1、什么是java?

java是一个既**面向对象**又**跨平台**的程序设计语言。

面向对象：

对象：万物皆对象，现实生活中的一个**实体。**是一个**具体的概念。**

类：是具有相同**属性**和**行为**的一组对象的集合。是一个**抽象的概念。**

属性： 描述对象的特征。

行为：描述对象的动作（功能）。

学生类：

张三同学 李四同学

属性：学号，姓名，年龄，成绩

行为：学习，吃饭，睡觉，玩

跨平台：

一次编写，多处运行。

可以理解为：**跨操作系统。**

2、java的特点

- 面向对象
  - 面向对象的特点：封装、继承、多态、（抽象）。
- 跨平台
- 简单
  - 去除了很多复杂的功能，比如说指针、运算符重载等。
- 安全性
- 多线程
  - j类似操作系统，在一个进程中可以有多个应用程序运行。

3、java的应用平台

JavaSE：

桌面应用程序，主要是java的基础

JavaEE：

企业级应用开发，比如说：淘宝、京东...

JavaME：主要是做手机端的应用程序

4、软件的分类：

C/S结构：

Client Service 客户端服务器，需要下载的程序，例如：微信，游戏。。。

B/S结构：

Browser Service 浏览器服务器，只需要一个浏览器，例如：淘宝。。。

5、JDK： java开发工具包

作用：提供了开发环境（可以编译，开发，运行）

1. bin目录：可执行的文件
2. lib目录：类库文件
3. jre目录：运行环境
    1. **JVM java虚拟机**
    2. java是运行在JVM上

> 注意：JDK， JRE，JVM 的关系?
>
> JDK 包含了 JRE, JRE 包含了JVM。

6、第一个应用程序

java中以类来描述对象的。

java文件以**.java为后缀。**

一般在新建项目的时候，我们把类进行分类，同一个功能的类放在同一个包下。

6.1 包

包的关键字：package ,用来存放文件。包名一般以域名的倒序写法。

例如：baidu.com ---> com baidu

zretc.com ---> com zretc

注意：多个单元之间用 点 分割，包名一般是小写的英文单词组成。

package 在代码的第一行，只能写一次。

6.2 代码

Demo.java

```java
package com.zretc;

// class 类的关键字，其后是类的名字，
//       驼峰命名法：类名首字母大写，其他的字母小写，如果有多个单词，首字母也要大写。例如：FirstDemo
// public 访问权限，范围；公共的，整个项目内都能访问到这个类（Demo）
// {} 作用域
public class Demo {
    // 程序的入口 ：  main()
    // 快捷键：psvm，是四个关键字的首字母
    public static void main(String[] args) {
        // 在控制台中打印：你好Java
        // 快捷键：sout
        // println() 打印在控制台上并换行
        // ln 换行
        // 不加ln 不换行
        System.out.print("你好Java!!!");
        System.out.println("你好Java!!!");
    }
}
// 整理代码格式 ：  ctrl+alt+l
```

7、变量

变量就是用来存储数据的一块内存。

变量在程序运行的过程中可以改变的量，称之为变量。

7.1 语法结构

语法结构：

① 先声明再赋值

数据类型 变量名；

变量 = 值；

例如：

```java
int a;
a = 10;

int a,b; // 同时声明多个变量，使用逗号分割
a = 10;
b = 20;
```

注意： a = 10，b=20； 错误写法

② 声明的同时直接初始化

数据类型 变量 = 值；

例如：

```java
int a = 10;

int a = 10,b = 20,c;
```

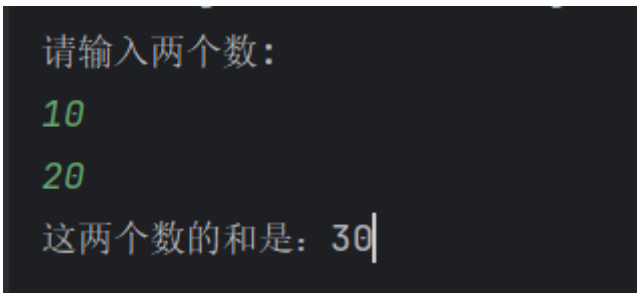
```
                c = 30;

// 注意： 在同一个作用域下不允许有同名的变量。
```

7.2 变量的命名规则：

1. 由字母、数字、下划线（_）、美元符($) 组成。
2. 不能以数字开头。
3. 不能使用关键字。
    1. 关键字：被java赋予了特殊含义的单词。例如：int package import public。。。
4. 区分大小写。
    1. 例如：ABC 和 abc 两个变量
5. 长度没有限制。

```
int ab123; 正确

int 12ab; 错误

int _12a; 正确

int 你好； 正确

int $ab%; 错误

int aabdhshdjashdjashdjsahdjahjdhsajd; 正确
```

7.3 练习




```
public class Demo4 {
    public static void main(String[] args) {
        System.out.println("请输入两个数:");
        Scanner sc = new Scanner(System.in);
        // 输入第一个数
        int num1 = sc.nextInt();
        // 输入第二数
        int num2 = sc.nextInt();
        // 输出这两个数的和
        System.out.println("这两个数的和是：" + (num1 + num2));
    }
}
```

```java
// 练习 ： 交换 n 和 m 两个变量中的值
        int n = 10;
        int m = 20;
        // 定义一个变量
        int temp = n; // 10
        n = m; // 20
        m = temp; // 10
        System.out.println("n = " + n);
        System.out.println("m = " + m);
```

8、数据类型

java中的数据类型分为两大类：**1. 基本数据类型　2. 引用数据类型**

- 基本数据类型（8个）
  - 整数型 （从小到大的顺序）
    - byte 字节型
      - -128~127 1字节 占8位二进制数
    - short 短整型
      - 2字节 16位二进制数
    - int 整型（默认的整数类型）
      - 4字节 32位
    - long 长整型， 需要再值的后面加L 或者 l
      - 8字节 64位
  - 浮点型（小数）
    - float 单精度浮点型
      - 4字节 32位
    - double 双精度浮点型（默认）
      - 8字节 64位
  - 字符型
    - char 标识符是一对单引号'',在单引号内有且仅有一个字符。
      - 2字节
      - char类型的前128个字符与ASICC码兼容 a --- 97， A --- 65
  - 布尔型 boolean
    - 值只有两个，true 正确的，成立的 ，false 错误的，不成立的
      - 1字节
- 引用数据类型
  - 除了8个基本数据类型以外的都是引用数据类型。
  - 大体分为： 类（class） / 接口（interface） / 数组（[]）
  - 常用的类：
    - String 字符串,标识符是一对双引号"",在双引号内可以有0个或多个字符。

```java
package com.zretc;
```

```java
import java.util.Scanner;

public class Demo1 {
    public static void main(String[] args) {
        // 8个基本数据类型
        // byte字节型 范围：-128~127
        byte b = 127;
        // 短整型
        short s = 128;
        // 默认的整型
        int i = 100;
        // 长整型,需要再值的后面加L 或者 l
        long lo = 10000000000L;

        // 浮点型
        // 单精度浮点型，需要在值的后面加F 或者 f
        float f = 10.1f;
        // 双精度浮点型 ,可选性的在值的后面加D 或者 d
        double d = 20.3D;
        double d2 = 20.3;

        // 字符型
        char c = '你';

        // 布尔型
        boolean bool = true;
        boolean bool2 = 3 < 2;

        // 引用数据类型：除了8种基本数据类型以外的都是引用数据类型
        // 大体分为： 类  接口  数组
        // 常用的类 ， 字符串String
        String str = ""; // 0个字符
        String str2 = "helloWorld";// 多个字符串

        // 举个例子：存储一个学生的信息，学号，姓名，年龄，性别，成绩
        String sId = "20210108";
        String sName = "tom";
        int age = 20;
        char sex = '男';
        double score = 92.5;
        System.out.println("学号：" + sId);
        System.out.println("姓名：" + sName);
        System.out.println("年龄：" + age);
        System.out.println("性别：" + sex);
        System.out.println("成绩：" + score);
    }
}
```

