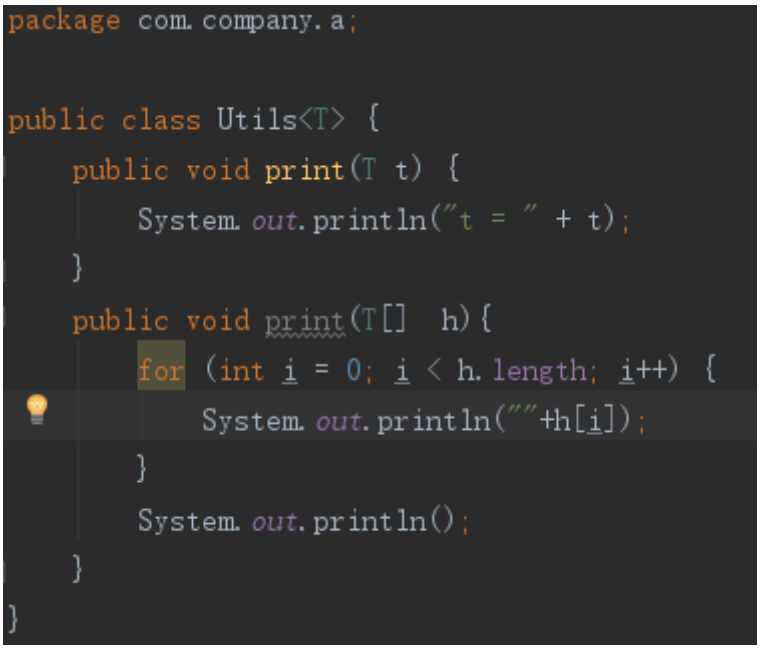
泛型:一中允许把数据类型作为参数进行传递的技术



```java
package com.company.a;

public class Utils<T> {
    public void print(T t) {
        System.out.println("t = " + t);
    }
    public void print(T[]  h){
        for (int i = 0; i < h.length; i++) {
            System.out.println(""+h[i]);
        }
        System.out.println();
    }
}
```

创建泛型类对象时,可以指定数据类型

```java
Utils<String>utils=new Utils<>();
utils.print("123");

int[]a= {1,2,3,4,};
```

创建泛型类对象时,没有指定数据类型,数据类型就为object

```java
Utils utils1=new Utils<>();
utils1.print("!124");
```

泛型必须为引用类型

```java
Utils<Integer> utils2=new Utils<>();
utils.print(123);
```

为了解决基本数据类型不符合面向对象特点,为每一种基本数据了类型提供与之对应的引用类型

包装类:基本数据类型对应的引用类型

基本数据类型	包装类
byte	byte
short	Short
int	integer
long	long
float	float
double	double
char	character
boolean	boolean

基本数据类型和包装类可以无缝转化

```
byte b1=10;
装箱:基本数据类型-包装
Byte b2=b1;
拆箱:包装-基本数据类型
byte b3=b1;
```

包装类提供属性和方法

```
//最大值
System.out.println(Byte.MAX_VALUE);
//最小值
System.out.println(Byte.MAX_VALUE);
//所占的二进制
System.out.println(Byte.SIZE);
//所占的字节数
System.out.println(Byte.BYTES);
```

```
Byte b=10;
//类型转化
System.out.println(b.shortValue());
System.out.println(b.shortValue());
```

```
//字符串转byte
```

```
byte b4=Byte.parseByte("12");
//字符串转Byte
Byte aByte=Byte.valueOf("21");
```

集合框架

集合框架的顶级接口

1.Collection接口

a.List接口

1.ArrayList

2.LinkedList

3.TreeList

4.Vector

b.Set接口

1.HashSet

2.LinkedHashSet

3.TreeSet

c.map接口

1.HashMap

2.LinkedHashMap

3.TreeMap

4.HashTable

集合框架的实现类如何选择?（10分）

```
Map实现类的选择: (3分)
1.有次序, 并且可以排序, 选择TreeMap
2.有次序, 但不能排序, 选择LinkedHashMap
3.其它选择HashMap或HashTable

HashMap和HashTable的选择? (2分)
1.HashTable是线程安全的; 效率低
```

```
2.HashMap是线程不安全的; 效率高

 Set实现类的选择: (3分)
 1.有次序, 并且可以排序, 选择TreeSet
 2.有次序, 但不能排序, 选择LinkedHashSet
 3.其它选择HashSet

 List实现类的选择: (3分)
 1.存储大量的数据, 经常做增加或删除, 选择LinkedList
 2.模拟栈的结构, 选择Stack
 3.其它选择ArrayList或Vector

 ArrayList和Vector选择? (2分)
 1.Vector是线程安全的; 效率低
 2.ArrayList是线程不安全的; 效率高
```