List

本节目标

> - 学习一个新的 java 语法泛型的使用
> - 学习一个新的 java 概念，包装类
> - `List` / `ArrayList` / `LinkedList` 的基本使用
> - ArrayList 类的使用
> - LinkedList 类的使用

1. 预备知识-泛型(Generic)

1.1 泛型的引入

问题：我们之前实现过的顺序表，只能保存 `int` 类型的元素，如果现在需要保存 指向 `Person` 类型对象的引用的顺序表，请问应该如何解决？如果又需要保存指向 `Book` 对象类型的引用呢？

回答：

1. 首先，我们在学习多态过程中已知一个前提，基类的引用可以指向子类的对象。
2. 其次，我们也已知 `Object` 是 java 中所有类的祖先类。

那么，要解决上述问题，我们很自然的想到一个解决办法，将我们的顺序表的元素类型定义成 `Object` 类型，这样我们的 `Object` 类型的引用可以指向 `Person` 类型的对象或者指向 `Book` 类型的对象了。

示例代码：

```
public class MyArrayList {
    private Object[] array;     // 保存顺序表的元素，即 Object 类型的引用
    private int size;           // 保存顺序表内数据个数

    public void add(Object o) { 尾插 }
    public Object get(int index) { 获取 index 位置的元素 }
    ...
}
```

这样，我们可以就可以很自由的存储指向任意类型对象的引用到我们的顺序表了。

示例代码：

```
MyArrayList books = new MyArrayList();
for (int i = 0; i < 10; i++) {
    books.add(new Book());  // 尾插 10 本书到顺序表
}

MyArrayList people = new MyArrayList();
for (int i = 0; i < 10; i++) {
    people.add(new Person());   // 尾插 10 个人到顺序表
}
```

遗留问题：现在的 MyArrayList 虽然可以做到添加任意类型的引用到其中了，但遇到以下代码就会产生问题。

```
MyArrayList books = new MyArrayList();
books.add(new Book);

// 将 Object 类型转换为 Person 类型，需要类型转换才能成功
// 这里编译正确，但运行时会抛出异常 ClassCastException
Person person = (Person)books.get(0);
```

提示：问题暴露的越早，影响越小。编译期间的问题只会让开发者感觉到，运行期间的错误会让所有的软件使用者承受错误风险。

所以我们需要一种机制，可以 1. 增加编译期间的类型检查 2. 取消类型转换的使用 **泛型就此诞生！**

1.2 泛型的分类

```
1. 泛型类

2. 泛型方法
```

1.3 泛型类的定义的简单演示

关于泛型类的定义，这里只是了解即可，我们重点学习泛型类的使用。

```
// 1. 尖括号 <> 是泛型的标志
// 2. E 是类型变量(Type Variable)，变量名一般要大写
// 3. E 在定义时是形参，代表的意思是 MyArrayList 最终传入的类型，但现在还不知道
public class MyArrayList<E> {
    private E[] array;
    private int size;

    ...
}
```

注意：泛型类可以一次有多个类型变量，用逗号分割。

1.4 泛型背后作用时期和背后的简单原理

1. 泛型是作用在编译期间的一种机制，即运行期间没有泛型的概念。

2. 泛型代码在运行期间，就是我们上面提到的，利用 Object 达到的效果（这里不是很准确，以后会做说明）。

1.5 泛型类的使用

```
// 定义了一个元素是 Book 引用的 MyArrayList
MyArrayList<Book> books = new MyArrayList<Book>();
books.add(new Book());

// 会产生编译错误，Person 类型无法转换为 Book 类型
books.add(new Person());

// 不需要做类型转换
Book book = book.get(0);

// 不需要做类型转换
// 会产生编译错误，Book 类型无法转换为 Person 类型
Person person = book.get(0);
```

通过以上代码，我们可以看到泛型类的一个使用方式：只需要在所有类型后边跟尖括号，并且尖括号内是真正的类型，即 E 可以看作的最后的类型。

注意： Book 只能想象成 E 的类型，但实际上 E 的类型还是 Object。

1.6 泛型总结

1. 泛型是为了解决某些容器、算法等代码的通用性而引入，并且能在编译期间做类型检查。
2. 泛型利用的是 Object 是所有类的祖先类，并且父类的引用可以指向子类对象的特定而工作。
3. 泛型是一种编译期间的机制，即 MyArrayList<Person> 和 MyArrayList<Book> 在运行期间是一个类型。
4. 泛型是 java 中的一种合法语法，标志就是尖括号 <>

2. 预备知识-包装类(Wrapper Class)

Object 引用可以指向任意类型的对象，但有例外出现了，8 种基本数据类型不是对象，那岂不是刚才的泛型机制要失效了?

