Week 1

AP Computer Science A

Homework bundle for this week

本周作业合集

exercises.lol

Contents 目录

1. Using Objects and Methods

Starter Quiz · 课前小测

AP Computer Science A Name: _____ Score: _____

1. An algorithm is best described as...
   - ☐ a finite, ordered set of steps that solves a problem
   - ☐ a single Java command
   - ☐ a type of computer
   - ☐ a run-time error
2. Which type stores whole numbers like 42?
   - ☐ int
   - ☐ double
   - ☐ boolean
   - ☐ String
3. Evaluate the Java expression 10 + 2 * 3.

   _____
4. In Java, the single = sign means...
   - ☐ store the right side into the left variable
   - ☐ test for equality
   - ☐ print a value
   - ☐ declare a type
5. What is the value of (int) 3.9 in Java?

   _____
6. x += 3 is exactly the same as...
   - ☐ x = x + 3
   - ☐ x = 3
   - ☐ x == x + 3
   - ☐ x + 3
7. An API documents a class's methods by telling you their...
   - ☐ names, parameters, and return values
   - ☐ internal machine code
   - ☐ memory addresses
   - ☐ file sizes
8. What does Math.pow(2, 3) equal (as a number)?

   _____
9. In Java, what is the value of the int expression 7 / 2?

   _____
10. What is 17 % 5 in Java (the remainder)?

    _____

1. Using Objects and Methods

Answers · 答案

AP Computer Science A

1. **a finite, ordered set of steps that solves a problem**

   Finite, ordered steps that finish — that's an algorithm.

2. **int**

   int stores integers; double stores decimals.

3. **16**

   * first (2*3=6), then + (10+6=16).

4. **store the right side into the left variable**

   = is assignment; == tests equality.

5. **3**

   Casting to int truncates (drops decimals) → 3, not 4.

6. **x = x + 3**

   Compound assignment is shorthand for x = x + 3.

7. **names, parameters, and return values**

   You program to the documented name/params/return, not the inner code.

8. **8**

   2 to the power 3 = 8 (returned as the double 8.0).

9. **3**

   int division drops the remainder: 7/2 = 3.

10. **2**

    17 = 3×5 + 2, so the remainder is 2.

AP COMPUTER SCIENCE A • EXERCISE SHEET

1.1 Introduction to Algorithms, Programming, and Compilers

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

KEY WORDS compiler 编译器 • program 程序 • syntax error 语法错误 • algorithm 算法 • logic error 逻辑错误

Objective

Build the skills to answer exam questions on **algorithms, programming, and compilers**.

You must be able to:

- describe an **algorithm** 算法 as a finite set of clear, ordered steps
- explain how **source code** 源代码 is written in Java
- describe how a **compiler** 编译器 translates Java into a runnable form
- distinguish a **syntax error** 语法错误 from a **logic error** 逻辑错误

1 Worked examples

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

Algorithm and program

An **algorithm** is a finite, ordered set of clear steps that solves a problem. A **program** carries out an algorithm; its **source code** is written in a language such as Java and its statements run in a defined order.

Compiler

A **compiler** translates Java source code into a form the computer can run.

Two kinds of error

- **Syntax error** — breaks the language's rules (e.g. a missing semicolon); the program will not compile.
- **Logic error** — compiles and runs, but gives the **wrong result**.

2 Practice

2.1 Define an algorithm. [1]

2.2 State what a compiler does. [1]

2.3 State the difference between a syntax error and a logic error. [2]

3 Exam-style questions

3.1 A compiler translates [1]

- **A** machine code into Java
- **B** Java source code into a runnable form
- **C** nothing at all
- **D** data into bits

3.2 A program that runs but produces the wrong answer has a [1]

- **A** syntax error
- **B** logic error
- **C** compiler error
- **D** case with no error

3.3 A Java statement is missing its semicolon.

(a) Name the type of error. [1]

(b) State whether the program will compile. [1]

(c) State the order in which a program's statements run. [1]

Solutions

2.1 a finite set of clear, ordered steps that solves a problem.

2.2 it translates Java source code into a form the computer can run.

2.3 a syntax error breaks the language's rules so it will not compile; a logic error compiles and runs but gives the wrong result.

3.1 B.

3.2 B.

3.3 (a) a syntax error. (b) no. (c) in the order written (top to bottom).

1.2 Variables and Data Types

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

KEY WORDS type 类型 • variable 变量 • primitive types 基本类型 • floating-point 浮点 • identifiers 标识符

Objective

Build the skills to answer exam questions on **variables and data types**.

You must be able to:

- understand that a **variable** 变量 is a named location in memory
- declare variables of the primitive types `int`, `double`, and `boolean`
- distinguish an integer from a **floating-point** 浮点 value
- use meaningful, valid **identifiers** 标识符

1 Worked examples

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

Variables and primitive types

A **variable** is a named memory location that stores a value. Each variable has a fixed **data type**:

- `int` — a whole number, e.g. `int age = 17;`
- `double` — a decimal (floating-point) number, e.g. `double price = 2.5;`
- `boolean` — `true` or `false`, e.g. `boolean done = false;`

Identifiers

Use clear, valid names (letters, digits, `_`; not starting with a digit), such as `studentCount`.

