泛型

编译器强类型检查，无需进行显示转换

```
T,E,K,V,?
泛型接口、泛型类、泛型方法
interface MyInterface<T>{...}
class MyClass<T>{...}
public <T> returnType functionName(T t){...}
```

1. 当T传入class内基本可以随意使用
2. 泛型继承时必须明确T
3. 注意ClassName(**T extends FatherClass**)与ClassName(**? extends FatherClass**)的区别
4. 不管为泛型的类型形参传入哪一种类型实参，对于`java`类型来说，他们依然被当成同一个类处理，在内存中也只占用一块内存空间，因此在静态方法，静态初始化块或者静态变量的声明和初始化中不允许使用类型形参。
5. 由于系统中并不会真正生成泛型，所以`instanceof`运算符后不能使用泛型类

泛型擦除

Java中的泛型**在编译器这个层次实现**，在生成的**Java字节码中式不包含泛型中的类型信息**的。使用泛型时候加上类型的参数，会在**编译器在编译的**时候去掉。这个过程就称为泛型擦除

协变性

数组具有协变性，而泛型不具有协变性

```
A extends B
A[] extends B[]
List<A> !extends List<B>
```

通配符 extends super

```
Apple->Fruit->Food
class Food{}
class Fruit extends  Food{}
class Apple extends  Fruit{}
//extends
List<? extends Fruit> fruits = new ArrayList<>();
fruits.add(new Food());//compile error
```

```
fruits.add(new Fruit());//compile error
fruits.add(new Apple());//compile error

fruits = new ArrayList<Food>();//compile error
fruits = new ArrayList<Fruit>();//compile success
fruits = new ArrayList<Apple>();//compile success

Fruit obj = fruits.get(0);//compile success

//super
List<? super Fruit> fruits = new AarrayList<>();
fruits.add(new Food());//compile error
fruits.add(new Fruit());//compile success
fruits.add(new Apple());//compile successs

fruits = new ArrayList<Food>();//compile success
fruits = new ArrayList<Fruit>();//compile success
fruits = new ArrayList<Apple>();//comple error

Fruit fruit = fruits.get(0);//compile error
```

1. extends 可用于的**返回类型限定**，**不能用于参数类型限定。**
2. super 可用于参数类型限定，不能用于返回类型限定。
3. 带有 super 超类型限定的通配符可以向泛型对易用写入，带有 extends 子类型限定的通配符可以向泛型对象读取。

Date

Date date = sdf.parse(String str_time);

String str_time = sdf.format(Date date);

```java
//String ->Date  Date date = sdf.parse(String str_time);
String str_time = "2019-01-10";
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
Date date = (Date) sdf.parse(str_time);
System.out.println(date);

//Date->String  sdf.format(Date date);
Date date = new Date();
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
String str_date  = sdf.format(date);
System.out.printlf(str_date);
```

hashCode equals ==

1. 同一个对象的hashCode值一定相等，逆否命题：Hashcode不同的两个对象一定不相同
2. 不同对象的hashCode有可能相等，也就是说hashCode相等不能判断两个对象相等
3. 如果两个对象调用equals()方法，返回结果为true，则两个对象的hashCode必定相等
4. hashCode不等，equals返回false
5. 重写equals()方法，必须重写hashCode方法，并且二者在逻辑上必须保持一致
6. ==操作符,**基本数据类型**比较的是值

   引用类型比较的是对象的地址

   在没有重写equals()方法时，调用equals()方法是Obejcet中的equals(),equals()就是用**==**来比较的