第5章 面向对象基础（上）

学习目标

- [ ] 初步了解面向对象的思想
- [ ] 能够明确类与对象关系
- [ ] 能够掌握类的定义格式
- [ ] 能够掌握创建对象格式
- [ ] 理解包的作用
- [ ] 掌握包的声明和导入
- [ ] 能够通过对象访问对象的非静态成员变量和非静态成员方法
- [ ] 能够区别类变量与实例变量
- [ ] 能够区别成员变量与局部变量
- [ ] 能够理解方法的调用执行机制
- [ ] 能够理解方法的参数传递机制
- [ ] 掌握方法的可变参数的使用
- [ ] 掌握方法重载的概念
- [ ] 能够判断出方法的重载
- [ ] 了解命令行参数
- [ ] 理解递归方法
- [ ] 理解对象数组

第五章 面向对象基础（上）

5.1 面向对象思想概述

5.1.1 概述

Java是一种计算机程序设计语言。所有的计算机程序一直都是围绕着两件事在进行的，程序设计就是用某种语言编写代码来完成这两件事，所以程序设计语言又称为编程语言。

1. 如何表示和存储数据
    - 基本数据类型的常量和变量：表示和存储一个个独立的数据
    - 对象：表示和存储与某个具体事物相关的多个数据（例如：某个学生的姓名、年龄、联系方式等）
    - 数据结构：表示和存储一组对象，数据结构有数组、链表、栈、队列、散列表、二叉树、堆……
2. 基于这些数据都有什么操作行为，其实就是实现什么功能
    - 数据的输入和输出
    - 基于一个或两个数据的操作：赋值运算、算术运算、比较运算、逻辑运算等
    - 基于一组数据的操作：统计分析、查找最大值、查找元素、排序、遍历等

面向对象和面向过程都是一种编程思想，基于不同的思想会产生不同的程序设计方法。Java语言是在面向对象思想的指引下去设计、开发计算机程序的，所以Java语言是一种面向对象的程序设计语言。而C语言是一种面向过程的程序设计语言。

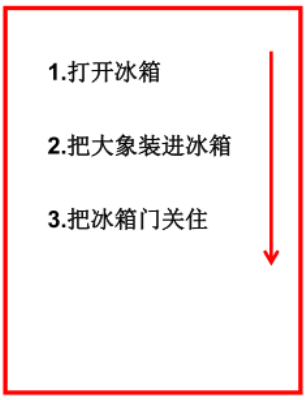
5.1.2 面向对象与面向过程的区别

1. 面向过程的程序设计思想（Process-Oriented Programming），简称POP
    - 关注的焦点是过程：过程就是操作数据的步骤，如果某个过程的实现代码在很多地方重复出现，那么就可以把这个过程抽象为一个函数，这样就可以大大简化冗余代码，也便于维护。
    - 代码结构：以函数为组织单位。独立于函数之外的数据称为全局数据，在函数内部的称为局部数据。
    - 以过程，步骤为主，考虑怎么做，程序员是具体执行者
    - 制约了软件的可维护性和可扩展性
2. **面向对象的程序设计思想**（Object Oriented Programming），简称OOP
    - 关注的焦点是类：**面向对象思想**就是在计算机程序设计过程中，参照现实中事物，将事物的**属性特征**、**行为特征**抽象出来，用类来表示。某个事物的一个具体个体称为实例或对象。
    - 代码结构：以类为组织单位。每种事物都具备自己的**属性**（即表示和存储数据，在类中用成员变量表示）和**行为/功能**（即操作数据，在类中用成员方法表示）。
    - 以对象（谁）为主，考虑谁来做，谁能做，程序员是指挥者
    - 面向对象仍然包含面向过程，只不过关注点变了，关注谁来做
    - 软件可重用性、可维护性和可扩展性强

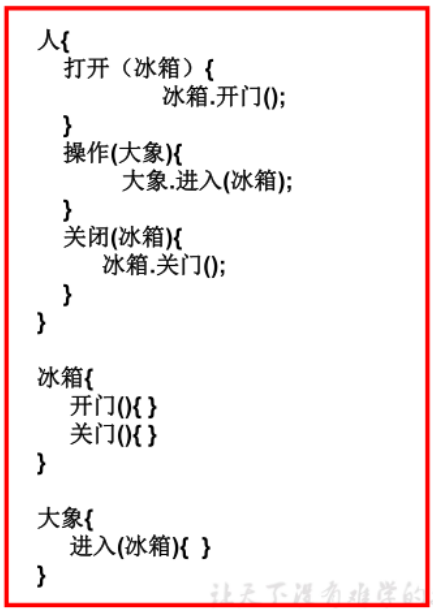
面向对象思想是一种更符合我们思考习惯的思想，它可以将复杂的事情简单化，并将我们从执行者变成了指挥者。

例子：吃饭，洗衣服

把大象装进冰箱



面向过程



面向对象

5.2 类和对象

5.2.1 类与对象的概念及关系

万物皆对象，环顾周围，你会发现很多对象，比如桌子，椅子，同学，老师，顾客，收银员等。

描述身边的对象：

如何描述对象？

对象的属性：姓名，年龄，体重，员工编号，部门等对象的静态特征

对象的行为：购买商品，收款，打印账单等对象的动态特征或行为特征或者功能

1. **什么是对象?**
    - **对象**：是一个具有特定属性和行为特征的具体事物。
2. **什么是类?**
    - **类**：是一类具有相同特征的事物的抽象描述，是一组相同**属性**和**行为**的对象的集合。
3. **类与对象的关系**
    - 类是对一类事物的描述，是**抽象的。**
    - 对象是一类事物的实例，是**具体的。**
    - **类是对象的模板，对象是类的实体。**

> 上例中的类和对象：
> **顾客**是一类事物的抽象描述，即为类，他们都有姓名，年龄，体重这些属性特征和购买商品的行为特征；**张三**是一个具体的顾客，即为对象。
> **收银员**是另一类事物的抽象描述，他们都有员工号，姓名、部门这些属性特征和收款、打印账单的行为特征；**李四**是一个具体的收银员。

举例描述类和对象：学生、手机、汽车、猫等

5.2.2 类的定义

Java中类的定义，就是把现实中类的概念用Java语言描述。

Java中用**class**关键字定义一个类，并定义类的成员：成员变量（属性）和成员方法（行为）。

类的定义格式

```
public class 类名 {
  //成员变量，描述这类事物的属性
  //成员方法，描述这类事物的行为
}
```

- **成员变量**：和以前定义变量几乎是一样的。只不过位置发生了改变。**在类中，方法外**，用于描述对象的属性特征。
- **成员方法**：和以前写的main方法格式类似。只不过功能和形式更丰富了。在类中，方法外，用于描述对象的行为特征。

定义类的代码举例：

```
//定义顾客类
public class Customer {
    //成员变量,描述属性特征
    String name;//姓名
    int age;//年龄
    int weight;//体重

    //成员方法，描述行为特征
    public void shopping(){
        System.out.println("购物...");
    }
}
```

