javase基础

一、基础数据类型

> 四类八种

- 整型： byte short int long（长整形）
- 浮点型： float（单精度） double （双精度）
- 布尔值： boolean
- 字符型： char

1、取值范围：

- 整型： byte -2^7 -> 2^7-1 10000000(-128) 其余的以此类推
- 浮点型： 符号位 + 阶码（指数） + 尾数
- char 一个字符连个字节 （unicode）

2、字符发展

ASCII（七位） -> ISO8859-1（单字节） -> unicode(两个字节) -> utf-8（可变长的字符集，英文 1个字节，中文3个字节）

GBK GB2312 中国的国标字符集

3、变量

> 变量的类型 变量的名字 = 具体的值;

- 定义和赋值两部分
- 他在内存里开辟了一片空间，可以存数据
- 他可以反复的到处使用
- 变量其实就是一个引用，他指向堆内存的一块区域，我们想操作一个对象只需要使用它的引用即可

4、标识符命名规则（名字）

- 可以用_和$但是不建议使用
- 可以用数字和字母，但是数字不能开头
- 不能使用关键字 super this class int double
- 所有的名字必须驼峰式命名，变量、方法首字母小写，类名首字母大写。静态常量全部大写多个单词用下划线分割。
- 尽量少用拼音，如果要用就整体都是拼音，不能拼音加因为，比如（isKong）

5、强制转化

整型里边小转大自动转，大转小需要强转，应为可能会丢失精度

整型和浮点型：浮点型转整型需要强转，反之不需要，但是他以科学计数法表示可能会丢失精度

char能不能和整型转： 就是两个字节，随便转，就能当成一个short。但是char类型的变量经过计算都是int。任何short,byte,int,char无论是逻辑运算还是数学运算结果都是int；

```
// 会保存，计算结果都是int
byte m = 1;
byte n = 3;
byte c = m + n;
```

二、运算符

1、逻辑运算符

> 逻辑运算真值表

与

条件1 condition1	条件2 condition2	结果
1	1	1
1	0	0
0	1	0
0	0	0

或

条件1 condition1	条件2 condition2	结果
1	1	1
1	0	1
0	1	1
0	0	0

非

条件1 condition1	结果
1	0
0	1

与： & 全部为真才是真，有一个是假就是假 串联电路 1 & 1 = 1； 0 & 0 = 0； 1& 0 = 0

或： | 全部为假才是假，有一个为真就是真 并联电路 1 | 1 = 1； 1 | 0 = 1； 0 | 0 = 0

非：！非真即使假，非假就是真

异或： ^ 相同为假，不同为真 1^1 = 0； 0^0 = 0； 1^0 = 1

双与（&&）：短路运算符， 前边为假直接返回假，后边的就不进行计算了 （常用）

双或 （||）：短路运算符， 前边为真直接返回真，后边的就不进行计算了（常用）

2、算术运算符

加 减 乘 除 取余（%）

3、赋值运算符

- =： 把后边的值赋给前边的变量
- ++： 自加一 arr[i++] 选运算，先把i给arr算，然后i=i+1； arr[++i] 选i=i+1，先把i给arr；arr[i+1] 就是一个值i不会变
- -- ： 自减一
- +=： i += 6 如同与 i= i+6
- -=： i -= 6 如同与 i= i-6

4、位移运算符

- >> 有符号右移 ,右移一位大致相当于什么除以2，除非溢出的末尾是零
- << 有符号左移，右移一位大致相当于什么乘以2，溢出的首位是零
- >>> 无符号右移

5、三目运算符

- 条件？ 结果一： 结果二；

三、流程控制语句

1、if语句

```java
if (i == 0){

} else {

}

if (i == 0){

}
if (i != 2){

}
```

2、switch

```java
// i可以是  byte short int String enum 的数据
switch (i){
    case 1:
        System.out.println();
    break;
    case 2:
        System.out.println();
    break;
    default:
       ystem.out.println();
    break;
}
```

3、while

```java
// 如果不是死循环，一定要相办法退出，退出机制
// 先判断满足条件才执行
boolean flag = true;
while (flag){

    if (如果达到某种条件){
        flag = false;
    }
}
// 死循环使用场景也挺多

do {} while (条件)    无论如何都要执行一次
```

4、for

```java
int i = 0;
for (; i < 10;){

     i++;
}
// 1、定义了一个变量，这个变量可以定义在任何地方，只用作用域能访问到即可，
//     如果i定义在了方法里，他的作用域就是整个方法。如果定义在分号处，就只能在for循环内使用
// 2、进入的条件，这个条件能很复杂 （i !=4 || i != 7）只要结果是true或者false即可
// 3、一个条件，用于退出，不写就是没有条件，他就无法退出。i++ 能不能写在for循环内部，可以。
for (;;)  死循环
```

5、几个关键字

（1）break

无论如何都要结束当前的全部循环，程序会继续向下执行。

（2）contiune

跳过本次循环继续下一次循环。

（3）打标签

for循环可已打标签，使用【break + 标签名】可以 退出 被打标签的循环

```java
flag:for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
        if( j > 0 ){
            break flag;
        }
        System.out.println("===="+j);
    }
}
```