8.1 类型转换

- 自动类型转换（隐式）
    - 小转大
- 强制类型转换（显示）、

- 大转小

```
// 类型转换
        // 自动类型转换 ：  小 转 大
        int a = 10;
        // 将int转成long
        long k = a;

        // 强制类型转换 ：  大 转 小
        byte x = (byte)k;

        // char类型的前128个字符与ASICC码兼容
        // a --- 97
        // A --- 65
        char cc = 'b';
        int t = cc;
        System.out.println("t = " + t); // 98
        int t2 = 65;
        char cc2 = (char) t2;
        System.out.println("cc2 = " + cc2); // A
```

9、运算符

- 1、算术运算符
  - +-/(取整) %(余数)
  - ++ （加1） --（减1）
  - 当有多个操作时，需要考虑++/--在变量的前面还是后面：
    - 在变量前，先加1再运算
    - 在变量后，先运算再加1
- 2、赋值运算符
  - = += -= *= /= %=
  - 例子： +=
  - ```
    int a = 10;
    // a = a + 20;
    // += 操作
    a += 20;
    ```
  - 可以自动强制类型转换
- 3、比较运算符
  - 运算的结果是boolean类型的
  - > >= < <= != ==
- 4、逻辑运算符
  - 用来连接两个条件，可以理解为是true和false之间的比较，结果也是boolean类型的
  - && 逻辑与
    - 并且，两个条件的结果都为true，结果为true
  - || 逻辑或
    - 或者，两个条件只要有一个为true,结果就是true
    ```

    - ! 逻辑非，取反
        - !true --> false
    - 逻辑运算符的短路机制：
        - &&逻辑与：&&左边的条件结果为false,右边不执行
        - ||逻辑或：||左边的条件结果为true,右边不执行
- 5、条件运算符（三目运算符、三元运算符）
    - 表达式1 ? 表达式2 ： 表达式3
    - 表达式1 一定是一个条件，结果为boolean类型
    - 如果表达式1的结果为true，执行表达式2，为false，执行表达式3

```
package com.zretc;

public class Demo2 {
    public static void main(String[] args) {
        // 算术运算符 + - * /(取整) %(余数)
        int a = 10;
        int b = 8;
        System.out.println(a + b);
        System.out.println(a - b);
        System.out.println(a * b);
        System.out.println(a / b); // 整数与整数相除时，取整
        System.out.println(a % b);

        System.out.println(a / 8.0);// 1.25

        // ++（自增1）  --（自减1）
        a++;
        System.out.println("a = " + a); // 11
        ++a;
        System.out.println("a = " + a); // 12
        // 当有多个操作时，需要考虑++/--在变量的前面还是后面：
        //  在变量前，先加1再运算
        //  在变量后，先运算再加1
        System.out.println("a = " + ++a); // a=13
        System.out.println("a = " + a++); // a=13
        System.out.println(a); // 14


        // 赋值运算符; = += -= *= /= %=
        // 执行顺序：  从右至左
        int c = 10;
        // c = c + 20;
        // += 操作
        c += 20;
        System.out.println("c = " + c);

        byte by = 5;
//        by = (byte) (by + 20);
        // += 可以自动强制类型转换
        by += 20;
        System.out.println("by = " + by);

        // 比较运算符,运算的结果是boolean类型的
        System.out.println(3 > 2); // true
```

```
        System.out.println(3 >= 2); // true
        System.out.println(3 < 2); // false
        System.out.println(3 <= 2); // false
        System.out.println(3 == 2); // false
        System.out.println(3 != 2); // true

        // 逻辑运算符
        // && 逻辑与，并且
        // 两个条件结果都成立，结果为true
        System.out.println(true && true);  // true
        System.out.println(true && false); // false
        System.out.println(false && true); // false
        System.out.println(false && false); // false

        // || 逻辑或，并且
        // 两个条件只要有一个成立，结果就是true
        System.out.println(true || true);  // true
        System.out.println(true || false); // true
        System.out.println(false || true); // true
        System.out.println(false || false); // false

        // !  逻辑非，取反
        System.out.println(!true); // false
        System.out.println(!(3<2)); // true


        // 逻辑运算符的短路机制：
        //    &&逻辑与：&&左边的条件结果为false,右边不执行
        //    ||逻辑或：||左边的条件结果为true,右边不执行

        int n = 10;
        int m = 11;
        boolean bool = n++ > 10 && --m == n;
        System.out.println("bool = " + bool); // false
        System.out.println("n = " + n); // 11
        System.out.println("m = " + m); // 11

        // 条件运算符
        int x = 20;
        int y = 30;
        // 比较大小
        int max = x > y ? x : y;
        System.out.println("最大数是: " + max);

    }
}
```

10、流程控制语句

10.1 条件语句

- if 语句
- if else 语句

- if -- else if -- else 语句

```
if 两个分支
 if(条件1){
      条件1为true时，执行的代码块;
 }else if(条件2){
      条件2为true时，执行的代码块;
 }else{
      条件1和条件2都为fasle时，执行的代码块;
 }
```

- switch case语句

```
switch case:拿同一个变量与多个不同的常量进行比较是否相等的应用程序。
switch(变量/表达式){
    case 常量1:
        执行的代码;
        [break;]
    case 常量2:
        执行的代码;
        [break;]
    case 常量3:
        执行的代码;
        [break;]
    ...
    [default:
        执行的代码;
        break;
    ]
}

注意：  1、变量的类型必须是: byte / short / int / char / String
       2、常量是一个固定的值，不能是一个范围。
       3、break：结束程序块。如果不加break,程序顺序执行。
       4、default ：  当所有的case都不匹配时，执行该代码块的内容。

       注意：[] 中的内容可写可不写，不是必须的。
```

10.2 循环

- for循环
- while循环
- do while循环

- 三种循环的区别：
  - for循环：已知循环次数的时候使用for
  - while 循环：不知道循环次数时使用，先判断再执行。
  - do while循环：不知道循环次数时使用，先执行再判断，不管条件成立与否至少执行一次。

```
for循环：
                for(;;){} 死循环

                for(初始化;循环条件;迭代语句){
                    循环体；
                }

                1、初始化：初始化循环的次数的变量
                2、循环条件：当循环条件为true时，执行循环体，为false时，循环结束。
                3、迭代语句：在适当的时候结束循环，防止出现死循环
                4、循环体：反复执行的代码
                5、执行顺序：初始化、循环条件、循环体、迭代语句、循环条件、循环体、迭代语
句......
```