实际上也确实如此，为了解决这个问题，java 引入了一类特殊的类，即这 8 种基本数据类型的包装类，在使用过程中，会将类似 int 这样的值包装到一个对象中去。

2.1 基本数据类型和包装类直接的对应关系

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

基本就是类型的首字母大写，除了 Integer 和 Character。

2.2 包装类的使用，装箱(boxing)和拆箱(unboxing)

```
int i = 10;

// 装箱操作，新建一个 Integer 类型对象，将 i 的值放入对象的某个属性中
Integer ii = Integer.valueOf(i);
Integer ij = new Integer(i);

// 拆箱操作，将 Integer 对象中的值取出，放到一个基本数据类型中
int j = ii.intValue();
```

2.3 自动装箱(autoboxing)和自动拆箱(autounboxing)

可以看到在使用过程中，装箱和拆箱带来不少的代码量，所以为了减少开发者的负担，java 提供了自动机制。

```
int i = 10;

Integer ii = i;             // 自动装箱
Integer ij = (Integer)i;    // 自动装箱

int j = ii;                 // 自动拆箱
int k = (int)ii;            // 自动拆箱
```

注意：自动装箱和自动拆箱是工作在编译期间的一种机制。

2.4 javap 反编译工具

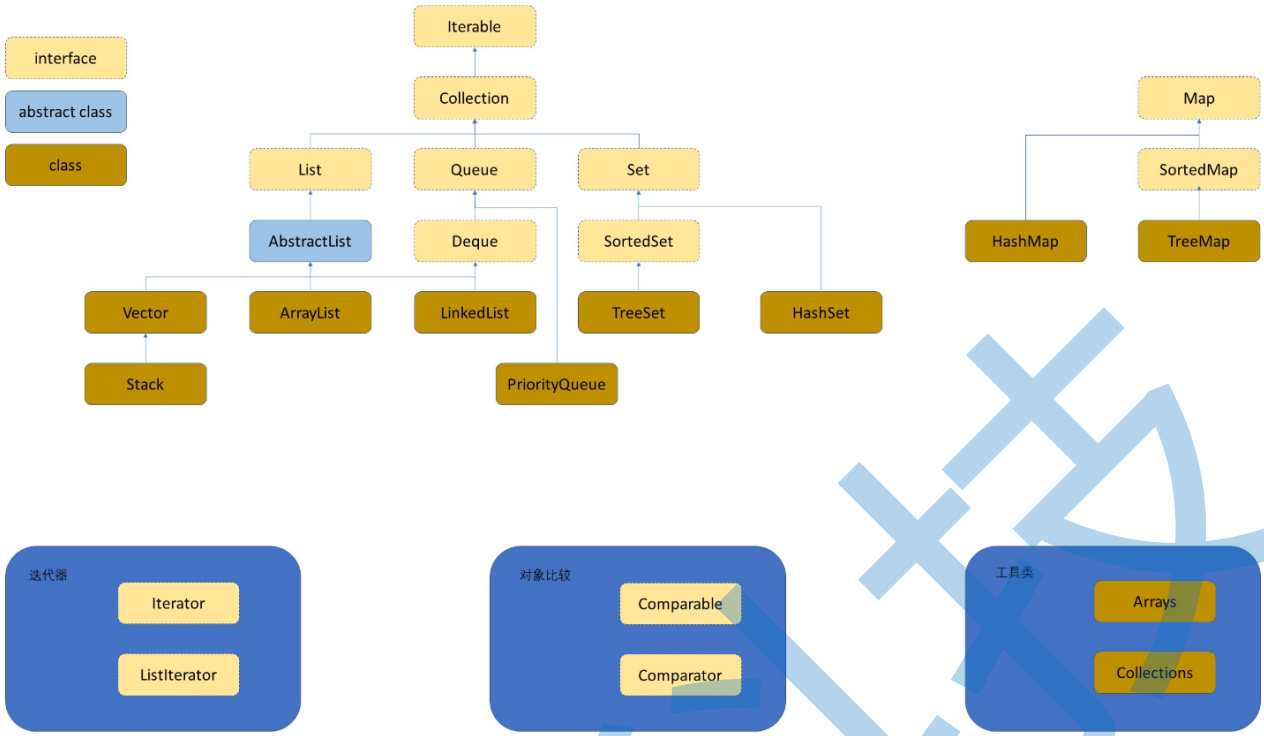
这里我们刚好学习一个 jdk 中一个反编译工具来查看下自动装箱和自动拆箱过程，并且看到这个过程是发生在编译期间的。

```
javap -c 类名称
```

```
Compiled from "Main.java"
public class Main {
  public Main();
    Code:
       0: aload_0
       1: invokespecial #1                  // Method java/lang/Object."<init>":()V
       4: return

  public static void main(java.lang.String[]);
    Code:
       0: bipush        10
       2: istore_1
       3: iload_1
       4: invokestatic  #2                  // Method java/lang/Integer.valueOf:
(I)Ljava/lang/Integer;
       7: astore_2
       8: iload_1
       9: invokestatic  #2                  // Method java/lang/Integer.valueOf:
(I)Ljava/lang/Integer;
      12: astore_3
      13: aload_2
      14: invokevirtual #3                  // Method java/lang/Integer.intValue:()I
      17: istore        4
      19: aload_2
      20: invokevirtual #3                  // Method java/lang/Integer.intValue:()I
      23: istore        5
      25: return
}
```

