泛型

1.创建泛型类（如ArrayList）实例

> 泛型就是一种可以接收数据类型的数据类型

```
static ArrayList<Song> songs_ = new ArrayList<Song>();
//用泛型对传入的元素做限制，希望传入的是Song类或者它的子类
public static void addSong(Song song){
    songs_.add(song);//以每一个Song对象作为一个列表的元素
}
```

2.泛型类型变量的声明和赋值

如果Song是music的子类，那么就能把一个ArrayList<Song> 的引用变量赋给ArrayList< music >的引用变量，因为ArrayList< music >的引用变量（假设是m) 能接收music类及其它的子类，当接收到Song子类的时候，就会将Song向上转型，这是m的运行类型是Song，编译类型是music，由此实现了多态和泛型的结合。

```
 ArrayList<Music> m = new ArrayList<Song>();
```

但是game类和music类没有任何关系，所以不能赋值.

3.自定义泛型类的笔记（跟着b站视频写的）

```
public class Generic03 {
    public static void main(String[] args) {
        Person<String> person = new Person<String>("John");
        person.getS();
        //在编译期间确定这个E到底是什么
        //下面所有E的部分全部换成String
    }
}
class Person<E>{
    //泛型类
    //该数据类型是定义Person对象的时候定义的。
    E s;
    public Person(E s) {
        this.s = s;
    }
    public E getS() {
        System.out.println(s.getClass());
        return s;
    }
}
```

4.自定义泛型接口的笔记（跟着b站视频写的）

```
interface I<U,R> {
    int n=10;
    //U name;不能这样写;
    //接口只能包含public static final 类型的变量，即常量
    R get(U u);
    //接口中的方法都是public static修饰的，只是没有显示出来
    void hi(R r);
    //jdk8中可以在接口中使用默认方法
    default R method(U u){
        return null;
    }
}

interface IA extends I<String,Integer>{

}
class AA implements IA{
    //当实现IA接口时，IA在继承I，指定了U和R的类型
    //实现接口方法的时候就用指定的类型自动替换
    @Override
    public Integer get(String s) {
        return 0;
    }
    @Override
    public void hi(Integer integer) {

    }
    @Override
    public Integer method(String s) {
        return IA.super.method(s);
    }
}
class BB implements I{
    //什么都没写的话类型默认是Object
    @Override
    //
    public Object get(Object o) {
        return null;
    }
    @Override
    public void hi(Object o) {

    }
    @Override
    public Object method(Object o) {
        return I.super.method(o);
    }
}
```

5.自定义泛型方法

```
public class CustomMethod {
    public static void main(String[] args) {
        Car car = new Car();
        car.fly("booma",100);
        //调用方法时必须传入参数，由编译器确定类型

    }
}
class Car{
    public void run(){

    }
    public <T,R> void fly(T t,R r){//泛型方法
        System.out.println("fly car");
        System.out.println(t.getClass().getName());
        System.out.println(r.getClass().getName());
    }

}
class Fish<T,R>{
    public void hi(T t){//这个不是泛型方法，而是该方法使用了泛型
        System.out.println("hi");
    }
}
```