 + 3 = 13
10	3	3	3 × 5 + 3 = 18
Total			59

Complete method `dogWalkShift`. Assume that `walkDogs` works as specified, regardless of what you wrote in part (a). You must use `walkDogs` appropriately to earn full credit.

```
/**
 * Performs an entire dog-walking shift and returns the amount
 * earned, in dollars, as described in part (b)
 * Preconditions: 0 <= startHour <= endHour <= 23
 *                maxDogs > 0
 */
public int dogWalkShift(int startHour, int endHour)
```

1. This question simulates birds or possibly a bear eating at a bird feeder. The following `Feeder` class contains information about how much food is in the bird feeder and simulates how much food is eaten. You will write two methods of the `Feeder` class.

```
public class Feeder
{
   /**
    *  The amount of food, in grams, currently in the bird feeder; initialized in the constructor and
    *  always greater than or equal to zero
    */
   private int currentFood;

   /**
    *  Simulates one day with numBirds birds or possibly a bear at the bird feeder,
    *  as described in part (a)
    *  Precondition: numBirds > 0
    */
   public void simulateOneDay(int numBirds)
   {  /* to be implemented in part (a) */  }

   /**
    *  Returns the number of days birds or a bear found food to eat at the feeder in this simulation,
    *  as described in part (b)
    *  Preconditions: numBirds > 0, numDays > 0
    */
   public int simulateManyDays(int numBirds, int numDays)
   {  /* to be implemented in part (b) */  }

   // There may be instance variables, constructors, or methods that are not shown.
}
```

GO ON TO THE NEXT PAGE.

(a) Write the `simulateOneDay` method, which simulates `numBirds` birds or possibly a bear at the feeder for one day. The method determines the amount of food taken from the feeder on this day and updates the `currentFood` instance variable. The simulation accounts for normal conditions, which occur 95% of the time, and abnormal conditions, which occur 5% of the time.

Under normal conditions, the simulation assumes that on any given day, only birds visit the feeder and that each bird at the feeder consumes the same amount of food. This standard amount consumed is between 10 and 50 grams of food, inclusive, in 1-gram increments. That is, on any given day, each bird might eat 10, 11, . . . , 49, or 50 grams of food. The amount of food eaten by each bird on a given day is randomly generated and each integer from 10 to 50, inclusive, has an equal chance of being chosen.

For example, a run of the simulation might predict that for a certain day under normal conditions, each bird coming to the feeder will eat 11 grams of food. If 10 birds come to the feeder on that day, then a total of 110 grams of food will be consumed.

If the simulated food consumed is greater than the amount of food in the feeder, the birds empty the feeder and the amount of food in the feeder at the end of the day is zero.

Under abnormal conditions, a bear empties the feeder and the amount of food in the feeder at the end of the day is zero.

The following examples show possible results of three calls to `simulateOneDay`.

- Example 1: If the feeder initially contains 500 grams of food, the call `simulateOneDay(12)` could result in 12 birds eating 20 grams of food each, leaving 260 grams of food in the feeder.
- Example 2: If the feeder initially contains 1,000 grams of food, the call `simulateOneDay(22)` could result in a bear eating all the food, leaving 0 grams of food in the feeder.
- Example 3: If the feeder initially contains 100 grams of food, the call `simulateOneDay(5)` could result in 5 birds attempting to eat 30 grams of food each. Since the feeder initially contains less than 150 grams of food, the feeder is emptied, leaving 0 grams of food in the feeder.

GO ON TO THE NEXT PAGE.

Complete the `simulateOneDay` method.

```
/**
 *  Simulates one day with numBirds birds or possibly a bear at the bird feeder,
 *  as described in part (a)
 *  Precondition: numBirds > 0
 */
public void simulateOneDay(int numBirds)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class Feeder

private int currentFood

public void simulateOneDay(int numBirds)
public int simulateManyDays(int numBirds, int numDays)
```

GO ON TO THE NEXT PAGE.

(b) Write the `simulateManyDays` method. The method uses `simulateOneDay` to simulate `numBirds` birds or a bear coming to the feeder on at most `numDays` consecutive days. The simulation returns the number of days that birds or a bear found food at the feeder.

Consider the following examples.

Value of `currentFood` and Method Call	**Possible Outcomes and Resulting Return Value**

`currentFood: 2400`
 	
`simulateManyDays(10, 4)`
	Day 1: `simulateOneDay` leaves 2100 grams of food in the feeder.
	Day 2: `simulateOneDay` leaves 1650 grams of food in the feeder.
	Day 3: `simulateOneDay` leaves 1500 grams of food in the feeder.
	Day 4: `simulateOneDay` leaves 1260 grams of food in the feeder.
	
	The simulation returns `4` because, on all four days of the simulation, birds or a bear found food at the feeder. The instance variable `currentFood` has the value `1260`.

`currentFood: 250`
 	
`simulateManyDays(10, 5)`
	Day 1: `simulateOneDay` leaves 150 grams of food in the feeder.
	Day 2: `simulateOneDay` leaves 0 grams of food in the feeder.
	
	The simulation returns `2` because, on two of the five simulated days, birds or a bear found food at the feeder. The instance variable `currentFood` has the value `0`.

`currentFood: 0`
 	
`simulateManyDays(5, 10)`
	The simulation returns `0` because no food was found at the feeder on any day. The instance variable `currentFood` has the value `0`.

GO ON TO THE NEXT PAGE.

Complete the `simulateManyDays` method. Assume that `simulateOneDay` works as intended, regardless of what you wrote in part (a). You must use `simulateOneDay` appropriately in order to receive full credit.

```
/**
 *  Returns the number of days birds or a bear found food to eat at the feeder in this simulation,
 *  as described in part (b)
 *  Preconditions: numBirds > 0, numDays > 0
 */
public int simulateManyDays(int numBirds, int numDays)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class Feeder

private int currentFood

public void simulateOneDay(int numBirds)
public int simulateManyDays(int numBirds, int numDays)
```


1. This question involves the `AppointmentBook` class, which provides methods for students to schedule appointments with their teacher. Appointments can be scheduled during one of eight class periods during the school day, numbered 1 through 8. A requested appointment has a duration, which is the number of minutes the appointment will last. The 60 minutes within a period are numbered 0 through 59. In order for an appointment to be scheduled, the teacher must have a block of consecutive, available minutes that contains at least the requested number of minutes in a requested period. Scheduled appointments must start and end within the same period.

The `AppointmentBook` class contains two helper methods, `isMinuteFree` and `reserveBlock`. You will write two additional methods of the `AppointmentBook` class.

```
public class AppointmentBook
{
   /**
    *  Returns true if minute in period is available for an appointment and returns
    *  false otherwise
    *  Preconditions: 1 <= period <= 8;  0 <= minute <= 59
    */
   private boolean isMinuteFree(int period, int minute)
   {  /* implementation not shown */  }

   /**
    *  Marks the block of minutes that starts at startMinute in period and
    *  is duration minutes long as reserved for an appointment
    *  Preconditions: 1 <= period <= 8; 0 <= startMinute <= 59;
    *      1 <= duration <= 60
    */
   private void reserveBlock(int period, int startMinute, int duration)
   {  /* implementation not shown */  }

   /**
    *  Searches for the first block of duration free minutes during period, as described in
    *  part (a). Returns the first minute in the block if such a block is found or returns -1 if no
    *  such block is found.
    *  Preconditions: 1 <= period <= 8; 1 <= duration <= 60
    */
   public int findFreeBlock(int period, int duration)
   {  /* to be implemented in part (a) */  }

   /**
    *  Searches periods from startPeriod to endPeriod, inclusive, for a block
    *  of duration free minutes, as described in part (b). If such a block is found,
    *  calls reserveBlock to reserve the block of minutes and returns true; otherwise
    *  returns false.
    *  Preconditions: 1 <= startPeriod <= endPeriod <= 8; 1 <= duration <= 60
    */
   public boolean makeAppointment(int startPeriod, int endPeriod,
                                  int duration)
   {  /* to be implemented in part (b) */  }

   // There may be instance variables, constructors, and methods that are not shown.
}
```

(a) Write the `findFreeBlock` method, which searches `period` for the first block of free minutes that is `duration` minutes long. If such a block is found, `findFreeBlock` returns the first minute in the block. Otherwise, `findFreeBlock` returns `-1`. The `findFreeBlock` method uses the helper method `isMinuteFree`, which returns `true` if a particular minute is available to be included in a new appointment and returns `false` if the minute is unavailable.

Consider the following list of unavailable and available minutes in period 2.

Minutes in Period 2	Available?
0–9 (10 minutes)	No
10–14 (5 minutes)	Yes
15–29 (15 minutes)	No
30–44 (15 minutes)	Yes
45–49 (5 minutes)	No
50–59 (10 minutes)	Yes

The method call `findFreeBlock(2, 15)` would return `30` to indicate that a 15-minute block starting with minute `30` is available. No steps should be taken as a result of the call to `findFreeBlock` to mark those 15 minutes as unavailable.

The method call `findFreeBlock(2, 9)` would also return `30`. Whenever there are multiple blocks that satisfy the requirement, the earliest starting minute is returned.

The method call `findFreeBlock(2, 20)` would return `-1`, since no 20-minute block of available minutes exists in period 2.

Complete method `findFreeBlock`. You must use `isMinuteFree` appropriately in order to receive full credit.

```
/**
 *  Searches for the first block of duration free minutes during period, as described in
 *  part (a). Returns the first minute in the block if such a block is found or returns -1 if no
 *  such block is found.
 *  Preconditions: 1 <= period <= 8; 1 <= duration <= 60
 */
public int findFreeBlock(int period, int duration)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

(b) Write the `makeAppointment` method, which searches the periods from `startPeriod` to `endPeriod`, inclusive, for the earliest block of `duration` available minutes in the lowest-numbered period. If such a block is found, the `makeAppointment` method calls the helper method `reserveBlock` to mark the minutes in the block as unavailable and returns `true`. If no such block is found, the `makeAppointment` method returns `false`.

Consider the following list of unavailable and available minutes in periods 2, 3, and 4 and three successive calls to `makeAppointment`.

Period	Minutes	Available?
2	0–24 (25 minutes)	No
2	25–29 (5 minutes)	Yes
2	30–59 (30 minutes)	No
3	0–14 (15 minutes)	Yes
3	15–40 (26 minutes)	No
3	41–59 (19 minutes)	Yes
4	0–4 (5 minutes)	No
4	5–29 (25 minutes)	Yes
4	30–43 (14 minutes)	No
4	44–59 (16 minutes)	Yes

The method call `makeAppointment(2, 4, 22)` returns `true` and results in the minutes 5 through 26, inclusive, in period 4 being marked as unavailable.

The method call `makeAppointment(3, 4, 3)` returns `true` and results in the minutes 0 through 2, inclusive, in period 3 being marked as unavailable.

The method call `makeAppointment(2, 4, 30)` returns `false`, since there is no block of 30 available minutes in periods 2, 3, or 4.

The following shows the updated list of unavailable and available minutes in periods 2, 3, and 4 after the three example method calls are complete.

Period	Minutes	Available?
2	0–24 (25 minutes)	No
2	25–29 (5 minutes)	Yes
2	30–59 (30 minutes)	No
3	0–2 (3 minutes)	No
3	3–14 (12 minutes)	Yes
3	15–40 (26 minutes)	No
3	41–59 (19 minutes)	Yes
4	0–26 (27 minutes)	No
4	27–29 (3 minutes)	Yes
4	30–43 (14 minutes)	No
4	44–59 (16 minutes)	Yes

Complete method `makeAppointment`. Assume that `findFreeBlock` works as intended, regardless of what you wrote in part (a). You must use `findFreeBlock` and `reserveBlock` appropriately in order to receive full credit.

