# Chapter 4 类、对象、方法

## 4.1构造方法

如果在定义类是时没有为类定义任何构造方法，则编译器自动为类添加一个默认构造方法（default constructor）。默认构造方法是无参数构造方法（no-args constructor），方法体为空。

## 4.2重载

是指在同一个类中定义多个方法，它们具有**相同的方法名**，但**参数列表不同**。

1. **参数个数相同，参数类型相同**

class Calculator {

// 重载方法 1

public int add(int a, int b) {

return a + b;

}

// 重载方法 2：参数类型不同

public double add(double a, double b) {

return a + b;

}}

1. **参数个数不同**

class Calculator {

// 重载方法 1

public int add(int a, int b) {

return a + b;

}

// 重载方法 2：参数个数不同

public int add(int a, int b, int c) {

return a + b + c;

}}

1. **参数顺序不同 （前提参数类型不同）**

class Printer {

// 重载方法 1

public void print(int a, String b) {

System.out.println("int: " + a + ", String: " + b);

}

// 重载方法 2：参数顺序不同

public void print(String a, int b) {

System.out.println("String: " + a + ", int: " + b);

}}

**注意：返回类型不同（不能单独作为重载的条件）**重载的核心是参数列表不同，而不是返回类型。

class Calculator {

public int add(int a, int b) {

return a + b;

}

// 编译错误：仅返回类型不同，不能构成重载

public double add(int a, int b) {

return a + b;

}}

注意：访问修饰符不同（不能单独作为重载的条件）

class Calculator {

public int add(int a, int b) {

return a + b;

}

// 编译错误：仅访问修饰符不同，不能构成重载

private int add(int a, int b) {

return a + b;

}}

注意：参数名称不同不能实现重载

class Calculator {

// 方法 1

public int add(int a, int b) {

return a + b;

}

// 方法 2：参数名称不同，但类型和数量相同

public int add(int x, int y) { // 编译错误

return x + y;

}

}

## 4.3重写

定义：指的是子类重新定义父类中已有的方法

条件：方法名相同、参数列表相同、返回类型相同、访问修饰符不能更严格、异常不能更宽泛

注意：不能重写父类中的static静态方法

不能重写final方法（被final修饰的方法不能被子类重写）

子类不能覆盖父类的private方法

### ****考试：重写与重载的区别****

| **特性** | **重写（Override）** | **重载（Overload）** |
| --- | --- | --- |
| **定义位置** | 父子类之间 | 同一个类或父子类中 |
| **方法名** | 必须相同 | 必须相同 |
| **参数列表** | 必须相同 | 必须不同（类型、个数或顺序） |
| **返回类型** | 必须相同（或子类） | 可以不同 |
| **访问修饰符** | 不能比父类方法更严格 | 可以不同 |
| **异常** | 不能抛出比父类方法更宽泛的异常 | 可以抛出不同的异常 |
| **多态性** | 运行时多态 | 编译时多态 |
| **目的** | 子类重新定义父类的行为 | 提供同一功能的多种实现方式 |

## 4.4 this关键字

this表示对象本身

作用1：解决局部变量与成员变量同名的问题

作用2：解决方法参数与成员变量同名的问题

作用3：调用该类的另一个构造方法

Employee (String name, int age, double salary)

public Employee(){

this("张明月", 28, 5000);

}

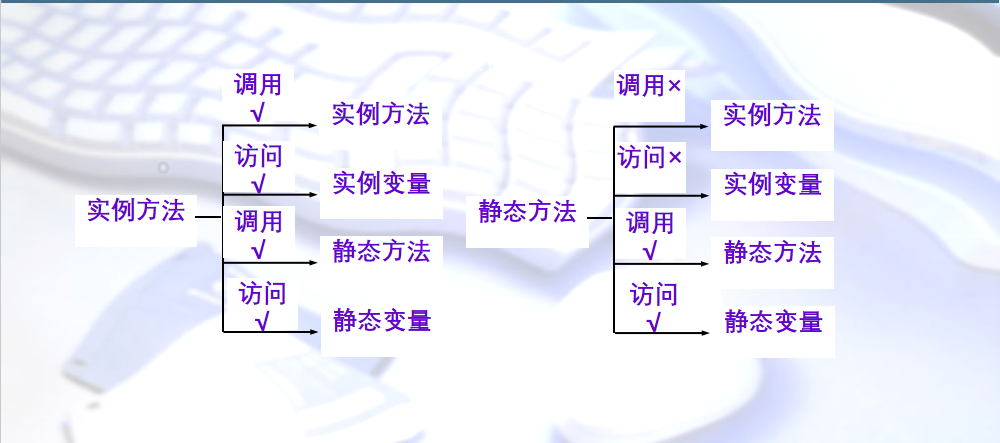
## 4.5 静态变量和静态方法

静态变量：如果成员变量用static修饰，则该变量称为静态变量或类变量

所有实例共享一块内存，通常使用类名访问

静态方法：如果成员方法用static修饰，则该方法称为静态方法或类方法

只能访问静态变量，通常使用类名访问



单例模式

SingleObject.java

public class SingleObject {

//创建 SingleObject 的一个对象

private static SingleObject instance = new SingleObject();

//让构造函数为 private，这样该类就不会被实例化

private SingleObject(){}

//获取唯一可用的对象

public static SingleObject getInstance(){

return instance;

}

public void showMessage(){

System.out.println("Hello World!");

}

}

SingletonPatternDemo.java

public class SingletonPatternDemo {

public static void main(String[] args) {

//不合法的构造函数

//编译时错误：构造函数 SingleObject() 是不可见的

//SingleObject object = new SingleObject();

//获取唯一可用的对象

SingleObject object = SingleObject.getInstance();

//显示消息

object.showMessage();

}

}

# Chapter 5 数组

## 5.1创建和使用数组

#### 创建

ArrayDemo.java

Double[] marks = new double[5];

marks[0] = 79;

marks[1] = 84.5;

marks[2] = 63;

marks[3] = 90;

marks[4] = 98;

System.out.println(marks[2]);

System.out.println(marks.length);

//输出每个元素的值

For( int i = 0 ; i<marks.length: i++){

System.out.println( marks[i] + “ “);

}

#### 增强的for 循环

for(元素类型 变量名：数组名) {

// 循环体

}

## 5.2 数组的应用

#### 复制 arraycopy

public static void arraycopy(Object src, int srcPos,

Object dest, int destPos,int length)

int[] source = {10,30,20,40};

int[] target = new int[source.length];

System.arraycopy(source, 0, target, 0, 4);

## 5.3 java.util.Arrays类

1、 Sort（）方法

double[] numbers = {5.0, 4.4, 1.9, 2.9, 3.8, 3.5};

java.util.Arrays.sort(numbers);

2、BinarySearch（）方法

var a = new int[]{1,5,7,3};

Arrays.sort(a); // 返回a的元素是：1 3 5 7

var i = Arrays.binarySearch(a,4);