四、数组

1、数组的定义

```java
int[] nums = {1,2,3};
int[] nums = new int[3];

类型[] 名字 = new 类型[长度];
```

java.lang.ArrayIndexOutOfBoundsException

2、数组的性质

（1）数组一旦建立不能改变长度。

（2）每个位置只能存一个值，多了会覆盖。

（3）编号从0开始，下标。

（4）他有个长度的属性，最后一个位置的编号是 长度-1。 0 - length-1

（5）数组里边可以是基本类型，也可以是引用类型。

3、数组的简单实用

（1）打擂台的形式找最大值

```
int[] salary = {4,5,0,6,7,8};
int maxIndex = 0 ;
for (int i = 1; i < salary.length; i++) {
    if(salary[i] > salary[maxIndex]){
        maxIndex = i;
    }
}
System.out.println("经过了n轮比赛得到的最大值得下标是："+maxIndex+"。值是："+salary[maxIndex]);

```

（2）通循环遍历打印数组的每一个值

```
for (int i = 0; i < salary.length; i++) {
    System.out.print(salary[i] +  " ");
}
```

（3）查找一个数组里存在的值。

```
int targetIndex = -1;
for (int i = 0; i < salary.length; i++) {
    if(salary[i] == 9){
        targetIndex = i;
    }
    break;
}
```

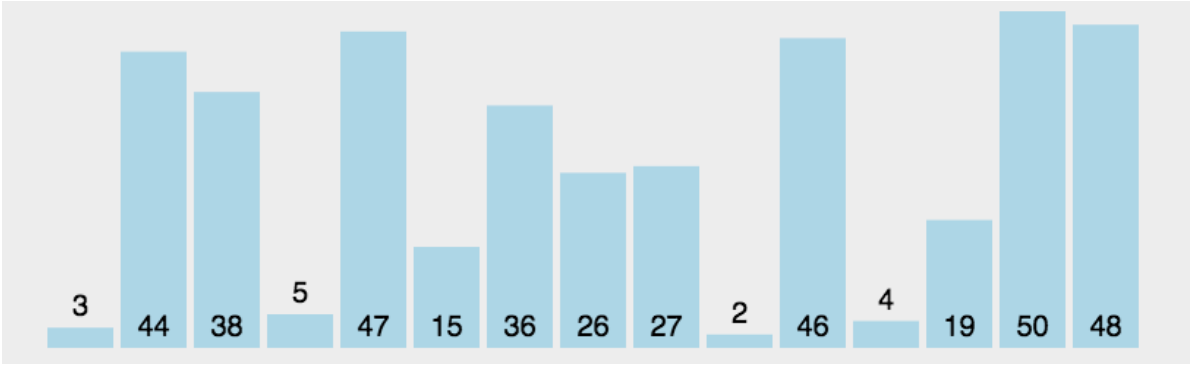
（4）元素的位移。

```
int[] salary = {4,5,0,6,7,8};

int temp = salary[0];
salary[0] = salary[1];
salary[1] = temp;
```

4、排序算法

（1）选择



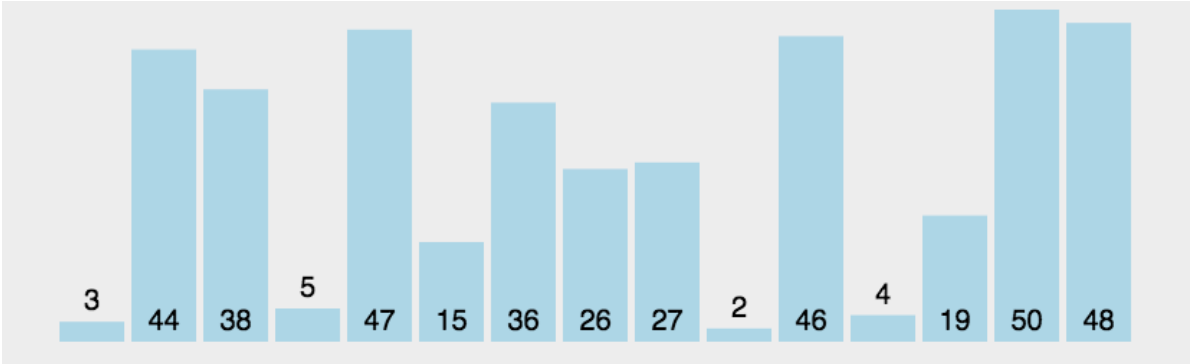
每一轮都是在找一个最小值放在未拍好序的元素的第一个

```
int[] nums = {4,6,0,8,7};
for (int j = 0 ; j < nums.length -1 ; j++){
    int minIndex = j;
    for (int i = j+1; i < nums.length; i++) {
        minIndex =  nums[i] < nums[minIndex] ? i : minIndex;
    }
    int temp = nums[minIndex];
    nums[minIndex] = nums[j];
    nums[j] = temp;
}

for (int i = 0; i < nums.length; i++) {
    System.out.print(nums[i] + " ");
}
```