```
while循环：
                [初始化；]
                while(循环条件){
                    循环体；
                     [迭代语句；]
                }

                执行顺序：先判断循环条件是否为true，为true执行循环体，为false循环结束。
```

```
do while循环：
                [初始化；]
                do{
                    循环体；
                    [迭代语句；]
                }while(循环条件);

                执行顺序：一上来不管循环条件是否为true，先执行循环体，
                        再判断循环条件，循环条件为true执行循环体，为false循环结束。
                简单来说：先执行，再判断，不管条件成立与否至少执行一次。
```

10.3 循环的关键字

- break : 用来结束 switch...case，for , while , do...while语句的。
  再循环中，如果出现嵌套循环，那么只能结束当前所在层的循环。
- continue : 跳过当前此次循环，直接执行下一次循环。

```java
// 练习：输入数字，直到输入-1循环结束
        Scanner sc = new Scanner(System.in);
        int sum = 0; // 接收和
        while(true){
            System.out.println("请输入一个数：");
            int n = sc.nextInt();
            // 判断输入的数是否为-1，是-1就结束循环
            if(n == -1){
                break;
            }
            sum += n;
        }
        System.out.println("输入的数字总和为：" + sum);
```

```java
/*
            嵌套循环
            九九乘法表：
                1*1=1
                1*2=2 2*2=4
                1*3=3 2*3=6 3*3=9
                。。。
         */
        // 行
        for (int i = 1; i <= 9 ; i++) {
            // 列
            for (int j = 1; j <= i ; j++) {
                // \代表的是转义字符
                // \t 制表符  \n 换行  \r 回车  \" 一个“
                System.out.print(j + "*" + i + "=" + j*i + "\t");
            }
            // 换行
            System.out.println();
        }

        /*
            *
            * *
            * * *
            * * * *
            * * * * *
         */
        for (int i = 1; i <= 5 ; i++) {
            for (int j = 1; j <= i ; j++) {
                System.out.print("* ");
            }
            System.out.println();
        }
```

11、数组

用来存储一组具有**相同数据类型**的元素的集合。

11.1 创建数组

创建数组的语法结构：

```
    1、 数据类型[] 数组名 = new 数据类型[长度];
        数据类型 数组名[] = new 数据类型[长度];
        int[] arr = new int[5];

    2、 数据类型 数组名[] = new 数据类型[]{元素1，元素2，元素3};
        int[] arr = new int[]{1,2,3,4,5};

    3、 数据类型 数组名[] = {元素1，元素2，元素3};
        int[] arr = {1,2,3,4,5};
```

存储数据的语法结构：

```
    数组名[下标] = 值；

    arr[0] = 1;// 第一个元素
    arr[1] = 2;// 第二个元素
    arr[4] = 5;// 第五个元素
```

取数据的语法结构：

```
    System.out.println(数组名[下标]);
    System.out.println("第五个元素是：" + arr[4]);
```

注意：

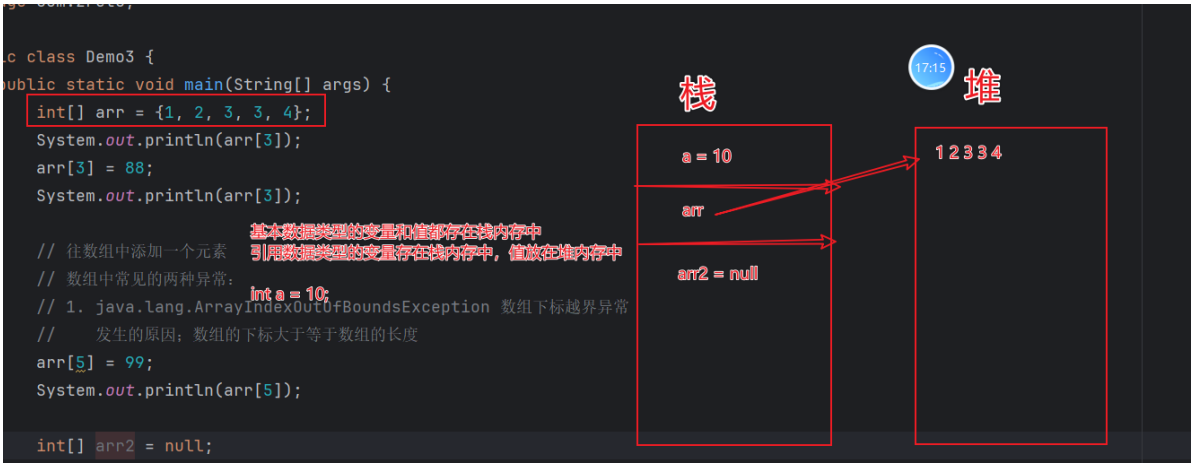
```
    1、创建数组时必须指定数组的长度。
    2、数组的长度可以为0，但是不能为负数。
    3、数组中元素的数据类型既可以是基本数据类型也可以是引用数据类型。
    4、下标，从 0 开始的，最大下标是数组的长度的减1。
    5、数组中的元素有默认值：
        byte short int long  ---  0
        float double --- 0.0
        char --- 空字符
        boolean --- false
        String --- null  (引用数据类型的默认值都是null)
    6、获取数组的长度：数组名.length
```

数组的遍历：将数组中的元素全部取出来。

```
    for (int i = 0;i < 数组名.length;i++){
        System.out.println(数组名[i]);
    }
```

foreach 遍历：专用于遍历数组，集合的。

```
    for(元素类型 变量 ： 数组/集合){
        System.out.println(变量);
    }
```



11.2 数组中常见的两种异常

数组中常见的两种异常：

1. java.lang.ArrayIndexOutOfBoundsException 数组下标越界异常

   发生的原因：数组的下标大于等于数组的长度

```
int[] arr = new int[5];
arr[5] = 99;
System.out.println(arr[5]);
```

2. java.lang.NullPointerException 空指针异常

   发生的原因：引用没有指向对象（没有对象）

```
int[] arr2 = null;
System.out.println(arr2[0]);
```

11.3 数组中的排序

1. 冒泡排序：

   两个相邻的元素比较大小，交换位置。

   从小到大的顺序：

   45 76 38 65 90 15 27

i		少比的次数
0	45 38 65 76 15 27 90	0
1	38 45 65 15 27 76	1
2	38 45 15 27 65	2
3	38 15 27 45	
4	15 27 38	
5	15 27	

   循环的次数：数组的长度减1

```
int[] arr = {45, 76, 38, 65, 90, 15, 27};
        // 冒泡：两个相邻的元素比较大小，交换位置
        for (int i = 0; i < arr.length - 1; i++) {
            // 循环比较
            for (int j = 0; j < arr.length - 1 - i; j++) {
                if (arr[j] > arr[j + 1]) {
                    // 交换位置
```

```
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }

        // 遍历数组
        for (int i : arr) {
            System.out.println(i);
        }
```

二维数组的创建方式：

```
    1、int[][] arr = new int[5][5];

    2、int arr[][] = new int[][]{{1,2},{3},{4,5}};

    3、int[] arr[] = {{1,2,3},{3,4}};