```
/**
 *  Searches periods from startPeriod to endPeriod, inclusive, for a block
 *  of duration free minutes, as described in part (b). If such a block is found,
 *  calls reserveBlock to reserve the block of minutes and returns true; otherwise
 *  returns false.
 *  Preconditions: 1 <= startPeriod <= endPeriod <= 8; 1 <= duration <= 60
 */
public boolean makeAppointment(int startPeriod, int endPeriod,
                               int duration)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class AppointmentBook

private boolean isMinuteFree(int period, int minute)
private void reserveBlock(int period, int startMinute,
   int duration)
public int findFreeBlock(int period, int duration)
public boolean makeAppointment(int startPeriod, int endPeriod,
   int duration)
```

1. This question involves simulation of the play and scoring of a single-player video game. In the game, a player attempts to complete three levels. A level in the game is represented by the `Level` class.

```
public class Level
{
   /** Returns true if the player reached the goal on this level and returns false otherwise */
   public boolean goalReached()
   {  /* implementation not shown */  }

   /** Returns the number of points (a positive integer) recorded for this level */
   public int getPoints()
   {  /* implementation not shown */  }

   // There may be instance variables, constructors, and methods that are not shown.
}
```

GO ON TO THE NEXT PAGE.

Play of the game is represented by the `Game` class. You will write two methods of the `Game` class.

```
public class Game
{
   private Level levelOne;
   private Level levelTwo;
   private Level levelThree;

   /** Postcondition: All instance variables have been initialized. */
   public Game()
   {  /* implementation not shown */  }

   /** Returns true if this game is a bonus game and returns false otherwise */
   public boolean isBonus()
   {  /* implementation not shown */  }

   /** Simulates the play of this Game (consisting of three levels) and updates all relevant
    *  game data
    */
   public void play()
   {  /* implementation not shown */  }

   /** Returns the score earned in the most recently played game, as described in part (a) */
   public int getScore()
   {  /* to be implemented in part (a) */  }

   /** Simulates the play of num games and returns the highest score earned, as
    *  described in part (b)
    *  Precondition: num > 0
    */
   public int playManyTimes(int num)
   {  /* to be implemented in part (b) */  }

   // There may be instance variables, constructors, and methods that are not shown.
}
```

GO ON TO THE NEXT PAGE.

(a) Write the `getScore` method, which returns the score for the most recently played game. Each game consists of three levels. The score for the game is computed using the following helper methods.

- The `isBonus` method of the `Game` class returns `true` if this is a bonus game and returns `false` otherwise.
- The `goalReached` method of the `Level` class returns `true` if the goal has been reached on a particular level and returns `false` otherwise.
- The `getPoints` method of the `Level` class returns the number of points recorded on a particular level. Whether or not recorded points are earned (included in the game score) depends on the rules of the game, which follow.

The score for the game is computed according to the following rules.

- Level one points are earned only if the level one goal is reached. Level two points are earned only if both the level one and level two goals are reached. Level three points are earned only if the goals of all three levels are reached.
- The score for the game is the sum of the points earned for levels one, two, and three.
- If the game is a bonus game, the score for the game is tripled.

GO ON TO THE NEXT PAGE.

The following table shows some examples of game score calculations.

	Level One Results	**Level Two Results**	**Level Three Results**	`isBonus` **Return Value**	**Score Calculation**
`goalReached` **Return Value:** `getPoints` **Return Value:**	`true` `200`	`true` `100`	`true` `500`	`true`	(200 + 100 + 500) × 3 = 2,400 The recorded points for levels one, two, and three are earned because the goals were reached in all three levels. The earned points are multiplied by 3 because `isBonus` returns `true`.
`goalReached` **Return Value:** `getPoints` **Return Value:**	`true` `200`	`true` `100`	`false` `500`	`false`	200 + 100 = 300 The recorded points for level one and level two are earned because the goal was reached in levels one and two. The recorded points for level three are not earned because the goal was not reached in level three.
`goalReached` **Return Value:** `getPoints` **Return Value:**	`true` `200`	`false` `100`	`true` `500`	`true`	200 × 3 = 600 The recorded points for only level one are earned because the goal was not reached in level two. The earned points are multiplied by 3 because `isBonus` returns `true`.
`goalReached` **Return Value:** `getPoints` **Return Value:**	`false` `200`	`true` `100`	`true` `500`	`false`	0 Because the goal in level one was not reached, no points are earned for any level.

Complete the `getScore` method.

```
/** Returns the score earned in the most recently played game, as described in part (a) */
public int getScore()
```

Begin your response at the top of a new page in the Free Response booklet and fill in the appropriate circle indicating the question number.
If there are multiple parts to this question, write the part letter with your response.

GO ON TO THE NEXT PAGE.

(b) Write the `playManyTimes` method, which simulates the play of `num` games and returns the highest game score earned. For example, if the four plays of the game that are simulated as a result of the method call `playManyTimes(4)` earn scores of `75`, `50`, `90`, and `20`, then the method should return `90`.

Play of the game is simulated by calling the helper method `play`. Note that if `play` is called only one time followed by multiple consecutive calls to `getScore`, each call to `getScore` will return the score earned in the single simulated play of the game.

Complete the `playManyTimes` method. Assume that `getScore` works as intended, regardless of what you wrote in part (a). You must call `play` and `getScore` appropriately in order to receive full credit.

```
/** Simulates the play of num games and returns the highest score earned, as
 *  described in part (b)
 *  Precondition: num > 0
 */
public int playManyTimes(int num)
```

Begin your response at the top of a new page in the Free Response booklet and fill in the appropriate circle indicating the question number.
If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class Level

public boolean goalReached()
public int getPoints()

public class Game

private Level levelOne
private Level levelTwo
private Level levelThree

public Game()
public boolean isBonus()
public void play()
public int getScore()
public int playManyTimes(int num)
```

GO ON TO THE NEXT PAGE.

1. The `APCalendar` class contains methods used to calculate information about a calendar. You will write two methods of the class.

```
public class APCalendar
{

   /** Returns true if year is a leap year and false otherwise. */
   private static boolean isLeapYear(int year)
   {  /* implementation not shown */  }

   /** Returns the number of leap years between year1 and year2, inclusive.
    *  Precondition: 0 <= year1 <= year2
    */
   public static int numberOfLeapYears(int year1, int year2)
   {  /* to be implemented in part (a) */  }

   /** Returns the value representing the day of the week for the first day of year,
    *  where 0 denotes Sunday, 1 denotes Monday, ..., and 6 denotes Saturday.
    */
   private static int firstDayOfYear(int year)
   {  /* implementation not shown */  }

   /** Returns n, where month, day, and year specify the nth day of the year.
    *  Returns 1 for January 1 (month = 1, day = 1) of any year.
    *  Precondition: The date represented by month, day, year is a valid date.
    */
   private static int dayOfYear(int month, int day, int year)
   {  /* implementation not shown */  }

   /** Returns the value representing the day of the week for the given date
    *  (month, day, year), where 0 denotes Sunday, 1 denotes Monday, ...,
    *  and 6 denotes Saturday.
    *  Precondition: The date represented by month, day, year is a valid date.
    */
   public static int dayOfWeek(int month, int day, int year)
   {  /* to be implemented in part (b) */  }

   // There may be instance variables, constructors, and other methods not shown.

}
```

(a) Write the static method `numberOfLeapYears`, which returns the number of leap years between `year1` and `year2`, inclusive.

In order to calculate this value, a helper method is provided for you.

- `isLeapYear(year)` returns `true` if `year` is a leap year and `false` otherwise.

Complete method `numberOfLeapYears` below. You must use `isLeapYear` appropriately to receive full credit.

```
/** Returns the number of leap years between year1 and year2, inclusive.
 *  Precondition: 0 <= year1 <= year2
 */
public static int numberOfLeapYears(int year1, int year2)
```

(b) Write the static method `dayOfWeek`, which returns the integer value representing the day of the week for the given date (`month`, `day`, `year`), where `0` denotes Sunday, `1` denotes Monday, ..., and `6` denotes Saturday. For example, 2019 began on a Tuesday, and January 5 is the fifth day of 2019. As a result, January 5, 2019, fell on a Saturday, and the method call `dayOfWeek(1, 5, 2019)` returns `6`.

As another example, January 10 is the tenth day of 2019. As a result, January 10, 2019, fell on a Thursday, and the method call `dayOfWeek(1, 10, 2019)` returns `4`.

In order to calculate this value, two helper methods are provided for you.

- `firstDayOfYear(year)` returns the integer value representing the day of the week for the first day of `year`, where `0` denotes Sunday, `1` denotes Monday, ..., and `6` denotes Saturday. For example, since 2019 began on a Tuesday, `firstDayOfYear(2019)` returns `2`.
- `dayOfYear(month, day, year)` returns *n*, where `month`, `day`, and `year` specify the *n*th day of the year. For the first day of the year, January 1 (`month = 1, day = 1`), the value `1` is returned. This method accounts for whether `year` is a leap year. For example, `dayOfYear(3, 1, 2017)` returns `60`, since 2017 is not a leap year, while `dayOfYear(3, 1, 2016)` returns `61`, since 2016 is a leap year.