（2）冒泡

每一轮都是两两相比，会将最大的冒到最后边



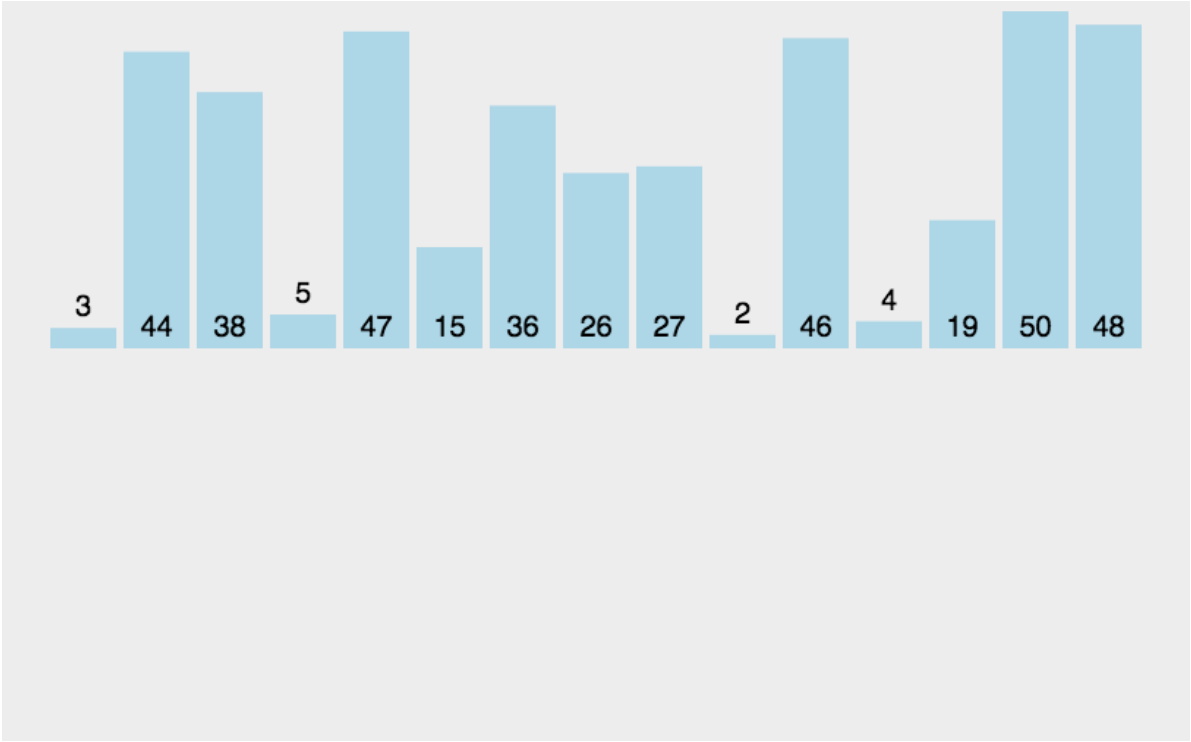

```
int[] nums = {3, 7, 4, 9, 5, 4, 1};

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

```
    }
}

for (int i = 0; i < nums.length; i++) {
    System.out.print(nums[i] + " ");
}
```

（3）插入



五、方法

> 权限修饰符 （static静态）(泛型）返回值类型 名字（参数）{}

1、 权限修饰符

	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 friendly	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

2、返回值类型

void 代表如返回值，能不能用return，能用代表终止方法；

3、static 不是必须的

静态方法和属性是属于类对象的，使用时不需要new对象，直接 `类名.方法名字()`,`类名.属性`

实例方法（不带static） 先要new对象

4、名字，驼峰式首字母小写

5、参数

参数可以有多个，方法定义的地方叫形参，方法调用的地方传入的参数叫实参

6、可变参数:

- 只能出现一次
- 必须放在最后

```
public void f1 (int... nums)
```

7、方法的重载

- 方法名一样
- 参数列表不一样
- 每一个方法都有个签名 就是用名字和参数签名的

六、面向对象

1、封装

需要什么对象，就创建一个什么类，万物皆对象（就是我没有）

```
public class User{
    // 私有的属性

    // 构造器 可以没有 ，可以有一个，也可有duoge

    // 实例方法

    // getter and Setter

    // toString(如果需要)
}
```

可能还有工具类和常量类，工具类里一般大多都是静态方法，常量类里大多是静态的常量

```
public static final String  ORDER_STATUS = "已发货";
```

1.1 属性

属性全部私有，使用getter取值，使用setter赋值、能做安全控制，比如我不想让别人设置值就不写setter，

1.2 方法

1.3 构造器

- new一个对象会主动调用他的构造器

- 构造器可有多个，可以有不同参数的构造器
- 构造器如果不写，或默认送你一个无参构造，一旦你写了有参构造方法，无参的就没有了，需要手写。

2、继承

- extends关键字，子类拥有父类（超类）一切（除了别private修饰的）。
- 构造子类是一定会先构造一个父类。
- 子类可以重写父类的方法。重写的方法需要加一个注解 `@override`
- Object是所有类的顶级父类

3、多态

- 有继承
- 有重写
- 有父类引用指向子类对象

