Writing Classes

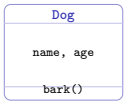
AP Computer Science A ▪ Unit 3

AP Computer Science A

```
class Dog
```


What we will cover

1. Abstraction & program design
2. The anatomy of a class
3. Constructors & methods
4. References & static members
5. Scope & the this keyword

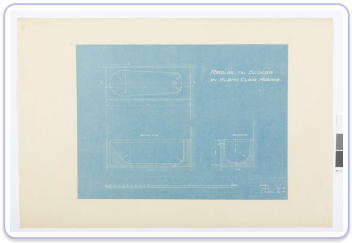


1 Design

Abstraction & program design

Abstraction 抽象 hides detail behind a simple interface, so we manage complexity one piece at a time.

> Good **program design** 程序设计 breaks a problem into classes, each with one clear responsibility. A well-designed class is easier to test, reuse, and change without breaking the rest.



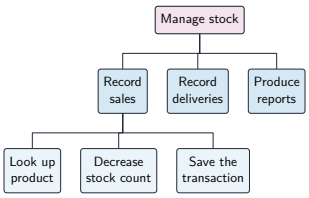
Class blueprint

Encapsulation, and the responsibility that comes with design

encapsulation 封装	keep **instance variables** 实例变量 `private` and expose them only through methods
accessor (getter) 访问器	returns a field's value without letting the caller change it
mutator (setter) 修改器	changes a field, and is the one place a validity check can live
overloading 重载	several methods with the same name and different parameter lists
system reliability 系统可靠性	the program doing its job without failure – sometimes a legal requirement, not just a quality goal

> Design also carries responsibility beyond the code: **intellectual property** 知识产权 in what you reuse, and **open source** 开源 licences that state the conditions of that reuse.

Decomposing a program into modules and sub-modules



> Decomposing a program into modules and sub-modules

2 Anatomy

The anatomy of a class

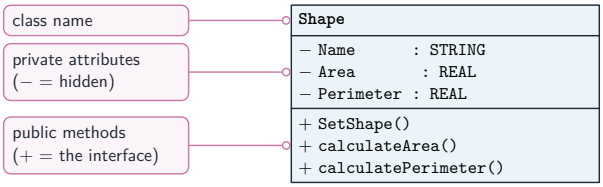
class Dog

```
private String name;
private int age;
```
fields 字段 *(the object's data)*

```
public Dog(String n) ...   constructor 构造函
public void bark() ...     methods 方法
```

Jigsaw design

A class diagram



A class diagram: private attributes and public methods

Constructors & methods

Constructor 构造函数
runs once at `new`; sets the object's initial **fields** 字段

Method 方法
a named action; may take **parameters** 形参 and return a value

A method **returns** 返回 a value with `return`; a `void` method returns nothing.

3 References & static

References & static members

An object variable holds a **reference** 引用 – a pointer to the object, not the object itself.
Passing it lets a method change the same object.

Instance member
belongs to **each object**: `d.bark()`

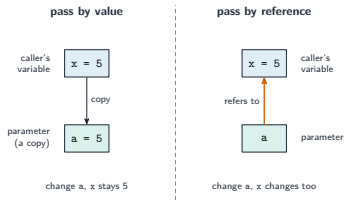
Static 静态 (class) member
belongs to the **class**:
`Math.sqrt(9)`

One more kind of variable, and one responsibility

Static (class) variable	A **static (class) variable** belongs to the class, not to any object: one copy shared by every instance – a counter of objects created, or a constant.
Instance variable	Each object has its own, so changing one object's copy changes nothing elsewhere.
How to spot it	Declared with `static`, and reached through the class name rather than an object reference.
Responsibility	**Legal and intellectual-property** rules apply to code as to anything else: licences, attribution, and other people's data.

Ask whether the value describes *one object* or *the class*. That question decides static or instance every time.

References & static members, in detail



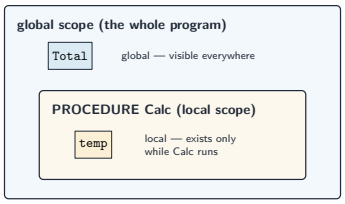
Java always passes

Scope & the this keyword

Scope 作用域 is where a name is visible. A local variable lives only inside its method; a field lives as long as the object.

`this` refers to the current object. Use it when a parameter name hides a field: `this.name = name;` sets the field from the parameter.

Scope & the this keyword, in detail



A global variable is visible everywhere; a local variable only inside its block

Key points

A class has **fields**, a **constructor** & **methods**

The constructor sets initial state at `new`

Object variables hold a **reference**; **static** belongs to the class

`this` names the current object

Worked example – passing a reference

`s` is a `Student` with score `50`; we call `tweak(s)`

Java passes the reference by value: `tweak` gets a copy of the arrow, pointing at the same object. So `s.setScore(90)` inside `tweak` does change the caller's object – but reassigning `s = new Student()` inside `tweak` does not.

You can change what the object contains; you cannot change which object the caller's variable points to.

Exam tips

- Design with methods and classes: encapsulate data as `private` fields and expose behaviour through public methods.
- Know the difference between an **object** and its **class**, and that objects are passed **by value** – the parameter gets a copy of the reference, so a method can change the object's state, but reassigning the parameter does not affect the caller (Java has no pass-by-reference).
- Traverse arrays and `ArrayLists` safely – size is `length` vs `.size()`, and removing during a loop shifts indices.
- **Trace** a **recursive** method to determine its result: find the base case first, then follow each recursive call to its return value (writing recursive code is outside the exam's scope).
- Recognise **inheritance** vocabulary – superclass, subclass, method overriding, and that every class is a subclass of `Object` (designing and implementing inheritance is outside the exam's scope).

4 Recap

Unit 3 – key takeaways

1. **Encapsulate**
   private fields, public methods
2. **Reference**
   passed by value – reassigning the parameter is local
3. **Lists**
   length vs .size(); removing shifts indices
4. **Recursion**
   find the base case, then follow each return

Check yourself

1. Can a method change the caller's object through a reference?
2. Can it change which object the caller's variable points to?
3. What does `static` mean – per object, or per class?
4. What does `this` refer to?

Answers **1.** yes **2.** no **3.** per class – shared by all objects **4.** the current object