Primitive Types & Using Objects

AP Computer Science A ▪ Unit 1

AP Computer Science A

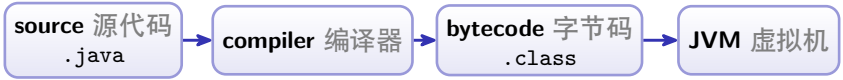
```
int x = 5;
```

What we will cover

1. Programs & compilers
2. Variables & data types
3. Expressions & casting
4. Methods & the Math class
5. Objects & strings

1 Foundations

From source code to running program



An **algorithm** 算法 is a step-by-step plan; a **program** 程序 is that plan in a **programming language** 编程语言 like Java. The compiler turns human-readable source into **bytecode** the JVM runs.

Documentation with Comments

Comments 注释 are ignored by the compiler but explain code to people.

Three comment styles

```
// a single-line comment
/* a block comment that
   may span several lines */
/** Returns the larger of a and b. */
int max(int a, int b)
```

A `/** ... */` **Javadoc** comment documents a method's purpose, parameters and return value, including its preconditions and postconditions.

Method Signatures

A **method signature** 方法签名 is the method's name plus its parameter types, e.g. `nextInt()` or `substring(int, int)`.

```
public  double  area (double w, double h)
```

access — return type — name — parameters: type and name

```
call:  double a = area(3.0, 4.5);
```

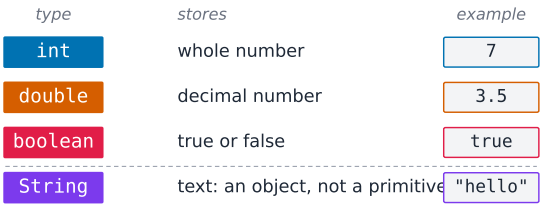
> A call must supply **arguments** 实参 that match the parameters in number, type and order.

Variables & primitive types

A **variable** 变量 is a named box that stores a value of a fixed **type** 类型. Declare it with the type first:

```
int count = 3;
double price = 2.5;
boolean done = false;
```

The main **primitive types** 基本类型 are `int`, `double` and `boolean`; `String` is a class, not a primitive.



Primitives and references

A primitive variable holds its value. A variable of a class type holds a **reference** 引用: an arrow to the object.

> Assigning a primitive copies the value; assigning a reference copies the arrow, so both names reach one object (an alias).

primitive

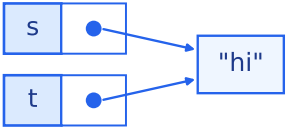
```
int y = x;
```



the value is copied

reference

```
String t = s;
```



both point to one object: an alias

2 Expressions

Expressions, casting & range

An **expression** 表达式 combines values with **operators** 运算符 ($+$ $-$ $*$ / $\%$), following order of operations.

Integer division & casting

`7 / 2` is 3 (integer division drops the remainder); `7 % 2` is `1`.

Cast 类型转换 to get a decimal: `(double) 7 / 2` is `3.5`. A value too big for its type causes **overflow** 溢出.

Compound assignment 复合赋值: `x += 3` means `x = x + 3`.

Integer division truncates, and the cast position decides

the trap	`7 / 2` is 3, not 3.5 – two `int`s divide to an `int`
the fix	cast one operand first: `(double) 7 / 2` is 3.5
the wrong fix	`(double) (7 / 2)` is 3.0 – the truncation already happened inside the brackets
so	the position of the cast decides whether the truncation happens at all
escape sequence 转义序列	`\n` and `\t` inside a String – a backslash changes what the next character means

Watch for integer division producing a truncated result where a decimal was wanted. It compiles, it runs, and it is silently wrong – a **logic error** 逻辑错误.

Assignment Statements and Input

An **assignment** 赋值 `x = expr;` evaluates the right side, then stores it in the left variable.

Read input with a `Scanner`

```
Scanner sc = new Scanner(System.in);
int n = sc.nextInt();
String name = sc.nextLine();
```

`x = x + 1;` is not algebra: it replaces the old value.