```
List<Integer> list  = new ArrayList<>();
list.add(1)
```

4、抽象类和接口

- 抽象类 : 拥有抽象方法的类，必须使用`abstract`修饰，可以有抽象方法，也可以没有。

  继承抽象类必须实现它的抽象方法。

- 接口：全部都是抽象方法（没有方法体的方法），`interface`

  实现接口（`implements`）必须实现所有的抽象方法

4、Object的几个方法

- equals
- hashcode（hash算法）
- 重写equals必须重写hashcode，主要是因为有hashmap，hashset这类的集合
- toString()

5、this和super

this指向本实例

super指向父类的实例

this和super可以当构造器使用，但是如果想使用super()必须放在第一行；

6、类的加载顺序

1. 父类的静态属性
2. 父类的静态代码块
3. 子类的静态属性
4. 子类的静态代码块
5. 父类的非静态属性
6. 父类的非静态代码块
7. 父类的构造器
8. 子类的非静态属性
9. 子类的非静态代码块
10. 子类的构造器

7、类型强转

父类引用 -> 子类对象 不需要强转　向下转型

子类引用 -> 父类的对象 需要强转 向上转型

七、集合

集合是存东西的，能存很多同一种类型（包含他的子类）的对象。

1、Collection

2、list（列表）

- 需要定义泛型
- ArrayList LinkedList
- 区别：自己看

3、set

- set和list的区别
- list： 一个是有序的，可重复 有序说的是插入的顺序和实际的位置相关
- set : 一个是无序的，不可重复
- hashset怎么判断一个对象是不是重复的，equals和hashcode这两个方法，如果两个对象不是用一个地址但是他们equal，并且hashcode一样就说明一样。

4、map（映射）

- 一听到hash就要想到无序
- hashmap最重要的集合，如果有兴趣可以看看我B站的hashmap源码解读，特别复杂。
- 存的是key，value键值对。
- hashmap的原理很重要，数据结构是数组+链表+红黑树

5、自动排序的map

TreeMap：数据结构是红黑树，这种结构天然就是有序的，字符串会按照字典序排序。

TreeSet ： 底层就是一个TreeMap

6、集合的遍历方式

> list的遍历方式： 普通for循环，增强for循环，迭代器（iterator）

> set的遍历方式：不能用普通for循环，只能用增强for和迭代器

```
// 增强for循环
for (Integer integer : set) {
    System.out.println(integer);
}
// 迭代器
Iterator<Integer> iterator = set.iterator();
while (iterator.hasNext()){
    System.out.println(iterator.next());
}
```

> hashmap的遍历方式： entrySet来遍历，还能用迭代器

```
// 第一种
for (Map.Entry entry : map.entrySet() ){
            System.out.println(entry.getKey());
            System.out.println(entry.getValue());
}
// 第二种
Set<String> keys = map.keySet();
for (String key : keys) {
    System.out.println(map.get(key));
}
// 第三种迭代器
Iterator<Map.Entry<String, User>> iterator = map.entrySet().iterator();
while (iterator.hasNext()){
    Map.Entry<String, User> next = iterator.next();
    System.out.println(next.getKey());
    System.out.println(next.getValue());
}
```

八、泛型

Java泛型设计原则：只要在编译时期没有出现警告，那么运行时期就不会出现ClassCastException异常.

泛型：**把类型明确的工作推迟到创建对象或调用方法的时候才去明确的特殊的类型**

参数化类型:

- **把类型当作是参数一样传递**
- <数据类型> **只能是引用类型**

1、泛型类

泛型类就是把泛型定义在类上，用户使用该类的时候，才把类型明确下来....这样的话，用户明确了什么类型，该类就代表着什么类型...用户在使用的时候就不用担心强转的问题，运行时转换异常的问题了。

- **在类上定义的泛型，在类的方法中也可以使用！**

```
/*
    1:把泛型定义在类上
    2:类型变量定义在类上,方法中也可以使用
 */
public class ObjectTool<T> {
    private T obj;

    public T getObj() {
        return obj;
    }

    public void setObj(T obj) {
        this.obj = obj;
    }
}
```

用户想要使用哪种类型，就在创建的时候指定类型。使用的时候，该类就会自动转换成用户想要使用的类型了。

```
public static void main(String[] args) {
    //创建对象并指定元素类型
    ObjectTool<String> tool = new ObjectTool<>();

    tool.setObj(new String("钟福成"));
    String s = tool.getObj();
    System.out.println(s);


    //创建对象并指定元素类型
    ObjectTool<Integer> objectTool = new ObjectTool<>();
    /**
         * 如果我在这个对象里传入的是String类型的,它在编译时期就通过不了了.
         */
    objectTool.setObj(10);
    int i = objectTool.getObj();
    System.out.println(i);
}
```

2、泛型方法

前面已经介绍了泛型类了，在类上定义的泛型，在方法中也可以使用.....

现在呢，我们可能就仅仅在**某一个方法上需要使用泛型**....**外界仅仅是关心该方法，不关心类其他的属性**...这样的话，我们在整个类上定义泛型，未免就有些大题小作了。

- 定义泛型方法....**泛型是先定义后使用的**

```
    //定义泛型方法..
    public <T> void show(T t) {
        System.out.println(t);

    }
```

- 测试代码：

用户传递进来的是什么类型，返回值就是什么类型了

```
    public static void main(String[] args) {
        //创建对象
        ObjectTool tool = new ObjectTool();

        //调用方法,传入的参数是什么类型,返回值就是什么类型
        tool.show("hello");
        tool.show(12);
        tool.show(12.5);

    }
```

3、泛型类派生出的子类

前面我们已经定义了泛型类，**泛型类是拥有泛型这个特性的类，它本质上还是一个Java类，那么它就可以被继承**

那它是怎么被继承的呢？？这里分两种情况

1. **子类明确泛型类的类型参数变量**
2. **子类不明确泛型类的类型参数变量**

3.1 子类明确泛型类的类型参数变量

- 泛型接口

```
/*
    把泛型定义在接口上
 */
public interface Inter<T> {
    public abstract void show(T t);

}
```

- 实现泛型接口的类.....

