

## 知识点列表

编号	名称	描述	级别
1	集合复制	Java 中默认的集合复制为浅表复制，理解浅表复制原理，掌握复制集合的两种方法：clone()和构造器复制	**
2	集合中的同步化	理解 Java 中线程安全和线程不安全的区别，相互转换的方法	**
3	数组与集合的转换	掌握数组变为 List，List 转换为数组的 API 方法	**
4	Collection 和 Collections 的区别	经典面试题，难度不大，注意理解	*
5	Map 的迭代	掌握迭代 Map 中的key与 value 以及 Map.Entry 的使用	**
6	静态内部类	掌握实现及使用	*
7	成员内部类	掌握实现及使用	*
8	局部内部类	掌握实现及使用	*
9	匿名内部类	熟练掌握匿名内部类的实现及使用	**

注：    \*\*"理解级别    \*\*\*"掌握级别    \*\*\*\*"应用级别

## 目录

### 1. 集合补遗

#### 1.1.集合复制 \*\*

**集合复制**，Java 默认的复制规则是浅表(浅层)复制

集合复制有 2 种方式：

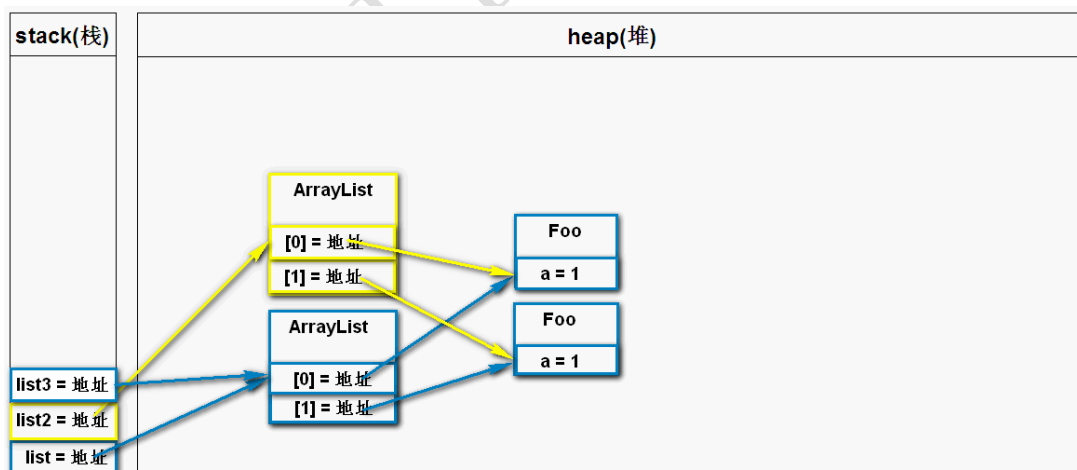
- 1) clone() 方法
  - clone()方法是 Object 定义的
- 2) 使用"复制构造器"
  - `Map map = new HashMap();`  
`Map map2 = new HashMap(map);`
  - `List list1 = new ArrayList();`  
`List list2 = new LinkedList(list1);`

#### 【案例 1】浅表复制\_clone()方法

```

CollectionClone.java
1 package corejava.day07.ch01;
2 import java.util.ArrayList;
3 public class CollectionClone {
4     public static void main(String[] args) {
5         ArrayList<Foo> list = new ArrayList<Foo>();
6         list.add(new Foo());
7         list.add(new Foo());
8         //1. 经过clone()产生一个新对象list2
9         ArrayList<Foo> list2 = (ArrayList<Foo>) list.clone();
10
11         //2. list3和list是同一个对象的引用
12         ArrayList<Foo> list3 = list;
13         System.out.println(list3==list); //true
14
15         //3. 浅表(浅层)复制测试:
16         //第一层复制了
17         System.out.println(list2==list); //false
18         //第二层元素没有被复制!
19         System.out.println(list2.get(0)==list.get(0)); //true
20     }
21 }
22 class Foo{
23     int a=1;
24 }
    
```