Class information for this question

```
public class APCalendar

private static boolean isLeapYear(int year)
public static int numberOfLeapYears(int year1, int year2)
private static int firstDayOfYear(int year)
private static int dayOfYear(int month, int day, int year)
public static int dayOfWeek(int month, int day, int year)
```

Complete method `dayOfWeek` below. You must use `firstDayOfYear` and `dayOfYear` appropriately to receive full credit.

```
/** Returns the value representing the day of the week for the given date
 *  (month, day, year), where 0 denotes Sunday, 1 denotes Monday, ...,
 *  and 6 denotes Saturday.
 *  Precondition: The date represented by month, day, year is a valid date.
 */
public static int dayOfWeek(int month, int day, int year)
```

1. This question involves reasoning about a simulation of a frog hopping in a straight line. The frog attempts to hop to a goal within a specified number of hops. The simulation is encapsulated in the following `FrogSimulation` class. You will write two of the methods in this class.

```
public class FrogSimulation
{
    /** Distance, in inches, from the starting position to the goal. */
    private int goalDistance;

    /** Maximum number of hops allowed to reach the goal. */
    private int maxHops;


    /** Constructs a FrogSimulation where dist is the distance, in inches, from the starting
     *  position to the goal, and numHops is the maximum number of hops allowed to reach the goal.
     *  Precondition: dist > 0; numHops > 0
     */
    public FrogSimulation(int dist, int numHops)
    {
       goalDistance = dist;
       maxHops = numHops;
    }


    /** Returns an integer representing the distance, in inches, to be moved when the frog hops.
     */
    private int hopDistance()
    {  /* implementation not shown */  }


    /** Simulates a frog attempting to reach the goal as described in part (a).
     *  Returns true if the frog successfully reached or passed the goal during the simulation;
     *          false otherwise.
     */
    public boolean simulate()
    {  /* to be implemented in part (a) */  }


    /** Runs num simulations and returns the proportion of simulations in which the frog
     *  successfully reached or passed the goal.
     *  Precondition: num > 0
     */
    public double runSimulations(int num)
    {  /* to be implemented in part (b) */  }
```

(a) Write the `simulate` method, which simulates the frog attempting to hop in a straight line to a goal from the frog's starting position of 0 within a maximum number of hops. The method returns `true` if the frog successfully reached the goal within the maximum number of hops; otherwise, the method returns `false`.

The `FrogSimulation` class provides a method called `hopDistance` that returns an integer representing the distance (positive or negative) to be moved when the frog hops. A positive distance represents a move toward the goal. A negative distance represents a move away from the goal. The returned distance may vary from call to call. Each time the frog hops, its position is adjusted by the value returned by a call to the `hopDistance` method.

The frog hops until one of the following conditions becomes true:

- The frog has reached or passed the goal.
- The frog has reached a negative position.
- The frog has taken the maximum number of hops without reaching the goal.

The following example shows a declaration of a `FrogSimulation` object for which the goal distance is 24 inches and the maximum number of hops is 5. The table shows some possible outcomes of calling the `simulate` method.

```
FrogSimulation sim = new FrogSimulation(24, 5);
```

	Values returned by `hopDistance()`	Final position of frog	Return value of `sim.simulate()`
Example 1	`5, 7, -2, 8, 6`	`24`	`true`
Example 2	`6, 7, 6, 6`	`25`	`true`
Example 3	`6, -6, 31`	`31`	`true`
Example 4	`4, 2, -8`	`-2`	`false`
Example 5	`5, 4, 2, 4, 3`	`18`	`false`

Class information for this question

```
public class FrogSimulation

private int goalDistance
private int maxHops

private int hopDistance()
public boolean simulate()
public double runSimulations(int num)
```

Complete method `simulate` below. You must use `hopDistance` appropriately to receive full credit.

```
/** Simulates a frog attempting to reach the goal as described in part (a).
 *  Returns true if the frog successfully reached or passed the goal during the simulation;
```

(b) Write the `runSimulations` method, which performs a given number of simulations and returns the proportion of simulations in which the frog successfully reached or passed the goal. For example, if the parameter passed to `runSimulations` is `400`, and 100 of the 400 `simulate` method calls returned `true`, then the `runSimulations` method should return `0.25`.

Complete method `runSimulations` below. Assume that `simulate` works as specified, regardless of what you wrote in part (a). You must use `simulate` appropriately to receive full credit.

```
/** Runs num simulations and returns the proportion of simulations in which the frog
 *  successfully reached or passed the goal.
```

4. This question involves the process of taking a list of words, called `wordList`, and producing a formatted string of a specified length. The list `wordList` contains at least two words, consisting of letters only.

   When the formatted string is constructed, spaces are placed in the gaps between words so that as many spaces as possible are evenly distributed to each gap. The equal number of spaces inserted into each gap is referred to as the *basic gap width.* Any *leftover spaces* are inserted one at a time into the gaps from left to right until there are no more leftover spaces.

   The following three examples illustrate these concepts. In each example, the list of words is to be placed into a formatted string of length 20.

   Example 1: `wordList: ["AP", "COMP", "SCI", "ROCKS"]`

   Total number of letters in words: 14
   Number of gaps between words: 3
   Basic gap width: 2
   Leftover spaces: 0

   Formatted string:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
A	P			C	O	M	P			S	C	I			R	O	C	K	S

   Example 2: `wordList: ["GREEN", "EGGS", "AND", "HAM"]`

   Total number of letters in words: 15
   Number of gaps between words: 3
   Basic gap width: 1
   Leftover spaces: 2

   The leftover spaces are inserted one at a time between the words from left to right until there are no more leftover spaces. In this example, the first two gaps get an extra space.

   Formatted string:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
G	R	E	E	N			E	G	G	S			A	N	D		H	A	M

   Example 3: `wordList: ["BEACH", "BALL"]`

   Total number of letters in words: 9
   Number of gaps between words: 1
   Basic gap width: 11
   Leftover spaces: 0

   Formatted string:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
B	E	A	C	H												B	A	L	L

   You will implement three `static` methods in a class named `StringFormatter` that is not shown.

(a) Write the `StringFormatter` method `totalLetters`, which returns the total number of letters in the words in its parameter `wordList`. For example, if the variable `List<String> words` is `["A", "frog", "is"]`, then the call `StringFormatter.totalLetters(words)` returns `7`. You may assume that all words in `wordList` consist of one or more letters.

Complete method `totalLetters` below.

```
/** Returns the total number of letters in wordList.
 *  Precondition: wordList contains at least two words, consisting of letters only.
 */
public static int totalLetters(List<String> wordList)
```

Part (b) begins on page 20.

(b) Write the `StringFormatter` method `basicGapWidth`, which returns the basic gap width as defined earlier.

Class information for this question

```
public class StringFormatter

public static int totalLetters(List<String> wordList)
public static int basicGapWidth(List<String> wordList,
                                int formattedLen)
public static int leftoverSpaces(List<String> wordList,
                                 int formattedLen)
public static String format(List<String> wordList, int formattedLen)
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Assume that `totalLetters` works as specified regardless of what you wrote in part (a). You must use `totalLetters` appropriately to receive full credit.

Complete method `basicGapWidth` below.

```
/** Returns the basic gap width when wordList is used to produce
 *  a formatted string of formattedLen characters.
 *  Precondition: wordList contains at least two words, consisting of letters only.
 *                formattedLen is large enough for all the words and gaps.
 */
public static int basicGapWidth(List<String> wordList,
                                int formattedLen)
```

Part (c) begins on page 22.

(c) Write the `StringFormatter` method `format`, which returns the formatted string as defined earlier. The `StringFormatter` class also contains a method called `leftoverSpaces`, which has already been implemented. This method returns the number of leftover spaces as defined earlier and is shown below.

```
/** Returns the number of leftover spaces when wordList is used to produce
 *  a formatted string of formattedLen characters.
 *  Precondition: wordList contains at least two words, consisting of letters only.
 *                formattedLen is large enough for all the words and gaps.
 */
public static int leftoverSpaces(List<String> wordList,
                                 int formattedLen)
{  /* implementation not shown */  }
```

Class information for this question

```
public class StringFormatter

public static int totalLetters(List<String> wordList)
public static int basicGapWidth(List<String> wordList,
                                int formattedLen)
public static int leftoverSpaces(List<String> wordList,
                                 int formattedLen)
public static String format(List<String> wordList, int formattedLen)
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Assume that `basicGapWidth` works as specified, regardless of what you wrote in part (b). You must use `basicGapWidth` and `leftoverSpaces` appropriately to receive full credit.

Complete method `format` below.

```
/** Returns a formatted string consisting of the words in wordList separated by spaces.
 *  Precondition: The wordList contains at least two words, consisting of letters only.
 *                formattedLen is large enough for all the words and gaps.
 *  Postcondition: All words in wordList appear in the formatted string.
 *    - The words appear in the same order as in wordList.
 *    - The number of spaces between words is determined by basicGapWidth and the
 *      distribution of leftoverSpaces from left to right, as described in the question.
 */
public static String format(List<String> wordList, int formattedLen)
```

STOP

END OF EXAM

AP COMPUTER SCIENCE A • EXERCISE SHEET

2.10 Implementing String Algorithms

Name: _____ Class: _____ Date: _____ **Total: 8 marks**

KEY WORDS traverse 遍历

Objective

Build the skills to answer exam questions on **implementing String algorithms**.

You must be able to:

- **traverse** 遍历 a String by looping over its indices from 0 to `length - 1`
- use `charAt` and `substring` inside a loop
- count characters or find a pattern one position at a time

1 Worked examples

Study these first. Each one shows the method for a question type used later.

■ Traversing a String

Loop over the valid indices 0 to `length() - 1` and read each character with `charAt`:

```
for (int i = 0; i < s.length(); i++) {
    char c = s.charAt(i);   // the character at index i
}
```

■ Counting and searching

By examining one position at a time, you can **count** particular characters or **find** a pattern within the String.

2 Practice

2.1 State the range of indices used to traverse a String `s`. [1]

2.2 State what `s.charAt(0)` returns. [1]

2.3 For `s = "apple"`, state the value of `s.charAt(1)`. [1]

---

3 Exam-style questions

3.1 To visit every character of a String, loop from `0` to [1]

- **A** `length`
- **B** `length - 1`
- **C** `length + 1`
- **D** `1`

3.2 `s.charAt(i)` returns [1]

- **A** a substring
- **B** the character at index `i`
- **C** the length
- **D** the index `i`

3.3 `String s = "hello";`.

(a) State `s.length()`. [1]

(b) State `s.charAt(4)`. [1]

(c) State the last valid index. [1]

Solutions

2.1 from 0 to `s.length() - 1`.

2.2 the first character of the String.

2.3 `'p'`.

3.1 B.

3.2 B.

3.3 (a) `5`. (b) `'o'`. (c) `4`.