2 Practice

2.1 Name the three primitive types above. [2]

2.2 State which type stores a whole number. [1]

2.3 Write a declaration for a `double` variable named `price` holding `4.99`. [1]

---

3 Exam-style questions

3.1 Which type stores `true` or `false`? [1]

- **A** `int`
- **B** `double`
- **C** `boolean`
- **D** `String`

3.2 A valid declaration of an integer is [1]

- **A** `int x = 3.5;`
- **B** `int x = 3;`
- **C** `double x = true;`
- **D** `int = 3;`

3.3 Consider `int count = 5;` and `double avg = 8.5;`.

(a) State the type of `count`. [1]

(b) State the type of `avg`. [1]

(c) State whether `boolean done = 1;` is valid. [1]

Solutions

2.1 `int`, `double`, `boolean`.

2.2 `int`.

2.3 `double price = 4.99;`.

3.1 **C**.

3.2 **B**.

3.3 (a) `int`. (b) `double`. (c) no — a `boolean` must be `true` or `false`.

AP COMPUTER SCIENCE A • EXERCISE SHEET

1.3 Expressions and Output

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

KEY WORDS integer division 整数除法 • expression 表达式 • operator precedence 运算符优先级 • type 类型 • casting 类型转换

Objective

Build the skills to answer exam questions on **expressions and output**.

You must be able to:

- evaluate arithmetic **expressions** 表达式 using `+`, `-`, `*`, `/`, and `%`
- understand **integer division** 整数除法 and the remainder operator `%`
- apply **operator precedence** 运算符优先级
- use `System.out.print` and `System.out.println`

1 Worked examples

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

■ Arithmetic and integer division

With two `int` operands, `/` is **integer division** (the fraction is dropped) and `%` gives the remainder: `7 / 2` is `3` and `7 % 2` is `1`.

■ Precedence and mixing types

`*`, `/`, `%` are done before `+`, `-`; use parentheses to change the order. Mixing an `int` and a `double` gives a `double`: `5 / 2.0` is `2.5`.

■ Output

`System.out.print` writes without a newline; `System.out.println` adds a newline at the end.

2 Practice

2.1 Evaluate `7 / 2` in Java (integer division). [1]

2.2 Evaluate `7 % 2`. [1]

2.3 State the difference between `print` and `println`. [1]

3 Exam-style questions

3.1 In Java, `9 / 4` evaluates to [1]

- **A** `2.25`
- **B** `2`
- **C** `3`
- **D** `1`

3.2 `System.out.println` differs from `print` by [1]

- **A** printing nothing
- **B** adding a newline at the end
- **C** rounding the value
- **D** casting the value

3.3 Evaluate the following.

(a) `10 % 3`. [1]

(b) `10 / 3`. [1]

(c) State the value **and type** of `10 / 3.0`. [1]

Solutions

2.1 3.

2.2 1.

2.3 `println` moves to a new line after printing; `print` does not.

3.1 **B**.

3.2 **B**.

3.3 (a) 1. (b) 3. (c) about 3.33, a `double`.

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

1.4 Assignment Statements and Input

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

KEY WORDS assignment 赋值 • input 输入 • assignment operator 赋值运算符 • variable 变量 • expression 表达式

Objective

Build the skills to answer exam questions on **assignment statements and input**.

You must be able to:

- use the **assignment operator** 赋值运算符 `=` to store an expression's value
- understand that assignment **replaces** the old value of a variable
- trace how a variable's value changes across assignments
- read user **input** 输入 with a `Scanner`

1 Worked examples

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

Assignment

`variable = expression;` evaluates the right side and stores it in the variable, **replacing** the old value. The left side must be the variable; the right side is the expression.

Tracing

```
int x = 4;
x = x + 3;   // x is now 7
```

Input

A `Scanner` reads values typed by the user, e.g. `int n = sc.nextInt();`.

2 Practice

2.1 State what the assignment operator does. [1]

2.2 Trace: `int y = 10; y = y * 2; y = y - 5;`. State the final value of `y`. [2]

2.3 Name the class commonly used to read user input. [1]

3 Exam-style questions

3.1 After `int a = 3; a = 8;`, the value of `a` is [1]

- **A** `3`
- **B** `8`
- **C** `11`
- **D** `5`

3.2 In `x = y + 2;`, the left side is [1]

- **A** the expression
- **B** the variable being assigned
- **C** a method call
- **D** program output

3.3 Trace: `int n = 5; n = n + 1; n = n * 3;`.

(a) State the value of `n` after `n = n + 1`. [1]

(b) State the value of `n` after `n = n * 3`. [1]

(c) Name the class used to read input. [1]

Solutions

2.1 it stores the value of the right-side expression in the left-side variable, replacing the old value.