System.out.println(i); // 输出：-3

在已排序的数组a中查找4，返回的结果-3（-2-1），它的含义要把4插入到数组a中，它的下标是应该2。

3、equals（）方法

static boolean equals(boolean[] a, boolean[] b)比较布尔型数组a与b是否相等。

4、fill（）方法

static void fill (int[] a, int val)：填充数组a中的每个元素。

# Chapter 6 字符串类

## 6.1 String类及基本操作

#### 6.1.1 创建String类对象

特殊方法

String str = “ java is cool”

一般方法

String str = new String(“java is cool”)

#### 6.1.2 字符串的基本操作

var s = "Java is cool";

System.out.println(s.length()); // 输出：12

System.out.println(s.substring(5,7)); // 输出：is

System.out.println(s.substring(8)); // 输出：cool

System.out.println(s.toUpperCase()); // 输出：JAVA IS COOL

System.out.println(s.toLowerCase()); // 输出：java is cool

#### 6.1.3 string对象的不变性

在Java程序中一旦创建一个String对象，就不能对其内容进行改变

var s = new String("Hello,world");

s.replace('o','A'); // s的值并没有改变

s = s.substring(0,6).concat("Java");

s.toUpperCase(); // s的值并没有改变

System.out.println(s); // 输出：Hello,Java

#### 6.1.4 字符串的比较

特别注意，不能使用“==”号比较字符串内容是否相等，请看下面代码。

var s1 = new String("Hello");

var s2 = new String("Hello");

System.out.println(s1 == s2); // 输出：false

这是因为在使用“==”比较引用类型的数据（对象）时，比较的是引用（地址）是否相等。

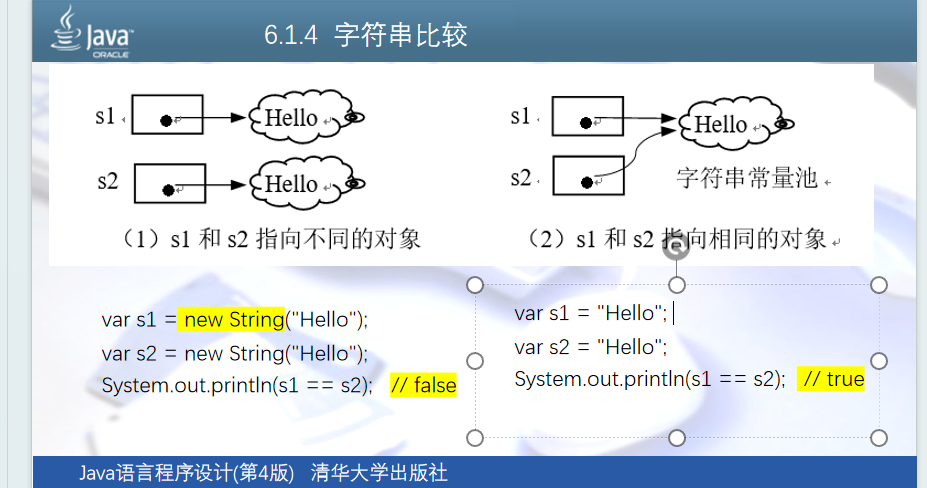
再请看下面一段代码：

var s1 = "Hello"; // 不是用构造方法创建的对象

var s2 = "Hello";

System.out.println(s1 == s2); // 输出：true

输出结果true。这两段代码的不同之处在于创建s1、s2对象的代码不同。这里的s1、s2是用字符串常量创建的两个对象。字符串常量存储和对象存储不同，字符串常量是存储在常量池中，对内容相同的字符串常量在常量池中只有一个副本，因此s1和s2是指向同一个对象



#### 6.1.5 字符串的查找和匹配

这些方法的返回值都是整数，查找成功返回字符或子串在原字符串中出现的下标位置（下标从0开始），查找不成功则返回-1。

public int indexOf(int ch, int fromIndex )

public int indexOf(String str)

public int indexOf(String str, int fromIndex )

#### 6.1.6 字符串的拆分和组合

var ss= "one little,two little,three little.";

String [] str = ss.split("[ ,.]");

System.out.println(str.length); // 输出：6

for(var s : str){

System.out.println(s);

}

在split()中指定的正则表达式“[ ,.]”的含义是使用空格、逗号或句点为分隔符拆分字符串。

String类的静态join()方法，实现将一个字符串数组按指定的分割符组合成一个字符串，它的功能与split()方法相反。下面代码演示了join()方法的使用。

var joined = String.join("\\","C:","javastudy","com");

System.out.println(joined); // 输出：C:\javastudy\com

String [] seasons = {"春", "夏", "秋", "冬"};

var s = String.join("-" ,seasons);

System.out.println(s); // 输出：春-夏-秋-冬

## 6.2 文本块



## 6.3 格式化数据

|  |  |
| --- | --- |
| 格式符 | 含  义 |
| %d | 结果被格式化成十进制整数 |
| %f | 结果被格式化成十进制浮点数 |
| %e | 结果以科学计数法格式输出 |
| %s | 结果以字符串输出 |
| %b | 结果以布尔值（true或false）形式输出 |
| %c | 结果以Unicode字符输出 |
| %n | 换行格式符，它不与参数对应。与\n含义相同，但%n是跨平台的 |

%d 用来输出十进制整数，可以有长度修饰。不可以应用到浮点数上，否则会报错.

System.out.printf("year = |%6d|%n", 2017); // 输出：year = | 2017|

%f 用来以小数方式输出 可以指定格式宽度和小数位，也可以仅指定小数位，不可以用到整数上，否则会报错

System.out.printf("|%8.3f|",2017.1234); // 输出：|2017.123|

%e 用来以科学计数法的格式输出浮点数。可以指定格式宽度和小数位，也可以仅指定小数位。指定的数据必须是浮点型值。int型值不能匹配%f和%e

System.out.printf("|%10.2e|%n",123.567); // 输出：| 1.24e+02|

“%c”用来以字符方式输出。它可以应用的数据类型有：char、Character、byte、Byte、short、Short，这些数据类型都能够转换成Unicode字符。

byte b = 65;

System.out.printf ("b = %c%n", b); // 输出：b = A

“%b”格式符可以用在任何类型的数据上。对于“%b”格式符号，如果参数值为null，结果输出false；如果参数是boolean或Boolean类型的数据，结果是调用String.valueOf()方法的结果，否则结果是true。

byte b = 0;

String s = null;

System.out.printf ("b1 = %b%n", b); // 输出：b1 = true

System.out.printf ("b2 = %b%n", true); // 输出：b2 = true

System.out.printf ("b3 = %b%n", s); // 输出：b3 = false

“%s”格式符也可以用在任何类型的数据上。对于“%s”格式符号，如果参数值为null，结果输出null；如果参数实现了Formatter接口，结果是调用args.formatTo()的结果，否则，结果是调用args.toString()的结果。