浅表复制内存结构图如下所示：



注：

- ✓ list2 是浅表复制，仅第 1 层复制了，第 2 层复制的是对象的引用
- ✓ list3 是赋值
- ✓ clone() 只能复制本类型

## 【案例 2】浅表复制\_“复制构造器”

```

1 package corejava.day07.ch02;
2
3 import java.util.ArrayList;
4
5
6
7 public class CollectionClone {
8     public static void main(String[] args) {
9         ArrayList<Foo> list = new ArrayList<Foo>();
10        list.add(new Foo());
11        list.add(new Foo());
12        //复制构造器演示：
13        //1. 所有集合都有“复制构造器”，是浅表复制
14        //2. “复制构造器”的参数是多态，可以在不同种类的集合间复制
15        LinkedList<Foo> list4 = new LinkedList<Foo>(list);
16        HashSet<Foo> set = new HashSet<Foo>(list);
17        System.out.println(list4.containsAll(list)); //true
18        System.out.println(set.containsAll(list)); //true
19    }
20 }
21 class Foo{
22     int a=1;
23 }

```

注：

- ✓ “复制构造器”可以复制不同类型的集合，使用起来较方便
- ✓ 继承自 Map 的集合同继承自 Collection 的集合不能相互复制（结构不同）

## 1.2. 同步化（线程安全）\*\*

同步化解决方案：

- 1) **Collections.synchronizedList()方法**可以将非线程安全的 list 包装为线程安全的  
`List list = new ArrayList();`  
`list = Collections.synchronizedList(list);` //转换以后就相当于 Vector
- 2) **Collections.synchronizedMap()方法**可以将非线程安全的 map 包装为线程安全的  
`HashMap map = new HashMap();`  
`map = Collections.synchronizedMap(map);`

## 1.3. 数组与集合的转换 \*\*

### 【案例 1】数组转 List

```

1 package corejava.day07.ch03;
2 import java.util.ArrayList;
3
4
5
6
7
8 public class Demo {
9     public static void main(String[] args) {
10         String[] names = {"A", "B", "C"};
11         //第1步：只读list（不能增删操作）
12         List list = Arrays.asList(names);
13         // list.add("cdd"); //UnsupportedOperationException
14         // list.remove(0); //UnsupportedOperationException
15         System.out.println(list);
16
17         //第2步：复制为全功能List
18         list = new ArrayList(list);
19         list.add("cdd");
20         System.out.println(list);
21
22         //第2步：(或)放入set集合
23         Set set = new HashSet(list);
24         System.out.println(set);
25     }
26 }

```

### 【案例 2】集合转数组

```

1 package corejava.day07.ch03;
2 import java.util.ArrayList;
3 public class Demo02 {
4     public static void main(String[] args) {
5         ArrayList list = new ArrayList();
6         list.add("a");
7         list.add("b");
8
9         //集合转Object[]
10        Object[] ary1 = list.toArray();
11        //集合转指定类型数组
12        String[] ary2 =
13            (String[])list.toArray(new String[]{});
14    }
15 }
16

```

#### 1.4. Collection 与 Collections \*

面试可能会遇到的题目，Collection 和 Collections 的区别是什么？

- ✓ **Collection** 抽象的集合概念，实现它的有 List 和 Set
- ✓ **Collections** 集合静态工具类，包含集合的工具方法，如 sort()等

#### 1.5. Map 的迭代 \*\*

- ✓ 对 key:value 进行迭代 `map.entrySet();`
- ✓ 对 key 进行迭代 `map.keySet();`
- ✓ 对 value 进行迭代 `map.values();`

#### 【案例 1】统计每个字符出现的次数

##### ● 版本 01

```

1 package corejava.day07.ch04;
2 import java.util.HashMap;
3
4 public class CharCounterDemo {
5     public static void main(String[] args) {
6         String str = "我醒了,我洗脸,我吃早餐";
7         Map<Character, Integer> map = count(str);
8         System.out.println(map);
9     }
10

```

```

11  /** 统计字符字数 */
12  public static Map<Character, Integer> count(String s){
13      Map<Character, Integer> map =
14          new HashMap<Character, Integer>();
15      for(int i=0; i<s.length(); i++){
16          char c = s.charAt(i);
17          if(map.containsKey(c)){//c字符是否在map中,是否统计过
18              int count = map.get(c) + 1;//取出原统计值,再加1
19              map.put(c, count);
20          }else{//没有统计过
21              map.put(c, 1);//第一次统计字符
22          }
23      }
24      return map;
25  }
26  }