1. This question involves the `Account` class, which is used to represent user accounts for a website. The `Account` class contains a helper method, `isAvailable`, which determines whether a username is available. You will write a constructor and a method in the `Account` class.

```
public class Account
{
   private String username; // To be initialized in part (a)

   /**
    *  Creates a username based on the parameter requestedName. If the
    *  username is unavailable, appends successive integers, beginning
    *  with 1, to requestedName until an available username is found,
    *  as described in part (a).
    */
   public Account(String requestedName)
   { /* to be implemented in part (a) */ }

   /**
    *  Returns true if the parameter str is an available username;
    *  returns false otherwise.
    */
   public static boolean isAvailable(String str)
   { /* implementation not shown */ }

   /**
    *  Returns a shortened version of username with each hyphen ("-")
    *  and the character before it removed, as described in part (b)
    *  Preconditions: username does not start or end with a hyphen.
    *                 username does not contain consecutive hyphens.
    *                 username.length() >= 2
    *  Postcondition: username is unchanged.
    */
   public String getShortenedName()
   { /* to be implemented in part (b) */ }

   /* There may be instance variables, constructors, and methods
      that are not shown. */
}
```

The following information applies to part A.

In part A, you will write the `Account` constructor, which creates a username based on the parameter `requestedName`.

A helper method, `isAvailable`, has been provided to determine whether a username is available. The `isAvailable` method returns `true` if its parameter is an available username and returns `false` otherwise.

If `requestedName` is an available username, the `Account` constructor will assign `requestedName` to the instance variable `username`. If not, the constructor will try different variations of `requestedName` until an available username is found and assigned to the instance variable `username`. The variations consist of `requestedName` followed by an integer, as in the following example.

- If `requestedName` is `"Luis-Cruz"` and this username is not available, then the constructor will check the availability of `"Luis-Cruz1"`, `"Luis-Cruz2"`, `"Luis-Cruz3"`, and so on, until an available username is found. The first available username found is assigned to `username`.
- If `requestedName` is `"PSmith"` and this username is available, then `"PSmith"` is assigned to `username`.

The following information applies to part B.

In part B, you will write the `getShortenedName` method, which returns a shortened version of the instance variable `username` with each hyphen (`"-"`) and the character immediately before it removed. If no hyphens appear in `username`, the value of `username` is returned.

For example, if `username` is `"Amy-Marie-Lin"`, `getShortenedName` should return `"AmMariLin"`.

As another example, if `username` is `"SammyB3"`, `getShortenedName` should return `"SammyB3"`.

Part A

Complete the `Account` constructor. You must use `isAvailable` appropriately in order to receive full credit.

```
/**
 *  Creates a username based on the parameter requestedName. If the
 *  username is unavailable, appends successive integers, beginning
 *  with 1, to requestedName until an available username is found,
 *  as described in part (a).
 */
public Account(String requestedName)
```

Part B

Complete method `getShortenedName`.

```
/**
 *  Returns a shortened version of username with each hyphen ("-")
 *  and the character before it removed, as described in part (b)
 *  Preconditions: username does not start or end with a hyphen.
 *                 username does not contain consecutive hyphens.
 *                 username.length() >= 2
 *  Postcondition: username is unchanged.
 */
public String getShortenedName()
```


3. This question involves the manipulation and analysis of a list of words. The following `WordChecker` class contains an `ArrayList<String>` to be analyzed and methods that are used to perform the analysis. You will write two methods of the `WordChecker` class.

```
public class WordChecker
{
   /** Initialized in the constructor and contains no null elements */
   private ArrayList<String> wordList;

   /**
    *  Returns true if each element of wordList (except the first) contains the previous
    *  element as a substring and returns false otherwise, as described in part (a)
    *  Precondition: wordList contains at least two elements.
    *  Postcondition: wordList is unchanged.
    */
   public boolean isWordChain()
   {  /* to be implemented in part (a) */  }

   /**
    *  Returns an ArrayList<String> based on strings from wordList that start
    *  with target, as described in part (b). Each element of the returned ArrayList has had
    *  the initial occurrence of target removed.
    *  Postconditions: wordList is unchanged.
    *        Items appear in the returned list in the same order as they appear in wordList.
    */
   public ArrayList<String> createList(String target)
   {  /* to be implemented in part (b) */  }

   // There may be instance variables, constructors, and methods that are not shown.
}
```

(a) Write the `isWordChain` method, which determines whether each element of `wordList` (except the first) contains the previous element as a substring. The following table shows two sample `isWordChain` method calls.

`wordList`	`isWordChain` **Return Value**	**Explanation**
`["an", "band", "band", "abandon"]`	`true`	Each element contains the previous element as a substring.
`["to", "too", "stool", "tools"]`	`false`	`"tools"` does not contain the substring `"stool"`.

Complete the `isWordChain` method.

```
/**
 *  Returns true if each element of wordList (except the first) contains the previous
 *  element as a substring and returns false otherwise, as described in part (a)
 *  Precondition: wordList contains at least two elements.
 *  Postcondition: wordList is unchanged.
 */
public boolean isWordChain()
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

(b) Write the `createList` method, which creates and returns an `ArrayList<String>`. The method identifies strings in `wordList` that start with `target` and returns a new `ArrayList` containing each identified string without the starting occurrence of `target`. Elements must appear in the returned list in the same order as they appear in `wordList`.

Consider an example where `wordList` contains the following strings.

```
["catch", "bobcat", "catchacat", "cat", "at"]
```

The following table shows the `ArrayList` returned by some calls to `createList`. In all cases, `wordList` is unchanged.

Method Call	`ArrayList` **Returned by** `createList`	**Explanation**
`createList("cat")`	`["ch", "chacat", ""]`	Only `"catch"`, `"catchacat"`, and `"cat"` begin with `"cat"`.
`createList("catch")`	`["", "acat"]`	Only `"catch"` and `"catchacat"` begin with `"catch"`.
`createList("dog")`	`[]`	None of the words in `wordList` begin with `"dog"`.

Complete the `createList` method.

```
/**
 *  Returns an ArrayList<String> based on strings from wordList that start
 *  with target, as described in part (b). Each element of the returned ArrayList has had
 *  the initial occurrence of target removed.
 *  Postconditions: wordList is unchanged.
 *      Items appear in the returned list in the same order as they appear in wordList.
 */
public ArrayList<String> createList(String target)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class WordChecker

private ArrayList<String> wordList

public boolean isWordChain()
public ArrayList<String> createList(String target)
```

3. Users of a website are asked to provide a review of the website at the end of each visit. Each review, represented by an object of the `Review` class, consists of an integer indicating the user's rating of the website and an optional `String` comment field. The comment field in a `Review` object ends with a period (`"."`), exclamation point (`"!"`), or letter, or is a `String` of length `0` if the user did not enter a comment.

```
public class Review
{
   private int rating;
   private String comment;

   /** Precondition: r >= 0
    *      c is not null.
    */
   public Review(int r, String c)
   {
      rating = r;
      comment = c;
   }

   public int getRating()
   {
      return rating;
   }

   public String getComment()
   {
      return comment;
   }

   // There may be instance variables, constructors, and methods that are not shown.
}
```

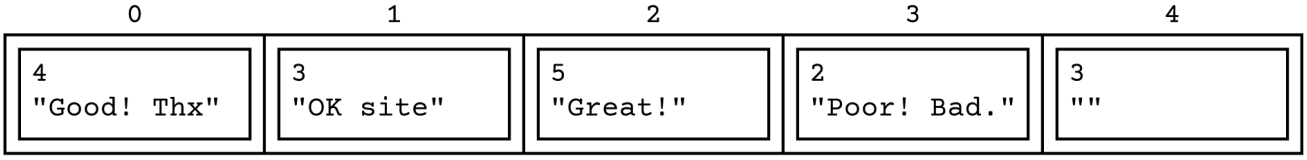
GO ON TO THE NEXT PAGE.

The `ReviewAnalysis` class contains methods used to analyze the reviews provided by users. You will write two methods of the `ReviewAnalysis` class.

```
public class ReviewAnalysis
{
    /** All user reviews to be included in this analysis */
    private Review[] allReviews;

    /** Initializes allReviews to contain all the Review objects to be analyzed */
    public ReviewAnalysis()
    {  /* implementation not shown */  }

    /** Returns a double representing the average rating of all the Review objects to be
     *  analyzed, as described in part (a)
     *  Precondition: allReviews contains at least one Review.
     *      No element of allReviews is null.
     */
    public double getAverageRating()
    {  /* to be implemented in part (a) */  }

    /** Returns an ArrayList of String objects containing formatted versions of
     *  selected user comments, as described in part (b)
     *  Precondition: allReviews contains at least one Review.
     *      No element of allReviews is null.
     *  Postcondition: allReviews is unchanged.
     */
    public ArrayList<String> collectComments()
    {  /* to be implemented in part (b) */  }
}
```

GO ON TO THE NEXT PAGE.

(a) Write the `ReviewAnalysis` method `getAverageRating`, which returns the average rating (arithmetic mean) of all elements of `allReviews`. For example, `getAverageRating` would return `3.4` if `allReviews` contained the following `Review` objects.



Complete method `getAverageRating`.

```
/** Returns a double representing the average rating of all the Review objects to be
 *  analyzed, as described in part (a)
 *  Precondition: allReviews contains at least one Review.
 *      No element of allReviews is null.
 */
public double getAverageRating()
```

Begin your response at the top of a new page in the Free Response booklet and fill in the appropriate circle indicating the question number.
If there are multiple parts to this question, write the part letter with your response.

Class information for this question

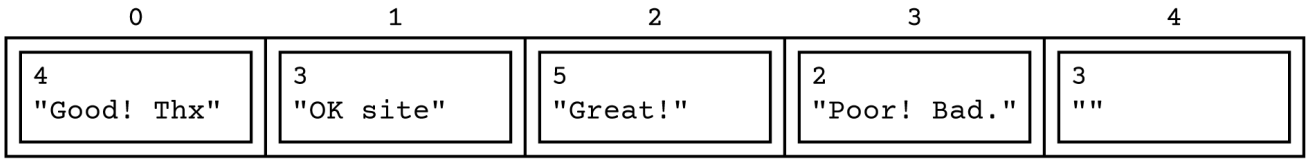
```
public class Review

private int rating
private String comment

public Review(int r, String c)
public int getRating()
public String getComment()

public class ReviewAnalysis

private Review[] allReviews

public ReviewAnalysis()
public double getAverageRating()
public ArrayList<String> collectComments()
```

GO ON TO THE NEXT PAGE.

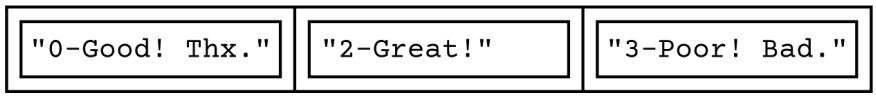
(b) Write the `ReviewAnalysis` method `collectComments`, which collects and formats only comments that contain an exclamation point. The method returns an `ArrayList` of `String` objects containing copies of user comments from `allReviews` that contain an exclamation point, formatted as follows. An empty `ArrayList` is returned if no comment in `allReviews` contains an exclamation point.

- The `String` inserted into the `ArrayList` to be returned begins with the index of the `Review` in `allReviews`.
- The index is immediately followed by a hyphen (`"-"`).
- The hyphen is followed by a copy of the original comment.
- The `String` must end with either a period or an exclamation point. If the original comment from `allReviews` does not end in either a period or an exclamation point, a period is added.

The following example of `allReviews` is repeated from part (a).



The following `ArrayList` would be returned by a call to `collectComments` with the given contents of `allReviews`. The reviews at index `1` and index `4` in `allReviews` are not included in the `ArrayList` to return since neither review contains an exclamation point.



Complete method `collectComments`.