练习：

定义学生类，汽车类

5.2.3 对象的创建与使用

类是对象的模板，所以通过类创建这个类的对象，或者说创建这个类的一个实例，这个过程称为类的实例化：

- **创建对象语法格式：**

  类名 对象名= new 类名 ();

  ```
  //创建顾客对象
  Customer c=new Customer();
  ```

- **使用对象的成员，使用“. ”操作**：

  使用成员变量：**对象名.属性**
  使用成员方法：**对象名.方法名()**

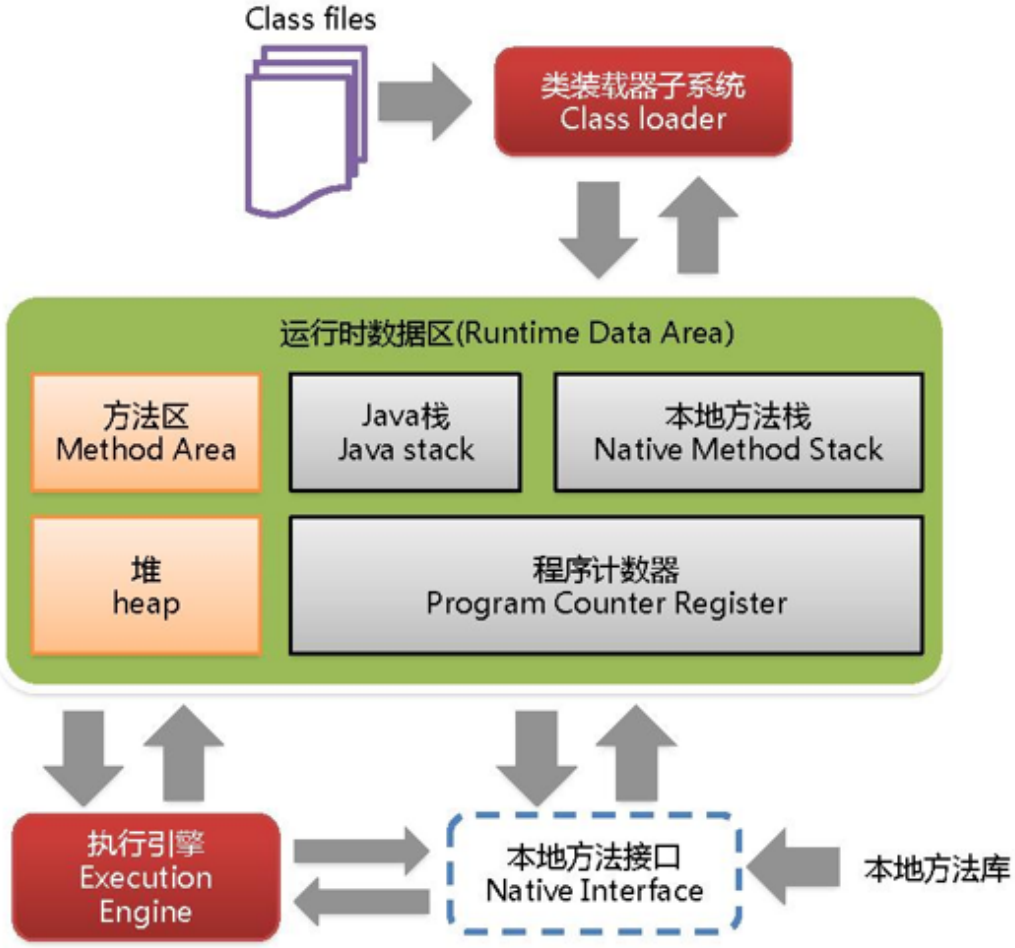
  ```
  c.name="张三"; //访问对象的属性，赋值
  c.age=18;
  System.out.println(c.name+"--"+c.age); //访问对象的属性，获取值
  c.shopping(); //访问对象的方法
  ```

练习：

定义手机类并创建对象再使用

5.2.4对象的内存分析

JVM内存结构图：



区域名称	作用
栈	虚拟机栈，用于存储正在执行的每个Java方法的局部变量表等。局部变量表存放了编译期可知长度的各种基本数据类型、对象引用，方法执行完，自动释放。
堆	存储对象（包括数组对象），new来创建的，都存储在堆内存。
方法区	存储已被虚拟机加载的类信息、常量、静态变量、即时编译器编译后的代码等数据。
程序计数器	程序计数器是CPU中的寄存器，它包含每一个线程下一条要执行的指令的地址
本地方法栈	当程序中调用了native的本地方法时，本地方法执行期间的内存区域

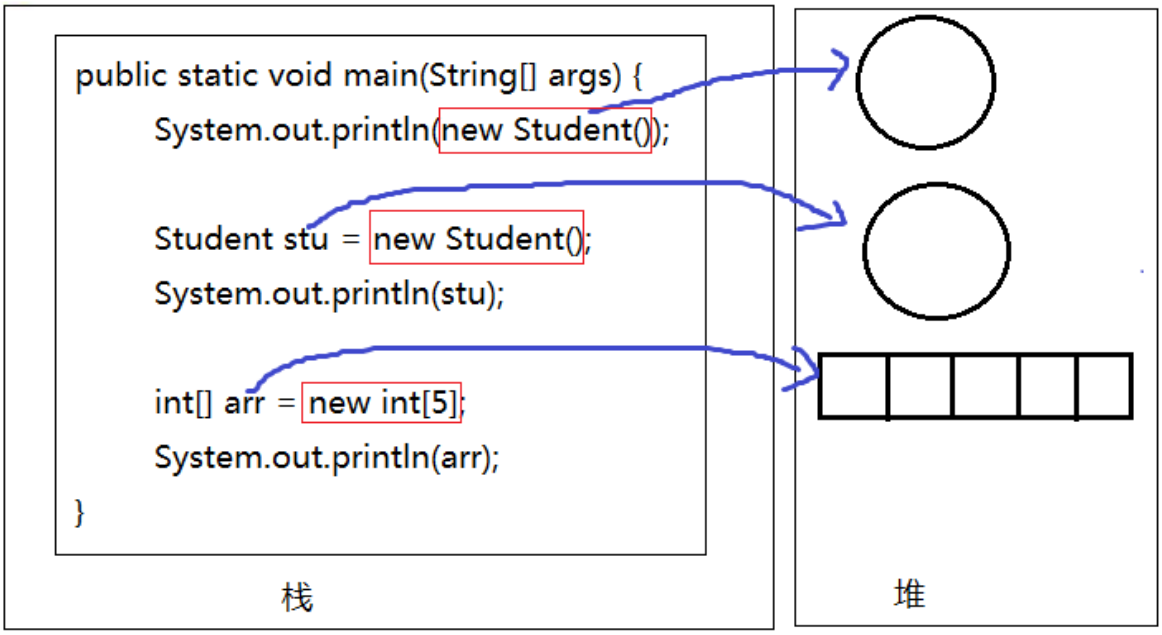
对象名中存储的是什么呢？答：对象地址

```
class Student{

}
public class TestStudent{
    //Java程序的入口
```

```
    public static void main(String[] args){
        System.out.println(new Student());//Student@7852e922

        Student stu = new Student();
        System.out.println(stu);//Student@4e25154f

        int[] arr = new int[5];
        System.out.println(arr);//[I@70dea4e
    }
}
//Student和TestStudent没有位置要求，谁在上面谁在下面都可以
//但是如果TestStudent类的main中使用了Student类，那么要求编译时，这个Student已经写好了，不写是不行的
//如果两个类都在一个.java源文件中，只能有一个类是public的
```

发现学生对象和数组对象类似，直接打印对象名和数组名都是显示“类型@对象的hashCode值"，所以说类、数组都是引用数据类型，引用数据类型的变量中存储的是对象的地址，或者说指向堆中对象的首地址。

那么像“Student@4e25154f”是对象的地址吗？不是，因为Java是对程序员隐藏内存地址的，不暴露内存地址信息，所以打印对象时不直接显示内存地址，而是JVM提取了对象描述信息给你现在，默认提取的是对象的运行时类型@代表对象唯一编码的hashCode值。



5.3 包（Package）

5.3.1 包的作用

（1）可以避免类重名：有了包之后，类的全名称就变为：包.类名

（2）分类组织管理众多的类

例如：

- java.lang----包含一些Java语言的核心类，如String、Math、Integer、 System和Thread等，提供常用功能
- java.net----包含执行与网络相关的操作的类和接口。
- java.io ----包含能提供多种输入/输出功能的类。
- java.util----包含一些实用工具类，如集合框架类、日期时间、数组工具类Arrays，文本扫描仪Scanner，随机值产生工具Random

- java.text----包含了一些java格式化相关的类
- java.sql和javax.sql----包含了java进行JDBC数据库编程的相关类/接口
- java.awt和java.swing----包含了构成抽象窗口工具集（abstract window toolkits）的多个类，这些类被用来构建和管理应用程序的图形用户界面(GUI)。

（3）可以控制某些类型或成员的可见范围

如果某个类型或者成员的权限修饰缺省的话，那么就仅限于本包使用

5.3.2 声明包的语法格式

```
package 包名;
```

> 注意：
>
> (1)必须在源文件的代码首行
>
> (2)一个源文件只能有一个声明包的语句

包的命名规范和习惯：
（1）所有单词都小写，每一个单词之间使用.分割
（2）习惯用公司的域名倒置

例如：com.atguigu.xxx;

> 建议大家取包名时不要使用“java.xx"包

5.3.3 如何跨包使用类

前提：被使用的类或成员的权限修饰符是>缺省的，即可见的

（1）使用类型的全名称

例如：java.util.Scanner input = new java.util.Scanner(System.in);

（2）使用import 语句之后，代码中使用简名称

import语句告诉编译器到哪里去寻找类。

import语句的语法格式：

```
import 包.类名;
import 包.*;
import static 包.类名.静态成员; //后面再讲
```

> 注意：
>
> 使用java.lang包下的类，不需要import语句，就直接可以使用简名称
>
> import语句必须在package下面，class的上面
>
> 当使用两个不同包的同名类时，例如：java.util.Date和java.sql.Date。一个使用全名称，一个使用简名称

