

|* 今天学习目标

- * 能够理解泛型概述
- * 能够掌握泛型类
 - * 箱
- * 能够掌握泛型方法
- * 能够掌握含有接口的泛型
- * 能够理解泛型通配符
- * 能够理解泛型上下限
 - extends super
 - * 抽出公共的方法
- * 能够使用集合完成综合案例
 - * 斗地主

-
- * 能够理解泛型概述
 - * 能够掌握泛型类
 - * 能够掌握泛型方法
 - * 能够掌握含有接口的泛型
 - * 能够理解泛型通配符
 - * 能够理解泛型上下限
 - * 能够使用集合完成综合案例

-
- * 回顾
 - * 自定义异常：extends Exception , RuntimeException , 重写构造器
 - * 日志：log4j (properties) , log4j2 (xml)
 - * 级别，输出地，格式，配置文件
 - * Logger(trace,debug,info,warn,error)

- * Appender(ConsoleAppender,FileAppender,.....)
- * Layout(PatternLayout,HTMLLayout,SimpleLayout)
- * 集合：容器--->引用类型
- * 与数组的区别：数组的长度不可变，集合的长度可变
- * Collection,List(ArrayList,Vector,LinkedList),Set(HashSet(HashMap的key), LinkedHashSet, TreeSet),Queue
- * 多线程并发集合
- * add,remove,contains,clear,isEmpty,addAll,removeAll,containsAll,toArray,size
- * Iterable---> Iterator(hasNext,next)
- * NoSuchElementException
- * 增强型for (JDK1.5)，迭代过程中，不可以删除或者增加元素
- * ConcurrentModificationException
- * 能够理解泛型概述
- * 案例

```

1 * 没有加上泛型
2 public static void main(String[] args) {
3     Collection container=new ArrayList();
4     // add:东邪西毒南帝北丐中神通
5     container.add("东邪");
6     container.add(2);
7     container.add("西毒");
8     container.add("南帝");
9     container.add("北丐");
10    container.add("中神通");
11    Iterator<String> ite = container.iterator();
12    while(ite.hasNext()) {
13        String value=ite.next();
14        System.out.println(value);
15    }
16 }
17 结果并报异常:
18 东邪

```

```

19 Exception in thread "main" java.lang.ClassCastException:
20     java.lang.Integer cannot be cast to java.lang.String
21     at com.lg.test1.Test1.main(Test1.java:19)
22
23 * 泛型:
24     Collection虽然可以存储各种对象,但实际上通常Collection只存储同一类型对象,例如String
25     因此在JDK5之后,新增了泛型(Generic)语法,让我们在设计API时可以指定类或方法支持泛型,
26     在编译时期进行语法检查,运行时擦除泛型。
27 * 什么是泛型?
28     * 可以在类或方法中预支地使用未知的类型。
29     * 用来灵活地将数据类型应用到不同的类、方法、接口当中,将数据类型作为参数进行传递。
30 * 添加泛型
31     public static void main(String[] args) {
32         Collection<String> container=new ArrayList<String>();
33         // add:东邪西毒南帝北丐中神通
34         container.add("东邪");
35         // container.add(2); // 编译期就报错了
36         container.add("西毒");
37         container.add("南帝");
38         container.add("北丐");
39         container.add("中神通");
40         Iterator<String> ite = container.iterator();
41         while(ite.hasNext()) {
42             String value=ite.next();
43             System.out.println(value);
44         }
45     }
46 * 泛型好处
47     * 将运行时期的ClassCastException,转移到了编译时期变成了编译失败。
48     * 避免了类型强转的麻烦