如果将上面代码中“%b”改为“%s”，输出结果如下所示。

b1 = 0

b2 = true

b3 = null

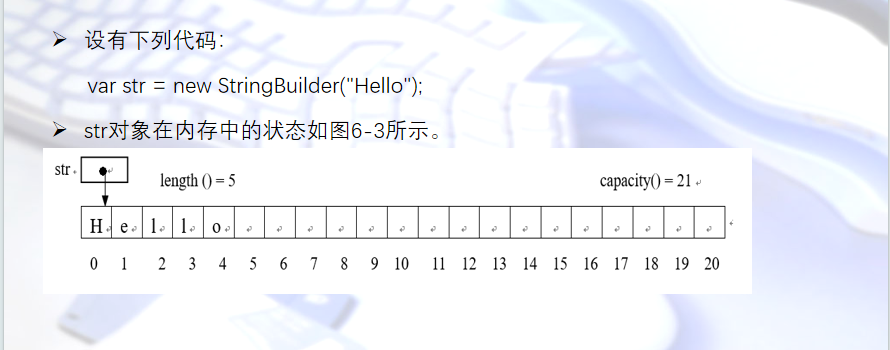
## 6.4 StringBuilder 类

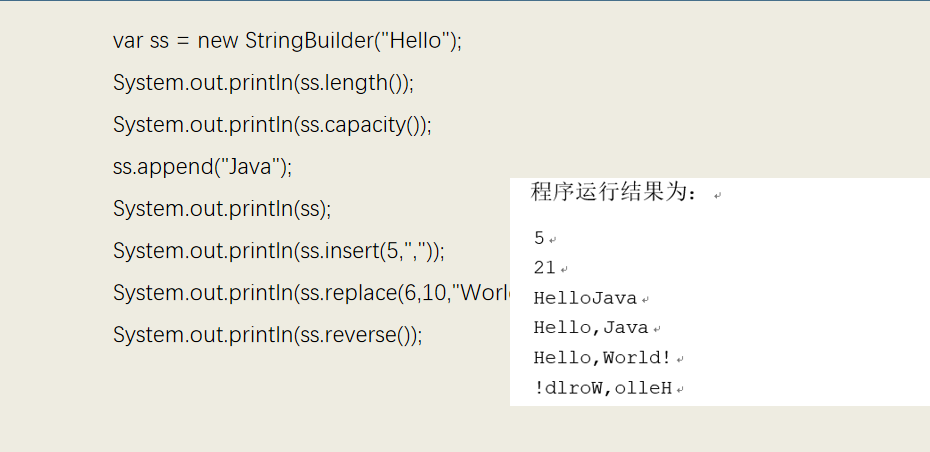
StringBuilder类表示可变字符串，即这个类的对象内容是可以修改的。

public StringBuilder()：创建一个没有字符的字符串缓冲区，初始容量为16个字符。此时length()方法的值为0，而capacity()方法的值为16。

public StringBuilder(int capacity)：创建一个没有字符的字符串缓冲区，capacity为指定的初始容量。

public StringBuilder(String str)：利用一个已存在的字符串对象str创建一个字符串缓冲区对象，另外再分配16个字符的缓冲区。





可变字符串 ss会改变，如果是String类型就不会改变

# Chapter 7 面向对象特征

三大特征：封装、多态、继承

## 封装

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 修饰符 | 同一个类 | 同一个包的类 | 不同包的子类 | 任何类 |
| private | √ | − | − | − |
| 缺省 | √ | √ | − | − |
| protected | √ | √ | √ | − |
| public | √ | √ | √ | √ |

## 继承

说明：1.子类继承父类中非private的成员变量和成员方法。

2 省略extends，定义的类继承Object类。

3 Java仅支持单继承。

#### 方法覆盖（重写）：

在子类中可以定义与父类中的名字、参数列表、返回值类型都相同的方法

说明：1.子类不能覆盖父类的private方法，也不能继承,如果在子类中定义了一个方法在父类中是private的，则这两个方法完全无关。

2.父类中static方法可以被继承，但不能被覆盖，如果子类中定义了与父类中的static方法完全一样的方法，那么父类中的方法被隐藏。父类中被隐藏的static方法仍然可以使用“类名.方法名()”形式调用。

#### @Override

为了避免在覆盖方法时写错方法头，可以使用@Override注解语 法:@Override注解表示其后的方法必须是覆盖父类的一个方法。如果具有该注解的方法没有覆盖父类的方法，编译器将报告一个错误。

#### Super 关键字 用来引用父类对象

1. 访问父类中被隐藏的成员变量

Super.variableName

1. 在子类中调用父类中被覆盖的方法

Super.methodName([paramlist])

1. 在子类中调用父类的构造方法

Super([paramlist])

#### 调用父类的构造方法

子类不能继承父类的构造方法

在创建子类对象时，必须先构造该类的所有父类对象。故在编写子类的构造方法时，必须保证它能调用父类的构造方法。（隐式或显式）

1.super显示调用

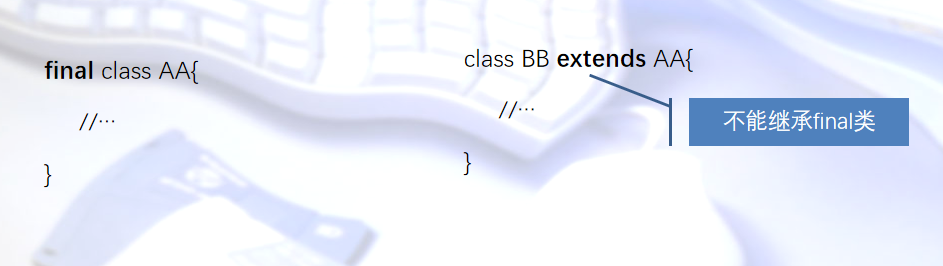
Super([paramlist])

2.调用父类的无参构造

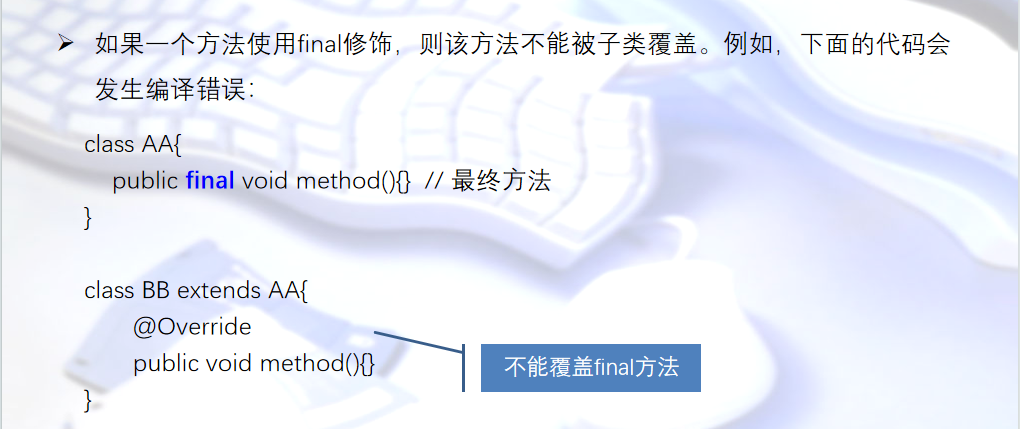
Super() 如果不显式调用，编译器会自动添加 （如果父类只有有参构造，则编译器不会给父类生成默认无参构造，子类如果没有显式调用父类的有参构造，此时父类没有无参构造，编译器会报错）