示例代码：

```
package com.atguigu.bean;

public class Student {
    // 成员变量
```

```
    private String name;
    private int age;

    // 构造方法
    public Student() {
    }

    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // 成员方法
    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public int getAge() {
        return age;
    }
}
```

```
package com.atguigu.test;

import java.util.Scanner;
import java.util.Date;
import com.atguigu.bean.Student;

public class Test{
    public static void main(String[] args){
        Scanner input = new Scanner(System.in);
        Student stu = new Student();
        String str = "hello";

        Date now = new Date();
        java.sql.Date d = new java.sql.Date(346724566);
    }
}
```

5.4 成员变量

5.4.1 变量的分类

根据定义位置不同分为：

- **局部变量**：定义在方法体内或其他局部区域内的变量（之前所使用的都是main方法中定义的变量，为局部变量）。

- **成员变量**：定义在类的成员位置，在方法体外，与方法（例如main方法）平行的位置。并且有修饰符修饰。

  根据修饰的不同成员变量又分为：

  - **类变量**：或叫**静态变量**，有static修饰的成员变量。（后面再讲）
  - **实例变量**：没有static修饰的成员变量。

5.4.2 成员变量的声明

语法格式：

```
class 类名{
    【修饰符】 数据类型  属性名;
}
```

> 说明：常用修饰符有public、缺省、private、protected、final、static
>
> 数据类型可以是任意基本数据类型和引用数据类型。
>
> 属性名即变量名，符合标识符的命名规则和规范。

示例：

```
//定义一个人类
public class Person{
    String name;
    char gender;
    int age;
}
```

5.4.3 实例变量

1、实例变量的特点

1. 实例变量的值是属于某个对象的
   - 必须通过对象才能访问实例变量
   - 每个对象的实例变量的值是独立的
2. **成员变量有默认初始值**（同数组元素默认初始值）

数据类型	默认值
byte，short，int，long	0
float，double	0.0
char	0或'\u0000'表现为空
boolean	false
数组，类，接口等引用类型	null

2、实例变量的访问

```
对象.实例变量
```

例如：

```
public class TestPerson {
    public static void main(String[] args) {
        Person p1 = new Person();
        p1.name = "张三";
        p1.age = 23;
        p1.gender = '男';

        Person p2 = new Person();
        /*
        （1）实例变量的值是属于某个对象的
        - 必须通过对象才能访问实例变量
        - 每个对象的实例变量的值是独立的
        （2）实例变量有默认值
         */
        System.out.println("p1对象的实例变量：");
        System.out.println("p1.name = " + p1.name);
        System.out.println("p1.age = " + p1.age);
        System.out.println("p1.gender = " + p1.gender);

        System.out.println("p2对象的实例变量：");
        System.out.println("p2.name = " + p2.name);
        System.out.println("p2.age = " + p2.age);
        System.out.println("p2.gender = " + p2.gender);
    }
}
```

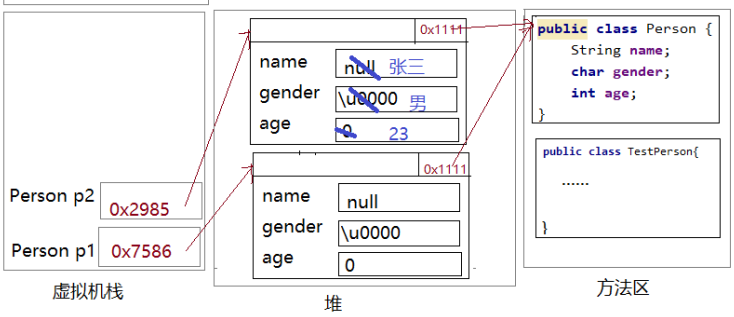
3、实例变量的内存分析

Java对象保存在内存中时，由以下三部分组成：

- 对象头
  - Mark Word：记录了和当前对象有关的GC、锁等信息。（后面再讲）
  - 指向类的指针：每一个对象需要记录它是由哪个类创建出来的，而Java对象的类数据保存在方法区，指向类的指针就是记录创建该对象的类数据在方法区的首地址。该指针在32位JVM中的长度是32bit，在64位JVM中长度是64bit。
  - 数组长度（只有数组对象才有）
- 实例数据
  - 即实例变量的值
- 对齐填充
  - 因为JVM要求Java对象占的内存大小应该是8bit的倍数，如果不满足该大小，则需要补齐至8bit的倍数，没有特别的功能。

```java
public class TestPerson {
    public static void main(String[] args) {
        Person p1 = new Person();
        p1.name = "张三";
        p1.age = 23;
        p1.gender = '男';

        Person p2 = new Person();
        System.out.println("p1对象的实例变量：");
        System.out.println("p1.name = " + p1.name);
        System.out.println("p1.age = " + p1.age);
        System.out.println("p1.gender = " + p1.gender);

        System.out.println("p2对象的实例变量：");
        System.out.println("p2.name = " + p2.name);
        System.out.println("p2.age = " + p2.age);
        System.out.println("p2.gender = " + p2.gender);
    }
}
```

```java
public class Person {
    String name;
    char gender;
    int age;
}
```



4、小结:实例变量与局部变量的区别

	实例变量	局部变量
声明的位置	直接声明在类的成员位置	声明在方法体中或其他局部区域内（方法声明上，构造方法，代码块等）
修饰符	public、private、final等	不能使用访问权限修饰符，可以使用final
内存加载位置	堆	栈
初始化值	有默认初始化值	无默认初始化值
生命周期	同对象的生命周期	随着方法的调用而存在，方法调用完毕即消失

练习题

（1）声明一个圆的图形类，有属性：半径
在测试类的main中，创建圆的2个对象，为半径属性赋值，并显示两个圆的半径值

（2）声明一个银行账户类，有属性：利率、账号、余额

在测试类的main中，创建账户类的两个对象，其中所有账户的利率是相同的，都是0.035，而账号和余额是不同的，并打印显示

（3）声明一个MyDate类型，有属性：年，月，日

声明另一个Employee类型，有属性：姓名（String类型），生日（MyDate类型）

在测试类中的main中，创建两个员工对象，并为他们的姓名和生日赋值，并显示

5.5 方法（Method）

成员变量是用来存储对象的数据信息的，那么如何表示对象的行为功能呢？就要通过方法来实现

5.5.1 方法的概念

方法也叫函数，是一个独立功能的定义，是一个类中最基本的功能单元。

把一个功能封装为方法的目的是，可以实现代码重用，从而简少代码量。

5.5.2 成员方法的分类

根据修饰不同方法主要分为两类：

- **实例方法**：没有static修饰的方法，必须通过实例对象来调用。
- **静态方法**：有static修饰的方法，也叫类方法，主要特点是可以由类名来调用。（后面再讲）

5.5.3 方法的声明

1. **方法声明的位置必须在类中方法外，在类的成员位置**
2. **语法格式**：

```
【修饰符】 返回值类型 方法名(【参数列表 】)【throws 异常列表】{
        方法体;
        【return 返回值;】
}
```

3. **格式说明**：
    1. 一个完整的方法 = 方法头 + 方法体。
        - 大括号内为方法体，主要来实现功能；
        - 大括号之前的内容是方法头，也称为方法签名。通常调用方法时只关注方法头即可。方法头包含5部分，有些部分可以缺省。
    2. **修饰符**： 修饰符后面详细讲，例如：public，static等都是修饰符
    3. **返回值类型**： 表示方法运行的结果的数据类型，与**"return 返回值"**搭配使用
    - 无返回值：void
    - 有返回值：可以是任意基本数据类型和引用数据类型
    4. **方法名**：给方法起一个名字，要符合标识符的命名规则，尽量见名知意，能准确代表该方法功能的名字
    5. **参数列表**：方法内部需要用到其他方法中的数据，需要通过参数传递的形式将数据传递过来，可以是基本数据类型、引用数据类型、也可以没有参数，什么都不写
    6. throws 异常列表：可选，在异常章节再讲
    7. **方法体**：特定功能的代码
    8. **return**：结束方法，可以返回方法的运行结果
    - 可以返回不同类型的数据，对应匹配的**返回值类型**。
    - 如果方法无返回值，可以省去return，并且返回值类型为**void**
4. **示例**：

```
public class MethodDefineDemo {

     /**
     * 无参无返回值方法的演示
     */
    void sayHello(){
        System.out.println("hello");
    }

    /**
     * 有参无返回值方法的演示
     * @param length int 第一个参数，表示矩形的长
     * @param width int 第二个参数，表示矩形的宽
     * @param sign char 第三个参数，表示填充矩形图形的符号
     */
    void printRectangle(int length, int width, char sign){
```

```
        for (int i = 1; i <= length ; i++) {
            for(int j=1; j <= width; j++){
                System.out.print(sign);
            }
            System.out.println();
        }
    }

    /**
     * 无参有返回值方法的演示
     * @return
     */
    int getIntBetweenOneToHundred(){
        return (int)(Math.random()*100+1);
    }

    /**
     * 有参有返回值方法的演示
     * @param a int 第一个参数，要比较大小的整数之一
     * @param b int 第二个参数，要比较大小的整数之二
     * @return int 比较大小的两个整数中较大者的值
     */
    int max(int a, int b){
        return a > b ? a : b;
    }
}
```