    4、int[][] arr = new int[5][];
        arr[1] = new int[3];
```

遍历：

```
        // 二维数组的元素个数是 3
        int[][] arr = {{1, 2, 3}, {4, 5}, {6, 7, 8, 9}};
        // 将数组中的元素取出来
        for(int i = 0;i < arr.length;i++){
            for (int j = 0; j < arr[i].length; j++) {
                System.out.println(arr[i][j]);
            }
        }

        for(int[] i : arr){
            for(int j : i){
                System.out.println(j);
            }
        }
```

12、类和方法

类 ： 具有相同属性和行为的一组对象的集合。

语法结构：
```
    访问权限 class 类名{
        // 属性 ： 描述对象的特征

        // 行为 ： 描述对象的动作（功能）

    }
```

属性 ： 在java当中是以变量的形式定义的。
行为 ： 在java当中是以方法（函数）的形式定义的。

12.1 属性

变量：
    成员变量，局部变量

成员变量与局部变量的区别：

1. 定义的位置：
        成员变量定义在类的里面，方法的外面。
        局部变量定义在方法中的变量。

2.作用域：
        成员变量在整个类中都能访问。
        局部变量只有在最近的一对{}中的有效。

3.有无默认值：
        成员变量有默认值，默认值与数组的默认值相同
        局部变量没有默认值

4.生命周期
        成员变量是随着对象的创建而产生，随着对象的释放而释放。
        局部变量是随着所在的方法被执行而产生，执行结束而释放。

12.2 方法

方法：
    参数：有参方法，无参方法
    返回值：有返回值，无返回值

语法结构：
    1）参数
        参考的是小括号内是否有变量

        修饰符 void 方法名(){ // 无参方法
            方法体；
        }

```
        修饰符 void 方法名(数据类型 变量1，数据类型 变量2){ // 有参方法
            方法体；
        }

    2）返回值
        参考的是void 和 return这两个关键字
        void ：无返回值的
        return :
            1.返回函数（将return后的值返回给调用者）
            2.结束函数

        注意： 有返回值的方法必须加return关键字，
               并且return后的值的类型必须与方法声明时的返回值类型一致。

        修饰符 void 方法名(){ // 无返回值的方法
            方法体；
        }

        修饰符 void 方法名(){ // 无返回值的方法
            方法体；
            return; // 结束函数
        }

        修饰符 返回值类型 方法名(数据类型 变量1){ // 无返回值的方法
            方法体；
            return 值； // 返回函数（将return后的值返回给调用者）
        }


调用方法：
    1、创建对象
        类名 对象名 = new 类名（）；
    2、根据对象来调用方法
        对象名.方法名（）； // 无返回值的，无参的
        对象名.方法名（实参1，实参2）； // 无返回值的，有参的
        返回值类型 变量 = 对象名.方法名（实参）；// 有返回值的，有参的

参数：
    实参：调用方法时传递的值
    形参：方法声明时的参数
    注意：实参和形参要保证个数、顺序、类型必须一致。
```

13、面向对象编程思想

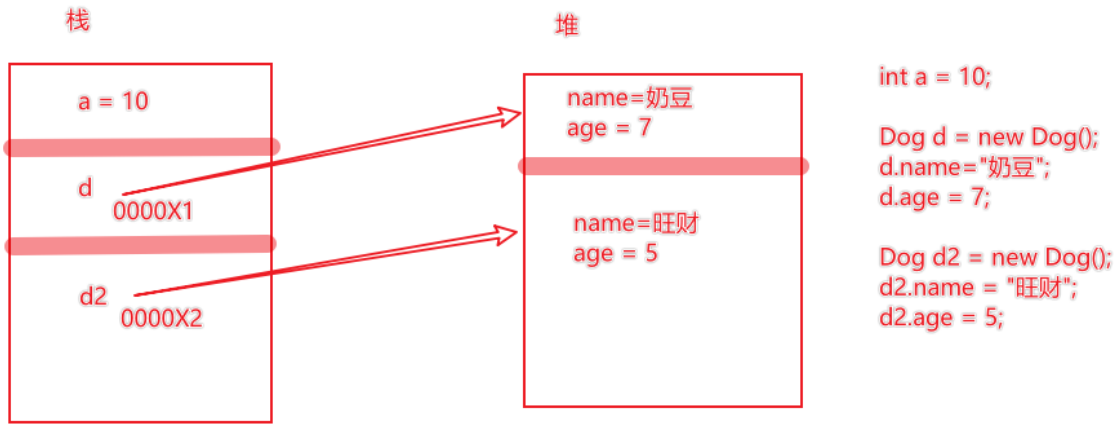
```
面向过程：就是对项目功能的需求进行一步一步分析的过程。
面向对象：就是在面向过程的基础上抽离出对象，对对象进行描述的过程。

淘宝：
    注册--登录--搜索--加入购物车--下单--确认订单--发货--派送--确认收货

    买家：
        注册--登录--搜索--加入购物车--下单--确认收货
    卖家：
        注册--登录--确认订单--发货
```

```
    快递员：
        注册--登录--派送

面向对象的三大特征：
    封装、继承、多态、（抽象）
```

```
public class Dog { // 狗类
    // 属性
    String type; // 品种
    String name;// 名字
    int age;//年龄
    String sex;//性别

    // 行为
    public void show(){ // 自我介绍
        System.out.println("品种：" + type + ",名字：" + name +
                ",年龄：" + age + ",性别：" + sex);
    }

    public void eat(String food){
        System.out.println(name + "喜欢吃" + food);
    }
}

public class DogMain {
    public static void main(String[] args) {
        // 1.创建对象
        Dog d1 = new Dog();
        // 2.访问对象的属性和行为
        d1.type = "泰迪";
        d1.name = "旺财";
        d1.age = 3;
        d1.sex = "男";
        d1.eat("骨头");
        d1.show();
    }
}
```

14、值传递

```
在java中有两种数据类型：
    基本数据类型的变量中存储的就是实实在在的值。
    引用数据类型的变量中存储的是指向堆内存中对象的地址的值。
```

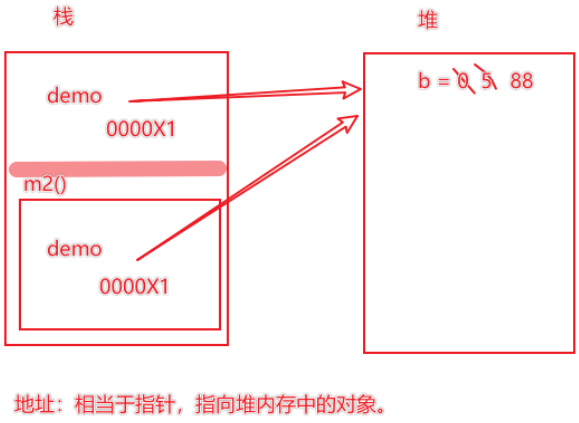


值传递：

当基本数据类型的变量作为参数传递时，传递的就是实实在在的值；

当引用数据类型的变量作为参数传递时，传递的是地址的值，

当两个引用的地址相同时，指向的是同一个对象，一个发生改变，另一个也改变。



地址：相当于指针，指向堆内存中的对象。

```
public class Demo4 {
   int b;//成员变量，属于对象
   public static void main(String[] args) {
      int a = 10;
      m1(a);