#### Final关键字

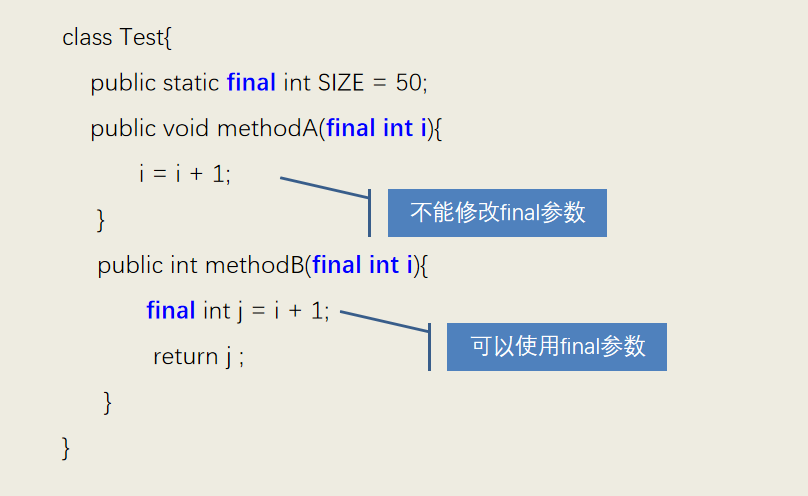
1.Final修饰类 ，则该类为最终类，不能被继承



2.Final修饰方法，则该方法不能被子类覆盖



3.Final修饰变量，则改变量为常值变量，一旦赋值便不能改变



#### 抽象类

定义：抽象类是一种**不能被实例化**的类，通常用作其他类的基类（父类）。它的主要作用是**定义子类的通用行为**，同时可以包含**具体方法**和**抽象方法**。 在类名前加 abstract关键字

特点：1.不能被实例化

2.可以包含抽象方法和具体方法（可以没有抽象方法）

抽象方法：只有声明，没有实现

具体方法：有完整的实现

3.可以包含成员变量、构造方法、静态方法等

4.子类必须实现抽象方法：如果一个类继承了抽象类，它必须实现抽象类中所有的抽象方法（除非子类也是抽象类）

## 抽象方法

定义：抽象方法时一种只有声明，没有实现的方法，必须在抽象类中定义。由具体子类实现。

特点：1. 只有方法签名，没有方法体

1. 必须定义在抽象类中
2. 子类必须实现抽象方法（除非子类也是抽象类）

#### Instanceof运算符

定义：用于检查一个对象是否是指定类或其子类的实例

例：



Fruit instanceof Apple 的结果是 true

fruit 是 Fruit 类型的引用，但实际指向的是 Apple 类的实例。因此，表达式 fruit instanceof Apple 的结果是 true。

# Chapter 9 接口 内部类

#### 接口

定义：使用Interface关键字定义，一种特殊的引用类型，用于定义一组方法的规范，而不提供具体的实现。

interface 接口名 {

// 常量（默认是 public static final）

数据类型 常量名 = 值;

// 抽象方法（默认是 public abstract）

返回类型 方法名(参数列表);

// 默认方法（Java 8 引入）

default 返回类型 方法名(参数列表) {

// 方法体

}

// 静态方法（Java 8 引入）

static 返回类型 方法名(参数列表) {

// 方法体

}

}

接口的实现：类使用implements 关键字实现接口，并必须实现接口中的所有抽象方法（除非是抽象类），一个类可以实现多个接口

接口的继承：子接口继承父接口中的常量、抽象方法、默认方法。且，接口可以多继承

接口类型的使用：接口是一种引用类型，任何实现该接口的实例都可以存储在该接口类型的变量中

interface Vehicle {

void start();

}

class Car implements Vehicle {

@Override

public void start() {

System.out.println("Car is starting...");

}}

public class Main {

public static void main(String[] args) {

Vehicle myVehicle = new Car(); // 向上自动类型转换

myVehicle.start(); // 输出: Car is starting...