5.5.4 实例方法的调用

方法必须先声明后使用，不调用不执行，调用一次执行一次。

1. **实例方法的调用格式**

```
对象名.实例方法(【实参列表】)
```

示例：

```
/**
 * 方法调用案例演示
 */
public class MethodInvokeDemo {
    public static void main(String[] args) {
        //创建对象
        MethodDefineDemo md = new MethodDefineDemo();

        System.out.println("----------------------方法调用演示----------------
----------");

        //调用MethodDefineDemo类中无参无返回值的方法sayHello
        md.sayHello();
        md.sayHello();
        md.sayHello();
        //调用一次，执行一次，不调用不执行

        System.out.println("------------------------------------------------
");
```

```
        //调用MethodDefineDemo类中有参无返回值的方法printRectangle
        md.printRectangle(5,10,'@');

        System.out.println("-------------------------------------------------
");
        //调用MethodDefineDemo类中无参有返回值的方法getIntBetweenOneToHundred
        md.getIntBetweenOneToHundred();//语法没问题，就是结果丢失

        int num = md.getIntBetweenOneToHundred();
        System.out.println("num = " + num);

        System.out.println(md.getIntBetweenOneToHundred());
        //上面的代码调用了getIntBetweenOneToHundred三次，这个方法执行了三次

        System.out.println("-------------------------------------------------
");
        //调用MethodDefineDemo类中有参有返回值的方法max
        md.max(3,6);//语法没问题，就是结果丢失

        int bigger = md.max(5,6);
        System.out.println("bigger = " + bigger);

        System.out.println("8,3中较大者是：" + md.max(8,9));
    }
}
```

回忆之前的代码：

```
//1、创建Scanner的对象
Scanner input = new Scanner(System.in);//System.in默认代表键盘输入

//2、提示输入xx
System.out.print("请输入一个整数："); //对象.非静态方法(实参列表)

//3、接收输入内容
int num = input.nextInt();  //对象.非静态方法()
```

2. **形参与实参的概念理解**
   - **形参**：在定义方法时方法名后面括号中声明的变量称为形式参数（简称形参）即形参出现在方法定义时。
   - **实参**：调用方法时方法名后面括号中的使用的值/变量/表达式称为实际参数（简称实参）即实参出现在方法调用时。
3. **方法调用的注意事项**
   - 调用方法时，实参的个数、类型、顺序必须要与形参列表一一对应
   - 调用方法时，如果方法有返回值，可以用与返回值类型相同的变量接受或直接处理返回值结果，如果方法的返回值类型是void，不需要也不能接收和处理返回值结果。
   - 方法调用表达式的结果可以不接收和处理，方法调用表达式直接加;成为一个独立的语句，这种情况，返回值丢失。

```
package com.atguigu.test04.method;

public class MethodReturnValue {
    public static void main(String[] args) {
```

```
        //创建对象
        MethodDefineDemo md = new MethodDefineDemo();

        //无返回值的都只能单独加;成一个独立语句
        //调用MethodDefineDemo类中无参无返回值的方法sayHello
        md.sayHello();
        //调用MethodDefineDemo类中有参无返回值的方法printRectangle
        md.printRectangle(5,10,'@');

        //有返回值的
        //(1)方法调用表达式可以作为赋值表达式的值
        int bigger = md.max(7,3);
        System.out.println("bigger = " + bigger);

        //(2)方法调用表达式可以作为计算表达式的一个操作数
        //随机产生两个[1,100]之间的整数，并求和
        int sum = md.getIntBetweenOneToHundred() +
md.getIntBetweenOneToHundred();
        System.out.println("sum = " + sum);

        //(3)方法调用表达式可以作为另一次方法调用的实参
        int x = 4;
        int y = 5;
        int z = 2;
        int biggest = md.max(md.max(x,y),z);
        System.out.println("biggest = " + biggest);

        //(4)方法调用表达式直接加;成为一个独立的语句，这种情况，返回值丢失
        md.getIntBetweenOneToHundred();
    }
}
```

5.5.5 本类内的实例变量和方法访问

在实例方法中还可以使用当前对象的其他成员。在Java中当前对象用this表示。

- this：在实例方法中，表示调用该方法的对象
- 如果没有歧义，完全可以省略this。

使用this的案例：矩形类

```
public class Rectangle {
    int length;
    int width;
    //求面积
    int area() {
        return this.length * this.width;
    }
    //求周长
    int perimeter(){
        return 2 * (this.length + this.width);
    }
    //打印矩形
    void print(char sign) {
        for (int i = 1; i <= this.width; i++) {
```

```
            for (int j = 1; j <= this.length; j++) {
                System.out.print(sign);
            }
            System.out.println();
        }
    }
    //打印矩形对象信息
    String getInfo(){
        return "长：" + this.length + "，宽：" + this.width +"，面积：" +
this.area() +"，周长：" + this.perimeter();
    }
}
```

测试类

```
public class TestRectangle {
    public static void main(String[] args) {
        Rectangle r1 = new Rectangle();
        Rectangle r2 = new Rectangle();

        System.out.println("r1对象：" + r1.getInfo());
        System.out.println("r2对象：" + r2.getInfo());

        r1.length = 10;
        r1.width = 2;
        System.out.println("r1对象：" + r1.getInfo());
        System.out.println("r2对象：" + r2.getInfo());

        r1.print('#');
        System.out.println("----------------------");
        r1.print('&');

        System.out.println("----------------------");
        r2.print('#');
        System.out.println("----------------------");
        r2.print('%');
    }
}
```

省略this.

```
public class Rectangle {
    int length;
    int width;

    int area() {
        return length * width;
    }

    int perimeter(){
        return 2 * (length + width);
    }

    void print(char sign) {
        for (int i = 1; i <= width; i++) {
```

```
            for (int j = 1; j <= length; j++) {
                System.out.print(sign);
            }
            System.out.println();
        }
    }

    String getInfo(){
        return "长：" + length + "，宽：" + width +"，面积：" + area() +"，周长：" +
perimeter();
    }
}
```

练习

1. **声明三角形类Triangle**

   （1）包含属性：高、宽

   （2）包含3个方法：

   1. 求面积、
   2. 求周长、
   3. 返回三角形对象的信息：长：xx，宽：xx，面积：xx，周长：xx

   （3）在测试类中进行测试

2. **声明一个圆类Circle**

   （1）有属性：半径radius

   （2）包含3个方法：

   1. 求面积、
   2. 求周长、
   3. 返回三角形对象的信息：长：xx，宽：xx，面积：xx，周长：xx

   （3）在测试类中进行测试

5.5.7 方法调用内存分析

方法不调用不执行，调用一次执行一次，每次调用会在栈中有一个入栈动作，即给当前方法开辟一块独立的内存区域，用于存储当前方法的局部变量的值，当方法执行结束后，会释放该内存，称为出栈，如果方法有返回值，就会把结果返回调用处，如果没有返回值，就直接结束，回到调用处继续执行下一条指令。

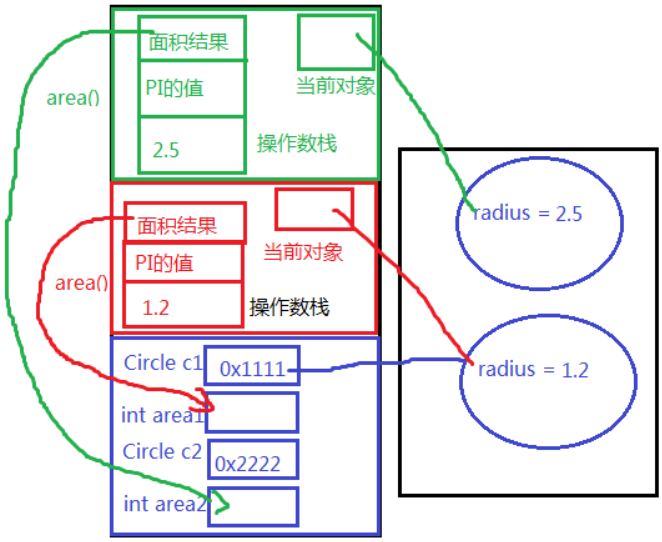
栈结构特点：先进后出，后进先出。

1. **示例一：**

```
public class TestCircle {
    public static void main(String[] args) {
        Circle c1 = new Circle();
        c1.radius = 1.2;
        int area1 = c1.area();

        Circle c2 = new Circle();
        c2.radius = 2.5;
        int area2 = c2.area();
    }
}
```

```
class Circle{
    double radius;
    public double area() {
        return Math.PI * radius * radius;
    }
}
```

```
public class TestCircle {
    public static void main(String[] args) {
        Circle c1 = new Circle();
        c1.radius = 1.2;
        int area1 = c1.area();

        Circle c2 = new Circle();
        c2.radius = 2.5;
        int area2 = c2.area();
    }
}
class Circle{
    double radius;
    public double area() {
        return Math.PI * radius * radius;
    }
}
```

