Java 讲义

中文版

Contents

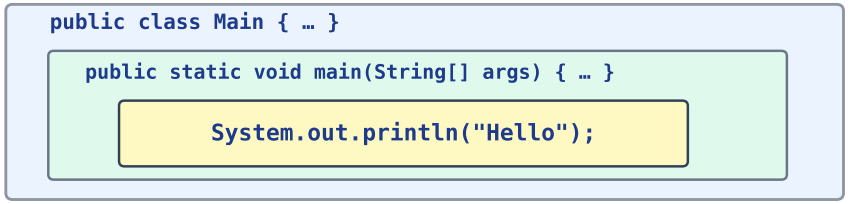
1 Java 基础

1.1 类、main 与输出

每个 Java 程序都住在一个类 (class) 里面。它从一个叫 `main` 的方法 (method) 开始。`System.out.println(...)` 打印一行;`System.out.print(...)` 打印但不换行。

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
        System.out.println("I am learning Java.");
    }
}
```

- Java 是编译 (compile) 型的: 编译器 (compiler) 先检查整个程序, 然后才运行。
- 每条语句 (statement) 都以分号 ; 结尾。



Exam tip: every program lives in a class; execution starts at main; end each statement with ;

Java programs live in a class and start at main

1.2 变量与基本类型

变量 (variable) 必须声明 (declare) 它的类型。常见的基本类型 (primitive type):`int`(整数)、`double`(小数)、`boolean`(`true`/`false`)、`char`(一个字符)。

```
public class Main {
    public static void main(String[] args) {
        int age = 17;
        double price = 9.99;
        boolean passed = true;
        System.out.println(age + " " + price + " " + passed);
    }
}
```

1.3 注释与风格

注释 (comment) 是 `//`(一行) 或 `/* ... */`(一块)。把花括号 `{ }` 里的代码缩进。类名首字母大写; 变量和方法用驼峰命名 (camelCase)(首字母小写)。

```
public class Main {
    public static void main(String[] args) {
        // greet the user
        String firstName = "Mei";
        System.out.println("Hi, " + firstName);
    }
}
```

常见错误

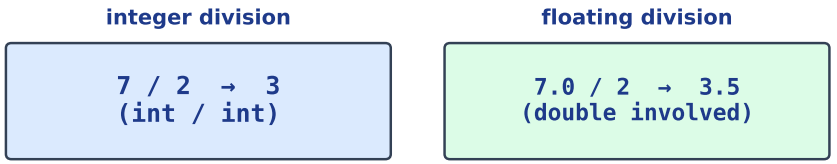
- 每条语句都以分号 `;` 结尾。
- `main` 必须正好是 `public static void main(String[] args)`。
- `println` 会换行,`print` 不会。

2 运算符与表达式

2.1 算术与赋值

Java 的算术 (arithmetic) 用 `+ - * /` 和 `%`(取余)。两个 `int` 相除时,`/` 是整数除法 (integer division)—— 会丢掉小数。`+=`、`-=`、`++` 是简写。

```
public class Main {
    public static void main(String[] args) {
        int a = 7, b = 2;
        System.out.println(a / b);    // 3 (integer division)
        System.out.println(a % b);    // 1
        double x = 7.0 / 2;           // 3.5 (one side is double)
        System.out.println(x);
    }
}
```



Exam tip: int/int drops the decimal; make one side double for a real quotient; % is remainder

int/int truncates; involve a double for a real quotient

2.2 使用对象:String、Math、包装类

有些值是带方法的对象 (object)。`String` 有 `.length()`、`.substring()`、`.toUpperCase()`。`Math` 有 `Math.max`、`Math.sqrt`、`Math.pow`。包装类 (wrapper class)(`Integer`、`Double`) 把基本类型包起来 —— 例如 `Integer.parseInt("42")`。

```
public class Main {
    public static void main(String[] args) {
        String s = "Hello";
        System.out.println(s.length());                  // 5
        System.out.println(s.toUpperCase());             // HELLO
        System.out.println(Math.max(3, 9));              // 9
        System.out.println(Integer.parseInt("42") + 1);  // 43
    }
}
```

`Math.random()` 返回一个从 `0.0` 到 (但不包含)`1.0` 的随机 (random)`double`。把它放大再强制转换, 就得到整数范围 —— 这是 AP 考试的惯用写法:

```
public class Main {
    public static void main(String[] args) {
        // 1 到 6 的随机整数 (掷骰子)
        int roll = (int) (Math.random() * 6) + 1;
        System.out.println(roll >= 1 && roll <= 6);   // true
    }
}
```

- `Integer.MAX_VALUE`(2147483647) 和 `Integer.MIN_VALUE` 是 `int` 的上下限; 超过就会绕回 (溢出)。
- 一个还没有指向任何对象的对象变量是 `null`; 对它调用方法会抛出 NullPointerException。

2.3 强制转换与类型转换

强制转换 (cast) 改变一个值的类型。`(int)` 丢掉小数; 当你需要精确结果时,`(double)` 可以避免整数除法。

```
public class Main {
    public static void main(String[] args) {
        double pi = 3.99;
        System.out.println((int) pi);            // 3
        int total = 7, n = 2;
        System.out.println((double) total / n);  // 3.5
    }
}
```

常见错误

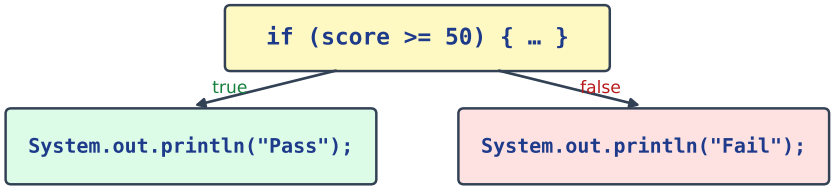
- 整数除法:`5 / 2` 得 2, 不是 2.5。先转换:`(double) 5 / 2`。
- 比较字符串 (以及其他对象) 用 `.equals()`, 不要用 `==`。
- 对两个对象用 `==` 判断的是它们是不是**同一个**对象, 而不是看起来是否相等。

3 布尔与选择

3.1 if / else

`if` 在条件 (condition) 为真时运行一个块;`else if` 和 `else` 增加更多情况。条件写在 `( )` 里, 块写在 `{ }` 里。

```
public class Main {
    public static void main(String[] args) {
        int score = 72;
        if (score >= 80) {
            System.out.println("A");
        } else if (score >= 60) {
            System.out.println("B");
        } else {
            System.out.println("fail");
        }
    }
}
```



Exam tip: always use braces; else if chains more cases; only one branch runs

if chooses the true branch; else the false branch

3.2 逻辑运算符与比较

用 `==`、`!=`、`<`、`>`、`<=`、`>=` 比较 —— 一次比较 (comparison) 得到一个 `boolean`。用 `&&`(与)、`||`(或)、`!`(非) 组合 —— 这些是逻辑运算符 (logical operator)。对 `String`, 要用 `.equals(...)`, **不要**用 `==`。

```
public class Main {
    public static void main(String[] args) {
        int age = 16;
        boolean member = true;
        System.out.println(age >= 18 && member);   // false
        String a = "hi";
        System.out.println(a.equals("hi"));        // true
    }
}
```

3.3 switch

`switch` 在许多固定值之间做选择。每个 `case` 以 `break` 结尾;`default` 是兜底。

```
public class Main {
    public static void main(String[] args) {
        int day = 3;
        switch (day) {
            case 1: System.out.println("Mon"); break;
            case 3: System.out.println("Wed"); break;
            default: System.out.println("other");
        }
    }
}
```

常见错误

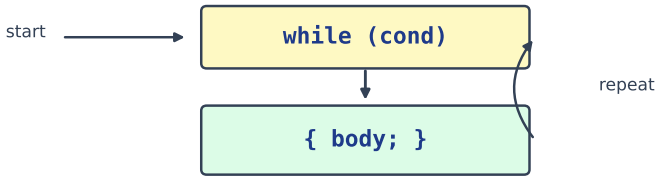
- 条件必须是 `boolean`;`if (x = 5)` 无法编译 (要用 `==`)。
- 每个 `switch` 分支都要有 `break;`, 否则会继续掉进下一个分支。
- `&&` 和 `||` 是逻辑运算符;`&` 和 `|` 是按位运算符。

4 循环

4.1 while 循环

`while` 循环在条件为真时重复。要在里面改变某些东西，否则它会永远循环。