}}

## 内部类

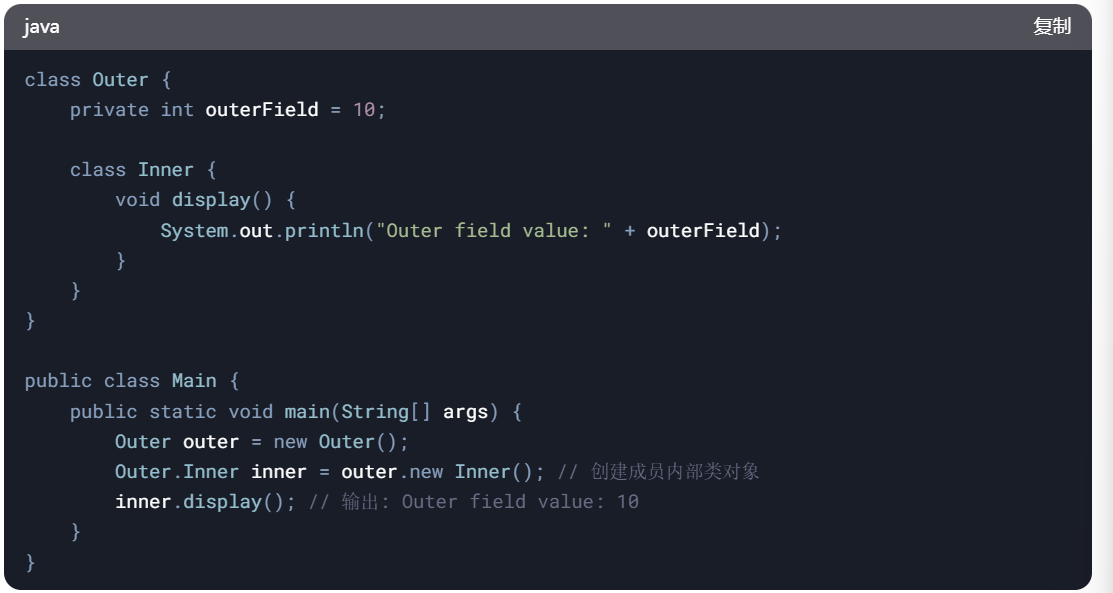
#### 成员内部类

定义： 是定义在一个类的内部，但没有使用static修饰的内部类。

特点： 可以访问外部类的所有成员（包括私有成员）

不能定义静态成员（除非是静态常量）

必须通过外部类的实例来创建

示例：

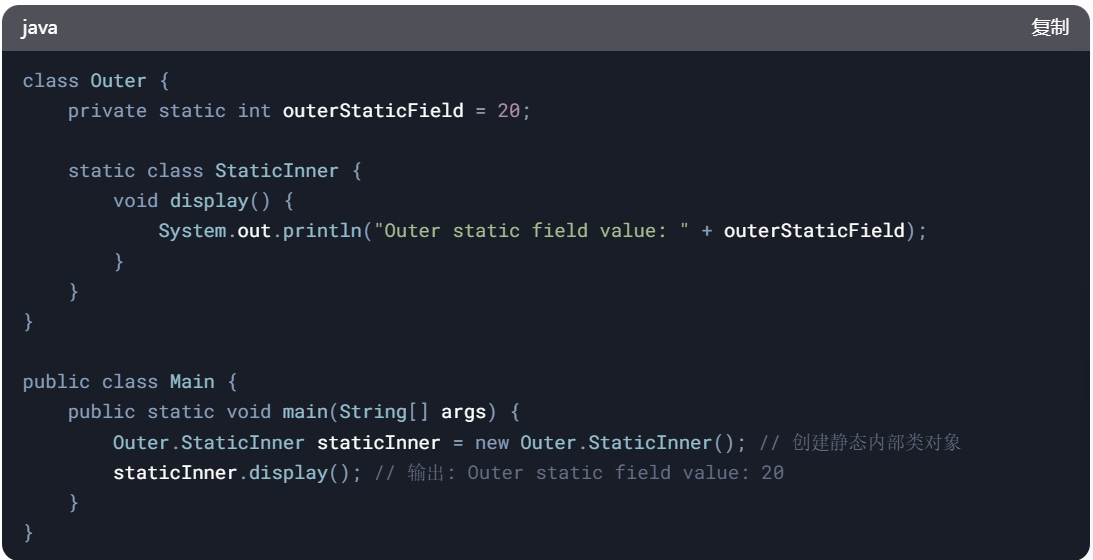
#### 静态内部类

定义： 定义在另一个类的内部，并使用static修饰的内部类

特点： 不能直接访问外部类的非静态成员

可以直接访问外部类的静态成员

不需要外部类的实例即可创建



#### 匿名内部类

定义：没有显式名称的内部类。

特点： 没有显式的类名

必须继承一个类或实现一个接口

通常用于简化代码



#### 局部内部类

定义：定义在方法或代码内部的类

特点：定义在方法或代码块内部

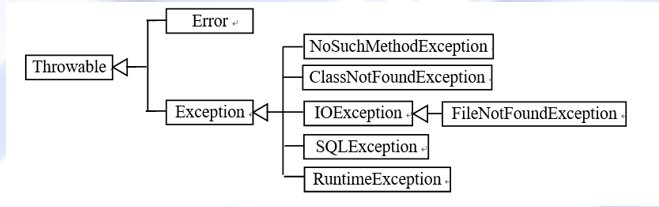
只能访问所在方法或代码块中的final变量

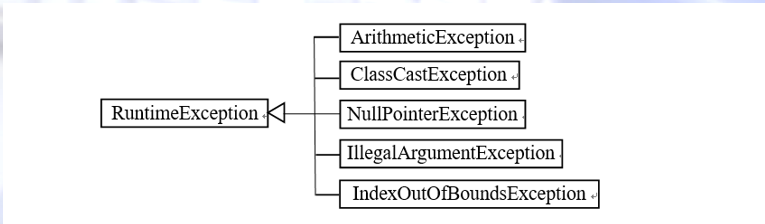
不能使用访问修饰符（如 public，private）



# Chapter 10 异常处理

## 异常类





#### 非检查异常

1. NullPointerException:空指针异常

String name = null;

System.out.print(name.length());

1. ArithmeticException :算数异常

Int a = 5;

Int b = a/0;

1. ClassCastException:对象转换异常

Object o = new Object();

String s = (String)o;

1. ArrayIndexOutOfBoundsException:数组下标越界异常

Int a[] = new int[5];

A[5] = 10;

1. NumberFormatException:数字格式化错误异常

Double d = double.parseDouble(“5madf”);

## 捕获异常

Try - catch - finally

throws语句声明抛出异常（则必须使用try-catch处理异常）

throw语句抛出异常

Try-with-resourse管理资源



要在一个catch语句中处理多个异常，需要使用“或”运算符（|）分隔多个异常



## 自定义异常

public class CustomException extends Exception {

   public CustomException(){}

   public CustomException(String message) {

        super(message);

   }

}



## 断言

Assert 关键字 ：用于在程序中检查某些条件是否为真。如果条件为假，assert会抛出AssertException异常

Assert 语法： 1. assert expression 布尔表达式为false则抛出异常

1. Assert expression : message 布尔表达式为false则抛 出异常，并将message作为错误信息



# Chapter 11 记录、注解、枚举类型

## 记录

Record 它用于简化不可变数据载体类的定义。

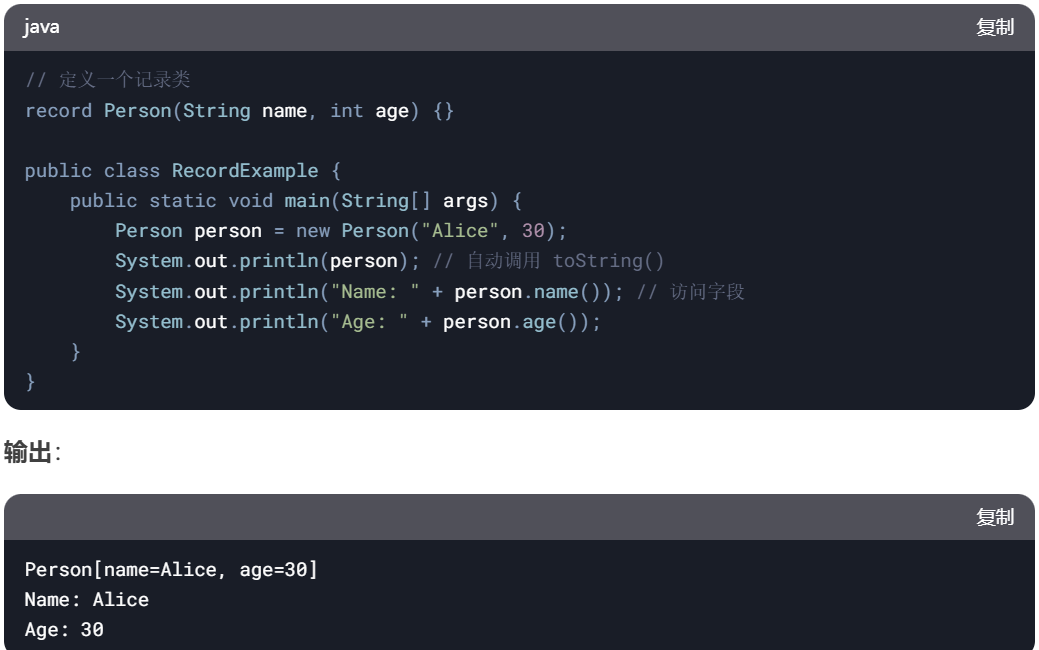
特点： 自动生成equals()、hashCode()、toString()方法