```
/** Returns an ArrayList of String objects containing formatted versions of
 *  selected user comments, as described in part (b)
 *  Precondition: allReviews contains at least one Review.
 *       No element of allReviews is null.
 *  Postcondition: allReviews is unchanged.
 */
public ArrayList<String> collectComments()
```

Begin your response at the top of a new page in the Free Response booklet and fill in the appropriate circle indicating the question number.
If there are multiple parts to this question, write the part letter with your response.

GO ON TO THE NEXT PAGE.

1. This question involves the `WordMatch` class, which stores a secret string and provides methods that compare other strings to the secret string. You will write two methods in the `WordMatch` class.

```
public class WordMatch
{
    /** The secret string. */
    private String secret;

    /** Constructs a WordMatch object with the given secret string of lowercase letters. */
    public WordMatch(String word)
    {
        /* implementation not shown */
    }

    /** Returns a score for guess, as described in part (a).
     *  Precondition: 0 < guess.length() <= secret.length()
     */
    public int scoreGuess(String guess)
    {  /* to be implemented in part (a) */  }

    /** Returns the better of two guesses, as determined by scoreGuess and the rules for a
     *  tie-breaker that are described in part (b).
     *  Precondition: guess1 and guess2 contain all lowercase letters.
     *                guess1 is not the same as guess2.
     */
    public String findBetterGuess(String guess1, String guess2)
    {  /* to be implemented in part (b) */  }
}
```

(a) Write the `WordMatch` method `scoreGuess`. To determine the score to be returned, `scoreGuess` finds the number of times that `guess` occurs as a substring of `secret` and then multiplies that number by the square of the length of `guess`. Occurrences of `guess` may overlap within `secret`.

Assume that the length of `guess` is less than or equal to the length of `secret` and that `guess` is not an empty string.

The following examples show declarations of a `WordMatch` object. The tables show the outcomes of some possible calls to the `scoreGuess` method.

```
WordMatch game = new WordMatch("mississippi");
```

Value of `guess`	**Number of Substring Occurrences**	**Score Calculation**: (**Number of Substring Occurrences**) x (**Square of the Length of** `guess`)	**Return Value of** `game.scoreGuess(guess)`
`"i"`	4	`4 * 1 * 1 = 4`	4
`"iss"`	2	`2 * 3 * 3 = 18`	18
`"issipp"`	1	`1 * 6 * 6 = 36`	36
`"mississippi"`	1	`1 * 11 * 11 = 121`	121

```
WordMatch game = new WordMatch("aaaabb");
```

Value of `guess`	**Number of Substring Occurrences**	**Score Calculation**: (**Number of Substring Occurrences**) x (**Square of the Length of** `guess`)	**Return Value of** `game.scoreGuess(guess)`
`"a"`	4	`4 * 1 * 1 = 4`	4
`"aa"`	3	`3 * 2 * 2 = 12`	12
`"aaa"`	2	`2 * 3 * 3 = 18`	18
`"aabb"`	1	`1 * 4 * 4 = 16`	16
`"c"`	0	`0 * 1 * 1 = 0`	0

Complete the `scoreGuess` method.

```
/** Returns a score for guess, as described in part (a).
 *  Precondition: 0 < guess.length() <= secret.length()
 */
public int scoreGuess(String guess)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class WordMatch

private String secret

public WordMatch(String word)
public int scoreGuess(String guess)
public String findBetterGuess(String guess1, String guess2)
```

(b) Write the `WordMatch` method `findBetterGuess`, which returns the better guess of its two `String` parameters, `guess1` and `guess2`. If the `scoreGuess` method returns different values for `guess1` and `guess2`, then the guess with the higher score is returned. If the `scoreGuess` method returns the same value for `guess1` and `guess2`, then the alphabetically greater guess is returned.

The following example shows a declaration of a `WordMatch` object and the outcomes of some possible calls to the `scoreGuess` and `findBetterGuess` methods.

```
WordMatch game = new WordMatch("concatenation");
```

Method Call	Return Value	Explanation
`game.scoreGuess("ten");`	`9`	`1 * 3 * 3`
`game.scoreGuess("nation");`	`36`	`1 * 6 * 6`
`game.findBetterGuess("ten", "nation");`	`"nation"`	Since `scoreGuess` returns `36` for `"nation"` and `9` for `"ten"`, the guess with the greater score, `"nation"`, is returned.
`game.scoreGuess("con");`	`9`	`1 * 3 * 3`
`game.scoreGuess("cat");`	`9`	`1 * 3 * 3`
`game.findBetterGuess("con", "cat");`	`"con"`	Since `scoreGuess` returns `9` for both `"con"` and `"cat"`, the alphabetically greater guess, `"con"`, is returned.

Complete method `findBetterGuess`.

Assume that `scoreGuess` works as specified, regardless of what you wrote in part (a). You must use `scoreGuess` appropriately to receive full credit.

```
/** Returns the better of two guesses, as determined by scoreGuess and the rules for a
 *  tie-breaker that are described in part (b).
 *  Precondition: guess1 and guess2 contain all lowercase letters.
 *                guess1 is not the same as guess2.
 */
public String findBetterGuess(String guess1, String guess2)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class WordMatch

private String secret

public WordMatch(String word)
public int scoreGuess(String guess)
public String findBetterGuess(String guess1, String guess2)
```

3. This question involves analyzing and modifying a string. The following `Phrase` class maintains a phrase in an instance variable and has methods that access and make changes to the phrase. You will write two methods of the `Phrase` class.

```
public class Phrase
{
   private String currentPhrase;

   /** Constructs a new Phrase object. */
   public Phrase(String p)
   {  currentPhrase = p;  }


   /** Returns the index of the nth occurrence of str in the current phrase;
    *  returns -1 if the nth occurrence does not exist.
    *  Precondition: str.length() > 0 and n > 0
    *  Postcondition: the current phrase is not modified.
    */
   public int findNthOccurrence(String str, int n)
   {  /* implementation not shown */  }


   /** Modifies the current phrase by replacing the nth occurrence of str with repl.
    *  If the nth occurrence does not exist, the current phrase is unchanged.
    *  Precondition: str.length() > 0 and n > 0
    */
   public void replaceNthOccurrence(String str, int n, String repl)
   {  /* to be implemented in part (a) */  }


   /** Returns the index of the last occurrence of str in the current phrase;
    *  returns -1 if str is not found.
    *  Precondition: str.length() > 0
    *  Postcondition: the current phrase is not modified.
    */
   public int findLastOccurrence(String str)
   {  /* to be implemented in part (b) */  }


   /** Returns a string containing the current phrase. */
   public String toString()
   {  return currentPhrase;  }
}
```

Part (a) begins on page 12.

(a) Write the `Phrase` method `replaceNthOccurrence`, which will replace the `nth` occurrence of the string `str` with the string `repl`. If the `nth` occurrence does not exist, `currentPhrase` remains unchanged.

Several examples of the behavior of the method `replaceNthOccurrence` are shown below.

Code segments	Output produced

`Phrase phrase1 = new Phrase("A cat ate late.");`
`phrase1.replaceNthOccurrence("at", 1, "rane");`
`System.out.println(phrase1);`
	`A crane ate late.`

`Phrase phrase2 = new Phrase("A cat ate late.");`
`phrase2.replaceNthOccurrence("at", 6, "xx");`
`System.out.println(phrase2);`
	`A cat ate late.`

`Phrase phrase3 = new Phrase("A cat ate late.");`
`phrase3.replaceNthOccurrence("bat", 2, "xx");`
`System.out.println(phrase3);`
	`A cat ate late.`

`Phrase phrase4 = new Phrase("aaaa");`
`phrase4.replaceNthOccurrence("aa", 1, "xx");`
`System.out.println(phrase4);`
	`xxaa`

`Phrase phrase5 = new Phrase("aaaa");`
`phrase5.replaceNthOccurrence("aa", 2, "bbb");`
`System.out.println(phrase5);`
	`abbba`

Class information for this question

```
public class Phrase

private String currentPhrase
public Phrase(String p)
public int findNthOccurrence(String str, int n)
public void replaceNthOccurrence(String str, int n, String repl)
public int findLastOccurrence(String str)
public String toString()
```

The `Phrase` class includes the method `findNthOccurrence`, which returns the `n`th occurrence of a given string. You must use `findNthOccurrence` appropriately to receive full credit.

Complete method `replaceNthOccurrence` below.

```
/** Modifies the current phrase by replacing the nth occurrence of str with repl.
 *  If the nth occurrence does not exist, the current phrase is unchanged.
 *  Precondition: str.length() > 0 and n > 0
 */
public void replaceNthOccurrence(String str, int n, String repl)
```

Part (b) begins on page 14.

(b) Write the `Phrase` method `findLastOccurrence`. This method finds and returns the index of the last occurrence of a given string in `currentPhrase`. If the given string is not found, `-1` is returned. The following tables show several examples of the behavior of the method `findLastOccurrence`.

```
Phrase phrase1 = new Phrase("A cat ate late.");
```

Method call	Value returned
`phrase1.findLastOccurrence("at")`	`11`
`phrase1.findLastOccurrence("cat")`	`2`
`phrase1.findLastOccurrence("bat")`	`-1`

Class information for this question

```
public class Phrase

private String currentPhrase
public Phrase(String p)
public int findNthOccurrence(String str, int n)
public void replaceNthOccurrence(String str, int n, String repl)
public int findLastOccurrence(String str)
public String toString()
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

You must use `findNthOccurrence` appropriately to receive full credit.

Complete method `findLastOccurrence` below.

```
/** Returns the index of the last occurrence of str in the current phrase;
 *  returns -1 if str is not found.
 *  Precondition: str.length() > 0
 *  Postcondition: the current phrase is not modified.
 */
```

2. This question involves two classes that are used to process log messages. A list of sample log messages is given below.

```
CLIENT3:security alert – repeated login failures
Webserver:disk offline
SERVER1:file not found
SERVER2:read error on disk DSK1
SERVER1:write error on disk DSK2
Webserver:error on /dev/disk
```

Log messages have the format *machineId*:*description*, where *machineId* identifies the computer and *description* describes the event being logged. Exactly one colon (":") appears in a log message. There are no blanks either immediately before or immediately after the colon.