```

* 能够掌握泛型类

* 泛型的意义在于代码的复用

* 定义格式：

* 修饰符 class 类名<代表泛型的变量> {}

* 查看ArrayList源码

* 盒子案例

```
1 * 装黄金的盒子
2 public class Gold {
3     @Override
4     public String toString() {
5         return "黄金";
6     }
7
8     public void light() {
9         System.out.println("金闪闪...");
10    }
11 }
12
13 public class GoldBox {
14     private Gold gold;
15
16     public Gold getGold() {
17         return gold;
18     }
19
20     public void setGold(Gold gold) {
21         this.gold = gold;
22     }
23
24 }
25
26 public static void main(String[] args) {
27     // 黄金的盒子
28     GoldBox boxGold=new GoldBox();
29     boxGold.setGold(new Gold());
30     System.out.println(boxGold.getGold());
31     boxGold.getGold().light();
32 }
33 结果:
34 黄金
35 金闪闪...
36
37 * 装硬币的盒子
```

```

38 public class Coin {
39
40     @Override
41     public String toString() {
42         return "硬币";
43     }
44
45     public void buy() {
46         System.out.println("拿着一块买两包辣条...");
47     }
48 }
49
50 public class CoinBox {
51     private Coin coin;
52
53     public Coin getCoin() {
54         return coin;
55     }
56
57     public void setCoin(Coin coin) {
58         this.coin = coin;
59     }
60
61 }
62
63 public static void main(String[] args) {
64     // 硬币的盒子
65     CoinBox coinBox=new CoinBox();
66     coinBox.setCoin(new Coin());
67     System.out.println(coinBox.getCoin());
68     coinBox.getCoin().buy();
69 }

```

70 结果:

71 硬币

72 拿着一块买两包辣条...

73

74 思考: 假如我要装其他的东西, 是不是每次都要创建各种类型的箱子?

75 * 用Object做个通用的箱子

```

76 public class ObjectBox {
77     private Object boj;

```

```

78
79     public Object getBoj() {
80         return boj;
81     }
82
83     public void setBoj(Object boj) {
84         this.boj = boj;
85     }
86
87 }
88
89 public static void main(String[] args) {
90     // 黄金的盒子
91     ObjectBox boxGold=new ObjectBox();
92     boxGold.setBoj(new Gold());
93     Gold gold=(Gold) boxGold.getBoj();// 有可能异常
94     System.out.println(gold);
95     gold.light();
96     System.out.println("-----");
97     // 硬币的盒子
98     ObjectBox coinGold=new ObjectBox();
99     coinGold.setBoj(new Coin());
100    Coin coin=(Coin) coinGold.getBoj();
101    System.out.println(coin);
102    coin.buy();
103 }
104 结果:
105 黄金
106 金闪闪...
107 -----
108 硬币
109 拿着一块买两包辣条...
110
111 思考:
112 * 使用Object作为箱子有什么问题?
113 * 类型强转比较麻烦
114 * 就像我们集合一样什么类型都可以装,在获取对象时候, 容易出现ClassCastException,
115
116 使用泛型类
117 public class Box<T> {

```

```

118     private T item;
119     public T getItem() {
120         return item;
121     }
122     public void setItem(T item) {
123         this.item = item;
124     }
125 }
126 public static void main(String[] args) {
127     // 黄金的盒子
128     Box<Gold> boxGold=new Box<Gold>();
129     boxGold.setItem(new Gold());
130     Gold gold=boxGold.getItem();
131     System.out.println(gold);
132     gold.light();
133     System.out.println("-----");
134     // 硬币的盒子
135     Box<Coin> coinGold=new Box<Coin>();
136     coinGold.setItem(new Coin());
137     Coin coin=coinGold.getItem();
138     System.out.println(coin);
139     coin.buy();
140 }
141 结果:
142 黄金
143 金闪闪...
144 -----
145 硬币
146 拿着一块买两包辣条...
147
148 总结: 泛型可以提高代码的复用性
149

```

* 能够掌握泛型方法

* 定义格式:

* 修饰符 <代表泛型的变量> 返回值类型 方法名(参数){ }

* 特点: 调用方法时, 确定泛型的类型

```

1  /**
2   * @author xiaozhao
3   * 泛型方法
4   */
5  public class Container {
6
7      public <E> void add(E element) {
8          System.out.println(element.getClass());
9      }
10 }
11 public static void main(String[] args) {
12     Container container=new Container();
13     container.add("123");
14     container.add(1);
15     container.add(1.8);
16 }
17 结果:
18 class java.lang.String
19 class java.lang.Integer
20 class java.lang.Double
21