```
/**
 * 子类明确泛型类的类型参数变量:
 */

public class InterImpl implements Inter<String> {
    @Override
    public void show(String s) {
        System.out.println(s);

    }
}
```

3.2 子类不明确泛型类的类型参数变量

- 当子类不明确泛型类的类型参数变量时，**外界使用子类的时候，也需要传递类型参数变量进来，在实现类上需要定义出类型参数变量**

```
/**
 * 子类不明确泛型类的类型参数变量:
 *       实现类也要定义出<T>类型的
 *
 */
public class InterImpl<T> implements Inter<T> {

    @Override
    public void show(T t) {
        System.out.println(t);

    }
}
```

测试代码:

```
public static void main(String[] args) {
    //测试第一种情况
    //Inter<String> i = new InterImpl();
    //i.show("hello");

    //第二种情况测试
    Inter<String> ii = new InterImpl<>();
    ii.show("100");

}
```

值得注意的是:

- **实现类的要是重写父类的方法，返回值的类型是要和父类一样的!**
- **类上声明的泛形只对非静态成员有效**

4、 类型通配符

为什么需要类型通配符？？？？我们来看一个需求.....

现在有个需求：**方法接收一个集合参数，遍历集合并把集合元素打印出来，怎么办?**

- 按照我们没有学习泛型之前，我们可能会这样做：

```
public void test(List list){
    for(int i=0;i<list.size();i++){
        System.out.println(list.get(i));

    }
}
```

上面的代码是正确的，**只不过在编译的时候会出现警告，说没有确定集合元素的类型**....这样是不优雅的...

- 那我们学习了泛型了，现在要怎么做呢？？有的人可能会这样做：

```
public void test(List<Object> list){
    for(int i=0;i<list.size();i++){
        System.out.println(list.get(i));
    }
}
```

这样做语法是没毛病的，但是这里十分值得注意的是：**该test()方法只能遍历装载着Object的集合！！！**

强调：**泛型中的`<Object>`并不是像以前那样有继承关系的，也就是说`List<Object>`和`List<String>`是毫无关系的！！！！**

那现在咋办？？？我们是不清楚List集合装载的元素是什么类型的，`List<Objcet>`这样是行不通的.....**于是Java泛型提供了类型通配符？**

所以代码应该改成这样：

```
public void test(List<?> list){
    for(int i=0;i<list.size();i++){
        System.out.println(list.get(i));
    }
}
```

?号通配符表示可以匹配任意类型，任意的Java类都可以匹配.....

现在非常值得注意的是，当我们使用?号通配符的时候：**就只能调对象与类型无关的方法，不能调用对象与类型有关的方法。**

记住，**只能调用与对象无关的方法，不能调用对象与类型有关的方法。**因为直到外界使用才知道具体的类型是什么。也就是说，在上面的List集合，我是不能使用add()方法的。**因为add()方法是把对象丢进集合中，而现在我是不知道对象的类型是什么。**

3.4.1设定通配符上限

首先，我们来看一下设定通配符上限用在哪里....

现在，我想接收一个List集合，它只能操作数字类型的元素【Float、Integer、Double、Byte等数字类型都行】，怎么做？？？

我们学习了通配符，但是如果直接使用通配符的话，该集合就不是只能操作数字了。因此我们需要**用到设定通配符上限**

```
List<? extends Number>
```

上面的代码表示的是：**List集合装载的元素只能是Number的子类或自身**

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


    //List集合装载的是Integer，可以调用该方法
    List<Integer> integer = new ArrayList<>();
    test(integer);

    //List集合装载的是String，在编译时期就报错了
    List<String> strings = new ArrayList<>();
    test(strings);

}


public static void test(List<? extends Number> list) {

}
```

3.4.2设定通配符下限

既然上面我们已经说了如何设定通配符的上限，那么设定通配符的下限也不是陌生的事了。直接来看语法吧

```
//传递进来的只能是Type或Type的父类
<? super Type>
```

设定通配符的下限这并不少见，在TreeSet集合中就有....我们来看一下

```
public TreeSet(Comparator<? super E> comparator) {
    this(new TreeMap<>(comparator));
}
```

5、通配符和泛型方法

大多时候，我们都可以使用泛型方法来代替通配符的.....

```java
//使用通配符
public static void test(List<?> list) {

}

//使用泛型方法
public <T> void  test2(List<T> t) {

}
```

上面这两个方法都是可以的.....那么现在问题来了，我们使用通配符还是使用泛型方法呢？？

原则：

- 如果**参数之间的类型有依赖关系**，或者返**回值是与参数之间有依赖关系**的。那么就使用**泛型方法**
- 如果**没有依赖关系**的，就使用**通配符**，通配符会**灵活一些.**

6、泛型擦除

泛型是**提供给javac编译器使用的**，它用于限定集合的输入类型，让编译器在源代码级别上，即挡住向集合中插入非法数据。但编译器编译完带有泛形的java程序后，**生成的class文件中将不再带有泛形信息**，以此使程序运行效率不受到影响，这个过程称之为“擦除”。

