Java

4.19:

在数组中关于 **System.arraycopy()** 和 **Arrays.copyOf()** 的应用

1. 删除数组中的一个元素

```
/**
* 删除数组元素
*
* @param arr   需要删除元素的数组
* @param index 需要删除元素的下标
* @return 返回删除元素之后的数组
*/
private static String[] delElement(String[] arr, int index) {
    // 剪切
    System.arraycopy(arr, index + 1, arr, index, arr.length - 1 - index);
    // 缩容
    arr = Arrays.copyOf(arr, arr.length - 1);
    return arr;
}
```

2. 添加一个元素到数组中

```
private static String[] addElement(String[] arr, String element) {
    // 扩容
    arr = Arrays.copyOf(arr, arr.length + 1);
    // 末位追加数据
    arr[arr.length - 1] = element;
    return arr;
 }
```

3. 关于字符串的方法:

```
// 以字符串 str 符开始,返回布尔值
string.startsWith(String str);
// 以字符串 str 结束,返回布尔值
string.endWith(String str)
// 包含字符串 str,返回布尔值
string.contains(String str)
// 分割字符,返回一个字符数组
string.split()
```

4.20

1. 补全字符串长度:

```
// 表示左对齐 并设定了字符串最少占据的字节长度
// 当str不足指定数字的字节长度时用空格补足
String.format("%-数字s",str)
```

2. 将数组以["数据 1","数据 2","数据 3"]的格式输出

```
int[]numbers={1,2,3,4};
Arrays.toString(numbers);
>>[1,2,3,4]
```

4.21

1. 拼接字符串:

```
"abc".contact("def");
>>"abcdef"
// "+" 的链接操作会不断调用new StringBuilder.
// new StringBuilder()--> append() --> toString --> ... -->
// --> new StringBuilder() --> append() --> toString()
// 会浪费大量内存空间
"abc"+"def";
>>"abcdef"
```

2. Stream 流遍历数组:

    - 打印:

```
Stream.of(数组).foreach(System.out::println);
Arrays.stream(数组).foreach(System.out::println)
```

- 获取数值

```
Stream.of(数组).foreach( e -> System.out.println("元素" + e);
```

3. compareTo() 与 equals() 与 ==

```
String str1 = "123";
String str2 = new String("123");
String str3 = "1234";
String str4 = "123";

System.out.println(str1.compareTo(str2));
>>0
System.out.println(str1.compareTo(str3));
>>-1
System.out.println(str1.equals(str2));
>>true
System.out.println(str1 == str2);
>>false
System.out.println(str1 == str4);
>>true
```

4. 获取子字符串

```
// 获取子字符串,从beginIndex(包括)到endIndex(不包括)结束
// string.subString(beginIndex, endIndex)
"123456".subString(1,3);
>>"23"
```

4.22

1. Integer.parseInt(str)与 Integer.valueOf(str)

```
Integer.parseInt(str) 将字符串转换成基本类型int
Integer.parseInt(str) 自动拆箱,所以可以用 ==,来判断

Integer.valueOf(str) 将字符串转换成Integer类型
Integer.valueOf(str) 可以直接对转换的对象调用Integer里面的方法,若数字在 -128~127之间,会直接取缓存,超过则创建新的对象
```

用integer1.equals(integer2) 来判断是否相等

4.23

1. 知道数组长度,但每次都在数组后面添加元素,可以不用定义初始长度再找下标的方法,可以利用每次数组扩容的方式在数组最后面添加元素

```
public static void queryCountInfoByGender() {
    String[] genders = {};
    int[] counts = {};
    int count = 0;
    boolean exist = false;
    for (String person : people) {
        String perGender = person.split("-")[2];
        for (String gender : genders) {
            if (perGender.equals(gender)) {
                exist = true;
                break;
            }
        }
        if (!exist) {
            genders = addElement(genders, perGender);
        }
        exist = false;
    }
    // 用要统计的信息在数据中遍历
    for (String gender : genders) {
        for (String person : people) {
            if (person.split("-")[2].equals(gender)) {
                count++;
            }
        }
        counts = Arrays.copyOf(counts, counts.length + 1);
        counts[counts.length - 1] = count;
        count = 0;
    }

    System.out.println(Arrays.toString(genders));
    System.out.println(Arrays.toString(counts));
}
```

2. 将字符串转化为字符的方法

```
"string".charAt(0);
>>'s'
```

3. 位运算:^异或:相同为 0,不同为 1

```
System.out.println(5 ^ 2);
// 5: 101
// 2: 010
// x: 111
>>7
```