```

* 能够掌握含有接口的泛型

* 定义格式：

* 修饰符 interface接口名<代表泛型的变量> {}

```

1  * 定义类时确定泛型的类型
2  public interface Collection<E> {
3      void add(E e);
4      E get();
5  }
6
7  public class CollectionImpl implements Collection<String>{
8      private String e;
9      @Override

```



```
10     public void add(String e) {
11         this.e=e;
12     }
13
14     @Override
15     public String get() {
16         return this.e;
17     }
18 }
19 public static void main(String[] args) {
20     Collection<String> c=new CollectionImpl();
21     c.add("xiaohei");
22     System.out.println(c.get());
23 }
24 结果:
25 xiaohei
26
27 * 始终不确定泛型的类型，直到创建对象时，确定泛型的类型
28 public interface Collection<E> {
29     void add(E e);
30     E get();
31 }
32 public class CollectionImpl2<E> implements Collection<E>{
33     private E e;
34     @Override
35     public void add(E e) {
36         this.e=e;
37     }
38
39     @Override
40     public E get() {
41         return this.e;
42     }
43 }
44 public static void main(String[] args) {
45     Collection<String> c=new CollectionImpl2<String>();
46     c.add("xiaohei");
47     System.out.println(c.get());
48
49     Collection<Integer> c1=new CollectionImpl2<Integer>();
```

```
50         c1.add(2);
51         System.out.println(c1.get());
52     }
53     结果:
54     xiaohei
55     2
```

* 能够理解泛型通配符

* 当使用泛型类或者接口时，传递的数据中，泛型类型不确定，可以通过通配符<?>表示

* 注意：泛型是没有继承关系的

* `Collection<Object> list = new ArrayList<String>();` 这种写法是错的

```
1  * 演示泛型是没有继承关系的
2  public class Parent {
3  }
4  public class Son extends Parent{
5  }
6  public class BoxUtils {
7      public static void setBox(Box<Parent> box) {
8          System.out.println(box);
9      }
10 }
11 public static void main(String[] args) {
12     Box<Parent> box=new Box<Parent>();
13     BoxUtils.setBox(box);
14     Box<Son> box1=new Box<Son>();
15     // BoxUtils.setBox(box1);// 编译时候出错
16 }
17
18 * 改为通配符
19 public class BoxUtils {
20     public static void setBox(Box<?> box) {
21         System.out.println(box);
22     }
23 }
```

```

24 public static void main(String[] args) {
25     Box<Parent> box=new Box<Parent>();
26     BoxUtils.setBox(box);
27     Box<Son> box1=new Box<Son>();
28     BoxUtils.setBox(box1);// 编译没出错
29 }
30
31

```

* 能够理解泛型上下限

* JAVA的泛型中可以指定一个泛型的上限和下限

* 泛型的上限：

* 格式：类型名称 <? extends 类> 对象名称

* 意义：只能接收该类型及其子类

* 泛型的下限：

* 格式：类型名称 <? super类> 对象名称

* 意义：只能接收该类型及其父类

```

1 public static void main(String[] args) {
2     // Object String Integer Number (Number是Integer父类)
3     java.util.Collection<Object> c1=new ArrayList<Object>();
4     java.util.Collection<String> c2=new ArrayList<String>();
5     java.util.Collection<Integer> c3=new ArrayList<Integer>();
6     java.util.Collection<Number> c4=new ArrayList<Number>();
7
8     // 上限:只能接收该类型及其子类
9     // getCollExtends(c1);// 出错 Object
10    // getCollExtends(c2);// 出错String
11    // getCollExtends(c3);// Integer
12    // getCollExtends(c4);// Number
13