The shorthand operators

compound assignment	`x += 5` means `x = x + 5`; likewise `-=`, `*=`, `/=`, `%=`
increment 自增 and decrement 自减	`x++` and `x--` add or subtract one
why they matter	they are everywhere in `for` loops, so you must read them fluently even if you never write them
the caution	`x /= 2` on an `int` still truncates – the shorthand does not change the type rules

> These are pure convenience: every one expands to a longer form you already know. Expand it mentally when tracing.

3 Methods & Math

Methods, the API & the Math class

A **method** 方法 is named, reusable code. Its **signature** 方法签名 is its name, parameters & return type.

API 应用程序接口 & libraries 库
documentation of ready-made classes to reuse

`Math` class
`Math.sqrt(9)`, `Math.pow`, `Math.random()`

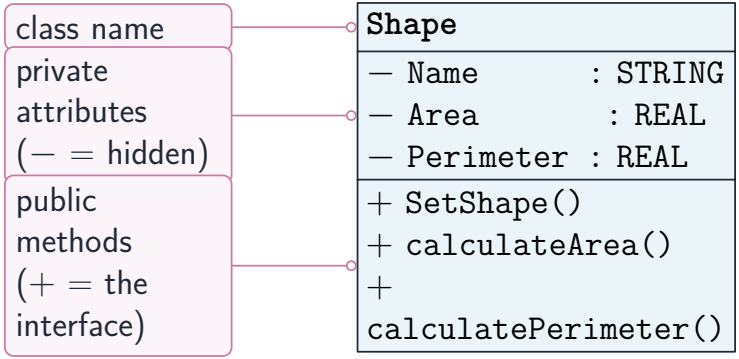
A **comment** 注释 (`// ...`) documents code for humans; the compiler ignores it.

Class methods, instance methods, and where code comes from

class (static) method 类方法	belongs to the class itself, so you call it on the class name: `Math.abs(x)`. No object needed
instance method	belongs to an object, so you call it on a reference: `s.length()`
a library 库	pre-written classes you can use; Java's is the API
compiled 编译	Java source is compiled once to bytecode, then run – so syntax errors surface before the program starts
abstraction 抽象	using a method by its name and contract, without knowing how it works inside

> If you find yourself creating an object just to call a method on it, check whether the method is `static` – `Math.sqrt` on a `new Math()` does not compile.

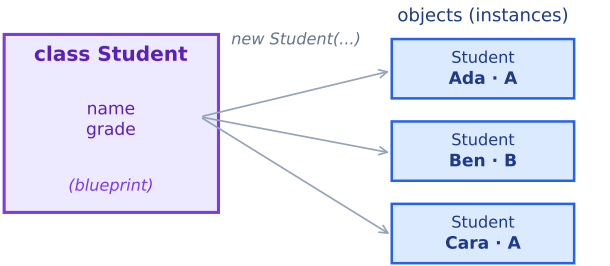
Methods, the API & the Math class, in detail



A class diagram: private attributes and public methods

4 Objects & strings

Objects: instances of classes



class = blueprint (type) · object = instance with its own field values

A **class** 类 is a blueprint; an **object** 对象 is a built **instance** 实例.

Instantiate 实例化 with `new`:

```
Scanner s = new
Scanner(...);
```

Call **instance methods** 实例方法 on an object with the dot operator: `s.nextInt()`.

Superclass and subclass

superclass 父类	holds attributes and behaviours shared by several subclasses
subclass 子类	`extends` the superclass, inheriting everything it has and adding its own
inheritance relationship 继承关系	an is-a relationship: a `Dog` *is a* `Animal`
method overriding 方法重写	a subclass redefines an inherited method with the same signature, and its version runs
why bother	shared code is written once, and code that works on the superclass works on every subclass

> Ask "is-a" before writing `extends`. A `Car` *has a* `Engine`, so that is a field, not a superclass – the commonest design error at this level.

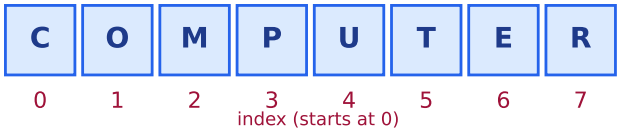
Object Creation and Storage (Instantiation)

Instantiation 实例化 creates an object with the `new` keyword, which calls a **constructor 构造函数**:

> Two references can point to the **same** object; comparing them with == compares addresses, not contents.