```
public class Main {
    public static void main(String[] args) {
        int n = 1;
        while (n <= 3) {
            System.out.println(n);
            n++;
        }
    }
}
```



Exam tip: condition is checked before each pass; body must eventually make it false

while checks the condition before each pass of the body

4.2 for 循环

`for` 循环把”开始、条件、步进”打包在一行里。当你知道次数时最适合用它。

```
public class Main {
    public static void main(String[] args) {
        for (int i = 0; i < 5; i++) {
            System.out.print(i + " ");
        }
        System.out.println();   // 0 1 2 3 4
    }
}
```


4.3 累加

累加器 (accumulator) 模式：在循环之前先建一个变量，然后每一轮更新它。

```
public class Main {
    public static void main(String[] args) {
        int total = 0;
        for (int i = 1; i <= 5; i++) {
            total += i;
        }
        System.out.println(total);   // 15
    }
}
```

4.4 嵌套循环

一个循环放在另一个循环里，就是嵌套循环 (nested loop)。外层每转一轮，内层都会完整运行一遍。

```
public class Main {
    public static void main(String[] args) {
        for (int r = 0; r < 3; r++) {
            for (int c = 0; c < 3; c++) {
                System.out.print("*");
            }
            System.out.println();
        }
    }
}
```

常见错误

- `for (int i = 0; i < n; i++)` 运行 `n` 次 (0 到 `n - 1`)；用 `<=` 会多跑一次。
- 不要在 `for (...)` 或 `while (...)` 后面紧跟分号 —— 那会得到一个空循环。
- 在 `for` 头部声明计数器，它的作用域就随循环结束。

5 字符串

5.1 String 方法

String(字符串) 是文本。常用方法:.length()、.charAt(i)、.substring(a, b)、.indexOf(x)、.toUpperCase()、.equals(...)。字符串是不可变 (immutable) 的 —— 每个方法都返回一个**新**字符串。

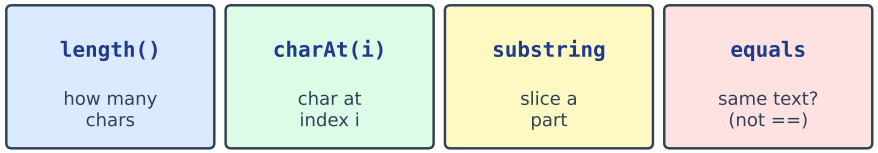
```
public class Main {
    public static void main(String[] args) {
        String s = "Python";
        System.out.println(s.length());          // 6
        System.out.println(s.charAt(0));         // P
        System.out.println(s.substring(0, 3));   // Pyt
        System.out.println(s.toUpperCase());     // PYTHON
    }
}
```

天天都会用到的方法:

方法	含义	示例 → 结果
.length()	有多少个字符	"Hi".length() → 2
.charAt(i)	取一个字符	"Hi".charAt(0) → H
.substring(a, b)	取一段, 到 b 之前为止	"Python".substring(0, 3) → Pyt
.substring(a)	从 a 到末尾	"Python".substring(3) → hon
.indexOf(x)	第一次出现的位置, 没有则 -1	"banana".indexOf("na") → 2
.equals(s)	文本相同?	"hi".equals("hi") → true
.compareTo(s)	顺序: 负数 / 0 / 正数	"apple".compareTo("banana") → 负数

compareTo 按字典顺序比较字符串 ——AP 考试的排序题会用到它:

```
public class Main {
    public static void main(String[] args) {
        String a = "apple", b = "banana";
        System.out.println(a.compareTo(b) < 0);    // true (apple 在前)
        System.out.println(a.compareTo("apple"));  // 0 (相等)
    }
}
```

Key String methods: length, charAt, substring, equals

5.2 构建与遍历字符串

用 + 连接字符串 (拼接,concatenation)。用循环和 .charAt(i) 访问每个字符。

```
public class Main {
    public static void main(String[] args) {
        String word = "banana";
        int count = 0;
        for (int i = 0; i < word.length(); i++) {
            if (word.charAt(i) == 'a') count++;
        }
        System.out.println(count);   // 3
    }
}
```

在循环里构建长字符串时,StringBuilder 快得多: 先逐段 append, 最后调用一次 .toString()。