      Demo4 demo = new Demo4();
      demo.b = 5;
      m2(demo);
       System.out.println(demo.b); // 88
   }

   public static void m1(int a){ // a=10
      System.out.println(a);
   }

   public static void m2(Demo4 demo){
      System.out.println(demo.b); // 5
      demo.b = 88;
      System.out.println(demo.b); // 88
   }
}
```

15、static关键字

static 是一个修饰符：

可以修饰成员方法、成员变量、成员代码块{}

static 修饰的成员是属于类的，当加载类的时候，就存在了。

static修饰的成员变量被所有对象共享。

static 修饰的成员访问方式：

1、类名调用

2、对象调用（但是没有意义，staitc修饰的成员属于类，不属于某一个对象。）

3、静态可以直接访问静态，

但是不能直接访问非静态，因为静态方法中没有对象。

16、构造方法

```
语法结构：
    修饰符 类名(){

    }

什么是构造方法：
    1、方法名与类名相同
    2、没有返回值也不需要写void关键字

构造方法的作用：
    1、给成员变量初始化
    2、创建对象

构造方法的特点：
    1、系统会为每一个类都提供一个默认无参的构造方法。
    2、如果手动写了一个有参构造方法，提供的默认无参构造方法失效。如果还想用这个默认无参构造方法，需要手动添加。


this代表的是当前对象：
this关键字的作用：
    1. 调用自身的成员变量和成员方法
    2. 区分成员变量和局部变量
    3. 构造方法之间相互调用
       注意：必须在有效代码的第一行调用
```

17、方法的重载

在**同一个类中**有两个或两个以上**方法名相同**，**参数列表不同**的方法，就够成了重载；

参数列表不同：个数、顺序、类型不同；

注意：重载与访问权限修饰符和有无返回值类型没有关系；

```java
public class Demo5 {

    public Demo5(String a){

    }

    public Demo5(){}
    public static void main(String[] args) {
        Demo5 d1 = new Demo5();
        Demo5 d2 = new Demo5("aa");
    }

    public void sum(int a,int b){}

    public void sum(double a,double b){}

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

```
    }
}
```

18、封装

将类的某些信息**隐藏在类的内部**，不允许外部随意访问和修改，而是通过该类提供的**公共的get/set**方法来访问私有成员。

封装的步骤：

1. 将属性私有化
2. 定义公共的get set方法访问这个私有的属性

访问权限：（从小到大的顺序）

- private 私有的，在本类中有效
- 默认的，在本包中有效
- protected 受保护的，在本包+外包子类中有效
- public 公共的，本包+外包（整个项目中）

19、继承

类与类之间的一种关系，子类继承父类，子类自动共享父类中的所有**非**私有成员。

继承的关键字： **extends**

```
访问权限 class 子类 extents 父类{
    类体；
}

注意： java中的继承是单继承。
       单继承： 一个子类只能有一个直接的父类。
       java中所有的类都继承自Object. Object是根类。


继承关系中成员的特点：
    成员变量：
        子类和父类出现同名的变量时：
                使用this引用本类中的成员
                使用super引用父类中的成员
                super 代表当前对象，引用的是继承父类中的成员
    构造方法：
        1.子类的构造方法默认会通过super()调用父类中的无参构造方法。
        2.如果父类中没有无参只有有参，子类中所有的构造方法都需要手动调用这个有参构造方法并初始化。

    成员方法：
        1、当子类中的方法，方法名，参数列表，返回值类型与父类中的方法相同时，构成了重写（覆盖）。
        2、子类中的方法访问权限必须大于等于父类中的方法访问权限。
        3、使用super关键字保留父类中原有的功能。
```

4、父类方法是静态的，子类重写也得是静态的。

注意：重写时，通过子类对象访问方法时，执行的一定是子类中的方法，因为子类中的方法功能更丰富更强大。

20、final 最终的

final 代表的是最终的，可以用来修饰类，方法变量：
修饰的类，叫最终类，不能被继承。
修饰的方法，叫最终方法，不能被重写，但是可以被继承。
修饰的变量，叫常量，只能赋值一次。

常量的命名规则：大写字母组成的,多个单词用下划线分割

21、抽象

模糊、不具体、笼统。
抽象的关键字 ： abstract
可以用来修饰类、方法；
修饰的类，叫抽象类
修饰的方法，叫抽象方法

抽象类：

```
public abstract class 类名{

}
```

抽象方法：

```
public abstract 返回值类型 方法名（）；
```

注意： 抽象方法只有方法的声明，没有方法的实现（方法体），以分号结束；

抽象的特点：
1、当一个类中有抽象方法了，那么这个类一定是抽象类。
2、抽象类不能实例化（new)。
3、抽象类一定是一个父类。
4、子类继承抽象类，要么重写抽象类中的所有抽象方法，要么就继续抽象下去。

22、多态

是指面向对象程序运行中，相同的信息发送给多个不同类型的对象，根据不同各类型的对象，调用给对应类型的方法，而有不同的行为。

简单来说：某一事物的多种存在形态。
在代码当中体现形式是，父类或接口的引用指向子类对象。

多态的好处：
    提高代码的扩展性，父类可以访问子类重写的内容

多态的弊端：
    父类引用不能访问子类的特有内容

多态：实际上就是将子类对象向上转型（父类形态）或者向下转型（本类形态）

注意：向下转型时，有可能发生java.lang.ClassCastException 类型转换异常
    解决办法：使用 instanceof关键字
            对象 instanceof 类型
            判断左边的对象是否属于右边的类型，属于返回true,否则返回false

多态时成员的特点：
    成员变量：
        简单来说：编译看左边，运行看左边

    成员方法：
        简单来说：编译看左边，运行看右边

23、接口

接口可以理解为比抽象更抽象
是一些列方法的声明，没有方法具体实现的一个集合。

```
public interface 接口名{
    常量;
    抽象方法;
}
```

接口中的成员：
    1、常量 ：
            默认被public static final 修饰
            访问方式：接口名.常量
                    子实现类.常量
                    实现类对象调用（没有意义）
    2、抽象方法 ：
            默认被public abstract修饰

实现类的语法结构：
```
    public class 类名 extends 类 implements 接口1，接口2{

    }
```

注意： 类与类之间是单继承
        类与接口之间是多实现
        接口不能实例化
        接口没有构造方法

24、内部类

在一个类的内部还有一个类，里面的这个类就是内部类。

```
修饰符 class 外部类{
    修饰符 class 内部类{

    }
}
```

成员内部类：

1、定义在类的里面，方法的外面，与成员变量、成员方法并列存在。
2、内部类可以访问外部类的成员。
3、外部类访问内部类中的内容，需要创建内部类对象，通过对象调用。
4、创建对象的语法结构：

```
外部类.内部类 对象名 = new 外部类().new 内部类();
OuterClass.InnerClass inner = new OuterClass().new InnerClass();
```

静态内部类：

1、静态内部类就是在成员内部类的前面加static关键字。
2、静态内部类可以直接访问外部类中的静态成员，但是不能直接访问非静态的成员。
3、创建静态内部类语法结构：

```
外部类.内部类 对象名 = new 外部类.内部类();
OuterClass2.InnerClass inner2 = new OuterClass2.InnerClass();
```

局部内部类;