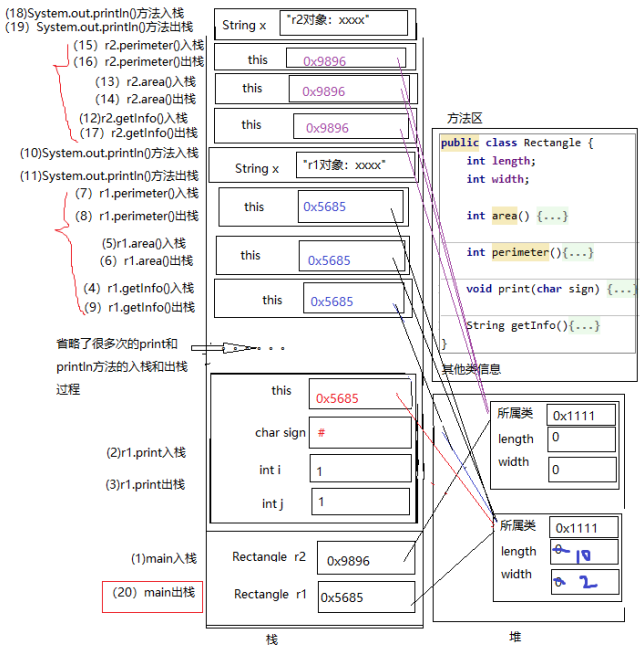


2. **示例二：**

```
public class MethodMemory {
    public static void main(String[] args) {
        Rectangle r1 = new Rectangle();
        Rectangle r2 = new Rectangle();
        r1.length = 10;
        r1.width = 2;
        r1.print('#');
        System.out.println("r1对象：" + r1.getInfo());
        System.out.println("r2对象：" + r2.getInfo());
    }
}
```

```
public class MethodMemory {
    public static void main(String[] args) {
        Rectangle r1 = new Rectangle();
        Rectangle r2 = new Rectangle();
        r1.length = 10;
        r1.width = 2;
        r1.print('#');
        System.out.println("r1对象：" + r1.getInfo());
        System.out.println("r2对象：" + r2.getInfo());
    }
}
```

```
public class Rectangle {
    int length;
    int width;

    int area() { return this.length * this.width;  }

    int perimeter() { return 2 * (this.length + this.width); }

    void print(char sign) {
        for (int i = 1; i <= this.width; i++) {
            for (int j = 1; j <= this.length; j++) {
                System.out.print(sign);
            }
            System.out.println();
        }
    }

    String getInfo(){
        return "长：" + this.length + ", 宽：" + this.width
                +", 面积：" + this.area() +", 周长：" + this.perimeter();
    }
}
```



5.6 方法参数的值传递机制

方法的参数传递机制：实参给形参赋值，那么反过来形参会影响实参吗？

- **方法的形参是基本数据类型时，形参值的改变不会影响实参；**
- **方法的形参是引用数据类型时，形参地址值的改变不会影响实参，但是形参地址值里面的数据的改变会影响实参，例如，修改数组元素的值，或修改对象的属性值。**
  - **注意：String、Integer等特殊类型容易错**

1、形参是基本数据类型

案例：编写方法，交换两个整型变量的值

```
package com.atguigu.test05.param;

public class PrimitiveTypeParam {
    void swap(int a, int b){//交换两个形参的值
        int temp = a;
        a = b;
        b = temp;
    }

    public static void main(String[] args) {
        PrimitiveTypeParam tools = new PrimitiveTypeParam();
        int x = 1;
        int y = 2;
        System.out.println("交换之前：x = " + x +",y = " + y);//1,2
        tools.swap(x,y);//实参x,y是基本数据类型，给形参的是数据的“副本”，调用完之后，x与y的值不变
        System.out.println("交换之后：x = " + x +",y = " + y);//1,2
    }
}
```

2、形参是引用数据类型

```
package com.atguigu.test05.param;

public class ReferenceTypeParam {
    void swap(MyData my){//形参my是引用数据类型，接收的是对象的地址值，形参my和实参data指向同一个对象
        //里面交换了对象的两个实例变量的值
        int temp = my.x;
        my.x = my.y;
        my.y = temp;
    }

    public static void main(String[] args) {
        ReferenceTypeParam tools = new ReferenceTypeParam();
        MyData data = new MyData();
        data.x = 1;
        data.y = 2;
        System.out.println("交换之前：x = " + data.x +",y = " + data.y);//1,2
        tools.swap(data);//实参是data，给形参my的是对象的地址值，调用完之后，x与y的值交换
        System.out.println("交换之后：x = " + data.x +",y = " + data.y);//2,1
    }
```

```
}
```

```
public class MyData{
    int x;
    int y;
}
```

3、形参是数组

```
package com.atguigu.test05.param;

public class ArrayTypeParam {
    void sort(int[] arr){//给数组排序，修改了数组元素的顺序，这里对arr数组进行排序，就相当于对nums数组进行排序
        for (int i = 1; i < arr.length; i++) {
            for (int j = 0; j < arr.length - i; j++) {
                if(arr[j] > arr[j+1]){
                    int temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
                }
            }
        }
    }

    void iterate(int[] arr){//输出数组的元素，元素之间使用空格分隔，元素打印完之后换行
                            //这个方法没有修改元素的值
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i]+" ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        ArrayTypeParam tools = new ArrayTypeParam();

        int[] nums = {4,3,1,6,7};
        System.out.println("排序之前：");
        tools.iterate(nums);//实参nums把数组的首地址给形参arr，这个调用相当于输出nums数组的元素
                            //对数组的元素值没有影响

        tools.sort(nums);//对nums数组进行排序

        System.out.println("排序之后：");
        tools.iterate(nums);//输出nums数组的元素
        //上面的代码，从头到尾，堆中只有一个数组，没有产生新数组，无论是排序还是遍历输出都是同一个数组
    }
}
```

4、形参指向新对象

```
package com.atguigu.test05.param;
```

```
public class AssignNewObjectToFormalParam {
    void swap(MyData my){
        my = new MyData(); //这里让my形参指向了新对象，此时堆中有两个MyData对象，和main中的data对象无关
        int temp = my.x;
        my.x = my.y;
        my.y = temp;

    }

    public static void main(String[] args) {
        //创建这个对象的目的是为了调用swap方法
        AssignNewObjectToFormalParam tools = new AssignNewObjectToFormalParam();

        MyData data = new MyData();
        data.x = 1;
        data.y = 2;
        System.out.println("交换之前：x = " + data.x +",y = " + data.y);//1,2
        tools.swap(data);//调用完之后，x与y的值交换？
        System.out.println("交换之后：x = " + data.x +",y = " + data.y);//1,2
    }
}
```

5.7 方法重载Overload

方法重载：同一个类中（本类声明的或继承自父类的）的方法，方法名相同，参数列表不同的情况，这就叫方法重载。

参数列表不同：指的是参数个数不同，数据类型不同，数据类型顺序不同。

注意：与方法的返回值类型无关

案例：用重载实现：

（1）定义方法求两个整数的和

（2）定义方法求三个整数的和

（3）定义方法求两个小数的和

（4）定义方法求一个小数与一个整数的和

```
package com.atguigu.test06.overload;

public class MathTools {
    //求两个整数的和
    public int sum(int a,int b){
        return a+b;
    }

    //求两个小数的和
    public double sum(double a, double b){
        return a+b;
    }

    //求三个整数的和
```

```
    public int sum(int a, int b, int c){
        return a+b+c;
    }

    //求一个小数与一个整数的和
    public  double sum(double a, int b){
        return a+b;
    }
}
```

调用方法时，多个方法都匹配，找最匹配的方法来调用。

```
package com.atguigu.test06.overload;

public class MethodOverloadMosthMatch {
    public static void main(String[] args) {
        MathTools tools = new MathTools();

        System.out.println(tools.sum(1, 3));
        System.out.println(tools.sum(1, 3, 8));
        System.out.println(tools.sum(1.7, 2.5));
        System.out.println(tools.sum(1.7, 2));
        System.out.println(tools.sum(1, 2.5));
    }
}
```