自动生成构造函数

字段是 final的，不可修改

语法： record 类名（参数列表）{}

示例：



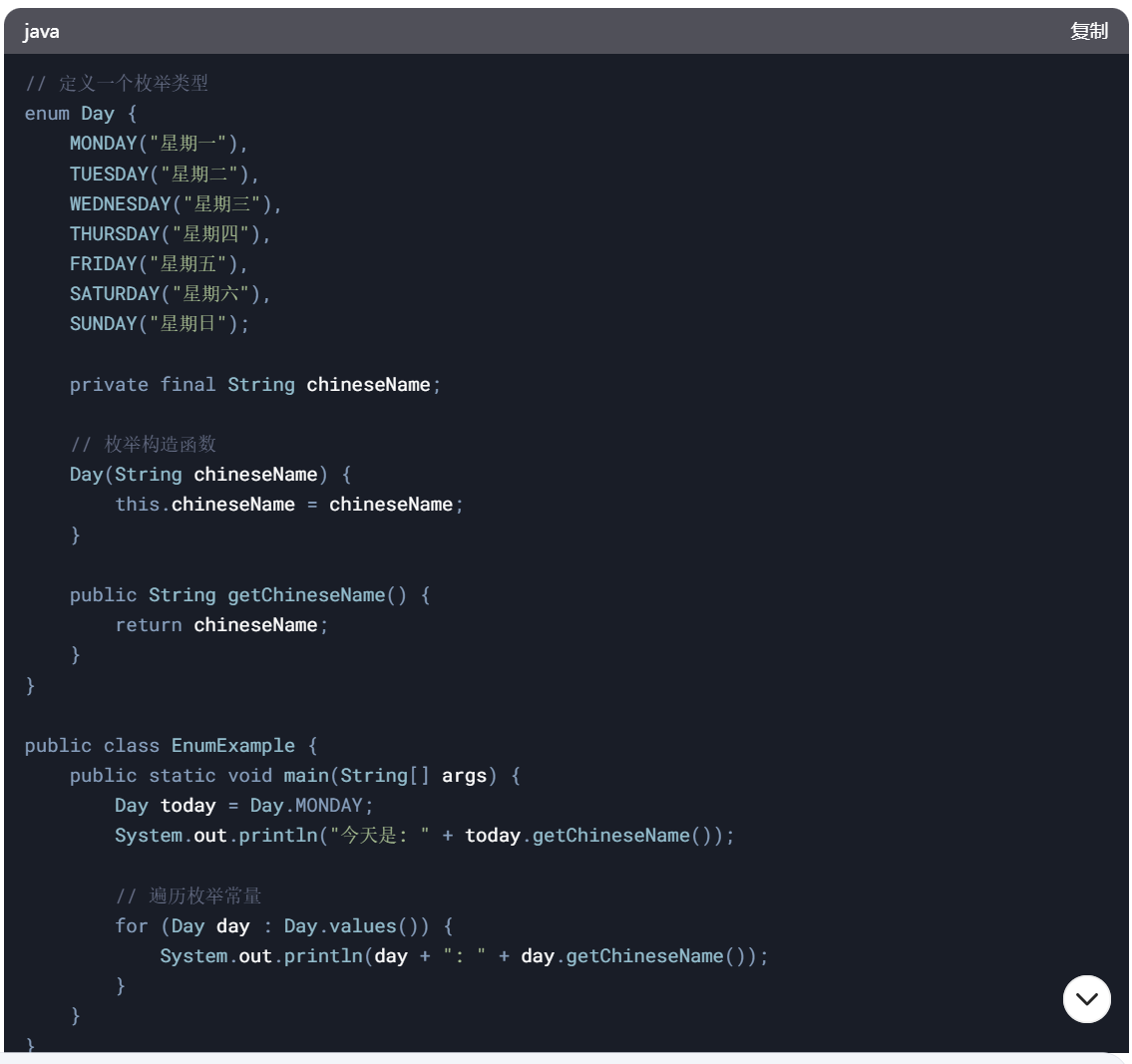
## 注释

@Override:表示重写父类方法

@Deprecated:表示方法或类已过时

@SuppressWarnings：抑制编译器警告

## 枚举

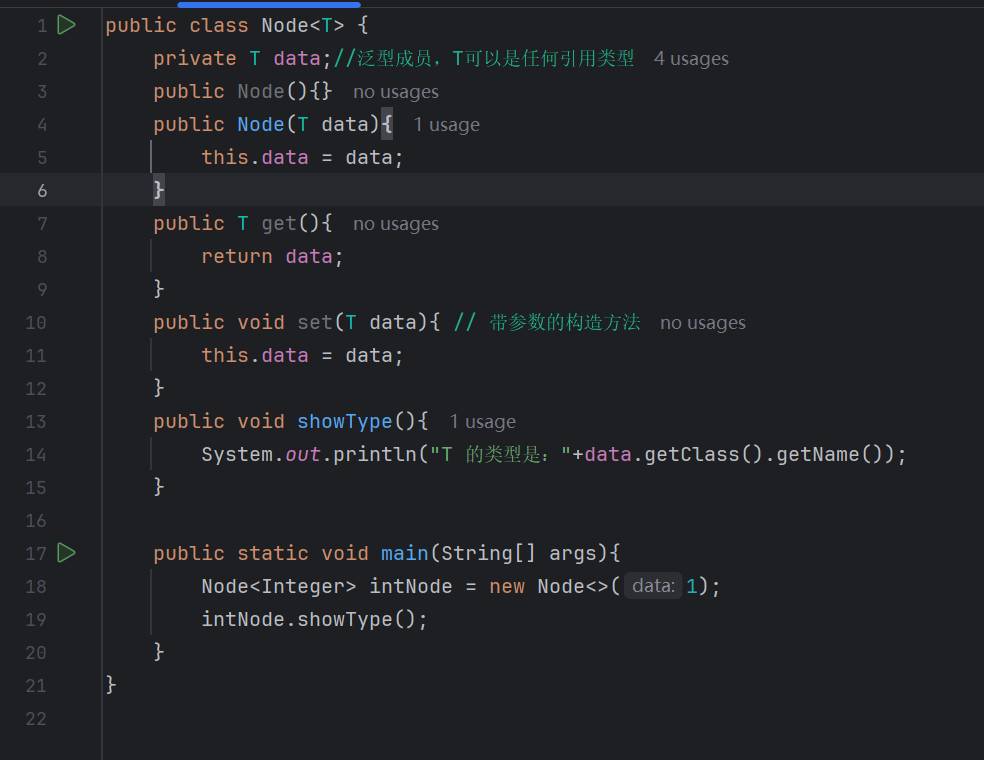


# Chapter 12 泛型与集合

## 12.1.泛型

### 12.1.1 泛型类

#### Node.java



### 考试第四大题的第一题 泛型接口+工厂模式

### 12.1.2泛型接口

即接口也可以带参数

public interface Readble<T> {

void read(T t);