2.2 $10 \times 2 = 20$, then $20 - 5 = 15$.

2.3 `Scanner`.

3.1 **B**.

3.2 **B**.

3.3 (a) `6`. (b) `18`. (c) `Scanner`.

AP COMPUTER SCIENCE A • EXERCISE SHEET

1.5 Casting and Range of Variables

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

KEY WORDS cast 强制类型转换 • casting 类型转换 • integer overflow 整数溢出 • truncates 截断 • type 类型

Objective

Build the skills to answer exam questions on **casting and the range of variables**.

You must be able to:

- use a **cast** 强制类型转换 such as `(int)` or `(double)` to convert a numeric type
- understand that casting a `double` to an `int` **truncates** 截断 the fraction
- describe how **integer overflow** 整数溢出 occurs
- round a `double` by adding 0.5 before casting to `int`

1 Worked examples

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

Casting

A **cast** converts a value from one numeric type to another. Casting a `double` to an `int` **truncates** (drops) the fractional part: `(int) 7.9` is `7`.

Range and overflow

An `int` can only hold values within a fixed **range**; a result too large causes **integer overflow**.

Rounding

To round to the nearest integer, add 0.5 first: `(int)(x + 0.5)`.

2 Practice

2.1 State what casting a `double` to an `int` does to the fractional part. [1]

2.2 Evaluate `(int) 8.7`. [1]

2.3 Show how to round `4.6` to the nearest `int` using a cast, and give the result. [2]

3 Exam-style questions

3.1 `(int) 3.99` evaluates to [1]

- **A** `4`
- **B** `3`
- **C** `3.99`
- **D** `0`

3.2 A result too large for the `int` type causes [1]

- **A** truncation
- **B** integer overflow
- **C** a syntax error
- **D** rounding

3.3 Let `double d = 5.8;`.

(a) State the value of `(int) d`. [1]

(b) State the value of `(int)(d + 0.5)`. [1]

(c) State what a cast from `double` to `int` drops. [1]

Solutions

2.1 it drops (truncates) it.

2.2 `8`.

2.3 `(int)(4.6 + 0.5) = (int) 5.1 = 5`.

3.1 **B**.

3.2 **B**.

3.3 (a) `5`. (b) `6`. (c) the fractional part.

1.6 Compound Assignment Operators

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

KEY WORDS compound assignment operators 复合赋值运算符 • decrement 自减 • increment 自增 • assignment 赋值 • variable 变量

Objective

Build the skills to answer exam questions on **compound assignment operators**.

You must be able to:

- use the **compound assignment operators** 复合赋值运算符 `+=`, `-=`, `*=`, `/=`, `%=`
- understand that `x += 3` is shorthand for `x = x + 3`
- use the **increment** 自增 (`++`) and **decrement** 自减 (`--`) operators
- trace a variable through a sequence of compound assignments

1 Worked examples

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

Compound assignment

`x += 3` means `x = x + 3`; similarly `-=`, `*=`, `/=`, `%=` update a variable using its current value.

Increment and decrement

`x++` adds 1 to `x`; `x--` subtracts 1.

Example

```
int x = 5;
x += 2;   // x is now 7
```

2 Practice

2.1 State what `x += 3` is shorthand for. [1]

2.2 Trace: `int x = 4; x *= 3; x -= 2;`. State the final value of `x`. [2]

2.3 State what `x++` does. [1]

3 Exam-style questions

3.1 `x -= 5` is shorthand for [1]

- **A** `x = 5`
- **B** `x = x - 5`
- **C** `x = x + 5`
- **D** `x = -5`

3.2 After `int n = 7; n++;`, `n` is [1]

- **A** `6`
- **B** `7`
- **C** `8`
- **D** `1`

3.3 Trace: `int a = 10; a /= 2; a++;`.

(a) State the value of `a` after `a /= 2`. [1]

(b) State the value of `a` after `a++`. [1]

(c) Rewrite `a += 4` in full (without the compound operator). [1]

Solutions

2.1 `x = x + 3`.

2.2 $4 \times 3 = 12$, then $12 - 2 = 10$.

2.3 it increases `x` by 1.

3.1 **B**.

3.2 **C**.

3.3 (a) `5`. (b) `6`. (c) `a = a + 4`.

AP COMPUTER SCIENCE A • EXERCISE SHEET

1.7 Application Program Interface (API) and Libraries

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

KEY WORDS class 类 • api 应用程序接口 • library 库 • abstraction 抽象 • reference 引用

Objective

Build the skills to answer exam questions on **the Application Program Interface (API) and libraries**.

You must be able to:

- understand that an **API** 应用程序接口 describes how to use a class
- explain that a **library** 库 is a collection of prewritten, reusable classes
- read an API entry to find a method's name, parameters, and return value
- locate classes in the AP Java **Quick Reference**

1 Worked examples

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

■ API and library

An **API** describes **how to use** a class without knowing how it is built inside. A **library** is a collection of prewritten classes that programmers reuse.