九、多线程

进程和线程的区别： 一个软件运行就是一个进程，一个进程可以运行多个线程，我们何以使用多个线程同时执行提升效率。

多个线程同时执行的时候，互相之间没有必然的顺序，他是有cpu的调度决定的。

> 启动线程必须调用start方法，不能调用run，调用run只是方法调用，start是个native的本地方法，会调度cpu的资源，开辟线程。

> 继承`Thread`类，重写`run`方法;

```java
public class MyThread extends Thread {
    public static void main(String[] args) {
        MyThread t1 = new MyTread("t1");
        t1.start();
    }

    public MyThread(){}

    public MyThread(String name) {
        super(name);
    }

    @Override
    public void run() {
        while (true){
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
```

```
                e.printStackTrace();
            }
            System.out.println("我是：" + Thread.currentThread().getName());
        }
    }
}
```

> 实现`runnable`接口，实现`run`方法;

```
public class MyRun implements Runnable {
    public static void main(String[] args) {
        Thread t1 = new Tread(new MyRun());
        t1.start();
    }
    @Override
    public void run() {
        for (int i = 0; i < 100; i++) {
            ThreadTest.SB.append(1);
        }
    }
}
```

十、常用类

1、String

是引用数据类型，他是final的，一旦创建就不能改变。

String，StringBuffer,StringBuilder的区别 面试常问。

- String是不可变得，线程安全的
- SringBuffer是可变字符串，线程安全
- StringBuilder是可变字符串，线程不安全
- 当有大量字符串拼接的代码时，没有线程安全的要求就用StringBuilder（一般情况），有要求就用StringBuffer。

2、工具类

Date Math Collections Arrays Canlader

以前咱写的复制的算法可能工具类一个copy方法就搞定了，不需要自己写了。

```
Date date = new Date();
SimpleDateFormate s = new SimpleDateFormate("yyyy-MM-dd hh:mm:ss");
String DateString = s.parse(date);
```

十一、枚举

1、定义枚举类型

定义枚举类型需要使用enum关键字，例如：

枚举可以用equals来比较，也可以用 == 来比较

```
public enum Direction {
    FRONT, BEHIND, LEFT, RIGHT;
}
Direction d = Direction.FRONT;
```

2、枚举与switch

枚举类型可以在switch中使用

```
Direction d = Direction.FRONT;
  switch(d) {
    case FRONT: System.out.println("前面");break;
    case BEHIND:System.out.println("后面");break;
    case LEFT:  System.out.println("左面");break;
    case RIGHT: System.out.println("右面");break;
    default:System.out.println("错误的方向");
}
Direction d1 = d;
System.out.println(d1);
```

3、枚举类的方法

- int compareTo(E e)：比较两个枚举常量谁大谁小，其实比较的就是枚举常量在枚举类中声明的顺序，例如FRONT的下标为0，BEHIND下标为1，那么FRONT小于BEHIND；
- boolean equals(Object o)：比较两个枚举常量是否相等；
- int hashCode()：返回枚举常量的hashCode；
- String name()：返回枚举常量的名字；
- int ordinal()：返回枚举常量在枚举类中声明的序号，第一个枚举常量序号为0；
- String toString()：把枚举常量转换成字符串；
- static T valueOf(Class enumType, String name)：把字符串转换成枚举常量。

1. 枚举类的构造器
   枚举类也可以有构造器，构造器默认都是private修饰，而且只能是private。因为枚举类的实例不能让外界来创建！

```
enum Direction {
    FRONT, BEHIND, LEFT, RIGHT;//[在枚举常量后面必须添加分号，因为在枚举常量后面还有其他成员时，分号是必须的。枚举常量必须在枚举类中所有成员的上方声明。]

    Direction()//[枚举类的构造器不可以添加访问修饰符，枚举类的构造器默认是private的。但你自己不能添加private来修饰构造器。] {
        System.out.println("hello");
    }
}
```

其实创建枚举项就等同于调用本类的无参构造器，所以FRONT、BEHIND、LEFT、RIGHT四个枚举项等同于调用了四次无参构造器，所以你会看到四个hello输出。

5、枚举类可以有成员

其实枚举类和正常的类一样，可以有实例变量，实例方法，静态方法等等，只不过它的实例个数是有限的，不能再创建实例而已。

```
enum Direction {
    FRONT("front"), BEHIND("behind"), LEFT("left"), RIGHT("right");

    private String name;

    Direction(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}
Direction d = Direction.FRONT;
System.out.println(d.getName());
```

十二、流

关键字

input 输入 output 输出 stream 流 writer 字符输入流 reader 字符输入流 File 文件

只要会了字节流就都会了

字节流能处理一切

1、流的分类表

| 分类 | 字节输入流 |字节输出流|字符输入流|字符输出流|

分类	字节输入流	字节输出流	字符输入流	字符输出流
抽象基类	InputStream	OutputStream	Reader	Writer
访问文件	FileInputStream	FileOutputStream	FileReader	FileWriter
访问数组	ByteArrayInputStream	ByteArrayOutputStream	CharArrayReader	CharArrayWriter
访问字符串			StringReader	StringWriter
缓冲流	BufferedInputStream	BufferedOutputStream	BufferedReader	BufferedWriter
转换流			InputStreamReader	OutputStreamWriter
对象流	ObjectInputStream	ObjectOutputStream		