1、定义在方法的里面，只能被该方法访问。
2、不能被访问权限与static关键字修饰。

匿名内部类：

1、没有名字的内部类，可以理解为是一个匿名的子类对象。
2、语法结构：

```
父类/接口 对象 = new 父类/接口(){
    子类内容;
};
```

3、匿名内部类实现的前提：要么继承一个类，要么实现一个接口。

25、常用类

25.1 String 字符串

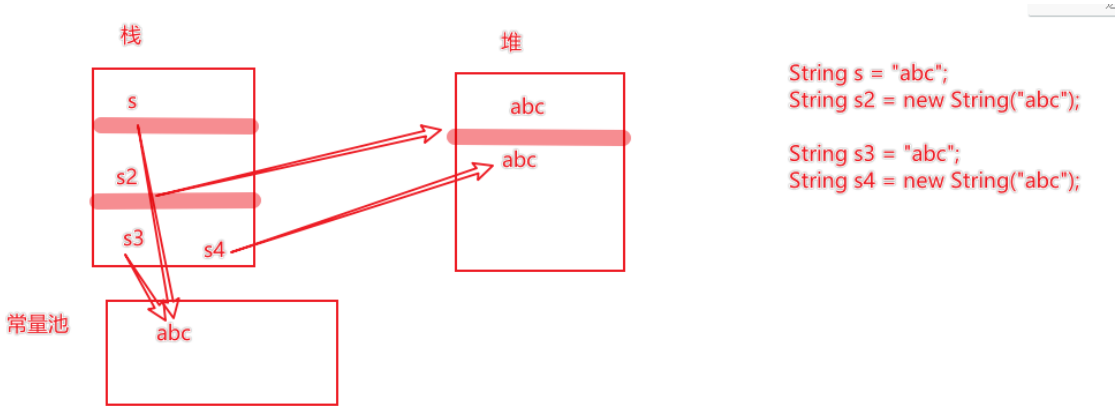
java.lang包下的，被final修饰，是一个常量，一旦赋值不能修改。

标识符：一对“”，里面可以有0~多个字符。

创建方式：

```
1、String s = "abc";
2、String s = new String("abc");
```

注意：使用=直接赋值的字符串，是一个常量，值存储在常量池中。只能赋值一次。
使用new关键创建的字符串，值存储在堆内存中。



常用方法：

charAt(int index)	根据索引检索字符
String concat(String str)	字符串拼接，相当于 + 号
boolean contains(CharSequence s)	判断当前字符串中是否包含指定的字符，包含返回true
endsWith(String suffix)	判断当前字符串是否以指定的字符结尾，是，返回true
startsWith(String suffix)	判断当前字符串是否以指定的字符开头，是，返回true
equals(Object anObject)	判断两个字符串是否相等，相等返回true
equalsIgnoreCase(String anotherString)	判断两个字符串是否相等，不区分大小写。相等返回true
int indexOf(String str)	检索指定字符所在的位置，不存在返回-1
int lastIndexOf(String str)	检索指定字符最后一次出现的位置，不存在返回-1
int length()	字符串的长度
String replace(char oldChar, char newChar)	替换
String[] split(String regex)	按照指定的字符，进行切割，返回一个数组
String substring(int beginIndex)	截取字符串
substring(int beginIndex, int endIndex)	截取字符串
char[] toCharArray()	转换成字符数组
String toLowerCase()	转换成小写字母
String toUpperCase()	转换成大写字母
trim()	去掉两端空白字符
String valueOf(boolean b)	将任意类型的数据转成字符串

```
== 和 equals():
        == 用来比较基本数据类型的值是否相等
            引用数据类型的地址是否相等
        equals() :
            比较内容是否相等
```

25.2 可变字符串

```
StringBuffer 和 StringBuilder 可变字符串，被 final 修饰。
    StringBuffer ，线程安全的，执行效率低。
    StringBuilder，线程不安全的，执行效率高。
```

常用方法：

append(boolean b)	追加
delete(int start, int end)	删除
deleteCharAt(int index)	删除指定位置的字符串
insert(int offset, boolean b)	插入
reverse()	反转

25.3 Math

```
Math 类包含用于执行基本数学运算的方法，如初等指数、对数、平方根和三角函数。
被 final 修饰。
该类的属性和方法都被 static 修饰，可以直接通过类名调用。

常用的方法：
    double ceil(double a) 向上取整
    double floor(double a) 向下取整

    max(int a, int b)  返回两个 int 值中较大的一个。
    min(int a, int b)  返回两个 int 值中较小的一个。
    round(double a)  四舍五入
    random()  随机数， double 值，该值大于等于 0.0 且小于 1.0。
```

字段摘要		
static double	E	比任何其他值都更接近 e（即自然对数的底数）的 double 值。
static double	PI	比任何其他值都更接近 pi（即圆的周长与直径之比）的 double 值。

25.4 Date 日期

```
构造方法：
    Date()
    Date(long date)
```

after(Date when)	测试此日期是否在指定日期之后。
before(Date when)	测试此日期是否在指定日期之前。
compareTo(Date anotherDate)	比较两个日期的顺序。 大于指定日期，返回正整数，小于，返回负整数，等于，返回0
long getTime()	毫秒

25.5 包装类

基本数据类型对应的引用数据类型

```
byte      ---     Byte
short     ---     Short
int       ---     Integer
long      ---     Long
float     ---     Float
double    ---     Double
char      ---     Character
boolean   ---     Boolean
```

```java
package com.zretc2;

public class Demo2 {
    public static void main(String[] args) {
//          包装类：基本数据类型对应的引用数据类型
        System.out.println("byte最大值:" + Byte.MAX_VALUE); // 127
        System.out.println("byte最小值:" + Byte.MIN_VALUE); // -128
        System.out.println("byte的二进制位数:" + Byte.SIZE); // 8

        // parseInt(String s)  将字符串转成int
        String s = "123";
        System.out.println(Integer.parseInt(s) + 8);

        // 打包：把基本数据类型转换成引用数据类型
        int i = 10;
        Integer in = i; // 直接打包
        // 拆包 ：  把引用数据类型转化成基本数据类型
        int a = in;
    }
}
```

26、异常

异常的根类：
    Throwable 类是 Java 语言中所有错误或异常的超类。
        -- Error : 错误
            用于指示合理的应用程序不应该试图捕获的严重问题，一般情况下解决不了。
        -- Exception : 异常
            指出了合理的应用程序想要捕获的条件，一般发生了可以解决的。

Exception异常分为两大类：
    1、编译时异常
        Exception 和 除了 RuntimeException 及其子类以外的都是编译时异常。
        编写代码时发生的异常
    2、运行时异常
        RuntimeException 以及他的子类都是运行时异常
        程序运行过程中抛出的异常

异常的解决方式：
    1、抛出异常
        throw : 定义在方法里，抛出的是异常对象
        throws : 定义在方法声明时，抛出的是异常类
    2、捕获异常
        try{
            可能发生异常的代码；
        }catch(异常类 对象名){
            解决异常的代码；
        }finally{
            用来释放资源，一定被执行的代码；
        }

        组合情况：
            1. try .. catch .. finally
            2. try .. catch
            3. try .. finlly