String manipulation

"COMPUTER".substring(0, 4) → "COMP"



valid indices: 0 … *s*.length() − 1 · *s*.charAt(*i*) is the char at *i*

A **String** 字符串 is an object – an ordered sequence of characters, indexed from **0**.

- `s.length()`
- `s.substring(0, 4)`
- `s.indexOf(``x'')`

Concatenate 拼接 with +: `` ``Hi, '' + name``.

The two most-tested String traps

`substring(a, b)`	includes index `a` but excludes `b` – so its length is `b - a`
comparison	use `.equals`, never `==` – `==` compares *references*, so two equal Strings can test false
immutable 不可变	a String cannot be changed; every "modification" returns a new String
indexing	`indexOf` returns `-1` when not found, and indices run `0` to `length()-1`

> Both traps compile cleanly and both give wrong answers at run time. They are among the most-tested points on the paper for exactly that reason.

Key points

Java **source** → compiler → **bytecode** → JVM

Strongly typed: `int`, `double`, `boolean`

`7/2=3`; **cast** for decimals; `x += 3`

A **class** is a blueprint; an **object** is an instance

Worked example – the excluded endpoint

`String s = "COMPUTER";` (indices 0–7)

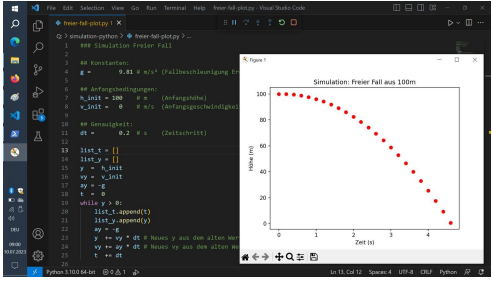
`s.length()` → 8
`s.substring(0, 4)` → "COMP" (index 4 is excluded)
`s.substring(4)` → "UTER" (index 4 to the end)
`s.indexOf("PU")` → 3
`s.indexOf("X")` → -1 (not found)

Miscounting `substring`'s excluded endpoint is the single most common slip on this paper.

Introduction to Algorithms, Programming, and Compilers

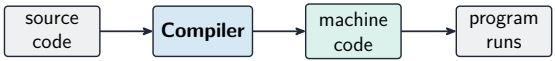
An **algorithm** 算法 is a finite, step-by-step procedure that solves a problem.

> A **program** 程序 expresses an algorithm in a language a computer can run.



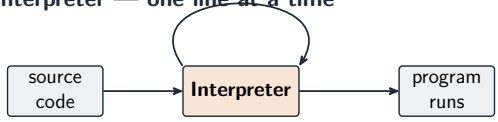
Compiler versus interpreter

Compiler — translate the whole program once



translated once; the machine code then runs many times

Interpreter — one line at a time



reads, translates and runs one line at a time (every run); stops at the first error

A compiler translates the whole program at once; an interpreter runs it line by line

Exam tips

- Trace code by hand line by line, tracking each variable's value in a table – the exam rewards careful tracing over guessing.
- Know Java's primitive types and that integer division truncates ($7/2$ gives 3); use a cast or a double for real division.
- Distinguish **compile-time** errors (syntax, types) from **run-time** errors – know the named ones: `ArithmeticException` (`int` $\div$ `0`), `NullPointerException` (method on a null reference), `StringIndexOutOfBoundsException` / `ArrayIndexOutOfBoundsException` – and **logic** errors (wrong output).
- Follow operator precedence and initialise every variable before you use it.
- On the free-response, write **complete, compilable** Java – return the right type and match the method header exactly.

5 Recap

Unit 1 – key takeaways

1 Trace

line by line in a table – no guessing

2 Types

$7/2 = 3$; cast or double for real division

3 Errors

compile-time vs run-time vs logic

4 FRQ

complete compilable Java; match the header

Check yourself

1. `"COMPUTER".substring(0,4)` gives what?
2. What does `indexOf` return when the string is absent?
3. Is `int` a primitive or a reference type?
4. What does the compiler turn source code into?

Answers **1.** `"COMP"` **2.** `-1` **3.** primitive **4.** bytecode, run by the JVM