}

实现接口的三种方法

1. 指定类中的泛型类型

Public class IntelligentRobot implements Readable<Book>{

Public void read(Book b){}

}

1. 创建 泛型类

Public class IntelligentRobot <U> implements Readable<U>{

Public void read(U t){}

}

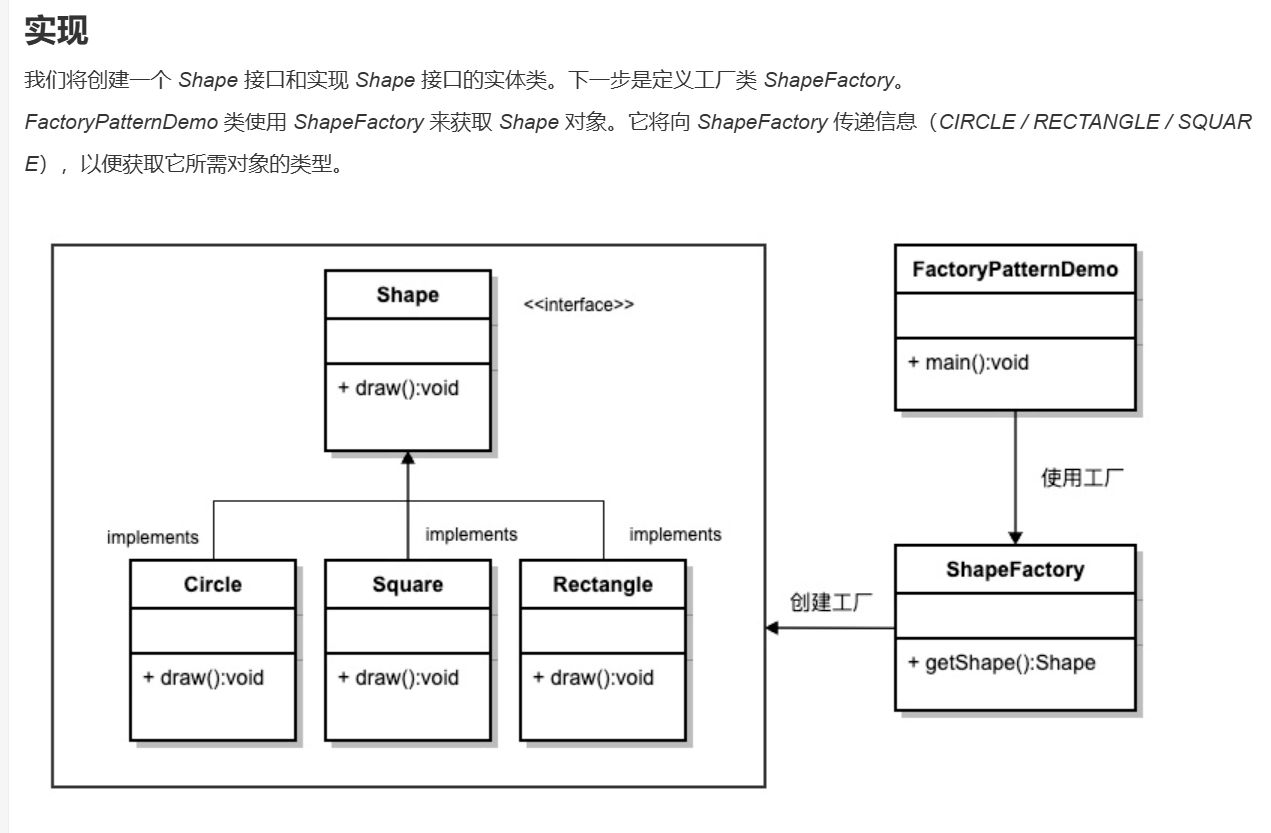
1. 不使用泛型

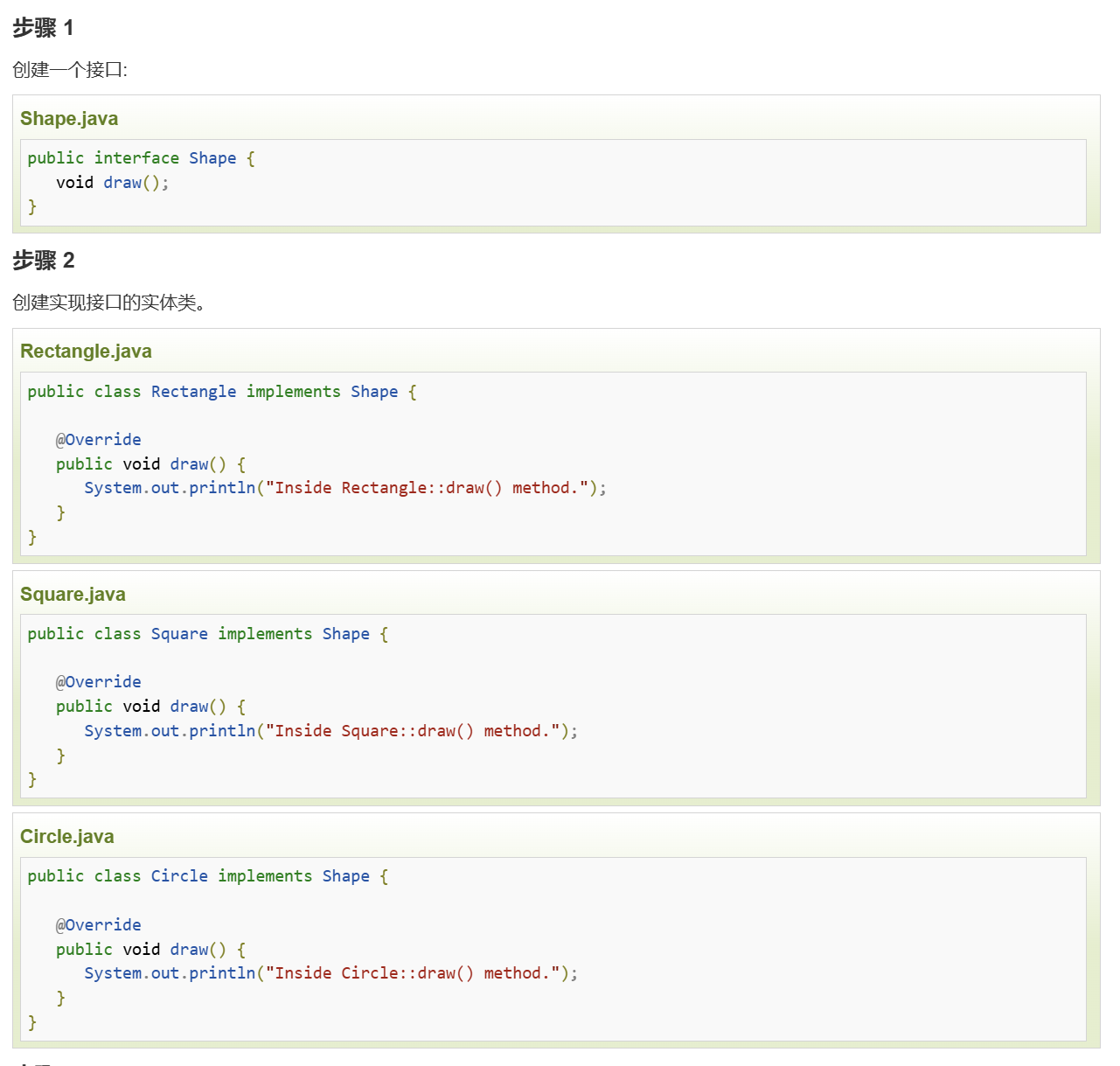
public class IntelligentRobot implements Readable {

public void read(Object o) { }

}

#### 工厂模式









**注意**：因为接口中定义了draw方法，所以可以通过 shape 类型直接调用draw（）函数。

故 步骤三 工厂创建getShape（）函数 返回类型 是 Shape，步骤四 使用工厂创建具体对象时 也是使用Shape 类型

### 12.1.3 泛型方法

定义语法

修饰符 <类型参数> 返回类型 方法名(参数列表) {

// 方法体

}

例如

public static <T> void swap(T[] array,int i, int j){

T temp = array[i];

array[i] = array[j];

array[j] = temp;