注意：
    1、catch 可以有多个，但是父类catch异常要放在最下面
    2、编译时异常一旦发生，必须处理，要么捕获，要么抛出。
        可以解决的话，就捕获，解决不了，就抛出，让调用者解决。
    3、运行时异常一旦发生，可选择性的处理。

```java
package com.zretc3;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

public class Demo1 {
    public static void main(String[] args) {
//        int[] arr = null;
//        // java.lang.NullPointerException : 空指针异常，没有对象
//        System.out.println(arr[0]);
```

```
        // java.lang.ArrayIndexOutOfBoundsException 数组下标越界
//        int[] arr = {1, 2, 3, 4, 5};
//        System.out.println(arr[5]);

        // java.lang.ArithmeticException 算数异常
//        double i = 10 / 0;
//        double i = 10 / 0.0;
//        System.out.println(i); // Infinity

        // 算出 距离2025-1-21 ， 还有多少天放假
        String str = "2025-1-21";
        // 将字符串转成 日期
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        try {
            Date date = sdf.parse(str); // ParseException 解析式异常
            System.out.println(date);

            Date date2 = new Date(); // 系统的当前日期
            // 将日期转成long类型的毫秒单位的时间
            long time1 = date.getTime();
            long time2 = date2.getTime();

            System.out.println((time1 - time2) / (24 * 60 * 60 * 1000));
        } catch (ParseException e) {
            throw new RuntimeException(e);
        }
    }
}
```

26.1 自定义异常

```
自定义异常需要继承 Exception 或者是 RuntimeException

语法结构：
    public class 类名 extentds Exception/RuntimeException{

    }

异常的注意事项：
    1、子类继承父类，重写父类方法时，如果父类方法抛出异常，子类方法可抛可不抛
       如果抛出异常，可以抛出父类异常或父类异常的子类
    2、如果父类方法没有抛出异常，子类也不能抛出异常
```

27、IO流

IO流，输入流和输出流，可以理解为就是对文件的**读和写**两个操作；

27.1 文件File类

文件和目录路径名的抽象表示形式。

`File` 类的实例是不可变的；也就是说，一旦创建，`File` 对象表示的抽象路径名将永不改变。

separator	**盘符**
File(String pathname)	构造方法
isDirectory()	是否是一个目录，存在且是一个目录时返回true
list()	返回一个数组，存储目录中的文件和目录
mkdir()	创建目录，如果父目录不存在，则不能创建目录
mkdirs()	创建目录，父目录不存在时，连同父目录一起创建
isFile()	判断是否是一个文件，存在且是一个文件返回true
createNewFile()	创建文件
getName()	文件或目录的名称
exists()	判断文件或目录是否存在
delete()	删除文件或目录，如果该目录下有子目录或文件的情况下，删除失败

27.2 文件的读写

```
IO操作，按照流向分为：输入流和输出流
       按照单位分为：  字节流和字符流

输入流(读)：
    程序从文件中读取数据
输出流（写）：
    程序向文件中写入数据

输入流：
    字节流: InputStream
             -- FileInputStream
             -- BufferedInputStream
    字符流: Reader
             -- FileReader
             -- BufferedReader

输出流：
    字节流: OutputStream
             -- FileOutputStream
             -- BufferedOutputStream
    字符流: Writer
             -- FileWriter
             -- BufferedWriter

以上都是输入流和输出流的根类，都被abstract修饰。
```

27.2.1 字节流

```
FileInputStream(File file)  读
```

read(byte[] b)	读取数据
close()	关闭流

```
public class Demo3 {
    public static void main(String[] args) {
        // 输入流：  读取文件中的数据
        try {
            FileInputStream f = new FileInputStream("D:/a.txt");
            // 可变字符串
            StringBuilder builder = new StringBuilder();
            byte[] b = new byte[1024];// 1024 字节等价于 1kb
            // 读
            int len = f.read(b); // 调用read() 返回读取数据的长度,如果读到流末尾返回-1
            while(len > 0){
                builder.append(new String(b,0,len));
                // 继续读取
                len = f.read(b);
            }
            System.out.println(builder);
            // 关闭流
            f.close();
        } catch (FileNotFoundException e) { // 找不到文件异常
            throw new RuntimeException(e);
        } catch (IOException e) { // IO异常
            throw new RuntimeException(e);
        }
    }
}
```

FileOutputStream(File file,boolean append)	写入的构造方法 第二个参数true,追加，默认覆盖
write(byte[] b)	写
flush()	刷新
close()	关闭流

```
package com.zretc;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

public class Demo4 {
    public static void main(String[] args) {
//        字节流的写入 ：  往文件中写入数据
```

```
//          创建输出流
        try {
            FileOutputStream f = new FileOutputStream("D:/a.txt");
            // 写
            // getBytes() 将字符串转成字节数组
            f.write("哈哈哈，马上下课".getBytes());
            // 刷新
            f.flush();
            // 关闭
            f.close();
        } catch (FileNotFoundException e) {
            throw new RuntimeException(e);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
}
```

练习：将D:/a.txt中的文件复制到E:/a.txt文件中

```
package com.zretc;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

public class Demo5 {
    public static void main(String[] args) {
        try {
            // 输入流
            FileInputStream fis = new FileInputStream("D:/a.txt");
            // 输出流
            FileOutputStream fos = new FileOutputStream("E:/a.txt",true); // 如果
文件不存在，自动创建,true追加，默认是覆盖
            // 一边读一边写
            byte[] b = new byte[1024];
            int len = fis.read(b);
            while(len > 0){
                // 将读取到的数据写入到E盘中
                fos.write(new String(b,0,len).getBytes());
                // 继续读
                len = fis.read(b);
            }
            // 刷新
            fos.flush();
            // 关闭
            fos.close();
            fis.close();
        } catch (FileNotFoundException e) {
            throw new RuntimeException(e);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
```

```
    }
}
```

27.2.2 字符流

FileReader(File file)	读的构造方法
read(char[] c)	读
close()	关闭流

FileWriter(File file,boolean bool)	写的构造方法
write(String str)	写
close()	关闭
flush()	刷新

```
package com.zretc;

import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class Demo6 {
    public static void main(String[] args) {
        try {
            FileReader fr = new FileReader("D:/a.txt");
            FileWriter fw = new FileWriter("E:/a.txt");

            char[] c = new char[1024];
            int len = fr.read(c);
            while(len > 0){
                fw.write(c,0,len);
                len = fr.read(c);
            }
            fw.flush();
            fw.close();
            fr.close();
        } catch (FileNotFoundException e) {
            throw new RuntimeException(e);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
}
```

27.2.3 带有缓冲区的读写

字节流中的读和写：

```
package com.zretc;

import java.io.*;

public class Demo7 {
    public static void main(String[] args) {
        try {
            BufferedInputStream bis = new BufferedInputStream(new
FileInputStream("D:/a.txt"));
            BufferedOutputStream bos = new BufferedOutputStream(new
FileOutputStream("E:/a.txt"));
            byte[] b = new byte[1024];
            int len = bis.read(b);
            while(len > 0){
                bos.write(b,0,len);
                len = bis.read(b);
            }
            bos.flush();
            bos.close();
            bis.close();
        } catch (FileNotFoundException e) {
            throw new RuntimeException(e);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
}
```