■ Reading an API

An API entry gives a method's **name**, its **parameters**, and its **return value** — enough to call it correctly.

■ Quick Reference

Reusing library classes supports **abstraction** and saves work; the AP Java **Quick Reference** lists the classes and methods you may use.

2 Practice

2.1 State what an API describes. [1]

2.2 Define a library. [1]

2.3 State one benefit of reusing library classes. [1]

3 Exam-style questions

3.1 An API describes [1]

- **A** how a class is built inside
- **B** how to use a class
- **C** the hardware
- **D** the compiler

3.2 A library is a collection of [1]

- **A** errors
- **B** prewritten reusable classes
- **C** variables
- **D** bytes

3.3 A student looks up how to call a method.

(a) Name what documents the method's parameters and return value. [1]

(b) State one benefit of using a library. [1]

(c) Name the reference used in the AP exam. [1]

Solutions

2.1 how to use a class (without knowing its internal details).

2.2 a collection of prewritten classes that can be reused.

2.3 it saves work and supports abstraction.

3.1 B.

3.2 B.

3.3 (a) the API. (b) it saves writing code from scratch. (c) the AP Java Quick Reference.

1.8 Documentation with Comments

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

KEY WORDS comments 注释 • postcondition 后置条件 • precondition 前置条件 • compiler 编译器 • program 程序

Objective

Build the skills to answer exam questions on **documentation with comments**.

You must be able to:

- write single-line **comments** 注释 (`//`) and block comments (`/* */`)
- understand that comments are ignored by the compiler
- state a **precondition** 前置条件 and a **postcondition** 后置条件
- use comments to explain intent

1 Worked examples

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

Comments

`// this is a single-line comment` and `/* this is a block comment */`. Comments are **ignored by the compiler** but help humans read the code.

Pre- and postconditions

- **Precondition** — what must be true **before** the code runs.
- **Postcondition** — what will be true **after** it runs.

Good comments explain **intent**, not obvious code.

2 Practice

2.1 State the two ways to write comments in Java. [2]

2.2 State whether comments affect how the program runs. [1]

2.3 Define a precondition. [1]

3 Exam-style questions

3.1 A single-line comment in Java starts with [1]

- **A** `/*`
- **B** `//`
- **C** `#`
- **D** `<!--`

3.2 Comments are [1]

- **A** run by the compiler
- **B** ignored by the compiler
- **C** errors
- **D** program output

3.3 A method assumes its input number is positive.

(a) Name this kind of assumption. [1]

(b) Name what is stated to be true after the method finishes. [1]

(c) State one good purpose of a comment. [1]

Solutions

2.1 `//` for a single line and `/* */` for a block.

2.2 no — they are ignored by the compiler.

2.3 something that must be true before the code runs.

3.1 **B**.

3.2 **B**.

3.3 (a) a precondition. (b) a postcondition. (c) to explain the intent / aid readability.

AP COMPUTER SCIENCE A • EXERCISE SHEET

1.9 Method Signatures

Name: _____ Class: _____ Date: _____ **Total: 10 marks**

KEY WORDS parameter 形参 • argument 实参 • method signature 方法签名 • method overloading 方法重载 • type 类型

Objective

Build the skills to answer exam questions on **method signatures**.

You must be able to:

- describe the parts of a **method signature** 方法签名: the name and parameter list
- distinguish a **parameter** 形参 from an **argument** 实参
- understand that arguments must match the number, order, and types of the parameters
- explain **method overloading** 方法重载

1 Worked examples

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

■ Method signature

A signature is the method's **name** plus its **parameter list**, e.g. `max(int a, int b)`.

■ Parameter vs argument

A **parameter** appears in the method **header** (`int a`); an **argument** is the value passed in the **call** (`max(3, 5)`). Arguments must match the parameters in number, order, and type.

■ Overloading

Overloading lets several methods share a name with **different parameter lists**.

2 Practice

2.1 State the two parts of a method signature. [2]

2.2 State the difference between a parameter and an argument. [2]

2.3 In the call `sum(3, 5)`, state whether `3` is a parameter or an argument. [1]

3 Exam-style questions

3.1 A method signature consists of the method name and its [1]

- **A** return value
- **B** parameter list
- **C** body
- **D** comments

3.2 Two methods with the same name but different parameter lists is called [1]

- **A** overriding
- **B** overloading
- **C** casting
- **D** recursion

3.3 A method `area(int w, int h)` is called with `area(4, 6)`.

(a) Name `w` and `h` in the header. [1]

(b) Name `4` and `6` in the call. [1]

(c) State whether `area(4)` is a valid call. [1]

Solutions

2.1 the method name and its parameter list.

2.2 a parameter is named in the method header; an argument is the value supplied in the call.

2.3 an argument.

3.1 B.

3.2 B.

3.3 (a) parameters. (b) arguments. (c) no — it has the wrong number of arguments.