```
public class Main {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder();
        for (int i = 1; i <= 5; i++) {
            sb.append(i).append(" ");
        }
        System.out.println(sb.toString().trim());   // 1 2 3 4 5
    }
}
```

常见错误

- 字符串不可变:s.toUpperCase() 返回一个新字符串, 要把结果存下来。

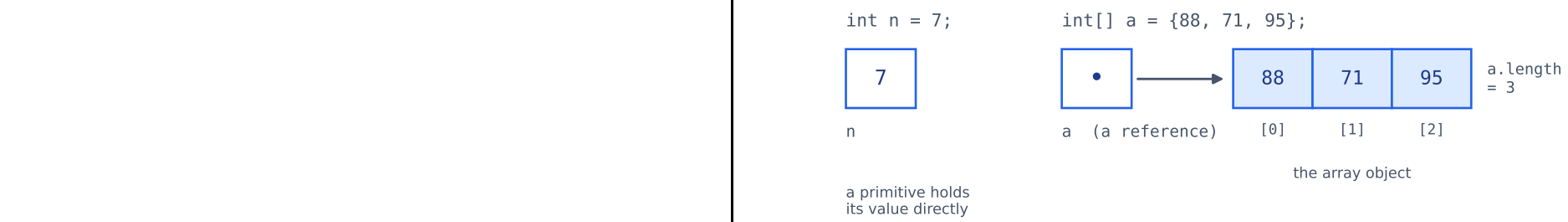
- 取字符用 `s.charAt(i)`; 长度是 `s.length()`(方法, 带 `()`)。
- 在大循环里用 `+=` 拼字符串很慢; 要用 `StringBuilder`。

6 数组

6.1 一维数组

数组 (array) 保存固定数量、同一类型的值。从 0 开始索引, 用 `.length` 取得大小。

```
public class Main {
    public static void main(String[] args) {
        int[] scores = {88, 71, 95};
        System.out.println(scores[0]);       // 88
        System.out.println(scores.length);   // 3
        scores[1] = 100;
        System.out.println(scores[1]);       // 100
    }
}
```



06-array-memory

基本类型直接存值; 数组变量存的是指向数组对象的引用

6.2 数组算法

用循环遍历数组, 求最大值、总和、计数, 或进行查找 (search)。增强 for(enhanced for, 即 for-each) 逐个读取每个值。

```
public class Main {
    public static void main(String[] args) {
        int[] a = {3, 9, 2, 7};
        int max = a[0], total = 0;
        for (int x : a) {
            if (x > max) max = x;
            total += x;
        }
        System.out.println(max + " " + total);   // 9 21
    }
}
```

6.3 二维数组

二维数组 (2-D array) 是由行和列组成的网格 (grid):`grid[row][col]`。

```
public class Main {
    public static void main(String[] args) {
        int[][] grid = {{1, 2, 3}, {4, 5, 6}};
        System.out.println(grid[1][2]);   // 6
        for (int[] row : grid) {
            for (int v : row) System.out.print(v + " ");
        }
        System.out.println();             // 1 2 3 4 5 6
    }
}
```

常见错误

- 数组的大小是 `a.length`(不带方括号, 也不带 `()`), 而且创建后固定。
- 合法下标是 `0` 到 `a.length - 1`;`a[a.length]` 会抛出 ArrayIndexOutOfBoundsException。
- 新建的 `int[5]` 会填满 0, 而不是空的。

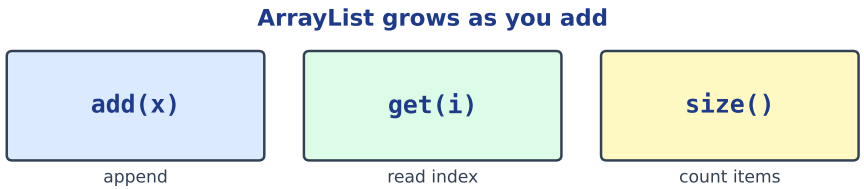
7 ArrayList

7.1 ArrayList 基础

`ArrayList` 是一个可变大小 (resizable) 的列表 —— 你添加或删除元素时它会自动增大或缩小。它存储对象, 所以要用包装类 (wrapper) 类型, 例如 `Integer`(不能用 `int`)。`<Integer>` 这部分是泛型 (generic) 类型。常用方法:`.add(x)`、`.get(i)`、`.set(i, x)`、`.size()`、`.remove(i)`。