The following `LogMessage` class is used to represent a log message.

```
public class LogMessage
{
   private String machineId;
   private String description;

   /** Precondition: message is a valid log message. */
   public LogMessage(String message)
   {  /* to be implemented in part (a) */  }

   /** Returns true if the description in this log message properly contains keyword;
    *          false otherwise.
    */
   public boolean containsWord(String keyword)
   {  /* to be implemented in part (b) */  }

   public String getMachineId()
   {  return machineId;  }

   public String getDescription()
   {  return description;  }

   // There may be instance variables, constructors, and methods that are not shown.
}
```

(a) Write the constructor for the `LogMessage` class. It must initialize the private data of the object so that `getMachineId` returns the *machineId* part of the message and `getDescription` returns the *description* part of the message.

Complete the `LogMessage` constructor below.

```
/** Precondition: message is a valid log message. */
public LogMessage(String message)
```

Part (b) begins on page 8.

(b) Write the `LogMessage` method `containsWord`, which returns `true` if the description in the log message *properly contains* a given keyword and returns `false` otherwise.

A description *properly contains* a keyword if all three of the following conditions are true.

- the keyword is a substring of the description;
- the keyword is either at the beginning of the description or it is immediately preceded by a space;
- the keyword is either at the end of the description or it is immediately followed by a space.

The following tables show several examples. The descriptions in the left table properly contain the keyword `"disk"`. The descriptions in the right table do not properly contain the keyword `"disk"`.

Descriptions that properly contain `"disk"`

`"disk"`
`"error on disk"`
`"error on /dev/disk disk"`
`"error on disk DSK1"`

Descriptions that do not properly contain `"disk"`

`"DISK"`
`"error on disk3"`
`"error on /dev/disk"`
`"diskette"`

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Assume that the `LogMessage` constructor works as specified, regardless of what you wrote in part (a).

Complete method `containsWord` below.

```
/** Returns true if the description in this log message properly contains keyword;
 *          false otherwise.
 */
public boolean containsWord(String keyword)
```

Part (c) begins on page 10.

(c) The `SystemLog` class represents a list of `LogMessage` objects and provides a method that removes and returns a list of all log messages (if any) that properly contain a given keyword. The messages in the returned list appear in the same order in which they originally appeared in the system log. If no message properly contains the keyword, an empty list is returned. The declaration of the `SystemLog` class is shown below.

```
public class SystemLog
{
    /** Contains all the entries in this system log.
     *  Guaranteed not to be null and to contain only non-null entries.
     */
    private List<LogMessage> messageList;

    /** Removes from the system log all entries whose descriptions properly contain keyword,
     *  and returns a list (possibly empty) containing the removed entries.
     *  Postcondition:
     *    – Entries in the returned list properly contain keyword and
     *      are in the order in which they appeared in the system log.
     *    – The remaining entries in the system log do not properly contain keyword and
     *      are in their original order.
     *    – The returned list is empty if no messages properly contain keyword.
     */
    public List<LogMessage> removeMessages(String keyword)
    {  /* to be implemented in part (c) */  }

    // There may be instance variables, constructors, and methods that are not shown.
}
```

Write the `SystemLog` method `removeMessages`, which removes from the system log all entries whose descriptions properly contain `keyword` and returns a list of the removed entries in their original order. For example, assume that `theLog` is a `SystemLog` object initially containing six `LogMessage` objects representing the following list of log messages.

```
CLIENT3:security alert – repeated login failures
Webserver:disk offline
SERVER1:file not found
SERVER2:read error on disk DSK1
SERVER1:write error on disk DSK2
Webserver:error on /dev/disk
```

The call `theLog.removeMessages("disk")` would return a list containing the `LogMessage` objects representing the following log messages.

```
Webserver:disk offline
SERVER2:read error on disk DSK1
SERVER1:write error on disk DSK2
```

After the call, `theLog` would contain the following log messages.

```
CLIENT3:security alert – repeated login failures
SERVER1:file not found
Webserver:error on /dev/disk
```

Assume that the `LogMessage` class works as specified, regardless of what you wrote in parts (a) and (b). You must use `containsWord` appropriately to receive full credit.

Complete method `removeMessages` below.

```
/** Removes from the system log all entries whose descriptions properly contain keyword,
 *  and returns a list (possibly empty) containing the removed entries.
 *  Postcondition:
 *    – Entries in the returned list properly contain keyword and
 *      are in the order in which they appeared in the system log.
 *    – The remaining entries in the system log do not properly contain keyword and
 *      are in their original order.
 *    – The returned list is empty if no messages properly contain keyword.
 */
public List<LogMessage> removeMessages(String keyword)
```

COMPUTER SCIENCE A
SECTION II
Time—1 hour and 45 minutes
Number of questions—4
Percent of total score—50

Directions: SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:
- Assume that the classes listed in the Java Quick Reference have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods will not receive full credit.

1. This question involves reasoning about one-dimensional and two-dimensional arrays of integers. You will write three static methods, all of which are in a single enclosing class, named `DiverseArray` (not shown). The first method returns the sum of the values of a one-dimensional array; the second method returns an array that represents the sums of the rows of a two-dimensional array; and the third method analyzes row sums.

   (a) Write a `static` method `arraySum` that calculates and returns the sum of the entries in a specified one-dimensional array. The following example shows an array `arr1` and the value returned by a call to `arraySum`.

	`arr1`				Value returned by `arraySum(arr1)`
0	1	2	3	4	
1	3	2	7	3	16

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Complete method `arraySum` below.

```
/** Returns the sum of the entries in the one-dimensional array arr.
 */
public static int arraySum(int[] arr)
```

Part (b) begins on page 4.

(b) Write a `static` method `rowSums` that calculates the sums of each of the rows in a given two-dimensional array and returns these sums in a one-dimensional array. The method has one parameter, a two-dimensional array `arr2D` of `int` values. The array is in row-major order: `arr2D[r][c]` is the entry at row `r` and column `c`. The method returns a one-dimensional array with one entry for each row of `arr2D` such that each entry is the sum of the corresponding row in `arr2D`. As a reminder, each row of a two-dimensional array is a one-dimensional array.

For example, if `mat1` is the array represented by the following table, the call `rowSums(mat1)` returns the array `{16, 32, 28, 20}`.

`mat1`

	0	1	2	3	4
0	1	3	2	7	3
1	10	10	4	6	2
2	5	3	5	9	6
3	7	6	4	2	1

Methods written in this question

```
public static int arraySum(int[] arr)
public static int[] rowSums(int[][] arr2D)
public static boolean isDiverse(int[][] arr2D)
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Assume that `arraySum` works as specified, regardless of what you wrote in part (a). You must use `arraySum` appropriately to receive full credit.

Complete method `rowSums` below.

```
/** Returns a one-dimensional array in which the entry at index k is the sum of
 *  the entries of row k of the two-dimensional array arr2D.
 */
public static int[] rowSums(int[][] arr2D)
```

Part (c) begins on page 6.

(c) A two-dimensional array is *diverse* if no two of its rows have entries that sum to the same value. In the following examples, the array `mat1` is diverse because each row sum is different, but the array `mat2` is not diverse because the first and last rows have the same sum.

`mat1`

	0	1	2	3	4	Row sums
0	1	3	2	7	3	16
1	10	10	4	6	2	32
2	5	3	5	9	6	28
3	7	6	4	2	1	20

`mat2`

	0	1	2	3	4	Row sums
0	1	1	5	3	4	14
1	12	7	6	1	9	35
2	8	11	10	2	5	36
3	3	2	3	0	6	14

Write a `static` method `isDiverse` that determines whether or not a given two-dimensional array is diverse. The method has one parameter: a two-dimensional array `arr2D` of `int` values. The method should return `true` if all the row sums in the given array are unique; otherwise, it should return `false`. In the arrays shown above, the call `isDiverse(mat1)` returns `true` and the call `isDiverse(mat2)` returns `false`.

Methods written in this question

```
public static int arraySum(int[] arr)
public static int[] rowSums(int[][] arr2D)
public static boolean isDiverse(int[][] arr2D)
```

Assume that `arraySum` and `rowSums` work as specified, regardless of what you wrote in parts (a) and (b). You must use `rowSums` appropriately to receive full credit.

Complete method `isDiverse` below.

```
/** Returns true  if all rows in arr2D have different row sums;
 *         false otherwise.
 */
public static boolean isDiverse(int[][] arr2D)
```

2. Consider a guessing game in which a player tries to guess a hidden word. The hidden word contains only capital letters and has a length known to the player. A guess contains only capital letters and has the same length as the hidden word.

   After a guess is made, the player is given a hint that is based on a comparison between the hidden word and the guess. Each position in the hint contains a character that corresponds to the letter in the same position in the guess. The following rules determine the characters that appear in the hint.

If the letter in the guess is …	**the corresponding character in the hint is**
also in the same position in the hidden word,	the matching letter
also in the hidden word, but in a different position,	`"+"`
not in the hidden word,	`"*"`

The `HiddenWord` class will be used to represent the hidden word in the game. The hidden word is passed to the constructor. The class contains a method, `getHint`, that takes a guess and produces a hint.

For example, suppose the variable `puzzle` is declared as follows.

```
HiddenWord puzzle = new HiddenWord("HARPS");
```

The following table shows several guesses and the hints that would be produced.

Call to `getHint`	**String returned**
`puzzle.getHint("AAAAA")`	`"+A+++"`
`puzzle.getHint("HELLO")`	`"H****"`
`puzzle.getHint("HEART")`	`"H*++*"`
`puzzle.getHint("HARMS")`	`"HAR*S"`
`puzzle.getHint("HARPS")`	`"HARPS"`

Write the complete `HiddenWord` class, including any necessary instance variables, its constructor, and the method, `getHint`, described above. You may assume that the length of the guess is the same as the length of the hidden word.

COMPUTER SCIENCE A
SECTION II
Time—1 hour and 45 minutes
Number of questions—4
Percent of total score—50

Directions: SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:
- Assume that the classes listed in the appendices have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods may not receive full credit.

1. This question involves reasoning about strings made up of uppercase letters. You will implement two related methods that appear in the same class (not shown). The first method takes a single string parameter and returns a scrambled version of that string. The second method takes a list of strings and modifies the list by scrambling each entry in the list. Any entry that cannot be scrambled is removed from the list.

   (a) Write the method `scrambleWord`, which takes a given word and returns a string that contains a scrambled version of the word according to the following rules.

   - The scrambling process begins at the first letter of the word and continues from left to right.
   - If two consecutive letters consist of an "A" followed by a letter that is not an "A", then the two letters are swapped in the resulting string.
   - Once the letters in two adjacent positions have been swapped, neither of those two positions can be involved in a future swap.

   The following table shows several examples of words and their scrambled versions.

`word`	Result returned by `scrambleWord(word)`
"TAN"	"TNA"
"ABRACADABRA"	"BARCADABARA"
"WHOA"	"WHOA"
"AARDVARK"	"ARADVRAK"
"EGGS"	"EGGS"
"A"	"A"
""	""

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Complete method `scrambleWord` below.