```

● 版本 02

```

CharCounterDemo02.java
1  package corejava.day07.ch04;
2  import java.util.HashMap;
3
4  public class CharCounterDemo02 {
5      public static void main(String[] args) {
6          String str = "我醒了,我洗脸,我吃早餐";
7          Map<Character, Integer> map = count(str);
8          System.out.println(map);
9      }
10
11     /** 统计字符字数 */
12     public static Map<Character, Integer> count(String s){
13         Map<Character, Integer> map =
14             new HashMap<Character, Integer>();
15         for(int i=0; i<s.length(); i++){
16             char c = s.charAt(i);
17             Integer count = map.get(c);
18             count = count==null?1:count+1;
19             map.put(c, count);
20         }
21         return map;
22     }
23 }

```

【案例 2】将 map 中统计出来的次数迭代打印，显示为“表格”形式

```
CharCounterDemo.java CopyOfCharCounterDemo.java
1 package corejava.day07.ch05;
2 import java.util.Collection;
7 public class CharCounterDemo {
8     public static void main(String[] args) {
9         String str =
10             "我醒了,我洗脸,我吃早餐,我出发,我到学校!";
11         Map<Character, Integer> map = count(str);
12
13         //1. 迭代Values:
14         //map.values() 返回全部值的集合, 可以迭代值
15         Collection<Integer> values = map.values();
16         int all = 0;
17         Iterator<Integer> ite = values.iterator();
18         while(ite.hasNext()){
19             int i = ite.next(); //使用泛型, 可以自动转换类型
20             all+=i;
21         }
22         System.out.println(all == str.length()); //true
23
24         //2. 迭代key:
25         //map.keySet() 返回key的Set类型集合, 显示统计表格
26         Set<Character> keys = map.keySet();
27         for (Iterator i = keys.iterator(); i.hasNext();) {
28             char c = (Character) i.next(); //没有泛型, 强制转换
29             int count = map.get(c); //自动拆包
30             System.out.println(
31                 c + "\t"
32                 + count + "\t" //字符统计
33                 + ((double) count/all*100) //出现频率(百分比)
34             );
35         }
36     }
37
38     public static Map<Character, Integer> count(String s){
39         Map<Character, Integer> map =
40             new HashMap<Character, Integer>();
41         for(int i=0; i<s.length(); i++){
42             char c = s.charAt(i);
43             Integer count = map.get(c);
44             count = count==null?1:count+1;
45             map.put(c, count);
46         }
47         return map;
48     }
49 }
```



Console

<terminated> CharCounterDemo (1) [Java Application] C:\Program Files\Java\jdk1.6.0\_06\bin\javaw.exe (Nov 3, 2011 2:5

true		
醒	1	4.545454545454546
脸	1	4.545454545454546
!	1	4.545454545454546
出	1	4.545454545454546
学	1	4.545454545454546
,	4	18.181818181818183
吃	1	4.545454545454546
发	1	4.545454545454546
我	5	22.727272727272727
到	1	4.545454545454546
校	1	4.545454545454546
了	1	4.545454545454546
洗	1	4.545454545454546
早	1	4.545454545454546
餐	1	4.545454545454546

【案例 3】将 map 中统计出来的次数，排序后迭代打印

```
CopyOfCharCounterDemo.java
1 package corejava.day07.ch06;
2 import java.util.ArrayList;
10
11 public class CopyOfCharCounterDemo {
12     public static void main(String[] args) {
13         String str =
14             "我醒了,我洗脸,我吃早餐,我出发,我到学校!";
15         Map<Character, Integer> map = count(str);
16
17         //3. 排序
18         //3.1 返回Entry类型(集合) Entry即键值对[key:value]
19         Set<Entry<Character, Integer>>
20             entrySet = map.entrySet();
21         //3.2 复制为List集合(便于排序)
22         ArrayList<Entry<Character, Integer>> list =
23             new ArrayList<Entry<Character, Integer>>(entrySet);
24         //3.3 按指定规则排序
25         Collections.sort(list, new ByValue());
26         //3.4 迭代输出排序以后的结果
27         for (Iterator<Entry<Character, Integer>>
28             i = list.iterator();
29             i.hasNext(); ) {
30             //每个元素是entry类型
```

```

31     Entry<Character, Integer> entry = i.next();
32     //entry 的 getKey() 返回是key(字符)
33     //entry 的 getValue() 返回的是Value(字符的统计结果)
34     int count = entry.getValue();
35     System.out.println(
36         entry.getKey() + "\t"
37         + count + "\t" +
38         (double) count/str.length()*100
39     );
40 }
41 }
42
43 public static Map<Character, Integer> count(String s){
44     Map<Character, Integer> map =
45         new HashMap<Character, Integer>();
46     for(int i=0; i<s.length(); i++){
47         char c = s.charAt(i);
48         Integer count = map.get(c);
49         count = count==null?1:count+1;
50         map.put(c, count);
51     }
52     return map;
53 }
54 }
55
56 /**自定义的排序比较规则，按照value 大小比较*/
57 class ByValue implements Comparator<Entry>{
58     public int compare(Entry o1, Entry o2) {
59         return
60             (Integer)o2.getValue() - (Integer)o1.getValue();
61     }
62 }