转换流能都将字符流转成字节流

二、小案例

1、文件的基本操作

```
@Test
public void testFile() throws Exception{
    //创建文件
    File file = new File("E:\\test\\b.txt");
    file.createNewFile();

    //查看文件夹下的文件
    File file2 = new File("E:\\test\\b.txt");
    String[] list = file2.list();
    for (int i = 0; i < list.length; i++) {
        System.out.println(list[i]);
    }
    //其他的方法自己查看
}
```

2、复制文件带有进度条

```
@Test
public void testFileInputStream() throws Exception {

    File file = new File("E:\\test\\a\\233.mp4");

    //拿到文件的大小
    long dataLength = file.length();

    //构建一个输入流，他的数据要流入内存，咱们的程序
    InputStream inputStream = new FileInputStream(file);

    //构建一个输出流，他的数据要从内存（咱们的程序）流到另一个文件夹
    OutputStream outputStream = new
FileOutputStream("E:\\test\\b\\233.mp4");

    //新建一个水泵，能存一点水，每次对对1k
    byte[] buffer = new byte[1024 *1024*50];
    Long currentLength = 0L;

    //如果read返回-1说明读完了
    int len;
    int showNumber = 0;
    while ( (len = inputStream.read(buffer)) != -1 ){
        outputStream.write(buffer,0,len);
        currentLength += len;
        //当下加载了百分之多少
        int currentPer  = (int)(((double)currentLength/dataLength)*100);
        //目的是不重复显示
        if(showNumber !=  currentPer){
            showNumber = currentPer;
            System.out.println("已经拷贝了百分之" + showNumber);
        }
    }
```

```
33
34      outputStream.flush();
35      outputStream.close();
36      inputStream.close();
37  }
```

3、字节流读文件

```
1   @Test
2   public void testInputStream() throws Exception{
3       //怼了一个输入流到文件上
4       InputStream wordInput = new FileInputStream("E:\\test\\a\\word.txt");
5       //建立缓冲区
6       byte[] bytes = new byte[1024];
7       int len;
8       while ( (len = wordInput.read(bytes)) != -1 ){
9           System.out.println(new String(bytes,0,len, Charset.forName("ISO8859-
    1")));
10      }
11
12      wordInput.close();
13  }
```

4、字符流对文件

```
1   @Test
2   public void testReader() throws Exception{
3       //怼了一个输入流到文件上
4       Reader reader = new FileReader("E:\\test\\a\\word.txt");
5       BufferedReader br = new BufferedReader(reader);
6       String str;
7       while ((str = br.readLine()) != null){
8           System.out.println(str);
9       }
10      reader.close();
11      br.close();
12  }
```

5、向文件里写内容

```
1   //这个用main方法测吧
2   public void testWriter() throws Exception{
3       //怼了一个输入流到文件上
4       Writer writer = new FileWriter("E:\\test\\a\\writer.txt");
5       BufferedWriter bw = new BufferedWriter(writer);
6       Scanner scanner = new Scanner(System.in);
7
8       while (true){
9           System.out.print("请输入：");
10          String words = scanner.next();
```

```
        bw.write(words);
        bw.flush();
    }
}
```

6、StringReader就是往String上怼

```
@Test
public void testStringReader() throws Exception{
    //怼了一个string
    OutputStream os = new FileOutputStream("E:\\test\\a\\user.txt");
    ObjectOutput oo  = new ObjectOutputStream(os);
    oo.writeObject(new User("王老师",3,4));
    oo.flush();
    oo.close();
    os.close();
}
```

7、对象流就是对new出来的对象进行序列化

该对象必须实现Serializable接口，才能被序列化。

对象在内存就是一堆0和1，任何文件在内存都是0和1，都能转化成字节数组，进行保存或者网络传输。

对象序列化也是一样的，就是把内存的对象以字节数组的形式上保存，我们能保存在磁盘，或者在网络中传输。

```
import java.io.Serializable;

/**
 * @author zn
 * @date 2020/2/14
 **/
public class User implements Serializable {
    private String name;

    private int age;

    private int gander;

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

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
```

```
27
28      public int getAge() {
29          return age;
30      }
31
32      public void setAge(int age) {
33          this.age = age;
34      }
35
36      public int getGander() {
37          return gander;
38      }
39
40      public void setGander(int gander) {
41          this.gander = gander;
42      }
43  }
44
```

```
1   @Test
2   public void testObjectOut() throws Exception{
3       //怼了一个string
4       InputStream is = new FileInputStream("E:\\test\\a\\user.txt");
5       ObjectInputStream oi  = new ObjectInputStream(is);
6       User user = (User)(oi.readObject());
7       System.out.println(user.getName());
8       is.close();
9       oi.close();
10  }
```

十三、异常

1、checked 检查性异常

去坐飞机，可能堵车，你只能提前走一会儿来预防，所以要在编译的时候就使用try，catch来预先捕获，并给出解决方案

FileNotFoundException IoException InterruptedException

这种异常继承自Exception， 必须捕获，并处理

2、运行时异常

去坐飞机，没带护照，这是你自己的原因，可以通过检查一下解决

ArrayIndexOutOfBoundsException ClassCastException 数学类异常(int i = 1 / 0)

