```
package com.southwind.demo;

import java.util.ArrayList;
import java.util.Collections;

public class Test {
    public static void main(String[] args) {
        ArrayList list = new ArrayList();
//        list.add("Hello");
//        list.add("Java");
//
Collections.addAll(list,"Java","JavaME","World");
//        System.out.println("排序之前");
//        System.out.println(list);
        //进行排序-》升序a
//        Collections.sort(list);
//        System.out.println("排序之后");
//        System.out.println(list);
        //查找元素在集合中的下标,二分查找法（集合中的元素必
须升序排列）
//        int index =
Collections.binarySearch(list,"Java");
//        System.out.println("Java 在 list 中的下
标"+index);
//        System.out.println(list);
//
Collections.replaceAll(list,"Java","Collections");
//        System.out.println(list);

        Collections.addAll(
                list,
```

```
                new User(1,"张三",30),
                new User(2,"李四",26),
                new User(3,"王五",18)
        );

        Collections.sort(list);

        System.out.println(list);

    }
}

class User implements Comparable{
    private Integer id;
    private String name;
    private Integer age;

    public User(Integer id, String name, Integer
age) {
        this.id = id;
        this.name = name;
        this.age = age;
    }

    @Override
    public String toString() {
        return "User{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", age=" + age +
                '}';
    }
```

```
    @Override
    public int compareTo(Object o) {
        if(o instanceof User){
            User user = (User) o;
            if(this.age < user.age){
                return 1;
            }else if(this.age == user.age){
                return 0;
            }else{
                return -1;
            }
        }
        return 0;
    }
}
```

泛型

泛型（Generics），是指在类定义时不指定类中信息的具体数据类型，而是暂时用一个标识符来替代，当外部实例化对象的时候再来指定具体的数据类型。

```
//定义 A 类的时候就指定了属性是 B 类型
public class A{
  private B b;

  public C test(D d){
    return new C();
  }
}
```

```
//定义 A 类的时候不指定属性的类型
public class A<T,E,M>{
  private T b;
  public E test(M m){
    return E;
  }
}

A<B,C,D> a = new A();
```

优点：这样做极大地提升程序的灵活性，提升类的扩展性，泛型可以指代类中成员变量的数据类型，方法的返回值类型以及方法的参数类型。

泛型的应用

自定义类中添加泛型

```
public class 类名<泛型1,泛型2,泛型3...>{
  private 泛型1 属性名;
  public 泛型2 方法名(泛型3){
    方法体
  }
}
```

```java
package com.southwind.demo3;

public class Time<T> {
    private T value;

    public T getValue() {
        return value;
    }

    public void setValue(T value) {
        this.value = value;
    }
}
```

```java
package com.southwind.demo3;

public class Test {
    public static void main(String[] args) {
        Time<Integer> time1 = new Time<>();
        time1.setValue(10);
        System.out.println("现在的时间
是"+time1.getValue());
        Time<String> time2 = new Time<>();
        time2.setValue("十点整");
        System.out.println("现在的时间
是"+time2.getValue());
    }
}
```

泛型用哪个字母都可以，关键是类定义处的字母和类中信息的字母保持一致。

```java
package com.southwind.demo4;

public class Time<H,M,S> {
    private H hour;
    private M minute;
    private S second;

    public H getHour() {
        return hour;
    }

    public void setHour(H hour) {
        this.hour = hour;
    }

    public M getMinute() {
        return minute;
    }

    public void setMinute(M minute) {
        this.minute = minute;
    }

    public S getSecond() {
        return second;
    }

    public void setSecond(S second) {
        this.second = second;
    }
}
```

```
package com.southwind.demo4;

public class Test {
    public static void main(String[] args) {
        Time<String,Integer,Float> time = new Time<>
();
        time.setHour("十点");
        time.setMinute(10);
        time.setSecond(10.0f);
        System.out.println("现在的时间
是"+time.getHour()+":"+time.getMinute()+":"+time.get
Second());
    }
}
```

泛型通配符

有一个参数为 ArrayList 的方法，希望这个方法即可接收泛型是 String 的集合，又可以接收泛型是 Integer 的集合，怎么实现?

多态在泛型中不适用

```
public class Test {

    public static void main(String[] args) {
        ArrayList<String> list1 = new ArrayList<>();
        ArrayList<Integer> list2 = new ArrayList<>();
        test(list1);
        test(list2);
    }

    public static void test(ArrayList<Object> list){

    }

}
```

```
public class Test {

    public static void main(String[] args) {
        ArrayList<String> list1 = new ArrayList<>();
        ArrayList<Integer> list2 = new ArrayList<>
();
        test(list1);
        test(list2);
    }

    public static void test(ArrayList<?> list){
        System.out.println(list);
    }

}
```

ArrayList<?> 表示可以使用任意的泛型类型对象，这样 test 方法具备通用性了。

泛型上限和下限

上限：表示实例化时具体的数据类型，可以是上限类型的子类或者是上限类型本身，用 extends 表示。

下限：表示实例化时具体的数据类型，可以是下限类型的父类或者是下限类型本身，用 super 表示。

类名<泛型标识 extends 上限类名>

类名<泛型标识 super 下限类名>

```
public class Time<T> {

    public static void main(String[] args) {
        test(new Time<Float>());
        test(new Time<Integer>());
        test(new Time<Number>());

        test2(new Time<String>());
        test2(new Time<Object>());
    }

    /**
     * 泛型上限
     * @param time
     */
    public static void test(Time<? extends Number>
time){

    }

    /**
     * 泛型下限
```

```
     * @param time
     */
    public static void test2(Time<? super String>
time) {

    }

}
```