```

注：

- ✓ **Map.Entry** 类型     Entry 就是键值对[key : value]，通过 map.entrySet()方法获得
- ✓ class ByValue{}最好写为静态内部类

输出结果

```

Console
<terminated> CopyOfCharCounterDemo [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Nov 3, 2011 3
我      5      22.727272727272727
,      4      18.181818181818183
醒      1      4.545454545454546
脸      1      4.545454545454546
!      1      4.545454545454546
出      1      4.545454545454546
学      1      4.545454545454546
吃      1      4.545454545454546
发      1      4.545454545454546
到校  1      4.545454545454546
了    1      4.545454545454546
洗    1      4.545454545454546
早餐  1      4.545454545454546

```

## 2. 内部类

1) 根据位置的不同，Java 中的内部类分为四种：

### ■ 静态内部类

- ◆ 使用 static 修饰，声明在类体中
- ◆ 静态内部类中可以访问外部类的静态成员

### ■ 成员内部类

- ◆ 声明在类体中，不使用 static，具有类的成员特征，也就是，必须有类的实例才能创建内部类实例
- ◆ 内部类实例可以访问共享外部类的成员变量（很常用）
- ◆ 如：链表的节点就可以定义为内部类

### ■ 局部内部类 把类声明在方法中，就是局部内部类，作用域

- ◆ 类似局部变量（很少见）

### ■ 匿名内部类

- ◆ 匿名类，非常常见，可以写在任何地方，就像一般的语句
- ◆ 语法更象是创建对象：Date d = new Date(){//...};
- ◆ 匿名类是对原类的一个继承，同时创建了实例，{} 就是继承以后的类体类体中可使用所有类的语法
- ◆ 匿名类不能写构造器
- ◆ 匿名类可以从抽象类或者接口继承，必须提供抽象方法的实现

2) 任何内部类都编译成独立的 class 文件

3) 最大的作用：封装！

- 匿名类可以封装内部概念：情侣间的“土豆”和菜市场的“土豆”是不一样的

- 4) 通过实例简单理解记忆语法，在今后的案例中灵活练习内部类

## 2.1. 静态内部类 \*

### 【案例 1】静态内部类的创建与调用

#### ● 版本 01

```
package corejava.day07.inner;
import corejava.day07.inner.Foo.Koo;
/** 静态内部类演示 */
public class StaticInnerClassDemo {
    public static void main(String[] args) {
        Koo koo = new Koo();
    }
}
class Foo{
    /** 静态内部类
     * Foo 类相当于Koo的包，为Koo声明了一个命名空间
     * 静态内部类的作用域，类似于静态变量，类加载以后就存在
     * 可以在静态内部类中访问静态成员(属性和方法)
     * */
    static class Koo{
    }
}
```

#### ● 版本 02

```
StaticInnerClassDemo.java
1 package corejava.day07.ch07;
2 /** 静态内部类演示 */
3 public class StaticInnerClassDemo {
4     public static void main(String[] args) {
5         Foo.Koo koo = new Foo.Koo();
6         System.out.println(koo.add()); //3
7     }
8 }
```

```

9 class Foo{
10     int a = 1;
11     static int b = 2;
12     /** 静态内部类，好处：将Koo 封装到 Foo内部
13      * Foo 类相当于Koo的包，为Koo声明了一个命名空间
14      * 静态内部类的作用域类似静态变量，类加载以后就存在
15      * 可以在静态内部类中访问静态成员(属性和方法)
16      */
17     static class Koo{
18         int add(){
19             //return a+b; //编译异常
20             return b+1;
21         }
22     }
23 }