```
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        ArrayList<Integer> nums = new ArrayList<Integer>();
        nums.add(10);
        nums.add(20);
        nums.add(30);
        System.out.println(nums.size());   // 3
        System.out.println(nums.get(1));   // 20
        nums.set(0, 99);
        System.out.println(nums);          // [99, 20, 30]
    }
}
```



Exam tip: ArrayList<Type> needs a type; indices start at 0 like arrays

ArrayList: add, get, size on a resizable list

7.2 ArrayList 算法与删除陷阱

`.remove(i)` 会把后面的每个元素向左移动 (shift) 一位。如果你一边让 `i` **递增**一边删除, 就会跳过下一个元素。解决办法: **倒着**循环, 或删除后不要让 `i` 递增。

```
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        ArrayList<Integer> nums = new ArrayList<Integer>();
        for (int n : new int[]{4, 7, 4, 9, 4}) nums.add(n);
        // 删除所有的 4 -- 倒着循环，这样删除时不会跳过元素。
        for (int i = nums.size() - 1; i >= 0; i--) {
            if (nums.get(i) == 4) nums.remove(i);
        }
        System.out.println(nums);   // [7, 9]
    }
}
```

常见错误

- `ArrayList` 用 `.size()`、`.get(i)`、`.add(...)`—— 不是数组那种 `[]`。
- 用下标正向遍历时删除元素会漏掉下一个 (remove bug)。要么倒着遍历, 要么用迭代器。
- 存的是对象而非基本类型: 用 `ArrayList<Integer>`,Java 会自动把 `int` 装箱。

8 编写类

8.1 字段、构造方法与方法

类 (class) 是对象的蓝图。它的字段 (field) 存储数据, 构造方法 (constructor) 初始化一个新对象, 方法 (method) 是它能做的动作。`this.name` 表示" 这个对象的 `name`"。用 `new` 创建对象。

```
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) {   // 构造方法
        this.name = name;
        this.age = age;
    }

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

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

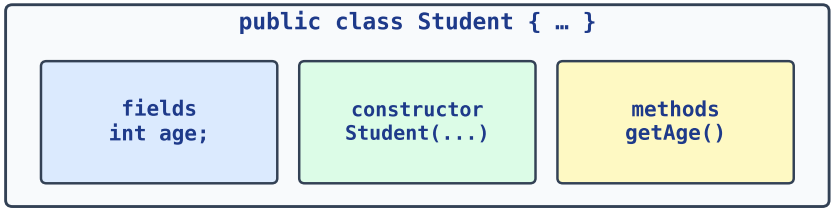
`static` 成员属于类本身，而不属于某个对象。静态方法用类名来调用 —— 就像 `Math.max`；静态字段由所有对象共享：

```
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;      // 被所有 Counter 对象共享

    public Counter() { count++; }

    public static int made() {         // 用类名调用:Counter.made()
        return count;
    }
}
```



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

Fields store state; constructor builds; methods act

8.2 封装

封装 (encapsulation) 是指把数据隐藏在方法后面。把字段标记为 `private`，这样外部代码就不能直接修改它们；再提供一个访问方法 (accessor，即 getter) 来读取，以及一个能安全修改它们的方法。方法可以**守护**数据 —— 这里要求存入的金额必须为正。

```
public class Main {
    public static void main(String[] args) {
        Account a = new Account(100);
        a.deposit(50);
        a.deposit(-999);                          // 被守护条件拒绝
        System.out.println(a.getBalance());       // 150
    }
}

class Account {
    private int balance;                          // 对外隐藏

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

    public void deposit(int amount) {
        if (amount > 0) balance += amount;        // 守护条件保持余额有效
    }

    public int getBalance() {                     // 访问方法 (getter)
        return balance;
    }
}
```

常见错误

- 构造方法与类同名，而且没有返回类型 (连 `void` 都没有)。
- 用 `this.field` 区分字段和同名的参数。
- 把字段设为 `private`，通过 getter/setter 方法访问 (封装)。