3. List 的使用



[List 的官方文档]

[ArrayList 的官方文档]

[LinkedList 的官方文档]

3.1 常见方法

List（线性表）：

方法	解释
boolean add(E e)	尾插 e
void add(int index, E element)	将 e 插入到 index 位置
boolean addAll(Collection<? extends E> c)	尾插 c 中的元素
E remove(int index)	删除 index 位置元素
boolean remove(Object o)	删除遇到的第一个 o
E get(int index)	获取下标 index 位置元素
E set(int index, E element)	将下标 index 位置元素设置为 element
void clear()	清空
boolean contains(Object o)	判断 o 是否在线性表中
int indexOf(Object o)	返回第一个 o 所在下标
int lastIndexOf(Object o)	返回最后一个 o 的下标
List<E> subList(int fromIndex, int toIndex)	截取部分 list

ArrayList（顺序表）：

方法	解释
ArrayList()	无参构造
ArrayList(Collection<? extends E> c)	利用其他 Collection 构建 ArrayList
ArrayList(int initialCapacity)	指定顺序表初始容量

LinkedList（链表）：

方法	解释
LinkedList()	无参构造

3.2 示例

```
import java.util.List;
import java.util.ArrayList;
import java.util.LinkedList;

public class ListDemo {
    public static void main(String[] args) {
        List<String> courses = new ArrayList<>();
        courses.add("C 语言");
        courses.add("Java SE");
```

```
        courses.add("Java Web");
        courses.add("Java EE");

        // 和数组一样，允许添加重复元素
        courses.add("C 语言");

        // 按照添加顺序打印
        System.out.println(courses);

        // 类似数组下标的方式访问
        System.out.println(courses.get(0));

        System.out.println(courses);
        courses.set(0, "计算机基础");
        System.out.println(courses);

        // 截取部分 [1, 3)
        List<String> subCourses = courses.subList(1, 3);
        System.out.println(subCourses);

        // 重新构造
        List<String> courses2 = new ArrayList<>(courses);
        System.out.println(courses2);

        List<String> courses3 = new LinkedList<>(courses);
        System.out.println(courses3);

        // 引用的转换
        ArrayList<String> courses4 = (ArrayList<String>)courses2;
        System.out.println(courses4);
        // LinkedList<String> c = (LinkedList<String>)course2; 错误的类型
        LinkedList<String> courses5 = (LinkedList<String>)courses3;
        System.out.println(courses5);
        // ArrayList<String> c = (ArrayList<String>)course3; 错误的类型
    }
}
```

运行结果

```
[C 语言, Java SE, Java Web, Java EE, C 语言]
C 语言
[C 语言, Java SE, Java Web, Java EE, C 语言]
[计算机基础, Java SE, Java Web, Java EE, C 语言]
[Java SE, Java Web]
[计算机基础, Java SE, Java Web, Java EE, C 语言]
[计算机基础, Java SE, Java Web, Java EE, C 语言]
[计算机基础, Java SE, Java Web, Java EE, C 语言]
[计算机基础, Java SE, Java Web, Java EE, C 语言]
```