5.8 可变参数

在JDK1.5之后，如果我们定义一个方法时，此时某个形参的类型可以确定，但是形参的个数不确定，那么我们可以使用可变参数。

格式：

```
【修饰符】 返回值类型 方法名(【非可变参数部分的形参列表,】参数类型... 形参名){  }
```

注意：

（1）一个方法最多只能有一个可变参数

（2）如果一个方法包含可变参数，那么可变参数必须是形参列表的最后一个

1. **示例一：　求n个整数的和**

```
public class Demo {
    public static void main(String[] args) {
        int[] arr = { 1, 4, 62, 431, 2 };
        int sum1 = getSum1(arr);
        System.out.println(sum1);

        int sum2 = getSum2(arr);
        System.out.println(sum2);
        int sum3 = getSum2(1, 4, 62, 431, 2);
        System.out.println(sum3);
    }
}
```

```
public class MathTools{
    // 完成数组 所有元素的求和
    // 原始写法
    public  int getSum1(int[] arr) {
        int sum = 0;
        for (int i = 0; i < arr.length; i++) {
            sum += arr[i];
        }

        return sum;
    }

    // 可变参数写法
    public  int getSum2(int... arr) {
        int sum = 0;
        for (int i = 0; i < arr.length; i++) {
            sum += arr[i];
        }
        return sum;
    }
}
```

2. **示例二：求1-n个整数中的最大值**

```
public class MathTools {
    public  int max(int num, int... others){
        int max = num;
        for (int i = 0; i < others.length; i++) {
            if(max < others[i]){
                max = num;
            }
        }
        return max;
    }
}
```

3. **示例三：字符串拼接**

   n个字符串进行拼接，每一个字符串之间使用某字符进行分割，如果没有传入字符串，那么返回空字符串""

```
public class StringTools {
    String concat(char seperator, String... args){
        String str = "";
        for (int i = 0; i < args.length; i++) {
            if(i==0){
                str += args[i];
            }else{
                str += seperator + args[i];
            }
        }
        return str;
    }
}
public class StringToolsTest {
```

```
    public static void main(String[] args) {
        StringTools tools = new StringTools();

        System.out.println(tools.concat('-'));
        System.out.println(tools.concat('-',"hello"));
        System.out.println(tools.concat('-',"hello","world"));
        System.out.println(tools.concat('-',"hello","world","java"));
    }
}
```

4. **可变参数的方法重载问题**

   求n个整数的最大值

```
 //求n整数的最大值
    public int max(int... nums) {
        int max = nums[0];//如果没有传入整数，或者传入null，这句代码会报异常
        for (int i = 1; i < nums.length; i++) {
            if (nums[i] > max) {
                max = nums[i];
            }
        }
        return max;
    }

/*    //求n整数的最大值
    public int max(int[] nums){  //编译就报错，与(int... nums)无法区分
        int max = nums[0];//如果没有传入整数，或者传入null，这句代码会报异常
        for (int i = 1; i < nums.length; i++) {
            if(nums[i] > max){
                max = nums[i];
            }
        }
        return max;
    }*/

    //求n整数的最大值
    public int max(int first, int... nums) {  //当前类不报错，但是调用时会引起多个方法同时匹配
        int max = first;
        for (int i = 0; i < nums.length; i++) {
            if (nums[i] > max) {
                max = nums[i];
            }
        }
        return max;
    }
```

5.9 命令行参数（了解）

通过命令行给main方法的形参传递的实参称为命令行参数


```
public class TestCommandParam{
    //形参：String[] args
    public static void main(String[] args){
        System.out.println(args);
        System.out.println(args.length);

        for(int i=0; i<args.length; i++){
            System.out.println("第" + (i+1) + "个参数的值是：" + args[i]);
        }
    }
}
```

命令行操作：

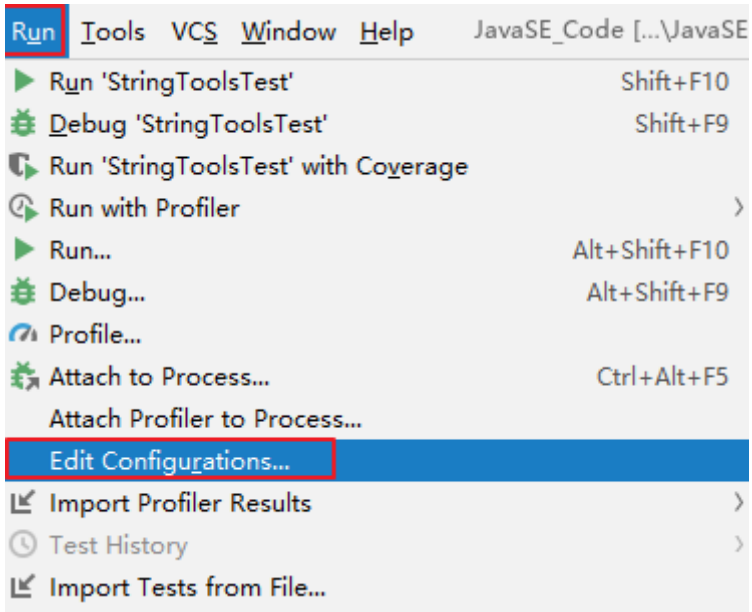
```
java TestCommandParam
```

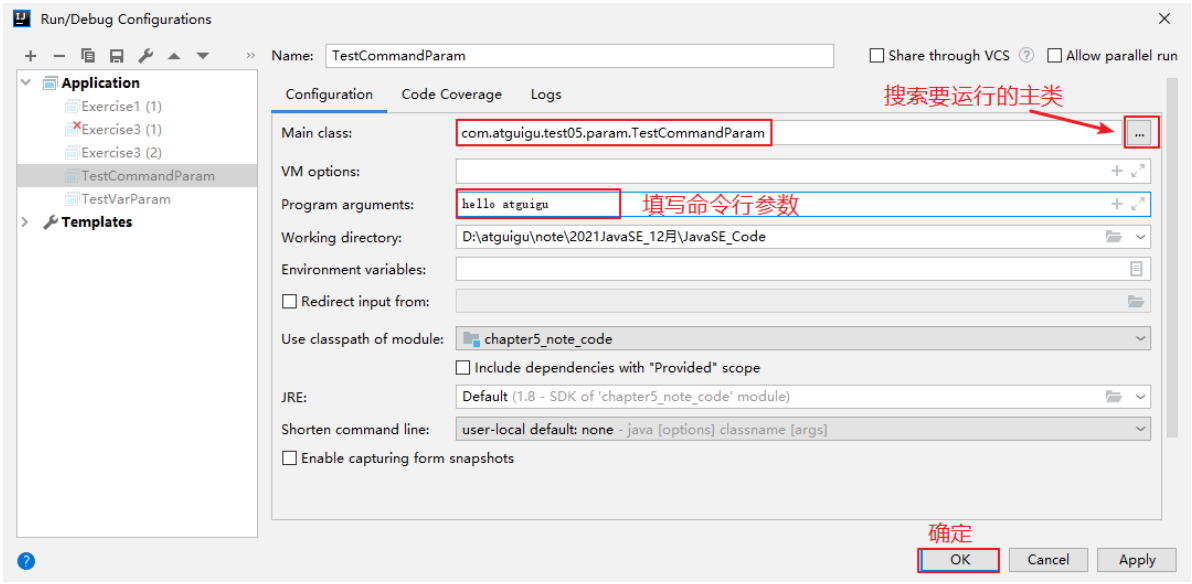
```
java TestCommandParam 1 2 3
```

```
java TestCommandParam hello atguigu
```