9 继承与多态

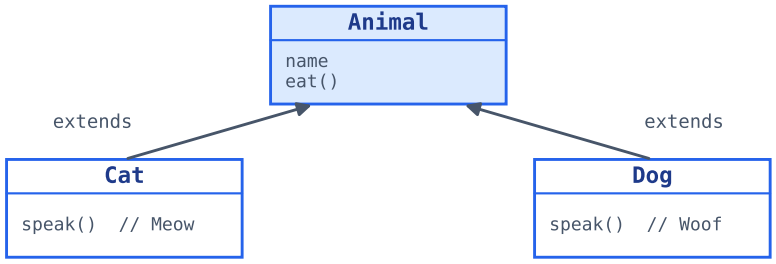
9.1 继承

继承 (inheritance) 让子类 (subclass) 复用父类 (superclass)。写 `class Cat extends Animal`,Cat 就免费获得了 `Animal` 的字段和方法。用 `super(...)` 调用父类的构造方法。

```
public class Main {
    public static void main(String[] args) {
        Cat c = new Cat("Milo");
        c.eat();      // Milo is eating  (从 Animal 继承而来)
        c.speak();    // Meow            (Cat 自己的方法)
    }
}

class Animal {
    protected String name;
    public Animal(String name) { this.name = name; }
    public void eat() { System.out.println(name + " is eating"); }
}

class Cat extends Animal {
    public Cat(String name) { super(name); }     // 调用 Animal 的构造方
↪   法
    public void speak() { System.out.println("Meow"); }
}
```



09-inheritance

Cat 和 *Dog* 都 *extends Animal:* 继承它的成员，再加上自己的方法

9.2 多态与 toString

子类可以重写 (override) 一个方法，以替换父类的版本。多态 (polymorphism) 是指一个 `Shape` 变量可以持有任何子类型,Java 会在运行时选择正确的 `toString`。`System.out.println(obj)` 会自动调用 `obj.toString()`。

```
public class Main {
    public static void main(String[] args) {
        Shape[] shapes = { new Circle(2), new Square(3) };
        for (Shape s : shapes) {
            System.out.println(s);          // 各自调用自己的 toString
        }
    }
}

class Shape {
    public String toString() { return "a shape"; }
}

class Circle extends Shape {
    private int r;
    public Circle(int r) { this.r = r; }
    public String toString() { return "Circle r=" + r; }     // 重写
}

class Square extends Shape {
    private int side;
    public Square(int side) { this.side = side; }
    public String toString() { return "Square side=" + side; }   // 重写
}
```

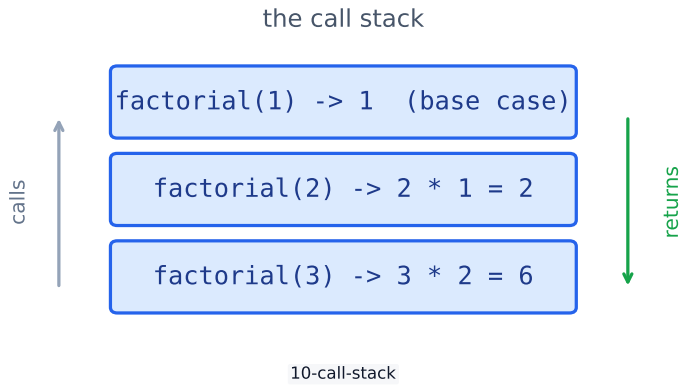
常见错误

- 重写方法的签名必须完全一致；加上 `@Override`，让编译器帮你抓错。
- `super(...)` 必须是子类构造方法的第一行。
- 子类对象**是**一个父类对象，但反过来不成立。

10 递归

10.1 递归

递归 (recursion) 是一个调用自己的方法。每个递归都需要一个基准情形 (base case, 即何时停止), 以及一个朝它靠近的递归调用 (recursive call)。没有基准情形它就永不停止, 并以栈溢出崩溃。



factorial(3) 的调用栈: 每个调用等待下层返回, 再按相反顺序返回

```
public class Main {
    public static void main(String[] args) {
        System.out.println(factorial(5));   // 120
    }

    public static int factorial(int n) {
        if (n <= 1) return 1;            // 基准情形
        return n * factorial(n - 1);     // 递归调用:n * (n-1)!
    }
}
```

跟踪一下:`factorial(5)` 等待 `factorial(4)`, 后者又等待 `factorial(3)`…… 一直到 `factorial(1)` 返回 1。然后结果再逐层相乘返回:`1 → 2 → 6 → 24 → 120`。

递归同样适用于字符串 —— 每次调用剥掉一个字符:

```
public class Main {
    public static void main(String[] args) {
        System.out.println(reverse("PYTHON"));   // NOHTYP
    }

    static String reverse(String s) {
        if (s.length() <= 1) return s;               // 基准情形
        return reverse(s.substring(1)) + s.charAt(0);
    }
}
```