```

#### 【案例 2】静态内部类实现比较规则

```

1 package corejava.day07.ch08;
2 import java.util.Arrays;
3 import java.util.Comparator;
4 /** 静态内部类演示 */
5 public class StaticInnerClassDemo {
6     public static void main(String[] args) {
7         String[] names = {"郭", "郭美", "郭美美"};
8         Arrays.sort(names, new ByLength()); //自定义排序
9         System.out.println(Arrays.toString(names));
10    }
11    //内部类实现的内部比较规则
12    static class ByLength implements Comparator<String>{
13        public int compare(String o1, String o2) {
14            return o2.length() - o1.length();
15        }
16    }
17 }

```

Console

<terminated> StaticInnerClassDemo [Java Application] C:\...jdk1.6.0\_06\bin\javaw.exe (Nov 3)

[郭美美, 郭美, 郭]

按长度排序

## 2.2. 成员内部类 \*

### 【案例】成员内部类演示

```

1 package corejava.day07.ch09;
2 import corejava.day07.ch09.Goo.Moo;
3 /** 成员内部类演示 */
4 public class InnerClassDemo {
5     public static void main(String[] args) {
6         //1. 在Goo创建之后, 才有Moo, 否则会出编译错误
7         //Moo moo = new Moo(); //编译错误
8
9         //2. 创建成员内部类
10        Goo goo = new Goo();
11        Moo moo = goo.new Moo(); //必须使用goo实例创建Moo实例
12
13        //3. goo创建的Moo实例可以访问goo实例的属性
14        Goo goo2 = new Goo();
15        goo2.a = 2;
16        Moo moo2 = goo2.new Moo();
17        System.out.println(moo.add() + "," + moo2.add());
18    }
19 }
20
21 class Goo{
22     int a = 1;
23     static int b = 2;
24     /**成员内部类: 不使用static说明的内部类,
25      * 成员内部类与实例变量具有相同的作用域 */
26     class Moo{ //成员内部类
27         int add(){
28             return a+b; //成员内部类优点: 可以共享成员变量
29         }
30     }
31 }

```

## 2.3. 局部内部类 \*

在方法内临时的创建一些类（只想在方法内部用）时，可以使用局部内部类

## 【案例】局部内部类演示

```

1 package corejava.day07.ch10;
2 import java.util.Arrays;
3 import java.util.Comparator;
4 /** 局部内部类演示 */
5 public class LocalInnerClassDemo {
6     public static void main(String[] args) {
7         //1. 可以共享访问局部变量，可局部变量必须final的
8         final int a = 2;
9         /** 局部内部类:
10          * 在方法中声明的内部类，作用域类似于局部变量 */
11         class Foo{ //局部内部类
12             int b = 1;
13             public int add(){
14                 return a+b; //a是final的局部变量
15             }
16         }
17         Foo f = new Foo();
18         System.out.println(f.add()); //3
19
20         //2. 按照字符串最后一个字母排序
21         String[] names = {"Andy", "Tom", "John", "Mac"};
22         class ByLastChar implements Comparator<String>{
23             public int compare(String o1, String o2) {
24                 return o1.charAt(o1.length()-1) -
25                     o2.charAt(o2.length()-1);
26             }
27         }
28         Arrays.sort(names, new ByLastChar());
29         System.out.println(Arrays.toString(names));
30     }
31 }

```

## 2.4. 匿名内部类 \*\*

匿名内部类，简称匿名类，比较常用。

```
package corejava.day07.inner;
/** 匿名内部类演示 */
public class AnnInnerClassDemo {
    public static void main(String[] main){
        Xoo xoo = new Xoo(){}; //匿名内部类
    }
}

class Xoo{
}
```

注：

✓ " {} "为类体，只有类体，没有类名

#### 【案例 1】匿名内部类演示

```
AnnInnerClassDemo.java
1 package corejava.day07.ch11;
2 /** 匿名内部类演示 */
3 public class AnnInnerClassDemo {
4     public static void main(String[] main){
5         //1. 匿名内部类new Xoo(){}是对Xoo的继承,并同时实例化
6         // 也可以说, new Xoo(){} 是Xoo子类实例, 是一个对象
7         Xoo xoo = new Xoo(){};
8
9         //2. "{}", 类体中可以声明大部分类的功能
10        // 比如覆盖该类的toString()方法
11        Xoo xool = new Xoo(){
12            @Override
13            public String toString() {
14                return "Hi , I am xool";
15            }
16        };
17        System.out.println(xoo);
18        System.out.println(xool);
19    }
20 }
21
22 class Xoo{ }
23
```