AP COMPUTER SCIENCE A • EXERCISE SHEET

1.10 Calling Class Methods

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

KEY WORDS class (static) method 类方法 • object 对象 • class 类 • expression 表达式 • type 类型

Objective

Build the skills to answer exam questions on **calling class methods**.

You must be able to:

- understand that a **class method** 类方法 (a `static` method) belongs to the class
- call one with `ClassName.methodName(arguments)`
- distinguish a value-returning method from a `void` method
- use a returned value inside a larger expression

1 Worked examples

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

■ Class (static) methods

A `static` method belongs to the **class**, not to an object, and is called as `ClassName.methodName(arguments)`, e.g. `Math.max(3, 8)`.

■ Returning a value vs `void`

Some methods **return** a value you can use; a `void` method returns **nothing**. A returned value can be used in a larger expression: `int x = Math.max(3, 8) + 1;`.

2 Practice

2.1 State how you call a `static` (class) method. [1]

2.2 State the difference between a value-returning method and a `void` method. [2]

2.3 State the value of `Math.max(3, 8)`. [1]

---

3 Exam-style questions

3.1 A class (static) method is called using [1]

- **A** `object.method()`
- **B** `ClassName.method()`
- **C** `new Class()`
- **D** just the method name

3.2 A `void` method [1]

- **A** returns a number
- **B** returns nothing
- **C** must be non-static
- **D** always causes an error

3.3 The call `Math.min(10, 4)` is used.

(a) State its return value. [1]

(b) State the general syntax for calling a static method. [1]

(c) State the value of `x` after `int x = Math.min(10, 4) + 1;`. [1]

Solutions

2.1 with `ClassName.methodName(arguments)`.

2.2 a value-returning method hands back a value the caller can use; a `void` method returns nothing.

2.3 8.

3.1 **B**.

3.2 **B**.

3.3 (a) 4. (b) `ClassName.methodName(arguments)`. (c) 5.

AP COMPUTER SCIENCE A • EXERCISE SHEET

1.11 Math Class

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

KEY WORDS math class 数学类 • pseudorandom 伪随机 • class 类 • type 类型 • object 对象

Objective

Build the skills to answer exam questions on **the Math class**.

You must be able to:

- call **Math class** 数学类 methods such as `Math.abs`, `Math.pow`, and `Math.sqrt`
- understand that `Math.pow(b, e)` computes b^e and returns a `double`
- generate a **pseudorandom** 伪随机 `double` in $[0, 1)$ with `Math.random`
- scale and shift `Math.random` to produce a random `int`

1 Worked examples

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

■ Common Math methods

- `Math.abs(-5)` is `5`.
- `Math.pow(2, 3)` is `8.0` (a `double`, since it computes 2^3).
- `Math.sqrt(9)` is `3.0`.

■ Random values

`Math.random()` returns a `double` in the range $[0, 1)$. To get a random `int` from 1 to 6:

```
int roll = (int)(Math.random() * 6) + 1;
```

2 Practice

2.1 State the value of `Math.abs(-7)`. [1]

2.2 State the value **and type** of `Math.pow(3, 2)`. [2]

2.3 State the range of values `Math.random()` can return. [1]

3 Exam-style questions

3.1 `Math.sqrt(16)` returns [1]

- **A** `4` (an `int`)
- **B** `4.0` (a `double`)
- **C** `8.0`
- **D** `256`

3.2 `Math.random()` returns a `double` in the range [1]

- **A** $[0, 1]$
- **B** $[0, 1)$
- **C** $[1, 10]$
- **D** any value at all

3.3 To roll a die: `(int)(Math.random() * 6) + 1`.

(a) State the smallest possible value. [1]

(b) State the largest possible value. [1]

(c) State the value of `Math.pow(5, 0)`. [1]

Solutions

2.1 7.

2.2 `9.0`, a `double`.

2.3 from 0 up to (but not including) 1.

3.1 **B**.

3.2 **B**.

3.3 (a) `1`. (b) `6`. (c) `1.0`.

1.12 Objects: Instances of Classes

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

KEY WORDS object 对象 • class 类 • instance 实例 • methods 方法 • attributes 属性

Objective

Build the skills to answer exam questions on **objects as instances of classes**.

You must be able to:

- understand that an **object** 对象 is a specific **instance** 实例 of a class
- explain that a **class** 类 is a blueprint defining state and behaviour
- distinguish the class (the type) from an object (a value of that type)
- describe an object's **attributes** 属性 and **methods** 方法

1 Worked examples

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

■ Class and object

A **class** is a **blueprint** that defines the state and behaviour of its objects. An **object** is a specific **instance** created from that blueprint.

■ Attributes and methods

An object's **attributes** are its data; its **methods** are the behaviours it can perform.

■ Many objects

Many separate objects can be created from the same class — each has its own attribute values.