归并排序 (merge sort) 是 AP 考试里的递归排序: 把数组分成两半, 分别递归排序, 再把两个有序的一半合并 (merge)。它是 O(n log n)—— 在大数组上远快于 $O(n^2)$ 的排序。

```
import java.util.Arrays;

public class Main {
    public static void main(String[] args) {
        int[] a = {5, 2, 9, 1, 7, 3};
        mergeSort(a, 0, a.length - 1);
        System.out.println(Arrays.toString(a));   // [1, 2, 3, 5, 7, 9]
    }

    static void mergeSort(int[] a, int lo, int hi) {
        if (lo >= hi) return;                 // 基准情形: 只剩一个元素
        int mid = (lo + hi) / 2;
        mergeSort(a, lo, mid);                // 排序左半
        mergeSort(a, mid + 1, hi);            // 排序右半
        merge(a, lo, mid, hi);                // 合并两半
    }

    static void merge(int[] a, int lo, int mid, int hi) {
        int[] tmp = new int[hi - lo + 1];
        int i = lo, j = mid + 1, k = 0;
        while (i <= mid && j <= hi) {
            if (a[i] <= a[j]) tmp[k++] = a[i++];
            else              tmp[k++] = a[j++];
        }
        while (i <= mid) tmp[k++] = a[i++];
        while (j <= hi)  tmp[k++] = a[j++];
        for (k = 0; k < tmp.length; k++) a[lo + k] = tmp[k];
    }
}
```

常见错误

- 递归必须有基准情形, 否则会抛出 StackOverflowError。
- 每次递归调用都要更**靠近**基准情形。
- 手动追踪一个小例子, 检查递归返回的值对不对。
- 归并排序里真正干活的是合并那一步; 递归只负责拆分数组。

11 查找与排序

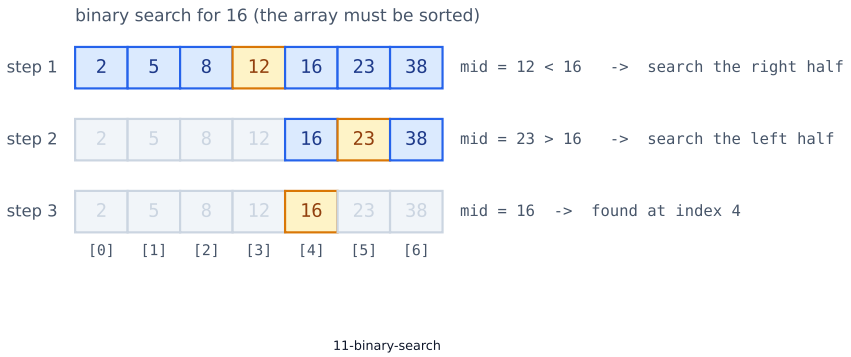
11.1 线性查找与二分查找

线性查找 (linear search) 逐个检查每个元素 —— 对任何数组都适用。二分查找 (binary search) 快得多, 但需要一个**已排序** (sorted) 的数组: 它查看中间值, 然后每一步丢掉一半。两者都返回索引, 找不到则返回 `-1`。

```
public class Main {
    public static void main(String[] args) {
        int[] a = {2, 5, 8, 12, 16, 23};      // 已排序, 所以二分查找可用
        System.out.println(linear(a, 12));    // 3
        System.out.println(binary(a, 12));    // 3
        System.out.println(binary(a, 9));     // -1 (未找到)
    }

    static int linear(int[] a, int target) {
        for (int i = 0; i < a.length; i++)
            if (a[i] == target) return i;
        return -1;
    }

    static int binary(int[] a, int target) {
        int lo = 0, hi = a.length - 1;
        while (lo <= hi) {
            int mid = (lo + hi) / 2;
            if (a[mid] == target) return mid;
            else if (a[mid] < target) lo = mid + 1;
            else hi = mid - 1;
        }
        return -1;
    }
}
```