#### 【案例 2】匿名内部类可以继承(实现)自抽象类、接口



```

1 package corejava.day07.ch11;
2 /** 匿名内部类演示 */
3 public class AnnInnerClassDemo {
4     public static void main(String[] main){
5         //1. 匿名类是一种特殊的局部内部类
6         final int a = 2;
7
8         //2. 匿名类可以从类继承匿名类
9         // 还可以从抽象类,接口继承(实现)匿名类
10        //2.1 从抽象类继承的匿名类实例
11        Yoo yoo = new Yoo(){
12            public int add() {
13                return a+1;//可以访问final局部变量
14            }
15        };
16        System.out.println(yoo.add());    // 3
17
18        //2.2 从接口继承(实现)的匿名类实例
19        Koo koo = new Koo(){
20            public int add() {
21                return a+1;
22            }
23        };
24        System.out.println(koo.add());    // 3
25    }
26 }
27
28 interface Koo{ int add(); }
29
30 abstract class Yoo{ abstract int add(); }
31

```

注：

- ✓ 第 12 行代码将继承的方法访问控制范围修改为 public  
Java 中的继承的方法范围可以放大，不能缩小

### 【案例 3】自定义比较 Card 大小

- 版本 01
  - Card.java
  - 请参考 day04 代码(略)
  - AnnInnerClassDemo.java

```
AnnInnerClassDemo03.java  AnnInnerClassDemo04.java
1 package corejava.day07.ch11;
2 import java.util.Arrays;
3
4 /** 匿名内部类演示 */
5 public class AnnInnerClassDemo03 {
6     public static void main(String[] main){
7         Card[] cards = {new Card(Card.SPADE, Card.TEN),
8                         new Card(Card.SPADE, Card.KING),
9                         new Card(Card.SPADE, Card.THREE),
10                        new Card(Card.SPADE, Card.ACE)};
11        //1. 自定义规则的比较器 (局部内部类)
12        Comparator<Card> byRank = new Comparator<Card>() {
13            public int compare(Card o1, Card o2) {
14                return o1.getRank() - o2.getRank();
15            }
16        };
17        Arrays.sort(cards, byRank);
18        System.out.println(Arrays.toString(cards));
19    }
20 }
```

- 版本 02

- Card.java

请参考 day04 代码(略)

- AnnInnerClassDemo.java

```
AnnInnerClassDemo04.java
1 package corejava.day07.ch11;
2 import java.util.Arrays;
3
4 /** 匿名内部类演示 */
5 public class AnnInnerClassDemo04 {
6     public static void main(String[] main){
7         Card[] cards = {new Card(Card.SPADE, Card.TEN),
8                         new Card(Card.SPADE, Card.KING),
9                         new Card(Card.SPADE, Card.THREE),
10                        new Card(Card.SPADE, Card.ACE)};
11        //2. 常用写法 (匿名内部类)
12        Arrays.sort(cards, new Comparator<Card>() {
13            public int compare(Card o1, Card o2) {
14                return o1.getRank() - o2.getRank();
15            }
16        });
17        System.out.println(Arrays.toString(cards));
18    }
19 }
```