字符流中的读和写：

BufferedReader：

String readLine() 读取一行，已到达流末尾，则返回 null

BufferedWriter：

newLine() 写入一个行分隔符。

```
package com.zretc;

import java.io.*;

public class Demo8 {
    public static void main(String[] args) {
        try {
            // 读
```

```
            BufferedReader br = new BufferedReader(new FileReader("D:/a.txt"));
            String s = br.readLine();
            while (s != null) {
                System.out.println(s);
                // 继续读取
                s = br.readLine();
            }
            // 关闭流
            br.close();

            // 写
            BufferedWriter bw = new BufferedWriter(new FileWriter("D:/a.txt"));
            bw.write("锄禾日当午，");
            // 换行
            bw.newLine();
            bw.write("汗滴禾下土。");
            bw.flush();
            bw.close();
        } catch (FileNotFoundException e) {
            throw new RuntimeException(e);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
}
```

28、集合

集合是专用于存储对象的。

集合和数组的区别：

1、数组既可以存储基本类型也可以存储引用类型。
集合只能存储引用类型

2、数组存储的元素个数是固定的。
集合的长度是没有限制的。

集合：

单列集合：存储一个元素
双列集合：存储一对元素

28.1 单列集合Collection

Collection 是单列集合的**跟接口。**

Collection的子集合：

- Set 集合： 无序不重复
- List 集合： 有序允许重复
- 有序：存与取的顺序是一致的，存：1,2,3　取：1,2,3
- 重复：集合中允许有相同的元素。 1，2,3,1,2

add(E e)	添加一个元素
addAll(Collection<? extends E> c)	将指定 collection 中的所有元素都添加到此 collection 中
remove(Object o)	删除一个元素
removeAll(Collection<?> c)	删除与指定集合中相同的元素（保留差集）
clear()	清空
contains(Object o)	判断是否包含指定的元素，包含返回true
containsAll(Collection<?> c)	是否包含指定集合中所有的元素，包含返回true
isEmpty()	判断集合是否为空，为空返回true
retainAll(Collection<?> c)	删除差集部分（求交集）
size()	集合的大小
toArray()	将集合转成数组
Iterator iterator()	迭代器（将数组中的元素存放到迭代器中）
hasNext()	判断迭代器中是否有下一个元素，有，有返回true
next()	获取迭代器中的下一个元素
remove()	移除迭代器返回的最后一个元素

28.1.1 Set集合

- Set集合是无序，不重复的
- Set集合的子集合
  - **HashSet**
    - 内部数据结构是哈希表，是不同步的。许使用 `null` 元素。
    - 什么是哈希表?
      - 是用来判断是否有相同元素的。
      - 1. 调用 hashCode(),判断两个对象的哈希值是否相同
        2. 如果哈希值相同的，调用equals()在判断内容是否相同
    - **LinkedHashSet**
      - 内部数据结构是链表+哈希表，不同步的
      - 有序，不重复
  - **TreeSet**
    - 内部数据结构是红黑树，是不同步的。按照自然顺序（字典顺序）排序。
    - 自定义的类不具备可比较性：
      - 1、实现 **Comparable** 接口，重写 compareTo(T o)，让自定义的类具备可比较性。

28.1.2 List 集合

- List集合是有序，允许重复的。
- List集合的子集合
  - ArrayList
    - 内部数据结构是数组，查询快，增删慢，不同步。
  - LinkedList
    - 内部数据结构是链表，增删快，查询慢，不同步。
  - Vector
    - 内部数据结构是数组，增删和查询都慢，同步。

add(int index, E element)	**在指定的位置添加元素**
addAll(int index, Collection<? extends E> c)	在指定的位置添加另一个集合中的所有元素
remove(int index)	删除指定位置的元素
get(int index)	获取指定位置的元素
set(int index, E element)	替换指定位置的元素
subList(int fromIndex, int toIndex)	截取一个子集合
indexOf(Object o)	检索指定元素第一次出现的位置
lastIndexOf(Object o)	检索指定元素最后一次出现的位置

LinkedList的特头的方法：

addFirst(E e)、offerFirst(E e)	**添加到开头**
addLast(E e)、offerLast(E e)	添加到末尾
getFirst()、peekFirst()	获取第一个
getLast()、peekLast()	获取最后一个元素
removeFirst()、pollFirst()	删除第一个
removeLast()、pollLast()	删除最后一个

28.2 双列集合 Map

- **接口 Map<K,V>**
- **类型参数：**

  `K` - 此映射所维护的键的类型

  `V` - 映射值的类型

  键是唯一的，不允许重复

- Map集合的子集合
  - HashMap
    - 内部数据结构是哈希表，不同步，允许有NULL值和NULL键
    - LinkedHashMap
      - 内部数据结构是链表+哈希表，不同步
      - 有序，不重复
  - HashTable
    - 内部数据结构是哈希表，同步，不允许有NULL值和NULL键
  - TreeMap
    - 内部数据结构是红黑树，是不同步的。按照自然顺序（字典顺序）排序。
    - 自定义的类不具备可比较性：
      - 1、实现 **Comparable** 接口，重写 compareTo(T o)，让自定义的类具备可比较性。

put(K key, V value) **putAll(Map<? extends K,? extends V> m)**	**添加**
remove(Object key)	删除
get(Object key)	根据键获取值
containsKey(Object key)	是否包含指定键
containsValue(Object value)	是否包含指定的值
isEmpty()	判断是否为空，为空返回true
clear()	清空集合
Set<Map.Entry<K,V>> **entrySet()**	将键值对（映射关系）存储到set
Set **keySet()**	将键存储到Set集合中
Collection values()	将值存储到Collection集合中

29、反射

反射的作用： 反编译

通过反射访问 Java 对象的，属性(`Field`)，方法(`Method`)，构造方法(`Constructor`)等。

获取 Class 类型对象的三种方式：

- **对象.getClass()** ： 返回此 `Object` 的运行时类。
- **类名.class**
- **Class.forName()**

30、设计模式

单例设计模式：保证一个类的对象在内存中的唯一性。

简单来说，就是这个类只能创建一个对象。

单例设计模式的两种写法：

1. **饿汉模式**
2. **懒汉模式**

饿汉模式，实现步骤：

- 1、私有化构造方法
- 2、创建一个私有的、静态的本类对象
- 3、定义一个公共的、静态的方法，访问这个本类对象

```
// 单例设计模式：  保证一个类的对象在内存中的唯一性
public class Single {
    // 2. 创建私有的，静态的本类对象
    private static Single s = new Single();
    // 1. 私有化构造方法
    private Single(){}

    // 3. 定义公共的静态的方法来访问本类对象
    public static Single getInstance(){
        return s;
    }
}

public class MainTest {
    public static void main(String[] args) {
//        Single s1 = new Single();
//        Single s2 = new Single();
        Single s1 = Single.getInstance();
        Single s2 = Single.getInstance();
        System.out.println(s1 == s2); // true
    }
}
```

```
// 懒汉模式
public class Single2 {
    // 2. 声明一个私有的，静态的本类对象
    private static Single2 s;
    // 1. 私有化构造方法
    private Single2(){}

    // 3. 定义公共的、静态的方法访问本类对象
    public static Single2 getInstance(){
        // 第一次访问该方法时创建对象
        if(s == null){
            s = new Single2();
        }
```

```
        return s;
    }
}
```