11-binary-search

二分查找每一步把范围减半，已排序数组的查找是 $O(\log n)$

11.2 选择排序与插入排序

选择排序 (selection sort) 反复找出剩余值中最小的那个，并把它交换到前面。插入排序 (insertion sort) 取出每个值，把它滑回到已排序部分中属于它的位置。

```java
import java.util.Arrays;

public class Main {
    public static void main(String[] args) {
        int[] a = {5, 2, 9, 1, 7};
        for (int i = 0; i < a.length - 1; i++) {
            int min = i;
            for (int j = i + 1; j < a.length; j++)
                if (a[j] < a[min]) min = j;
            int t = a[min]; a[min] = a[i]; a[i] = t;   // 交换到位
        }
        System.out.println(Arrays.toString(a));   // [1, 2, 5, 7, 9]
    }
}
```



```java
import java.util.Arrays;

public class Main {
    public static void main(String[] args) {
        int[] a = {5, 2, 9, 1, 7};
        for (int i = 1; i < a.length; i++) {
            int key = a[i], j = i - 1;
            while (j >= 0 && a[j] > key) {    // 把较大的值向右移
                a[j + 1] = a[j];
                j--;
            }
            a[j + 1] = key;                   // 把 key 放进空位
        }
        System.out.println(Arrays.toString(a));   // [1, 2, 5, 7, 9]
    }
}
```

各算法的速度对比:

算法	时间
线性查找	O(n)
二分查找	O(log n)，仅限已排序数组
选择排序 / 插入排序	$O(n^2)$
归并排序 (见主题 10)	O(n log n)

常见错误

- 二分查找只对**已排序**的数组有效。
- 线性查找是 O(n)；二分查找是 O(log n)，但要先排好序。
- 选择排序和插入排序都是 $O(n^2)$—— 好懂，但大数据上很慢。

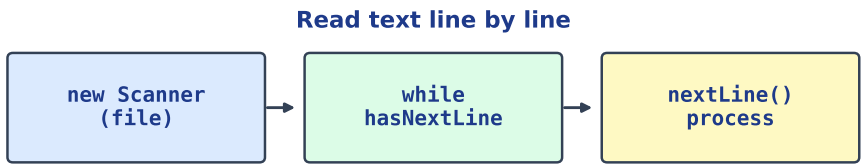
12 文件与 FRQ

12.1 用 Scanner 读写文本文件

`Scanner` 一次读取一行文本。读取真实文件时写 `new Scanner(new File("scores.txt"))`; 这里我们包装一个 `String`, 这样示例在任何地方都能运行。用 `.split(" ")` 把一行拆分 (split) 成几部分, 用 `Integer.parseInt(...)` 从文本中解析 (parse) 出数字。

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        // 真实文件:  Scanner in = new Scanner(new File("scores.txt"));
        String data = "Alice 80\nBob 95\nCara 72";
        Scanner in = new Scanner(data);
        int total = 0, count = 0;
        while (in.hasNextLine()) {
            String line = in.nextLine();
            String[] parts = line.split(" ");      // 按空格拆分
            total += Integer.parseInt(parts[1]);
            count++;
        }
        System.out.println("average = " + (total / count));   // average
↪   = 82
    }
}
```



Exam tip: hasNextLine guards the loop; split + parseInt turn a line into data

Scanner reads lines while hasNextLine is true

12.2 AP FRQ 题型

AP CS A 考试有四道自由作答 (free-response) 题, 每道都有固定的形式:

- **第 1 题 —— 方法与控制:** 按给定规范写方法; 循环、`if`、`String`/`Math`。
- **第 2 题 —— 类的设计:** 根据描述写一个完整的类 (字段、构造方法、方法)。
- **第 3 题 —— 数组 / ArrayList:** 处理一维数组或 `ArrayList`(查找、计数、构建新列表)。
- **第 4 题 —— 二维数组:** 按行和列遍历一个网格。

技巧始终相同: 读懂规范, 严格按描述写出方法, 返回正确的类型。

```
public class Main {
    public static void main(String[] args) {
        // 第 1 题风格: 按规范实现一个方法, 然后接受测试。
        System.out.println(countEven(new int[]{4, 7, 10, 3, 6}));   // 3
    }

    /** 返回 arr 中有多少个偶数。 */
    public static int countEven(int[] arr) {
        int count = 0;
        for (int x : arr)
            if (x % 2 == 0) count++;
        return count;
    }
}
```

常见错误

- `nextInt()` 会把换行符留下, 导致紧接的 `nextLine()` 读到空行 —— 先把它读掉。
- 读之前先检查 `hasNext()`, 避免读过文件末尾。
- 在 FRQ 里仔细看方法头: 返回类型和参数要完全对上。