3.4 练习-扑克牌

```java
public class Card {
    public int rank;    // 牌面值
    public String suit; // 花色

    @Override
    public String toString() {
        return String.format("[%s %d]", suit, rank);
    }
}
```

```java
import java.util.List;
import java.util.ArrayList;
import java.util.Random;

public class CardDemo {
    public static final String[] SUITS = {"♠", "♥", "♣", "♦"};
    // 买一副牌
    private static List<Card> buyDeck() {
        List<Card> deck = new ArrayList<>(52);
        for (int i = 0; i < 4; i++) {
            for (int j = 1; j <= 13; j++) {
                String suit = SUITS[i];
                int rank = j;
                Card card = new Card();
                card.rank = rank;
                card.suit = suit;

                deck.add(card);
            }
        }

        return deck;
    }

    private static void swap(List<Card> deck, int i, int j) {
        Card t = deck.get(i);
        deck.set(i, deck.get(j));
        deck.set(j, t);
    }

    private static void shuffle(List<Card> deck) {
        Random random = new Random(20190905);
        for (int i = deck.size() - 1; i > 0; i--) {
            int r = random.nextInt(i);
            swap(deck, i, r);
        }
    }

    public static void main(String[] args) {
        List<Card> deck = buyDeck();
        System.out.println("刚买回来的牌:");
        System.out.println(deck);
```

```
        shuffle(deck);
        System.out.println("洗过的牌:");
        System.out.println(deck);
        // 三个人，每个人轮流抓 5 张牌
        List<List<Card>> hands = new ArrayList<>();
        hands.add(new ArrayList<>());
        hands.add(new ArrayList<>());
        hands.add(new ArrayList<>());

        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 3; j++) {
                hands.get(j).add(deck.remove(0));
            }
        }

        System.out.println("剩余的牌:");
        System.out.println(deck);
        System.out.println("A 手中的牌:");
        System.out.println(hands.get(0));
        System.out.println("B 手中的牌:");
        System.out.println(hands.get(1));
        System.out.println("C 手中的牌:");
        System.out.println(hands.get(2));
    }
}
```

运行结果

```
刚买回来的牌:
[[♠ 1], [♠ 2], [♠ 3], [♠ 4], [♠ 5], [♠ 6], [♠ 7], [♠ 8], [♠ 9], [♠ 10], [♠ 11], [♠ 12],
[♠ 13], [♥ 1], [♥ 2], [♥ 3], [♥ 4], [♥ 5], [♥ 6], [♥ 7], [♥ 8], [♥ 9], [♥ 10], [♥ 11],
[♥ 12], [♥ 13], [♣ 1], [♣ 2], [♣ 3], [♣ 4], [♣ 5], [♣ 6], [♣ 7], [♣ 8], [♣ 9], [♣ 10],
[♣ 11], [♣ 12], [♣ 13], [♦ 1], [♦ 2], [♦ 3], [♦ 4], [♦ 5], [♦ 6], [♦ 7], [♦ 8], [♦ 9],
[♦ 10], [♦ 11], [♦ 12], [♦ 13]]
洗过的牌:
[[♥ 11], [♥ 6], [♣ 13], [♣ 10], [♥ 13], [♠ 2], [♦ 1], [♥ 9], [♥ 12], [♦ 5], [♥ 8], [♠
6], [♠ 3], [♥ 5], [♥ 1], [♦ 6], [♦ 13], [♣ 12], [♦ 12], [♣ 5], [♠ 4], [♣ 3], [♥ 7], [♦
3], [♣ 2], [♠ 1], [♦ 2], [♥ 4], [♦ 8], [♠ 10], [♦ 11], [♥ 10], [♦ 7], [♣ 9], [♦ 4], [♣
8], [♣ 7], [♠ 8], [♦ 9], [♠ 12], [♠ 11], [♣ 11], [♦ 10], [♠ 5], [♠ 13], [♠ 9], [♠ 7],
[♣ 6], [♣ 4], [♥ 2], [♣ 1], [♥ 3]]
剩余的牌:
[[♦ 6], [♦ 13], [♣ 12], [♦ 12], [♣ 5], [♠ 4], [♣ 3], [♥ 7], [♦ 3], [♣ 2], [♠ 1], [♦ 2],
[♥ 4], [♦ 8], [♠ 10], [♦ 11], [♥ 10], [♦ 7], [♣ 9], [♦ 4], [♣ 8], [♣ 7], [♠ 8], [♦ 9],
[♠ 12], [♠ 11], [♣ 11], [♦ 10], [♠ 5], [♠ 13], [♠ 9], [♠ 7], [♣ 6], [♣ 4], [♥ 2], [♣
1], [♥ 3]]
A 手中的牌:
[[♥ 11], [♣ 10], [♦ 1], [♦ 5], [♠ 3]]
B 手中的牌:
[[♥ 6], [♥ 13], [♥ 9], [♥ 8], [♥ 5]]
C 手中的牌:
[[♣ 13], [♠ 2], [♥ 12], [♠ 6], [♥ 1]]
```

3.3 面试题练习

1. 杨辉三角

内容重点总结

- 学习泛型的使用语法
- 学习包装类的概念和自动装/拆箱
- 掌握 List 的使用

课后作业

- 博客总结泛型的用法
- 博客总结包装类的知识
- 代码完成 List 的使用