```
/** Scrambles a given word.
 *  @param word the word to be scrambled
 *  @return the scrambled word (possibly equal to word)
 *  Precondition: word is either an empty string or contains only uppercase letters.
 *  Postcondition: the string returned was created from word as follows:
 *    - the word was scrambled, beginning at the first letter and continuing from left to right
 *    - two consecutive letters consisting of "A" followed by a letter that was not "A" were swapped
 *    - letters were swapped at most once
 */
public static String scrambleWord(String word)
```

Part (b) begins on page 4.

(b) Write the method `scrambleOrRemove`, which replaces each word in the parameter `wordList` with its scrambled version and removes any words that are unchanged after scrambling. The relative ordering of the entries in `wordList` remains the same as before the call to `scrambleOrRemove`.

The following example shows how the contents of `wordList` would be modified as a result of calling `scrambleOrRemove`.

Before the call to `scrambleOrRemove`:

	0	1	2	3	4
`wordList`	"TAN"	"ABRACADABRA"	"WHOA"	"APPLE"	"EGGS"

After the call to `scrambleOrRemove`:

	0	1	2
`wordList`	"TNA"	"BARCADABARA"	"PAPLE"

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Assume that `scrambleWord` is in the same class as `scrambleOrRemove` and works as specified, regardless of what you wrote in part (a).

Complete method `scrambleOrRemove` below.

```
/** Modifies wordList by replacing each word with its scrambled
 *  version, removing any words that are unchanged as a result of scrambling.
 *  @param wordList the list of words
 *  Precondition: wordList contains only non-null objects
 *  Postcondition:
 *    - all words unchanged by scrambling have been removed from wordList
 *    - each of the remaining words has been replaced by its scrambled version
 *    - the relative ordering of the entries in wordList is the same as it was
 *          before the method was called
 */
public static void scrambleOrRemove(List<String> wordList)
```

AP COMPUTER SCIENCE A • EXERCISE SHEET

2.11 Nested Iteration

Name: _____ Class: _____ Date: _____ **Total: 9 marks**

KEY WORDS nested loop 嵌套循环 • iteration 迭代

Objective

Build the skills to answer exam questions on **nested iteration**.

You must be able to:

- write a **nested loop** 嵌套循环 by placing one loop inside another
- understand that the **inner loop** runs fully for each pass of the outer loop
- calculate how many times the inner body runs in total

1 Worked examples

Study these first. Each one shows the method for a question type used later.

■ Nested loops

A loop inside another loop. The **inner loop runs completely** for **every** pass of the outer loop:

```
for (int i = 0; i < 3; i++)
    for (int j = 0; j < 4; j++)
        // this body runs 3 × 4 = 12 times
```

■ Counting total runs

Total inner-body runs = (outer passes) × (inner passes per outer pass).

2 Practice

2.1 State what a nested loop is. [1]

2.2 State how many times the inner body runs if the outer loop runs 4 times and the inner 5 times. [2]

2.3 State what runs fully for each pass of the outer loop. [1]

3 Exam-style questions

3.1 In a nested loop, the inner loop runs [1]

- **A** once in total
- **B** fully for each pass of the outer loop
- **C** never
- **D** only on the last pass

3.2 An outer loop of 3 and an inner loop of 3 run the inner body [1]

- **A** 3
- **B** 6
- **C** 9
- **D** 1

times.

3.3 `for (int i = 0; i < 3; i++) for (int j = 0; j < 4; j++) ....`

(a) State the number of outer passes. [1]

(b) State the inner passes per outer pass. [1]

(c) State the total number of inner-body runs. [1]

Solutions

2.1 a loop placed inside the body of another loop.

2.2 $4 \times 5 = 20$.

2.3 the inner loop.

3.1 B.

3.2 C.

3.3 (a) 3. (b) 4. (c) 12.

3. The `CourseRecord` class is used to store information about a student enrolled in a course. A partial definition of the `CourseRecord` class is shown.

```
public class CourseRecord
{
   /**
    *  Returns a unique ID for the student associated with this
    *  CourseRecord
    */
   public String getStudentID()
   { /* implementation not shown */ }

   /**
    *  Returns a nonnegative integer representing the number of times the
    *  student associated with this CourseRecord has been absent in the
    *  course
    */
   public int getAbsences()
   { /* implementation not shown */ }

   /* There may be instance variables, constructors, and methods
      that are not shown. */
}
```

The `Attendance` class maintains two `ArrayLists` of `CourseRecord` objects, named `historyList` and `mathList`. A partial definition of the `Attendance` class is shown.

```
public class Attendance
{
   /* The students enrolled in a history course */
   private ArrayList<CourseRecord> historyList;

   /* The students enrolled in a math course */
   private ArrayList<CourseRecord> mathList;

   /**
    *  Returns the number of students who are enrolled in both
    *  the history course and the math course but have more
    *  absences in the history course than the math course
    *  Preconditions:
    *        No student ID appears multiple times in historyList.
    *        No student ID appears multiple times in mathList.
    *        historyList and mathList do not contain any null elements.
    *  Postcondition:
    *        historyList and mathList are unchanged.
    */
   public int moreHistoryThanMathAbsences()
   { /* to be implemented */ }

   /* There may be instance variables, constructors, and methods
      that are not shown. */
}
```

Write the `Attendance` method `moreHistoryThanMathAbsences`. The method should return the number of students who are enrolled in both the history course and the math course but have more absences in the history course than the math course.

For example, suppose `historyList` and `mathList` have the following contents.

`historyList`:

Student ID	`"rc29"`	`"br98"`	`"dr03"`	`"ot32"`	`"sq98"`	`"ry00"`
Number of Absences	1	1	2	2	3	1

`mathList`:

Student ID	`"fr27"`	`"sq98"`	`"dr03"`	`"dk12"`	`"ot32"`	`"js33"`	`"ry00"`
Number of Absences	2	1	2	1	1	0	3

In this example, there are four student IDs (`"dr03"`, `"ot32"`, `"sq98"`, and `"ry00"`) that appear in both `historyList` and `mathList`. Of these, only `"ot32"` and `"sq98"` have more absences in the history course than the math course, so the `moreHistoryThanMathAbsences` method should return `2`.

Complete method `moreHistoryThanMathAbsences`.

```
/**
 *  Returns the number of students who are enrolled in both
 *  the history course and the math course but have more
 *  absences in the history course than the math course
 *  Preconditions:
 *        No student ID appears multiple times in historyList.
 *        No student ID appears multiple times in mathList.
 *        historyList and mathList do not contain any null elements.
 *  Postcondition:
 *        historyList and mathList are unchanged.
 */
public int moreHistoryThanMathAbsences()
```

1. This question involves the `WordMatch` class, which stores a secret string and provides methods that compare other strings to the secret string. You will write two methods in the `WordMatch` class.

```
public class WordMatch
{
    /** The secret string. */
    private String secret;

    /** Constructs a WordMatch object with the given secret string of lowercase letters. */
    public WordMatch(String word)
    {
        /* implementation not shown */
    }

    /** Returns a score for guess, as described in part (a).
     *  Precondition: 0 < guess.length() <= secret.length()
     */
    public int scoreGuess(String guess)
    {  /* to be implemented in part (a) */  }

    /** Returns the better of two guesses, as determined by scoreGuess and the rules for a
     *  tie-breaker that are described in part (b).
     *  Precondition: guess1 and guess2 contain all lowercase letters.
     *                guess1 is not the same as guess2.
     */
    public String findBetterGuess(String guess1, String guess2)
    {  /* to be implemented in part (b) */  }
}
```

(a) Write the `WordMatch` method `scoreGuess`. To determine the score to be returned, `scoreGuess` finds the number of times that `guess` occurs as a substring of `secret` and then multiplies that number by the square of the length of `guess`. Occurrences of `guess` may overlap within `secret`.

Assume that the length of `guess` is less than or equal to the length of `secret` and that `guess` is not an empty string.

The following examples show declarations of a `WordMatch` object. The tables show the outcomes of some possible calls to the `scoreGuess` method.

```
WordMatch game = new WordMatch("mississippi");
```

Value of `guess`	**Number of Substring Occurrences**	**Score Calculation**: (**Number of Substring Occurrences**) x (**Square of the Length of** `guess`)	**Return Value of** `game.scoreGuess(guess)`
`"i"`	4	`4 * 1 * 1 = 4`	4
`"iss"`	2	`2 * 3 * 3 = 18`	18
`"issipp"`	1	`1 * 6 * 6 = 36`	36
`"mississippi"`	1	`1 * 11 * 11 = 121`	121

```
WordMatch game = new WordMatch("aaaabb");
```

Value of `guess`	**Number of Substring Occurrences**	**Score Calculation**: (**Number of Substring Occurrences**) x (**Square of the Length of** `guess`)	**Return Value of** `game.scoreGuess(guess)`
`"a"`	4	`4 * 1 * 1 = 4`	4
`"aa"`	3	`3 * 2 * 2 = 12`	12
`"aaa"`	2	`2 * 3 * 3 = 18`	18
`"aabb"`	1	`1 * 4 * 4 = 16`	16
`"c"`	0	`0 * 1 * 1 = 0`	0

Complete the `scoreGuess` method.

```
/** Returns a score for guess, as described in part (a).
 *  Precondition: 0 < guess.length() <= secret.length()
 */
public int scoreGuess(String guess)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class WordMatch

private String secret

public WordMatch(String word)
public int scoreGuess(String guess)
public String findBetterGuess(String guess1, String guess2)
```

(b) Write the `WordMatch` method `findBetterGuess`, which returns the better guess of its two `String` parameters, `guess1` and `guess2`. If the `scoreGuess` method returns different values for `guess1` and `guess2`, then the guess with the higher score is returned. If the `scoreGuess` method returns the same value for `guess1` and `guess2`, then the alphabetically greater guess is returned.

The following example shows a declaration of a `WordMatch` object and the outcomes of some possible calls to the `scoreGuess` and `findBetterGuess` methods.