```

```

14      // 下限:只能接收该类型及其父类
15      getCollSuper(c1);//Object
16  //      getCollSuper(c2);// 出错 String
17  //      getCollSuper(c3);// 出错Integer
18      getCollSuper(c4);//Number
19  }
20
21      public static void getCollExtends(java.util.Collection<? extends Number> co
22
23  }
24
25      public static void getCollSuper(java.util.Collection<? super Number> coll)
26
27  }

```

* 统计一个数组中大于某个特定元素的个数

```

1  * Integer
2      * Utils
3      public static int statisticsGreaterCount(Integer[] items,Integer item) {
4          int count=0;
5          for(int var:items) {
6              if(var>item) {
7                  count++;
8              }
9          }
10         return count;
11     }
12     public static void main(String[] args) {
13         Integer[] arr= {1,68,9,8,120};
14         int count = Utils.statisticsGreaterCount(arr, 10);
15         System.out.println(count);
16     }
17     结果:
18     2
19 * Gold

```

```
20 public class Gold{
21     private double weight;
22
23     public Gold() {
24         super();
25     }
26
27     public Gold(double weight) {
28         super();
29         this.weight = weight;
30     }
31
32     @Override
33     public String toString() {
34         return "黄金";
35     }
36
37     public void light() {
38         System.out.println("金闪闪...");
39     }
40
41     public double getWeight() {
42         return weight;
43     }
44
45     public void setWeight(double weight) {
46         this.weight = weight;
47     }
48
49 }
50 Utils:
51     public static int statisticsGreaterCount(Gold[] items,Gold item) {
52         int count=0;
53         for(Gold var:items) {
54             if(var.getWeight(>item.getWeight())) {
55                 count++;
56             }
57         }
58         return count;
59     }
```

```
60 public static void main(String[] args) {
61     Gold[] golds=new Gold[5];
62     for (int i = 0; i < golds.length; i++) {
63         golds[i]=new Gold(i+10);
64     }
65     Gold gold=new Gold(11);
66     int count = Utils.statisticsGreaterCount(golds, gold);
67     System.out.println(count);
68
69 }
```

70 结果:

71 3

72

73 * Coin

74

```
75 public class Coin {
76     private int value;
77
78     public Coin() {
79         super();
80     }
81
82     public Coin(int value) {
83         super();
84         this.value = value;
85     }
86
87     @Override
88     public String toString() {
89         return "硬币";
90     }
91
92     public void buy() {
93         System.out.println("拿着一块买两包辣条...");
94     }
95
96     public int getValue() {
97         return value;
98     }
99 }
```

```

100     public void setValue(int value) {
101         this.value = value;
102     }
103 }
104 public static int statisticsGreaterCount(Coin[] items,Coin item) {
105     int count=0;
106     for(Coin var:items) {
107         if(var.getValue()>item.getValue()) {
108             count++;
109         }
110     }
111     return count;
112 }
113 public static void main(String[] args) {
114     Coin[] Coins=new Coin[5];
115     for (int i = 0; i < Coins.length; i++) {
116         Coins[i]=new Coin(i+10);
117     }
118     Coin Coin=new Coin(12);
119     int count = Utils.statisticsGreaterCount(Coins, Coin);
120     System.out.println(count);
121
122 }
123 结果:
124 2
125
126 * 思考: 可不可以写个公共的方法复用代码 (利用泛型)
127 public class Utils2 {
128     // 思考: 可不可以写个公共的方法 (利用泛型)
129     public static <T extends Comparable<T>> int statisticsGreaterCount(T[] item
130         int count=0;
131         for(T var:items) { //边界符号
132             if(var.compareTo(item)>0) { // short, int, double, long, float, byte
133                 count++;
134             }
135         }
136         return count;
137     }
138 }
139 Gold

```

```

140 * 实现Comparable接口
141     public class Gold implements Comparable<Gold>
142 * 实现compareTo方法
143 @Override
144     public int compareTo(Gold o) {
145         return (int) (weight-o.weight);
146     }
147 Coin:
148 * 实现Comparable接口
149     public class Coin implements Comparable<Gold>
150 * 实现compareTo方法
151 @Override
152     public int compareTo(Coin o) {
153         return value-o.value;
154     }
155 * 测试：工具类Utils2 --> 把Utils改成Utils2测试结果
156 结果：效果一样
157