}

必须在方法返回值前指定泛型

## 12.2集合

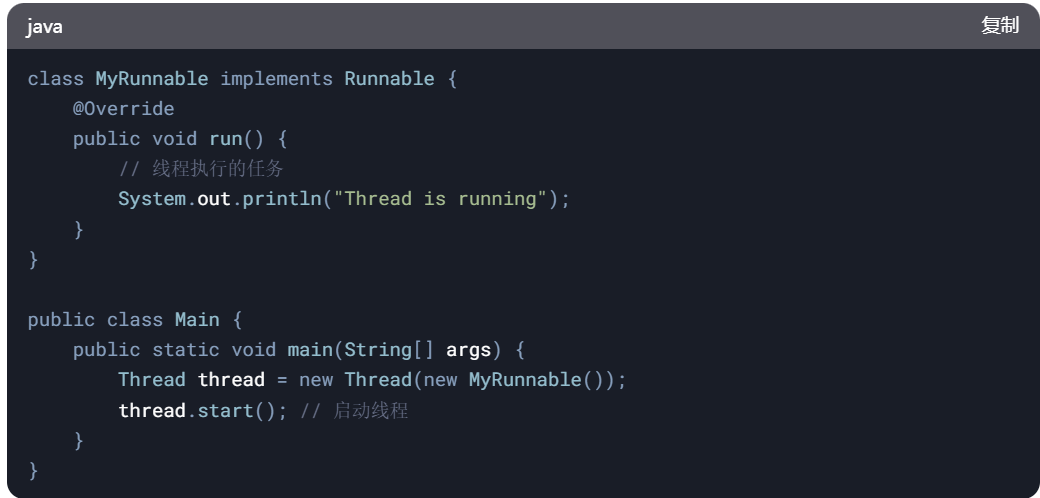
# Chapter 18 多线程

## 创建线程

**继承Thread类**

****

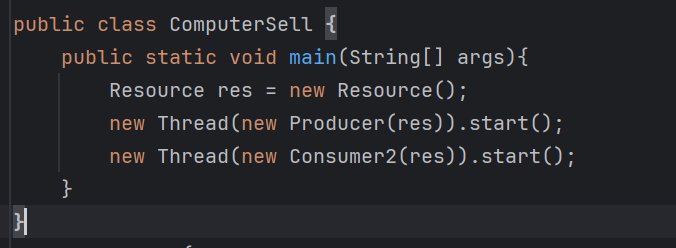
**实现Runnable接口**

****

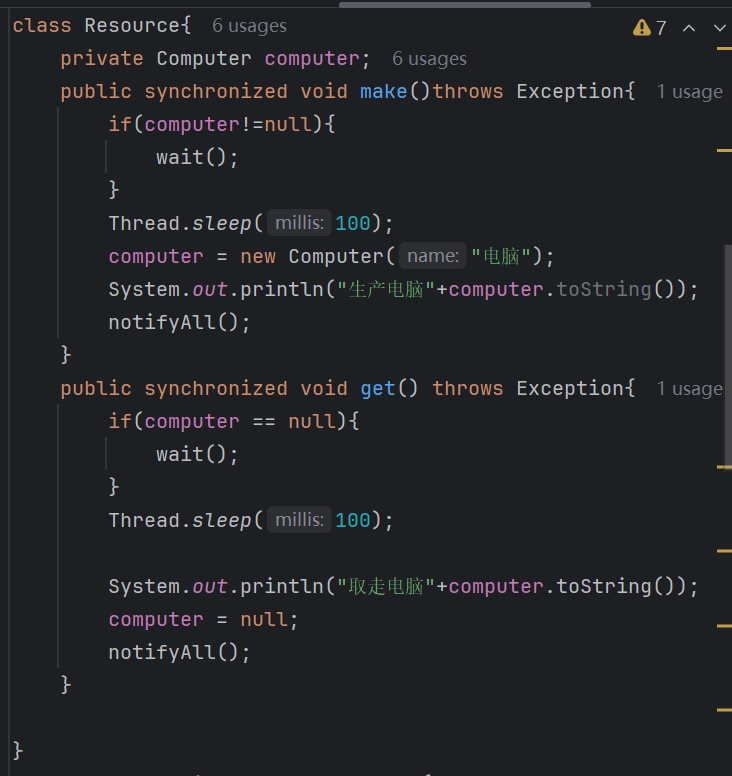
**Synchronized关键字**

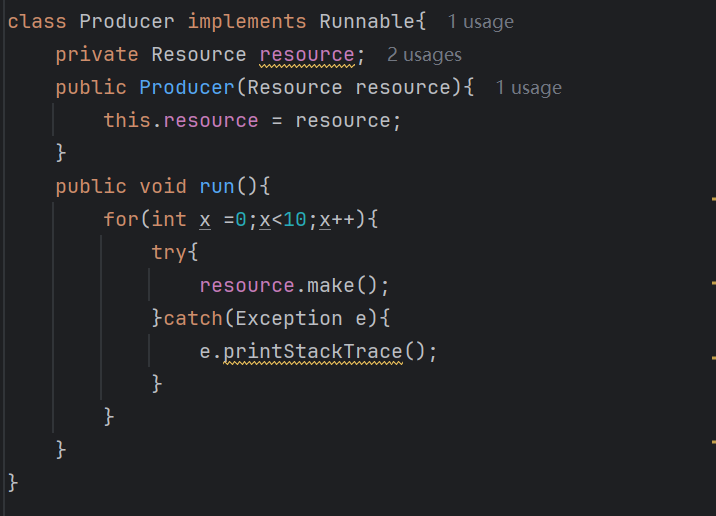
此关键字可以用来修饰方法或代码块，确保同一时间只有一个线程可以执行该代码。

示例：











示例：请说明什么是Java中多线程的同步问题和死锁问题，并编写一个多线程程序：实例化2个线程，一个线程打印字符A-T，一个线程打印数字1-60，2个线程同时执行，要求打印格式为A123 B456……T585960（请写出主要代码）。

多线程的同步问题：当多个线程同时访问和修改共享资源时，可能会导致数据不一致或程序行为异常。这是因为线程的执行顺序是不确定的 ，可能会导致竞态条件。

思锁问题：多个线程在执行过程中，因为争夺资源而造成的一种互相等待现象，导致这些线程都无法继续执行。

四个必要条件：

1. 互斥条件：资源一次只能被一个线程占用。
2. 占有并等待：线程持有至少一个资源，并等待获取其他被占用的资源。
3. 不可抢占：线程已有的资源不能被其他线程强行抢占，只能由线程自己释放。
4. 巡皇等待：存在一个线程等待的循环链，每个线程都在等待下一个线程所持有的资源。
5. 