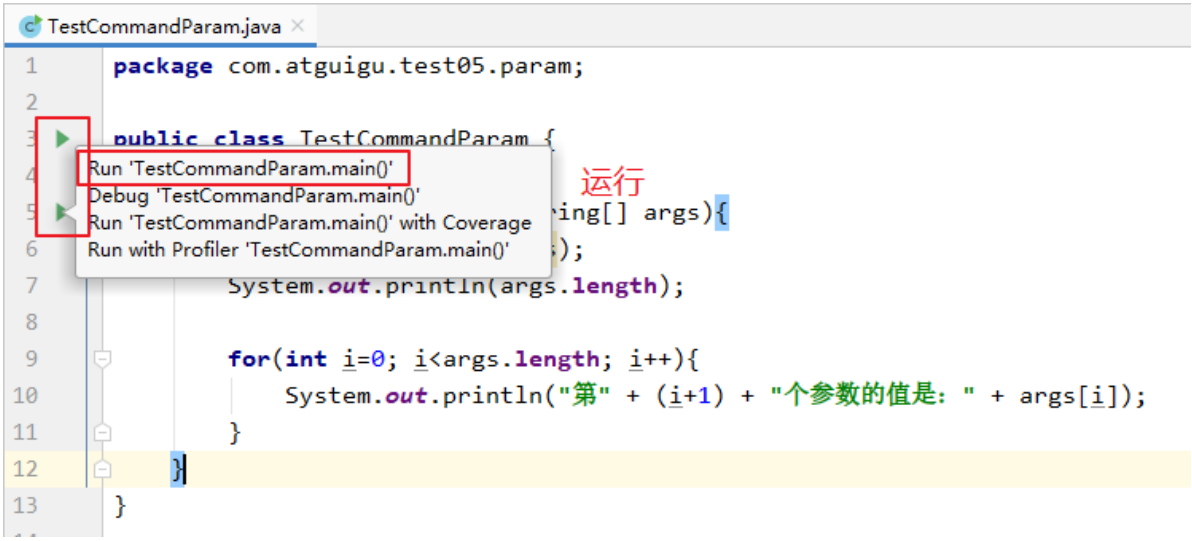
IDEA中操作：

（1）配置运行参数





（2）运行程序



5.10 递归 Recursion

- **递归：指在当前方法内调用自己的这种现象。**
- **递归的分类:递归分为两种，直接递归和间接递归。**
  - **直接递归称为方法自身调用自己。**
  - **间接递归可以A方法调用B方法，B方法调用C方法，C方法调用A方法。**
- **注意事项：**
  - **递归一定要有条件限定，保证递归能够停止下来，否则会发生栈内存溢出。(无穷递归，类似死循环)**
  - **在递归中虽然有限定条件，但是递归次数不能太多。否则也会发生栈内存溢出。**

1. **示例一：计算1-100之间所有自然数的和**

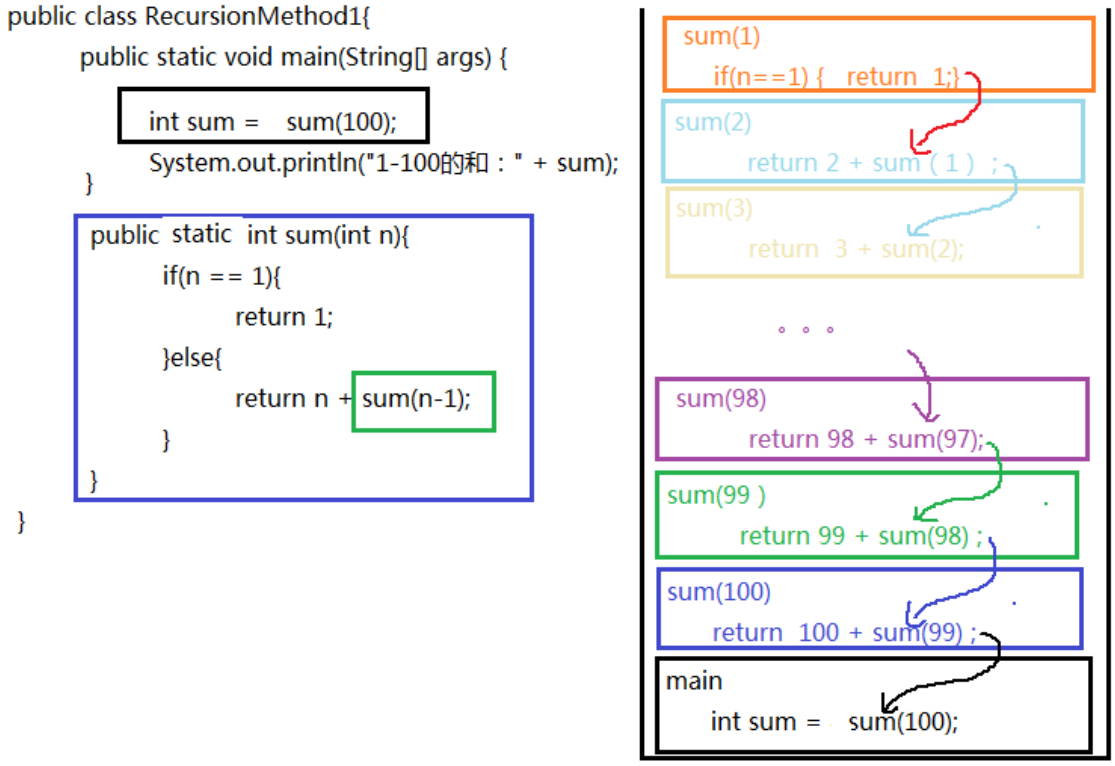
   循环实现：

```
public class RecursionMethod1{
    public static void main(String[] args) {
        int sum = sum(100);
        System.out.println("1-100的和：" + sum);
```

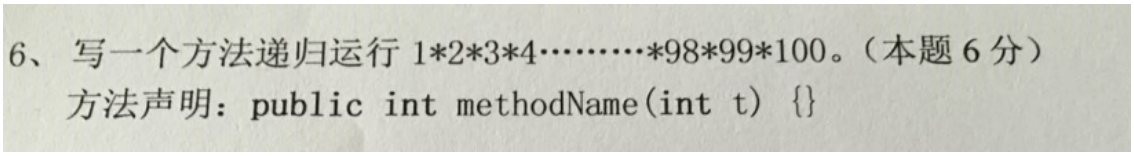
```
    }

    public static int sum(int n){
        int sum=0;
        for(int i=1;i<=n;i++){
            sum+=i;
        }
        return sum;
    }
}
```

递归实现：

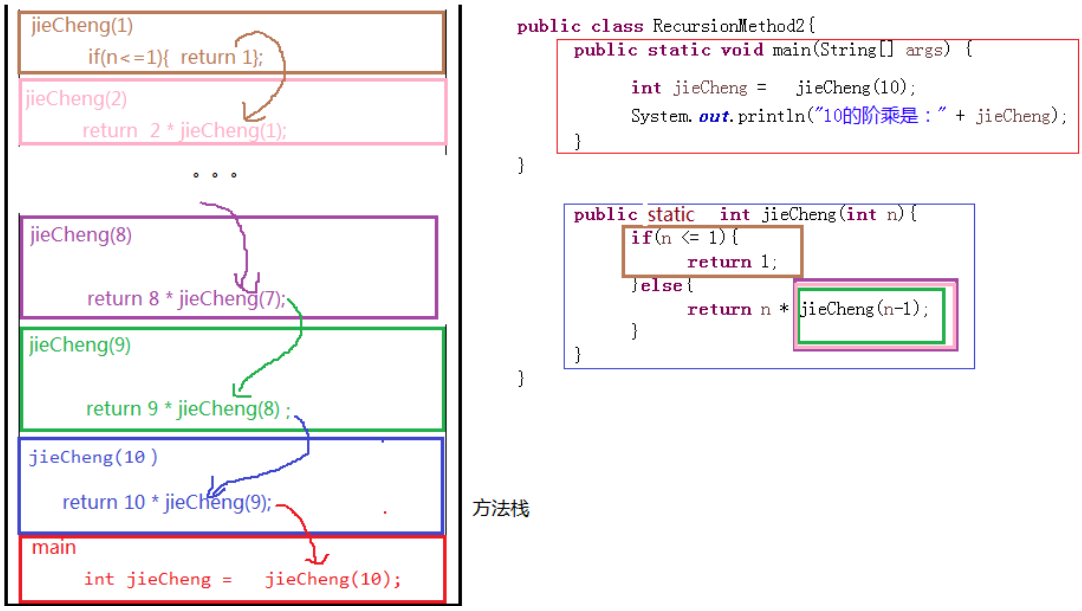
```
public class RecursionMethod1{
    public static void main(String[] args) {
        int sum = sum(100);
        System.out.println("1-100的和：" + sum);
    }

    public static int sum(int n){
        if(n == 1){
            return 1;
        }else{
            return n + sum(n-1);
        }
    }
}
```



2. **示例二：求n!**



```java
public class RecursionMethod2{
    public static void main(String[] args) {
        int jieCheng = jieCheng(10);
        System.out.println("10的阶乘是：" + jieCheng);
    }
    public static int jieCheng(int n){
        if(n <= 1){
            return 1;
        }else{
            return n * jieCheng(n-1);
        }
    }
}
```