#### 【案例 4】贪吃蛇\_03

**练习描述：**增加贪吃蛇吃豆豆的功能

**修改步骤：**

**第 1 步** 将蛇 (Worm) 作为面板 (WormPane) 的内部类

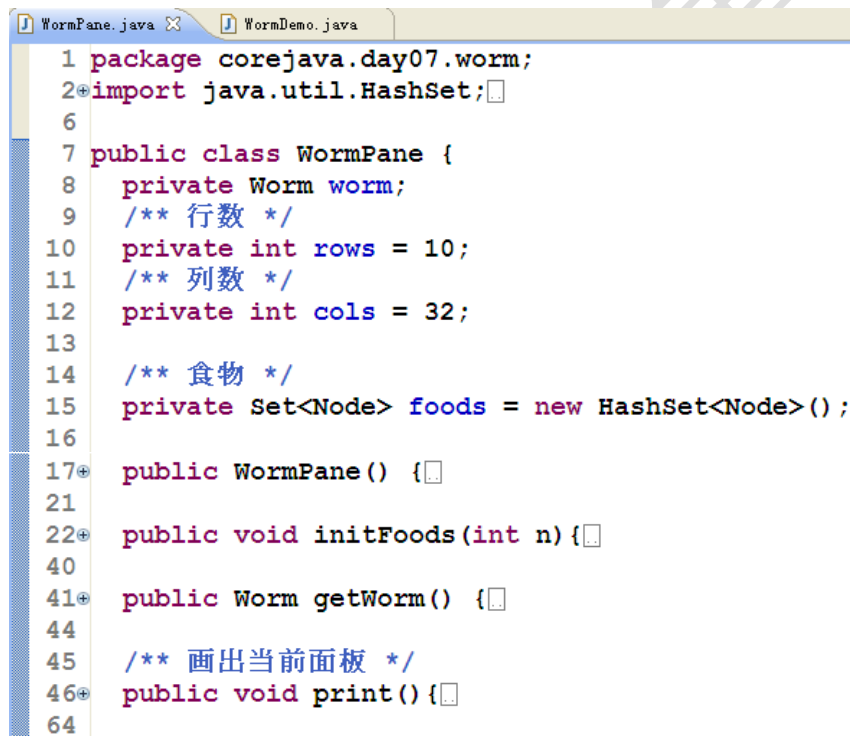
**第 2 步** 插入新节点到头部

**第 3 步** 在 WormDemo 中, import WormPane.Worm

- **Node.java**

请参考 day05-day06 代码 (略)

- **WormPane.java**



```

1 package corejava.day07.worm;
2 import java.util.HashSet;
6
7 public class WormPane {
8     private Worm worm;
9     /** 行数 */
10    private int rows = 10;
11    /** 列数 */
12    private int cols = 32;
13
14    /** 食物 */
15    private Set<Node> foods = new HashSet<Node>();
16
17    public WormPane() {}
21
22    public void initFoods(int n) {}
40
41    public Worm getWorm() {}
44
45    /** 画出当前面板 */
46    public void print() {}
64
    
```

```

65 public class Worm {
66     //<Node> 约束了集合中元素的类型, nodes中只能放置Node实例
67     private LinkedList<Node> nodes =
68         new LinkedList<Node>();
69     //当前默认的行走方向
70     private int dir;
71
72     public static final int UP = -10;
73     public static final int DOWN = 10;
74     public static final int LEFT = -1;
75     public static final int RIGHT = 1;
76
77
78     public Worm() {}
79
80     public Worm(LinkedList<Node> nodes) {}
81
82     /** 走一步 */
83     public void step() {
84         //找到头节点
85         Node head = nodes.getFirst(); //相当于: get(0)
86         //更具当前方向计算新节点
87         int i = head.getI() + dir/10;
88         int j = head.getJ() + dir%10;
89         head = new Node(i, j);
90         //第2步
91         //插入新节点到头部
92         nodes.addFirst(head); //相当于: add(0, head)
93         if (foods.contains(head)) { //吃到食物
94             foods.remove(head);
95             return; //就不删除节点
96         }
97         //删除末尾节点
98         nodes.removeLast(); //相当于: remove(nodes.size()-1)
99     }
100
101     /** 换个方向走一步 */
102     public void step(int dir) {}
103
104     public boolean contains(int i, int j) {
105         return nodes.contains(new Node(i, j));
106     }
107
108     public String toString() {}
109 }
110
111
112

```

● WormDemo.java

```

1 package corejava.day07.worm;
2 import java.util.Scanner;
3 //第3步：import 导入 WormPane.Worm
4 import corejava.day07.worm.WormPane.Worm;
5
6 public class WormDemo {
7     public static void main(String[] args) {
8
9     }
10 }

```

- 运行结果：蛇可以吃豆豆了

```

| ##### |
| #      |
| #      |
| #      |
| 0      |
|          0
|
| ##### |
[[4,2], [3,2], [2,2], [1,2], [1,3], [1,4], [1,5], [1,6], [1,7], [1,8]]

```