```
WordMatch game = new WordMatch("concatenation");
```

Method Call	Return Value	Explanation
`game.scoreGuess("ten");`	`9`	`1 * 3 * 3`
`game.scoreGuess("nation");`	`36`	`1 * 6 * 6`
`game.findBetterGuess("ten", "nation");`	`"nation"`	Since `scoreGuess` returns `36` for `"nation"` and `9` for `"ten"`, the guess with the greater score, `"nation"`, is returned.
`game.scoreGuess("con");`	`9`	`1 * 3 * 3`
`game.scoreGuess("cat");`	`9`	`1 * 3 * 3`
`game.findBetterGuess("con", "cat");`	`"con"`	Since `scoreGuess` returns `9` for both `"con"` and `"cat"`, the alphabetically greater guess, `"con"`, is returned.

Complete method `findBetterGuess`.

Assume that `scoreGuess` works as specified, regardless of what you wrote in part (a). You must use `scoreGuess` appropriately to receive full credit.

```
/** Returns the better of two guesses, as determined by scoreGuess and the rules for a
 *  tie-breaker that are described in part (b).
 *  Precondition: guess1 and guess2 contain all lowercase letters.
 *                guess1 is not the same as guess2.
 */
public String findBetterGuess(String guess1, String guess2)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class WordMatch

private String secret

public WordMatch(String word)
public int scoreGuess(String guess)
public String findBetterGuess(String guess1, String guess2)
```

2. This question involves reasoning about pairs of words that are represented by the following `WordPair` class.

```
public class WordPair
{

    /** Constructs a WordPair object. */
    public WordPair(String first, String second)
    {  /* implementation not shown */  }


    /** Returns the first string of this WordPair object. */
    public String getFirst()
    {  /* implementation not shown */  }


    /** Returns the second string of this WordPair object. */
    public String getSecond()
    {  /* implementation not shown */  }

}
```

You will implement the constructor and another method for the following `WordPairList` class.

```
public class WordPairList
{

    /** The list of word pairs, initialized by the constructor. */
    private ArrayList<WordPair> allPairs;


    /** Constructs a WordPairList object as described in part (a).
     *  Precondition: words.length >= 2
     */
    public WordPairList(String[] words)
    { /* to be implemented in part (a) */ }


    /** Returns the number of matches as described in part (b).
     */
    public int numMatches()
    { /* to be implemented in part (b) */ }
```

(a) Write the constructor for the `WordPairList` class. The constructor takes an array of strings `words` as a parameter and initializes the instance variable `allPairs` to an `ArrayList` of `WordPair` objects.

A `WordPair` object consists of a word from the array paired with a word that appears later in the array. The `allPairs` list contains `WordPair` objects `(words[i], words[j])` for every `i` and `j`, where $0 \le i < j <$ `words.length`. Each `WordPair` object is added exactly once to the list.

The following examples illustrate two different `WordPairList` objects.

Example 1

```
String[] wordNums = {"one", "two", "three"};
WordPairList exampleOne = new WordPairList(wordNums);
```

After the code segment has executed, the `allPairs` instance variable of `exampleOne` will contain the following `WordPair` objects in some order.

```
("one", "two"), ("one", "three"), ("two", "three")
```

Example 2

```
String[] phrase = {"the", "more", "the", "merrier"};
WordPairList exampleTwo = new WordPairList(phrase);
```

After the code segment has executed, the `allPairs` instance variable of `exampleTwo` will contain the following `WordPair` objects in some order.

```
("the", "more"), ("the", "the"), ("the", "merrier"),
("more", "the"), ("more", "merrier"), ("the", "merrier")
```

Class information for this question

```
public class WordPair

public WordPair(String first, String second)
public String getFirst()
public String getSecond()

public class WordPairList

private ArrayList<WordPair> allPairs

public WordPairList(String[] words)
public int numMatches()
```

Complete the `WordPairList` constructor below.

```
/** Constructs a WordPairList object as described in part (a).
 * Precondition:
```

(b) Write the `WordPairList` method `numMatches`. This method returns the number of `WordPair` objects in `allPairs` for which the two strings match.

For example, the following code segment creates a `WordPairList` object.

```
String[] moreWords = {"the", "red", "fox", "the", "red"};
WordPairList exampleThree = new WordPairList(moreWords);
```

After the code segment has executed, the `allPairs` instance variable of `exampleThree` will contain the following `WordPair` objects in some order. The pairs in which the first string matches the second string are shaded for illustration.

```
("the", "red"), ("the", "fox"), ("the", "the"),
("the", "red"), ("red", "fox"), ("red", "the"),
("red", "red"), ("fox", "the"), ("fox", "red"),
("the", "red")
```

The call `exampleThree.numMatches()` should return `2`.

Class information for this question

```
public class WordPair

public WordPair(String first, String second)
public String getFirst()
public String getSecond()

public class WordPairList

private ArrayList<WordPair> allPairs

public WordPairList(String[] words)
public int numMatches()
```

Complete method `numMatches` below.

```
/** Returns the number of matches as described in part (b).
 */
```

4. This question involves reasoning about arrays of integers. You will write two static methods, both of which are in a class named `ArrayTester`.

```
public class ArrayTester
{

    /** Returns an array containing the elements of column c of arr2D in the same order as
     *  they appear in arr2D.
     *  Precondition: c is a valid column index in arr2D.
     *  Postcondition: arr2D is unchanged.
     */
    public static int[] getColumn(int[][] arr2D, int c)
    {  /* to be implemented in part (a) */  }


    /** Returns true if and only if every value in arr1 appears in arr2.
     *  Precondition: arr1 and arr2 have the same length.
     *  Postcondition: arr1 and arr2 are unchanged.
     */
    public static boolean hasAllValues(int[] arr1, int[] arr2)
    {  /* implementation not shown */  }


    /** Returns true if arr contains any duplicate values;
     *          false otherwise.
     */
    public static boolean containsDuplicates(int[] arr)
    {  /* implementation not shown */  }


    /** Returns true if square is a Latin square as described in part (b);
     *          false otherwise.
     *  Precondition: square has an equal number of rows and columns.
     *                square has at least one row.
     */
    public static boolean isLatin(int[][] square)
    {  /* to be implemented in part (b) */  }
```

(a) Write a static method `getColumn`, which returns a one-dimensional array containing the elements of a single column in a two-dimensional array. The elements in the returned array should be in the same order as they appear in the given column. The notation `arr2D[r][c]` represents the array element at row `r` and column `c`.

The following code segment initializes an array and calls the `getColumn` method.

```
int[][] arr2D = { { 0, 1, 2 },
                  { 3, 4, 5 },
                  { 6, 7, 8 },
                  { 9, 5, 3 } };

int[] result = ArrayTester.getColumn(arr2D, 1);
```

When the code segment has completed execution, the variable `result` will have the following contents.

```
result: {1, 4, 7, 5}
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Complete method `getColumn` below.

```
/** Returns an array containing the elements of column c of arr2D in the same order as they
 *  appear in arr2D
```

(b) Write the static method `isLatin`, which returns `true` if a given two-dimensional square array is a *Latin square*, and otherwise, returns `false`.

A two-dimensional square array of integers is a Latin square if the following conditions are true.

- The first row has no duplicate values.
- All values in the first row of the square appear in each row of the square.
- All values in the first row of the square appear in each column of the square.

Examples of Latin Squares

1	2	3
2	3	1
3	1	2

10	30	20	0
0	20	30	10
30	0	10	20
20	10	0	30

Examples that are NOT Latin Squares

1	2	1
2	1	1
1	1	2

Not a Latin square because the first row contains duplicate values

1	2	3
3	1	2
7	8	9

Not a Latin square because the elements of the first row do not all appear in the third row

1	2
1	2

Not a Latin square because the elements of the first row do not all appear in either column

The `ArrayTester` class provides two helper methods: `containsDuplicates` and `hasAllValues`. The method `containsDuplicates` returns `true` if the given one-dimensional array `arr` contains any duplicate values and `false` otherwise. The method `hasAllValues` returns `true` if and only if every value in `arr1` appears in `arr2`. You do not need to write the code for these methods.

Class information for this question

```
public class ArrayTester

public static int[] getColumn(int[][] arr2D, int c)
public static boolean hasAllValues(int[] arr1, int[] arr2)
public static boolean containsDuplicates(int[] arr)
public static boolean isLatin(int[][] square)
```

Complete method `isLatin` below. Assume that `getColumn` works as specified, regardless of what you wrote in part (a). You must use `getColumn`, `hasAllValues`, and `containsDuplicates` appropriately to receive full credit.

```
/** Returns true if square is a Latin square as described in part (b);
 *          false otherwise.
 *  Precondition: square has an equal number of rows and columns.
 *                square has at least one row.
 */
public static boolean isLatin(int[][] square)
```

STOP

END OF EXAM

AP COMPUTER SCIENCE A • EXERCISE SHEET

2.12 Informal Run-Time Analysis

Name: _____ Class: _____ Date: _____ **Total: 8 marks**

KEY WORDS run-time analysis 运行时间分析 • nested loop 嵌套循环

Objective

Build the skills to answer exam questions on **informal run-time analysis**.

You must be able to:

- count how many times a statement executes as a measure of a program's work
- understand that a loop over n items runs its body about n times
- explain why a **nested loop** over n items runs its inner body about n^2 times

1 Worked examples

Study these first. Each one shows the method for a question type used later.

■ Counting executions

We estimate a program's work by counting how many times a statement runs.

- A single loop over n items runs its body about n times.
- A **nested loop** over n items runs its inner body about $n \times n = n^2$ times.

So nested loops grow much faster than single loops as n increases.

■ Worked count

For a loop that runs n times **inside** a loop that runs n times, the inner statement runs n^2 times. If $n = 5$ that is 25 executions; if n doubles to 10 it jumps to 100 – four times the work. A single loop would only double.

2 Practice

2.1 State roughly how many times a single loop over n items runs its body. [1]

2.2 State roughly how many times a nested loop over n items runs its inner body. [1]

2.3 A single loop runs over 100 items. State roughly how many times its body runs. [1]

---

3 Exam-style questions

3.1 A single loop over n items runs its body about [1]

- **A** 1
- **B** n
- **C** n^2
- **D** 2^n

times.

3.2 A nested loop over n items runs its inner body about [1]

- **A** n
- **B** n^2
- **C** $\log n$
- **D** 1

times.

3.3 A nested loop processes $n = 10$ items.

(a) State roughly how many inner-body runs there are. [1]

(b) State whether this grows faster than a single loop. [1]

(c) State roughly how many runs a single loop over 10 items has. [1]

Solutions

2.1 about n times.

2.2 about n^2 times.

2.3 about 100 times.

3.1 B.

3.2 B.

3.3 (a) about 100 (10^2). (b) yes. (c) about 10.