3. **示例三：计算斐波那契数列（Fibonacci）的第n个值**

   规律：一个数等于前两个数之和，

   f(0) =1，

   f(1) = 1，

   f(2) = f(0) + f(1) =2，

   f(3) = f(1) + f(2) = 3，

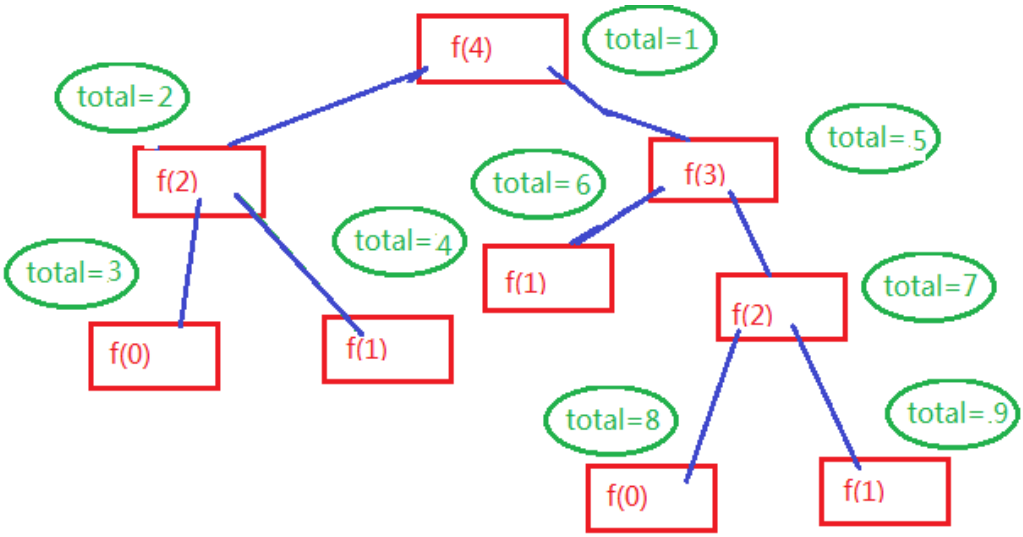
   f(4) = f(2) + f(3) = 5

   …

   f(n) = f(n-2) + f(n-1);

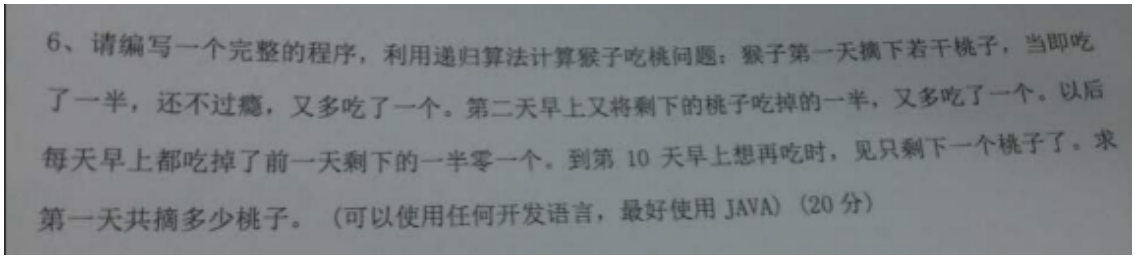
```java
public class RecursionMethod3{
    public static void main(String[] args) {
        Count c = new Count();

        System.out.println("f(10)：" + c.f(10));
        System.out.println("f方法被调用的总次数：" + c.total);
    }
}
class Count{
    int total = 0;
    public int f(int n){
```

```
            total++;
            if(n <= 1){
                return 1;
            }else{
                return f(n-2) + f(n-1);
            }
        }
    }
```



4. **练习**

描述：猴子吃桃子问题，猴子第一天摘下若干个桃子，当即吃了所有桃子的一半，还不过瘾，又多吃了一个。第二天又将仅剩下的桃子吃掉了一半，又多吃了一个。以后每天都吃了前一天剩下的一半多一个。到第十天，只剩下一个桃子。试求第一天共摘了多少桃子?


6、请编写一个完整的程序，利用递归算法计算猴子吃桃问题：猴子第一天摘下若干桃子，当即吃了一半，还不过瘾，又多吃了一个。第二天早上又将剩下的桃子吃掉的一半，又多吃了一个。以后每天早上都吃掉了前一天剩下的一半零一个。到第 10 天早上想再吃时，见只剩下一个桃子了。求第一天共摘多少桃子。（可以使用任何开发语言，最好使用 JAVA）（20 分）

5.11 对象数组

数组是用来存储一组数据的容器，一组基本数据类型的数据可以用数组装，那么一组对象也可以使用数组来装。

即数组的元素可以是基本数据类型，也可以是引用数据类型。当元素是引用数据类型时，我们称为对象数组。

> **注意：对象数组，首先要创建数组对象本身，即确定数组的长度，然后再创建每一个元素对象，如果不创建，数组的元素的默认值就是null，所以很容易出现空指针异常NullPointerException。**

1. **示例：**

（1）定义圆Circle类，包含radius半径属性，getArea()求面积方法，getPerimeter()求周长方法，String getInfo()返回圆对象的详细信息的方法

（2）在测试类中创建长度为5的Circle[]数组，用来装5个圆对象，并给5个圆对象的半径赋值为[1,10)的随机值

```
class Test16_ObjectArray{
    public static void main(String[] args){
        //要在数组中存储5个圆对象
        //声明一个可以用来存储圆对象的数组
        Circle[] arr = new Circle[5];
        //for(int i=0; i<arr.length; i++){
        //  System.out.println(arr[i]);
        //}
        //System.out.println(arr[0].radius);//NullPointerException

        //给元素赋值
        //元素的类型是：Circle，应该给它一个Circle的对象
        //arr[0] = 1.2;//错误的
        //arr[0]相当于它是一个Circle类型的变量，也是对象名，必须赋值为对象
        /*
        arr[0] =  new Circle();
        arr[0].radius = 1.2;
        System.out.println(arr[0].radius);
        */

        //创建5个对象，半径随机赋值为[1,10)的随机值
        //Math.random()==>[0,1)
        //Math.random()*9==>[0,9)
        //Math.random()*9+1==>[1,10)
        for(int i=0; i<arr.length; i++){
            arr[i] = new Circle();//有对象才有半径
            arr[i].radius = Math.random()*9+1;
        }


        //遍历显示圆对象的信息
        for(int i=0; i<arr.length; i++){
            //arr[i]是一个Circle的对象，就可以调用Circle类中的属性和方法
            System.out.println(arr[i].getInfo());
        }
    }
}
class Circle{
    double radius;
    public double getArea(){
        return 3.14 * radius * radius;
    }
    public double getPerimeter(){
        return 3.14 * 2 * radius;
    }
    public String getInfo(){
        return "半径：" + radius +"，面积：" + getArea() + "，周长：" +
getPerimeter();
    }
}
```

2. **对象数组的内存图分析**

3. **练习1**

   （1）定义学生类Student

声明姓名和成绩实例变量，

getInfo()方法：用于返回学生对象的信息

（2）测试类ObjectArrayTest的main中创建一个可以装3个学生对象的数组，并且按照学生成绩排序，显示学生信息

```
public class ObjectArrayTest {
    public static void main(String[] args) {
        Student[] arr = new Student[3];
        arr[0] = new Student();
        arr[0].name = "张三";
        arr[0].score = 89;

        arr[1] = new Student();
        arr[1].name = "李四";
        arr[1].score = 84;

        arr[2] = new Student();
        arr[2].name = "王五";
        arr[2].score = 85;

        for (int i = 1; i < arr.length; i++) {
            for (int j = 0; j < arr.length-1; j++) {
                if(arr[j].score > arr[j+1].score){
                    Student temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
                }
            }
        }

        for (int i = 0; i < arr.length; i++) {
            System.out.println(arr[i].getInfo());
        }
    }
}
class Student{
    String name;
    int score;
    public String getInfo(){
        return "姓名：" + name + ",成绩：" + score;
    }
}
```

```
class Test18_ObjectArrayExer2_2{
    public static void main(String[] args){
        //创建一个可以装3个学生对象的数组
        Student[] arr = new Student[3];//只是申明这个数组，可以用来装3个学生，此时里面没有学生对象

        //从键盘输入
        java.util.Scanner input = new java.util.Scanner(System.in);
        for(int i=0;i<arr.length; i++){
            System.out.println("请输入第" + (i+1) + "个学生信息：");
            arr[i] = new Student();
```

```
            System.out.print("姓名：");
            arr[i].name = input.next();

            System.out.print("成绩：");
            arr[i].score = input.nextInt();
        }

        //先显示一下目前的顺序
        for(int i=0; i<arr.length; i++){
            System.out.println(arr[i].getInfo());
        }

        System.out.println("----------------------------------------");
        //冒泡排序
        for(int i=1; i<arr.length; i++){
            for(int j=0; j<arr.length-i; j++){
                //arr[j] > arr[j+1]//错误的
                if(arr[j].score > arr[j+1].score){
                    //交换两个元素，这里是两个学生对象，所以temp也得是Student类型
                    Student temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
                }
            }
        }
        //再显示一下目前的顺序
        for(int i=0; i<arr.length; i++){
            System.out.println(arr[i].getInfo());
        }
    }
}
class Student{
    String name;
    int score;//使用int或double都可以

    public String getInfo(){
        return "姓名：" + name +"，成绩：" + score;
    }
}
```