Writing classes

Java Reference

Fields, constructor & methods

A class 类 is a blueprint for objects. Its fields 字段 store data, its constructor 构造方法 sets up a new object, and its methods 方法 are the actions. `this.name` means "this object's `name`". Create an object with `new`.

```
public class Main {
    public static void main(String[] args) {
        Dog d = new Dog("Rex", 3);
        System.out.println(d.describe());   // Rex is 3 years old
        d.haveBirthday();
        System.out.println(d.describe());   // Rex is 4 years old
    }
}

class Dog {
    private String name;
    private int age;

    public Dog(String name, int age) {   // constructor
        this.name = name;
        this.age = age;
    }

    public String describe() {
        return name + " is " + age + " years old";
    }

    public void haveBirthday() {
        age++;
    }
}
```

A `static` member belongs to the class itself, not to any one object. Call a static method on the class name — like `Math.max` — and a static field is shared by every object:

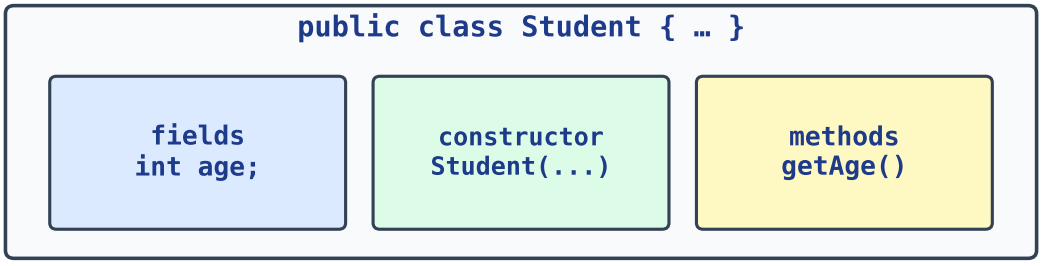
```
public class Main {
    public static void main(String[] args) {
        System.out.println(Counter.made());   // 0
```

```
        new Counter();
        new Counter();
        System.out.println(Counter.made());   // 2
    }
}

class Counter {
    private static int count = 0;      // shared by ALL Counter objects

    public Counter() { count++; }

    public static int made() {         // called on the class:
↪   Counter.made()
        return count;
    }
}
```



Exam tip: constructor name matches the class; fields store state; methods do work

Fields store state; constructor builds; methods act

Encapsulation

Encapsulation 封装 means hiding data behind methods. Mark fields `private` so outside code can't touch them directly; expose an accessor 访问方法 (getter) to read, and a method to change them safely. The method can **guard** the data — here a deposit must be positive.

```
public class Main {
    public static void main(String[] args) {
        Account a = new Account(100);
        a.deposit(50);
```

```
        a.deposit(-999);                          // rejected by the guard
        System.out.println(a.getBalance());       // 150
    }
}

class Account {
    private int balance;                          // hidden from outside

    public Account(int start) {
        balance = start;
    }

    public void deposit(int amount) {
        if (amount > 0) balance += amount;        // guard keeps balance
↪   valid
    }

    public int getBalance() {                     // accessor (getter)
        return balance;
    }
}
```

Common mistakes

- A constructor has the class name and no return type (not even `void`).
- Use `this.field` to tell a field apart from a parameter with the same name.
- Make fields `private` and reach them through getter/setter methods (encapsulation).