4. 静态块:静态代码块主要用于类的初始化。它只执行一次，并且在同属于一个类的 main 函数之前执行。

   - 静态代码块会在类被加载时自动执行。
   - 静态代码块只能定义在类里面，不能定义在方法里面。

```
class Test01 {
    static {
        System.out.println("1234");
    }
    public static void main(String[] args){
    }
}
>>"1234"
```

   - 静态代码块里的变量都是局部变量，只在块内有效。
   - 一个类中可以定义多个静态代码块，按顺序执行。
   - 静态代码块只能访问类的静态成员，而不允许访问实例成员。

```
class Test02 {
   // 成员变量(类比C的"全局变量")
    static int a = 0;
    static {
        // 局部变量
        a = 0;
        // 定义int整型变量b,只能在静态块中使用,无法被静态块外部调用
        int b = 1;
    }
}
```

   - 一个类可以使用不包含在任何方法体中的静态代码块，当类被载入时，静态代码块被执行，且只被执行一次，静态块常用来执行类属性的初始化.

```
class Test03 {
   static {
       System.out.println("run!");
   }
```

```
}

class Test04 {
    public static void main(String[] args) {
       new Test03();
    }
}
>>"run!"
```

5. 关于类中的变量

```
    class Demo {
        public int id;
        public String name;
        @Override
        public String toString() {
            // 每次调用同String都会使id自增1;
            id ++;
            return id + " ";
        }
    }
    Demo demo = new Demo();
    // 这个操作也会让id自增.
    System.ouy.println(demo.id++);
    // 更有,在对成员变量进行字符串补空格的操作如果也是
    demo.name = String.format("%-12s", demo.name);
    // 上述操作也会改变引用中成员变量的数值,str会被补足.
    //然后将补足后的字符串赋值给原字符串demo.name.
```

4.24

1. 什么是类?

   把某一类有相同特征(属性)和行为(方法)抽象出来的概念,叫做类.类是不占用内存的.

2. 什么是对象

   表示在现实生活中存在的事物.Java 中通过 new 关键字来创建一个对象.对象占用内存.

3. 面向对象的三大特征:**封装,继承,多态**

- 封装:将类的信息隐藏到内部.
  - 访问修饰符:public , protected , default , private
    - private: 本类.

    - default: 本类,同包
    - protected: 本类,同包,子类
    - public: 任何地方都可使用
- getter/setter
- 构造方法
- 方法重载
    - 同一类
    - 方法名一致
    - 方法的参数列表不一样(类型/个数/顺序)

```
...
public void show(int num) {}
public void show(float f) {}
...
public void show(int num, String str) {}
public void show(String str, int num) {}
...
```

- 构造方法重载
    - 同一类
    - 方法名一致
    - 方法的参数列表不一样(类型/个数/顺序)
    - **方法的返回值对方法的重载无影响**

```
public Job01() {
    System.out.println("无参构造方法");
}

public Job01(int num) {
    System.out.println("int : " + num);
}
```

- **static** 关键字
    - 类在被加载时同时被加载.
    - 只会加载一次,并且内存中只有一块内存空间保存
    - 可以修饰类/方法

> 1. static 修饰的方法叫静态方法(类方法),该方法属于这个类,通过 *类名.静态方法名(实参);* 调用(用 *对象.方法名(实参)* 也能调用
> 2. 没有被 static 修饰的叫做成员方法,成员方法的调用方式为: *对象.方法名(实参)*

```
public class Job02 {
   static String name;
   int age;

public Job02(String name, int age) {
   this.name = name;
   this.age = age;
}

public static void main(String[] args) {
   Job02 j1 = new Job02("晓峰", 20);
   Job02 j2 = new Job02("圆圆", 16);

   System.out.println(j1.name);
   System.out.println(j1.age);
   System.out.println(j2.name);
   System.out.println(j2.age);
 }
}
>>圆圆
>>20
>>圆圆
>>16
```

- 继承
  - public class 类名 extends 父类 { }
  - 子类只能继承父类中非私有化的属性和方法
  - 一个子类只能有一个父类,一个父类可以有多个子类
  - 继承关系满足: xxx(子类) is a xxx(父类)
  - 一个 B 类继承 A 类:
    1. B 类是 A 类的子类(派生类)
    2. 类是 B 类的超类(基类,父类)