```

* 能够使用集合完成综合案例

* 斗地主案例

* 54张牌打乱顺序

* 三个玩家参与游戏

* 三人交替摸牌

* 每人17张牌

* 最后三张留作底牌

- 1 * 思路
- 2 * 准备一副牌
- 3 * 洗牌
- 4 * 发牌
- 5 * 创建三个玩家（每个玩家17张牌，每人依次摸1张）
- 6 * 创建底牌集合（最后三张牌）
- 7 * 查看牌


```
8 * 叫地主
9 * 实现
10 * 实现
11 /**
12  * 牌
13 */
14 public class Card {
15     /**
16      * 花色
17      */
18     private String color;
19     /**
20      * 数字
21      */
22     private String number;
23
24
25     public Card() {
26         super();
27     }
28     public Card(String color, String number) {
29         super();
30         this.number = number;
31         this.color = color;
32     }
33     public String getNumber() {
34         return number;
35     }
36     public void setNumber(String number) {
37         this.number = number;
38     }
39     public String getColor() {
40         return color;
41     }
42     public void setColor(String color) {
43         this.color = color;
44     }
45     @Override
46     public int hashCode() {
47         final int prime = 31;
```

```

48         int result = 1;
49         result = prime * result + ((color == null) ? 0 : color.hashCode());
50         result = prime * result + ((number == null) ? 0 : number.hashCode());
51         return result;
52     }
53     @Override
54     public boolean equals(Object obj) {
55         if (this == obj)
56             return true;
57         if (obj == null)
58             return false;
59         if (getClass() != obj.getClass())
60             return false;
61         Card other = (Card) obj;
62         if (color == null) {
63             if (other.color != null)
64                 return false;
65         } else if (!color.equals(other.color))
66             return false;
67         if (number == null) {
68             if (other.number != null)
69                 return false;
70         } else if (!number.equals(other.number))
71             return false;
72         return true;
73     }
74     @Override
75     public String toString() {
76         return (color!=null)?color+number:number;
77     }
78
79 }
80
81 /**
82  *
83  * 玩家
84  */
85 public class Player {
86     private String name;
87     public List<Card> cards=new ArrayList<Card>();