这种异常继承RuntimeException，不需要捕获，需要通过检查来预防。

3、错误 error

stackOutOfMemoryError　比如递归出不去

4、自定义异常

很多时候我们需要自己定义一些异常来帮助我们处理一些业务。

十四、算法

1、排序

- 选择排序
- 冒泡排序
- 插入排序

- 希尔排序
- 快速排序
- 归并排序（分而治之，归并）

- 堆排序　完全二叉树 大根堆 小根堆
- 桶排序
- 基数排序
- 计数排序

2、查找

二分查找

3、链表

数组和链表的区别：

- 数组：按下标查找快，插入效率慢；
- 链表：查找慢，插入快。

4、队列和栈 FIFO FILO

5、hash算法，

可以将任何的文件转化成一个定长的字符串。

hash碰撞： 当两个不同的文件生成的字符长一样了，就叫hash碰撞

回到hashmap 取余数后值一样就叫hash碰撞

5、递归

方法自己调用自己

一定要有出口，没有出口就会栈内存溢出。

典型的案例：斐波那契数列

```
/**
     * 第一个 0 第二个是1  以后都是前两个的和
     * 自己调用自己叫递归
     * 完成斐波那契数列  自己找一个联系递归的例子
     * @param count
     * @return
     */
public static int fibonacci(int count){
    if (count == 1){
        return 0;
    }
    if (count == 2){
        return 1;
    }
    if (count < 1){
        System.out.println("您输入的不合法");
        return -1;
    }
    return fibonacci(count - 1) + fibonacci(count - 2);
}
```

5、超级集合

> 项目代码

> 接口定义统一规范

```
package cn.itnanls.util;

/**
 * @author zn
 * @date 2020/2/7
 **/
public interface Super<T> {

    //添加数据的方法
    void add(T data);


    //根据下标查询数字
    T get(int index)

    //查看当前有多少个数字
    int size();

}
```

```java
package cn.itnanls.util;

/**
 * @author zn
 * @date 2020/2/7
 **/
public class SuperArray<T> extends Super<T> {

    //维护一个数组,要想什么都存，就要使用顶级父类
    private Object[] array;
    //当前最后一个数字的下边，要为-1 ，以为数组的第一个下标为0
    private int currentIndex = -1;

    //构造是初始化
    public SuperArray(){
        array = new Object[8];
    }

    //添加数据的方法
    public void add(T data){
        System.out.println("我是数组的实现！---add");
        currentIndex++;
        //自动扩容
        if(currentIndex > array.length-1){
            array = dilatation(array);
        }
        array[currentIndex] = data;
    }


    //根据下标查询数字
    public T get(int index){
        System.out.println("我是数组的实现---get");
        return (T)array[index];
    }

    //查看当前有多少个数字
    public int size(){
        return currentIndex + 1;
    }

    //数组扩容的方法
    private Object[] dilatation(Object[] oldArray){
        Object[] newArray = new Object[oldArray.length * 2];
        for (int i = 0; i < oldArray.length; i++) {
            newArray[i] = oldArray[i];
        }
        return newArray;
    }

    //验证下标是否合法
    private boolean validateIndex(int index) {
        //只要有一个不满足就返回false
        return index <= currentIndex && index >= 0;
    }
```

```

}

```

```
package cn.itnanls.util;

/**
 * @author zn
 * @date 2020/2/11
 **/
public class SuperLinked<T> extends Super<T> {

    private Node head = null;
    private Node tail = null;

    private int length = 0;

    //添加元素
    public void add(T data){
        System.out.println("我是链表的实现-----add");
        Node<T> node = new Node<>();
        node.setNum(data);
        if (length == 0) {
            //如果第一次添加一共就一个节点
            head = node;
        }else{
            //和尾巴拉手
            tail.setNextNode(node);
        }
        //把心添加进来的当成尾巴
        tail = node;
        length ++;
    }

    //根据下标查询数字,非常有意思的写法
    public T get(int index){
        System.out.println("我是链表的实现------get");
        if(index > length){
            return null;
        }
        //小技巧
        Node targetNode = head;
        for (int i = 0; i < index; i++) {
            targetNode = targetNode.getNextNode();
        }
        return (T)(targetNode.getNum());
    }

    //查看当前有多少个数字
    public int size(){
        return length;
    }

    class Node<T> {

        //存储的真实数据
        private T num;
```

```
        //写一个节点
        private Node nextNode = null;

        public T getNum() {
            return num;
        }

        public void setNum(T num) {
            this.num = num;
        }

        public Node getNextNode() {
            return nextNode;
        }

        public void setNextNode(Node nextNode) {
            this.nextNode = nextNode;
        }
    }
}

```

```
package cn.itnanls.util;

/**
 * @author zn
 * @date 2020/2/11
 **/
public abstract class Super<T> {

    /**
     * 标记所有的子类实现必须有add方法，添加数据
     * @param data
     */
    public abstract void add(T data);

    /**
     * 标记所有的子类实现必须有get方法，获取数据
     * @param index
     * @return
     */
    public abstract T get(int index);

    /**
     * 标记所有的子类实现必须有size方法，数据大小
     * @return
     */
    public abstract int size();

}

```