复习：

1. 方法：需要重复使用某段逻辑的时候，就将这段逻辑提取成新的形式。

定义格式：修饰符 返回值类型 方法名(参数列表){方法体; return 返回值;} --- 形参，实参 --- 方法名+参数列表 -> 方法签名

public static int add(int i, int j){} --- add(int, int)

方法的重载：在同一个类中存在了方法名一致而参数列表不同（参数的个数/对应位置上的参数类型）的方法

方法的传值：基本类型传递的是实际值；对象传递的是地址，地址不变的前提下可以改变其中的属性或者元素。

方法的递归：在方法中调用了自己本身。当递归层次太深会出现StackOverflowError --- 栈溢出错误 --- 递归的思路往往是逆推的思维方式

2. 面向对象与面向过程的比较：都是思维方式。面向过程是注重流程中的每一个步骤，清楚每一个步骤中的细节（自己做）；面向对象是注重对象，只要拥有了对象就能使用对象身上的功能（找别人做）。面向对象是基于面向过程的。相对简单的事务使用面向过程，相对复杂的事务使用面向对象。--- 面向对象更适合于团队的开发；面向过程一般而言比面向对象更适合于效率开发。

3. 类与对象的关系：类是对象的概括/抽取，对象是类的具体实现

4. 对象的内存存储：对象是存储在堆内存中，在栈内存中会存储对象的地址引用。对象在存储到内存中之后其中的属性会自动赋予默认值。--- 对象在使用完成之后不一定立即移出堆内存而是在不定的某个时刻被回收。

5. 成员变量和局部变量：定义位置、作用范围、内存位置、生命周期

### 构造方法

特点：与类同名而没有返回值类型的方法

作用：创建对象

### this关键字

this关键字在类中调用本类里面的属性和方法，代表当前在活动的对象。

this语句 -> this(参数列表) - 表示在本类的构造方法中调用本类对应形式的其他的构造方法 - this语句必须放在构造方法的第一行

### 代码块

构造代码块：定义在类中用{}包起来的代码 --- 在创建对象的时候先于构造方法执行

局部代码块：定义在方法中用{}包起来的代码 --- 提高了栈内存的利用率

## 面向对象的特征

封装、继承、多态（抽象）

### 封装

将代码进行提取和总结，抽取成不同的形式 --- 封装

体现形式：方法、类（属性的私有化 - 将属性限制为私有的，然后提供对外的设置和获取的方法，然后在方法中进行限制使数据更加符合常理）

优势：提高了代码的复用性，保证了数据的合法性

### 权限修饰符

指在Java中用于限定使用范围的关键字

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 本类中 | 子类中 | 同包类中 | 其他类中 |
| public | 可以 | 可以 | 可以 | 可以 |
| protected | 可以 | 可以 | 可以 | 不可以 |
| 默认 | 可以 | 同包子类可以 | 可以 | 不可以 |
| private | 可以 | 不可以 | 不可以 | 不可以 |

练习：定义一个类表示矩形（长方形），提供获取周长和面积的方法

## 继承

如果一些类中含有相同的代码，那么可以将这些相同的代码提取到一个新的类中，然后通过extends关键字让原来的类和新的类产生关系 --- 继承。 通过继承，原来的类就称之为了子类（派生类），新的类就成了父类（超类/基类）。

子类通过继承父类可以使用父类中的一部分方法和属性

注意：子类继承了父类的全部的数据域（方法+属性），但是继承之后只有一部分数据域对子类可见

在Java中，支持的是类和类之间的单继承 -> 一个子类只能继承一个父类，但是一个父类可以有多个子类

单继承一定优于多继承吗？ - 不对

多继承比单继承能够更好的提高代码的复用性

class A {

public int m(){return 9;}

}

class B {

public boolean m(){return false;}

}

class C extends A, B{}

C c = new C();

c.m(); // 返回值类型能确定吗？

多继承导致在调用方法的时候可能产生歧义

优势：提高代码的复用性，避免方法的调用产生歧义

### super关键字

在子类中用于表示父类对象的引用，可以在子类中调用父类中的方法的属性。

super语句 --- 子类在继承父类之后，子类的构造方法中会含有一个super语句。如果没有手动指定super语句，那么默认使用super()调用父类无参的构造；如果父类只提供了含参构造，那么子类就必须手动提供对应形式的super语句 --- super语句必须在子类构造方法的首行

### 方法的重写/覆盖

在父子类中存在了方法签名相同的非静态方法。遵循 “两等两小一大”原则：

1. 方法签名相同

2. 如果父类中的方法的返回值类型是基本类型/void，那么子类重写的方法的返回值类型与父类一致

class A {

public void m(){}

}

class B extends A {

public void m(){}

}

3. 如果父类中的方法的返回值类型是引用类型，那么子类在重写方法的时候，返回值类型要么与父类一致，要么是父类方法返回值类型的子类

class A {}

class B extends A {}

class C {

public A m(){return null;}

}

class D extends C {

public B m(){return null;}

}

4. 子类重写的方法的权限修饰符的范围要大于等于父类中对应方法的权限修饰符的范围

class A {

public void m(){}

}

class B extends A {

public void m(){}

}

注意:如果父类中的方法用private修饰，那么这个方法对子类不可见，所以此时与子类中的方法构不成重写

### 多态

编译时多态：方法的重载

add(2, 4) -> add(int, int)

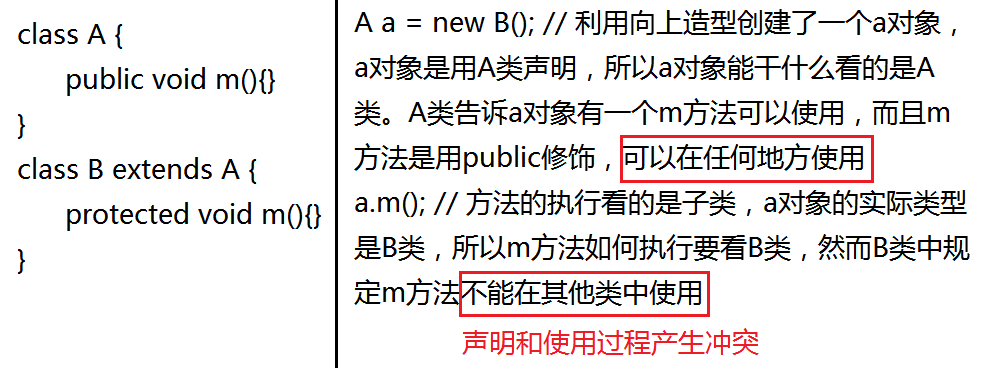
add(3, 1, 7) -> add(int, int, int)

运行时多态：向上造型和方法的重写 --- 基于继承的

注意：如果使用向上造型来创建对象，那么这个对象所能调用的方法看的是父类中的声明，方法如何执行看的是子类中的实现过程

#### 重写的理解

1. 子类重写的方法的权限修饰符的范围要大于等于父类中对应方法的权限修饰符的范围



2. 如果父类中的方法的返回值类型是引用类型，那么子类在重写方法的时候，返回值类型要么与父类一致，要么是父类方法返回值类型的子类

注意：Java中所有的基本类型之间没有继承关系，之所以能够自动提升，是因为所表示的范围是否能够包含