2 Practice

2.1 State the relationship between a class and an object. [1]

2.2 State the difference between an attribute and a method. [2]

2.3 State whether many objects can be created from one class. [1]

3 Exam-style questions

3.1 An object is [1]

- **A** a blueprint
- **B** an instance of a class
- **C** a primitive type
- **D** a compiler

3.2 A class defines an object's [1]

- **A** only its data
- **B** only its methods
- **C** its state and behaviour
- **D** the hardware

3.3 A `Dog` class is written.

(a) State what a specific dog created from it is called. [1]

(b) Name an object's data. [1]

(c) Name an object's behaviours. [1]

Solutions

2.1 a class is a blueprint; an object is a specific instance created from it.

2.2 an attribute is the object's data; a method is a behaviour the object can perform.

2.3 yes.

3.1 **B**.

3.2 **C**.

3.3 (a) an object (an instance). (b) its attributes. (c) its methods.

AP COMPUTER SCIENCE A • EXERCISE SHEET

1.13 Object Creation and Storage (Instantiation)

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

KEY WORDS reference variable 引用变量 • alias 别名 • constructor 构造方法 • instantiation 实例化 • null 空值

Objective

Build the skills to answer exam questions on **object creation and storage (instantiation)**.

You must be able to:

- create an object by calling a **constructor** 构造方法 with `new`
- understand that a **reference variable** 引用变量 stores the object's location
- describe what it means for a reference to hold `null`
- recognize that two references can form an **alias** 别名

1 Worked examples

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

■ Creating an object

`Dog d = new Dog();` — the `new` keyword allocates memory and calls the **constructor** (instantiation).

■ Reference variables and null

A **reference variable** stores the object's **location**, not the object itself. A reference of value `null` refers to **no object**.

■ Alias

`Dog e = d;` makes `e` and `d` refer to the **same** object — an alias.

2 Practice

2.1 State the keyword used to create an object. [1]

2.2 State what a reference variable stores. [1]

2.3 State what `null` means. [1]

3 Exam-style questions

3.1 An object is created with the keyword [1]

- **A** `make`
- **B** `new`
- **C** `create`
- **D** `class`

3.2 A reference variable that refers to no object holds [1]

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

3.3 `Cat a = new Cat(); Cat b = a;`.

(a) State how many `Cat` objects exist. [1]

(b) State what `b` refers to. [1]

(c) Name the situation of two references to one object. [1]

Solutions

2.1 `new`.

2.2 the location (memory address) of the object.

2.3 it refers to no object.

3.1 **B**.

3.2 **B**.

3.3 (a) one. (b) the same object as `a`. (c) an alias.

1.14 Calling Instance Methods

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

KEY WORDS instance method 实例方法 • nullpointerexception 空指针异常 • instance 实例 • class 类 • object 对象

Objective

Build the skills to answer exam questions on **calling instance methods**.

You must be able to:

- call an **instance method** 实例方法 with `objectName.methodName(arguments)`
- understand that it acts on the **state** of that object
- explain why calling a method on a `null` reference causes a **NullPointerException** 空指针异常
- distinguish calling an instance method from a class method

1 Worked examples

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

> ■ **Instance method call**
>
> Call it on an object: `objectName.methodName(arguments)`. It acts on the **state** of the object it is called on, and its return value can be used in a larger expression.
>
> ■ **The null trap**
>
> Calling a method on a reference that is `null` causes a **NullPointerException**.
>
> ■ **Instance vs class method**
>
> An **instance** method is called on an **object**; a **class** (`static`) method is called on the **class**.

2 Practice

2.1 State the syntax to call an instance method. [1]

2.2 State what happens if you call a method on a `null` reference. [1]

2.3 State the difference between an instance method and a class method. [2]

3 Exam-style questions

3.1 An instance method is called on [1]

- **A** the class
- **B** an object
- **C** a primitive value
- **D** the compiler

3.2 Calling a method on a `null` reference throws a [1]

- **A** syntax error
- **B** NullPointerException
- **C** overflow error
- **D** nothing at all

3.3 `String s = "hi";`.

(a) State the value of `s.length()`. [1]

(b) State whether `length` is an instance or a class method. [1]

(c) State what `s = null; s.length();` causes. [1]

Solutions

2.1 `objectName.methodName(arguments)`.

2.2 it throws a NullPointerException.

2.3 an instance method is called on an object and uses its state; a class method is called on the class.

3.1 **B**.

3.2 **B**.

3.3 (a) 2. (b) an instance method. (c) a NullPointerException.

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.

AP COMPUTER SCIENCE A • EXERCISE SHEET

1.15 String Manipulation

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

KEY WORDS string 字符串 • concatenation 拼接 • immutable 不可变 • object 对象 • reference 引用

Objective

Build the skills to answer exam questions on **String manipulation**.