```

```
88     public String getName() {
89         return name;
90     }
91     public void setName(String name) {
92         this.name = name;
93     }
94 }
95
96
97 /**
98  *
99  * 生成牌盒的工具类
100  */
101 public class CardUtils {
102     public static final String[] COLORS= {"♠", "♥", "♣", "♦"};
103     public static final String[] NUMBERS= {"2", "3", "4", "5", "6", "7", "8", "9", "10",
104     /**
105      * 获取一副牌
106      * @return
107      */
108     public static List<Card> getCards(){
109         List<Card> cards=new ArrayList<Card>();
110         for (int i = 0; i < COLORS.length; i++) { // 4
111             for (int j = 0; j < NUMBERS.length; j++) { //13
112                 Card card=new Card(COLORS[i],NUMBERS[j]);
113                 cards.add(card);
114             }
115         }
116         // 大小王
117         Card sCard=new Card();
118         sCard.setNumber("小王");
119         Card bCard=new Card();
120         bCard.setNumber("大王");
121         cards.add(sCard);
122         cards.add(bCard);
123         return cards;
124     }
125 }
126
127 public class Client {
```

```
128     public static void main(String[] args) {
129         // 1 准备一副牌
130         List<Card> cards = CardUtils.getCards();
131         System.out.println("新牌: ");
132         System.out.println(cards);
133
134         // 2 洗牌
135         Collections.shuffle(cards);
136         System.out.println("洗完牌: ");
137         System.out.println(cards);
138
139         // 3 发牌
140         // 创建3个玩家
141         Player player1=new Player();
142         player1.setName("刘备");
143         Player player2=new Player();
144         player2.setName("张飞");
145         Player player3=new Player();
146         player3.setName("关羽");
147         // 创建底牌盒子
148         List<Card> finalCards=new ArrayList<Card>();
149         for (int i = 0; i < cards.size(); i++) {
150             Card card=cards.get(i);
151             if(i<51) {
152                 if(i%3==0) {
153                     player1.cards.add(card);
154                 }else if(i%3==1) {
155                     player2.cards.add(card);
156                 }else if(i%3==2) {
157                     player3.cards.add(card);
158                 }
159             }else { // 51,52,53
160                 // 最后三张给底牌
161                 finalCards.add(card);
162             }
163         }
164         // 查看牌
165         System.out.println("查看牌: ");
166         System.out.println(player1.getName()+":"+player1.cards);
167         System.out.println(player2.getName()+":"+player2.cards);
```

```

168     System.out.println(player2.getName()+":"+player3.cards);
169     System.out.println("底牌"+":"+finalCards);
170
171     // 叫地主
172     System.out.println("恭喜"+player1.getName()+"获得抢到地主");
173     player1.cards.addAll(finalCards);
174     finalCards.clear();
175     System.out.println("查看牌: ");
176     System.out.println(player1.getName()+":"+player1.cards);
177     System.out.println(player2.getName()+":"+player2.cards);
178     System.out.println(player2.getName()+":"+player3.cards);
179     System.out.println("底牌"+":"+finalCards);
180
181 }
182 }
183
184 结果:
185 新牌:
186 [♠2, ♠3, ♠4, ♠5, ♠6, ♠7, ♠8, ♠9, ♠10, ♠J, ♠Q, ♠K, ♠A, ♥2, ♥3, ♥4, ♥5, ♥6, ♥7, ♥8, ♥9, ♥10, ♥J, ♥Q, ♥K, ♥A]
187 洗完牌:
188 [♦5, ♣4, ♣K, ♣6, ♣K, ♠5, ♦6, ♦J, ♦K, ♣J, ♣J, ♥7, ♠3, 小王, ♥5, ♣7, ♠10, 大王, ♥6, ♥8, ♥9, ♥10, ♥J, ♥Q, ♥K, ♥A]
189 查看牌:
190 刘备:[♦5, ♣6, ♦6, ♣J, ♠3, ♣7, ♥J, ♣A, ♠6, ♠4, ♥A, ♠7, ♦10, ♦4, ♠3, ♥4, ♠Q]
191 张飞:[♣4, ♠K, ♦J, ♠J, 小王, ♠10, ♥3, ♥2, ♥K, ♦Q, ♥Q, ♠8, ♠9, ♠9, ♥10, ♥6, ♦7]
192 张飞:[♣K, ♠5, ♦K, ♥7, ♥5, 大王, ♠2, ♠10, ♠2, ♦2, ♦8, ♠5, ♥9, ♦3, ♥8, ♠8, ♦9]
193 底牌:[♦A, ♠A, ♠Q]
194 恭喜刘备获得抢到地主
195 查看牌:
196 刘备:[♦5, ♣6, ♦6, ♣J, ♠3, ♣7, ♥J, ♣A, ♠6, ♠4, ♥A, ♠7, ♦10, ♦4, ♠3, ♥4, ♠Q, ♦A, ♥5, ♥6, ♥7, ♥8, ♥9, ♥10, ♥J, ♥Q, ♥K, ♥A]
197 张飞:[♣4, ♠K, ♦J, ♠J, 小王, ♠10, ♥3, ♥2, ♥K, ♦Q, ♥Q, ♠8, ♠9, ♠9, ♥10, ♥6, ♦7]
198 张飞:[♣K, ♠5, ♦K, ♥7, ♥5, 大王, ♠2, ♠10, ♠2, ♦2, ♦8, ♠5, ♥9, ♦3, ♥8, ♠8, ♦9]
199 底牌:[ ]
200

```