You must be able to:

- create **String** 字符串 objects and combine them with **concatenation** 拼接 (`+`)
- call methods such as `length`, `substring`, and `indexOf`
- understand that a String is **immutable** 不可变
- compare strings with `equals` and `compareTo`, not `==`

1 Worked examples

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

■ Strings and methods

Strings are made from literals and joined with `+`. They are **zero-indexed** (the first character is at index 0) and **immutable** (methods return **new** strings):

- `"hello".length()` is `5`
- `"hello".substring(1, 3)` is `"el"` (indices 1 and 2)
- `"hello".indexOf("l")` is `2`

■ Comparing strings

Use `s.equals(t)` for equality and `s.compareTo(t)` for order — **not** `==`, which compares references.

2 Practice

2.1 State the result of `"cat" + "fish"`. [1]

2.2 State the value of `"data".length()`. [1]

2.3 State why you should compare strings with `equals`, not `==`. [1]

3 Exam-style questions

3.1 Strings in Java are [1]

- **A** mutable
- **B** immutable
- **C** numbers
- **D** booleans

3.2 The first character of a String is at index [1]

- **A** `1`
- **B** `0`
- **C** `-1`
- **D** its length

3.3 `String s = "computer";`.

(a) State the value of `s.length()`. [1]

(b) State the value of `s.substring(0, 3)`. [1]

(c) State the correct way to test whether `s` equals `"computer"`. [1]

Solutions

2.1 `"catfish"`.

2.2 4.

2.3 `==` compares references (identity), while `equals` compares the actual characters.

3.1 **B**.

3.2 **B**.

3.3 (a) 8. (b) `"com"`. (c) `s.equals("computer")`.

2. This question involves the `SignedText` class, which contains methods that are used to include a signature as part of a string of text. You will write the complete `SignedText` class, which contains a constructor and two methods.

The `SignedText` constructor takes two `String` parameters. The first parameter is a first name and the second parameter is a last name. The length of the second parameter is always greater than or equal to 1.

The `getSignature` method takes no parameters and returns a formatted signature string constructed from the first and last names according to the following rules.

- If the first name is an empty string, the returned signature string contains just the last name.
- If the first name is not an empty string, the returned signature string is the first letter of the first name, a dash `("-")`, and the last name, in that order.

The `addSignature` method returns a possibly revised copy of its `String` parameter. The parameter will contain at most one occurrence of the object's signature, at either the beginning or the end of the parameter. The returned string is created from the parameter according to the following rules.

- If the object's signature does not occur in the `String` parameter of the method, the returned `String` is the value of the parameter with the signature added to the end.
- If the object's signature occurs at the end of the `String` parameter, the returned `String` is the unchanged value of the parameter.
- If the object's signature occurs at the beginning of the `String` parameter, the returned `String` is the value of the original parameter with the signature removed from the beginning and appended to the end of the parameter.

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

Statement	Method Call Return Value (blank if none)	Explanation
`SignedText st1 = new SignedText("", "Wong");`		The `SignedText` object `st1` has an empty first name and last name `"Wong"`.
`String temp = st1.getSignature();`	`"Wong"`	
`SignedText st2 = new SignedText("henri", "dubois");`		The `SignedText` object `st2` has first name `"henri"` and last name `"dubois"`.
`temp = st2.getSignature();`	`"h-dubois"`	
`SignedText st3 = new SignedText("GRACE", "LOPEZ");`		The `SignedText` object `st3` has first name `"GRACE"` and last name `"LOPEZ"`.
`temp = st3.getSignature();`	`"G-LOPEZ"`	
`SignedText st4 = new SignedText("", "FOX");`		The `SignedText` object `st4` has an empty first name and last name `"FOX"`.
`String text = "Dear";`		
`temp = st4.addSignature(text);`	`"DearFOX"`	The signature does not occur in the `addSignature` parameter, so the returned string is the value of the parameter with the signature appended.
`text = "Best wishesFOX";`		
`temp = st4.addSignature(text);`	`"Best wishesFOX"`	The signature occurs at the end of the `addSignature` parameter, so the returned string is the unchanged value of the parameter.

Statement	Method Call Return Value (blank if none)	Explanation
`text = "FOXThanks";`		
`temp = st4.addSignature(text);`	`"ThanksFOX"`	The signature occurs at the beginning of the `addSignature` parameter, so the returned string is the value of the original parameter with the signature removed from the beginning and appended to the end of the parameter.
`text = "G-LOPEZHello";`		
`temp = st3.addSignature(text);`	`"HelloG-LOPEZ"`	The signature occurs at the beginning of the `addSignature` parameter, so the returned string is the value of the original parameter with the signature removed from the beginning and appended to the end of the parameter.

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

2. This question involves methods that distribute text across lines of an electronic sign. The electronic sign and the text to be displayed on it are represented by the `Sign` class. You will write the complete `Sign` class, which contains a constructor and two methods.

The `Sign` class constructor has two parameters. The first parameter is a `String` that contains the message to be displayed on the sign. The second parameter is an `int` that contains the *width* of each line of the sign. The width is the positive maximum number of characters that can be displayed on a single line of the sign.

A sign contains as many lines as are necessary to display the entire message. The message is split among the lines of the sign without regard to spaces or punctuation. Only the last line of the sign may contain fewer characters than the width indicated by the constructor parameter.

The following are examples of a message displayed on signs of different widths. Assume that in each example, the sign is declared with the width specified in the first column of the table and with the message `"Everything on sale, please come in"`, which contains 34 characters.

Width of the Sign	Sign Display

15	`Everything on s`
	`ale, please com`
	`e in`

17	`Everything on sal` `e, please come in`
40	`Everything on sale, please come in`

In addition to the constructor, the `Sign` class contains two methods.

The `numberOfLines` method returns an `int` representing the number of lines needed to display the text on the sign. In the previous examples, `numberOfLines` would return `3`, `2`, and `1`, respectively, for the sign widths shown in the table.

The `getLines` method returns a `String` containing the message broken into lines separated by semicolons (`;`) or returns `null` if the message is the empty string. The constructor parameter that contains the message to be displayed will not include any semicolons. As an example, in the first row of the preceding table, `getLines` would return `"Everything on s;ale, please com;e in"`. No semicolon should appear at the end of the `String` returned by `getLines`.

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

Statement	Method Call Return Value (blank if none)	Explanation
`String str;`		
`int x;`		
`Sign sign1 = new Sign("ABC222DE", 3);`		The message for `sign1` contains 8 characters, and the sign has lines of width 3.
`x = sign1.numberOfLines();`	`3`	The sign needs three lines to display the 8-character message on a sign with lines of width 3.
`str = sign1.getLines();`	`"ABC;222;DE"`	Semicolons separate the text displayed on the first, second, and third lines of the sign.
`str = sign1.getLines();`	`"ABC;222;DE"`	Successive calls to `getLines` return the same value.
`Sign sign2 = new Sign("ABCD", 10);`		The message for `sign2` contains 4 characters, and the sign has lines of width 10.
`x = sign2.numberOfLines();`	`1`	The sign needs one line to display the 4-character message on a sign with lines of width 10.
`str = sign2.getLines();`	`"ABCD"`	No semicolon appears, since the text to be displayed fits on the first line of the sign.
`Sign sign3 = new Sign("ABCDEF", 6);`		The message for `sign3` contains 6 characters, and the sign has lines of width 6.
`x = sign3.numberOfLines();`	`1`	The sign needs one line to display the 6-character message on a sign with lines of width 6.
`str = sign3.getLines();`	`"ABCDEF"`	No semicolon appears, since the text to be displayed fits on the first line of the sign.
`Sign sign4 = new Sign("", 4);`		The message for `sign4` is an empty string.
`x = sign4.numberOfLines();`	`0`	There is no text to display.
`str = sign4.getLines();`	`null`	There is no text to display.
`Sign sign5 = new Sign("AB_CD_EF", 2);`		The message for `sign5` contains 8 characters, and the sign has lines of width 2.
`x = sign5.numberOfLines();`	`4`	The sign needs four lines to display the 8-character message on a sign with lines of width 2.
`str = sign5.getLines();`	`"AB;_C;D_;EF"`	Semicolons separate the text displayed on the four lines of the sign.

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

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.

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. The `StringChecker` interface describes classes that check if strings are valid, according to some criterion.

```
public interface StringChecker
{
   /** Returns true if str is valid. */
   boolean isValid(String str);
}
```

A `CodeWordChecker` is a `StringChecker`. A `CodeWordChecker` object can be constructed with three parameters: two integers and a string. The first two parameters specify the minimum and maximum code word lengths, respectively, and the third parameter specifies a string that must not occur in the code word. A `CodeWordChecker` object can also be constructed with a single parameter that specifies a string that must not occur in the code word; in this case the minimum and maximum lengths will default to 6 and 20, respectively.

The following examples illustrate the behavior of `CodeWordChecker` objects.

Example 1

```
StringChecker sc1 = new CodeWordChecker(5, 8, "$");
```

Valid code words have 5 to 8 characters and must not include the string `"$"`.

Method call	Return value	Explanation
`sc1.isValid("happy")`	`true`	The code word is valid.
`sc1.isValid("happy$")`	`false`	The code word contains `"$"`.
`sc1.isValid("Code")`	`false`	The code word is too short.
`sc1.isValid("happyCode")`	`false`	The code word is too long.

Example 2

```
StringChecker sc2 = new CodeWordChecker("pass");
```

Valid code words must not include the string `"pass"`. Because the bounds are not specified, the length bounds are 6 and 20, inclusive.

Method call	Return value	Explanation
`sc2.isValid("MyPass")`	`true`	The code word is valid.
`sc2.isValid("Mypassport")`	`false`	The code word contains `"pass"`.
`sc2.isValid("happy")`	`false`	The code word is too short.
`sc2.isValid("1,000,000,000,000,000")`	`false`	The code word is too long.

Write the complete `CodeWordChecker` class. Your implementation must meet all specifications and conform to all examples.

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 `n`th occurrence of the string `str` with the string `repl`. If the `n`th 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)
```

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

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)
```