计算机科学与技术学院 2019-2020 学年第 2 学期考试试卷

Java 语言程序设计　　A 卷(开卷)

姓名_____班级_____学号_____考试日期_2020-5-17__

题号	一	二	三	四	五	总分	核对人
题分	25	20	20	15	20	100	
得分							

得分	评卷人

一、阅读下面程序并回答问题 (共 25 分)

1: 阅读下列程序，并回答问题。（15 分）

```
class BankAccount{
    private double myBalance;  //账户余额
    public BankAccount() { myBalance = 0;}
    public BankAccount(double balance) { myBalance = balance; }
    public void deposit(double amount) { myBalance += amount; }
    public void withDraw(double amount) { myBalance -= amount; }
    public double getBalance() { return myBalance;}
}

class SavingsAccount extends BankAccount{
    private double myInterestRate ;  //存款利息
    public SavingsAccount(){ //实现代码这里没有显示 }
    public SavingsAccount(double balance, double rate){ //实现代码这里没有显示 }
    //向账户里添加利息
    public void addInterest(){ //实现代码这里没有显示 }
}

class CheckingAccount extends BankAccount{
    public CheckingAccount(double balance){ //实现代码这里没有显示}

    //CheckingAccount 取钱时可能需要扣除的手续费
    private static final double FEE = 2.0;
    //如果取钱后账户余额小于 MIN_BALANCE，就用扣除 FEE，允许账户余额为负数
    private static final double MIN_BALANCE = 50.0;

    public void withDraw(double amount){ //实现代码这里没有显示 }
}
```

(1) 类 SavingsAccount 的缺省构造函数的下面几个实现是否正确，无论正确与否都需要说明原因。（3 分）

① myInterestRate = 0.0; super();

② super(); myInterestRate = 0;

③ super();

答案：

①错误。super 必须是第一条语句。原因回答错误扣 0.5 分。结论和原因都正确给 1 分。

②正确。只要答案是正确，就给 1 分。

③正确。只要答案是正确，就给 1 分。

(2) 类 SavingsAccount 的带参数构造函数的下面几个实现是否正确，无论正确与否都需要说明原因。（3 分）

① myBalance = balance; myInterestRate = rate;

② getBalance() = balance; myInterestRate = rate;

③ super(); myInterestRate = rate;

④ super(balance); myInterestRate = rate;

答案：

①错误。myBalance 是父类的私有成员，子类不能访问。原因回答错误扣 0.5 分。结论和原因都正确给 1 分。

②错误。函数调用 getBalance()不能出现在=左边。原因回答错误扣 0.5 分。结论和原因都正确给 1 分。

③正确。只要答案是正确，就给 0.5 分。

④正确。只要答案是正确，就给 0.5 分。

(3) 类 CheckingAccount 的构造函数的下面几个实现是否正确，无论正确与否都需要说明原因。（3 分）

① super(balance);

② super(); deposit(balance);

③ deposit(balance);

答案：

①正确。只要答案是正确，就给 1 分。

②正确。只要答案是正确，就给 1 分。

③正确。只要答案是正确，就给 1 分。

(4) 类 CheckingAccount 的 withDraw 方法的下面几个实现是否正确，无论正确与否都需要说明原因。（3 分）

① super.withDraw(amount);

```
if( myBalance < MIN_BALANCE)
        super.withDraw(amount);
```

② withDraw(amount);

```
if( getBalance() < MIN_BALANCE)
        withDraw(FEE);
```

③ super.withDraw(amount);
if(getBalance() < MIN_BALANCE)
super.withDraw(FEE);

④ myBalance -= amount;
if(myBalance < MIN_BALANCE)
myBalance -= amount;

答案：

①错误。myBalance 是父类的私有成员，子类不能访问。原因回答错误扣 0.5 分。结论和原因都正确给 0.5 分。

②错误。withDraw 方法递归调用。原因回答错误扣 0.5 分。结论和原因都正确给 1 分。

③正确。只要答案是正确，就给 1 分。

④错误。myBalance 是父类的私有成员，子类不能访问。原因回答错误扣 0.5 分。结论和原因都正确给 0.5 分。

(5) 基于以上类的定义，若实例化下面 3 个对象：
BankAccount b = new BankAccount(1400);
BankAccount s= new SavingsAccount(1000,0.04);
BankAccount c = new CheckingAccount(500);
则下面的语句是否正确，无论正确与否都需要说明原因。（3 分）

① s.addInterest();

② s.withDraw(500);

③ b.deposit(200);

答案：

① 错误。s 对象声明类型是 BankAccount，没有 addInterest 方法。原因回答错误扣 0.5 分。结论和原因都正确给 1 分。

②正确。只要答案是正确，就给 1 分。

③正确。只要答案是正确，就给 1 分。

2: 阅读下列程序，并回答问题。（10 分）

```
class A {
    public void method1(){ System.out.println("A's method1"); }
    public static void method1(int n){ System.out.println("A's static method1");}
}

class B extends A {
    public void method1(){ System.out.println("B's method1"); }

    public static void method1(long n){ System.out.println("B's static method1"); }

    public void method2(){ System.out.println("B's method2"); }
}
```

基于以上类的定义，实例化了如下的对象，同时通过这些语句调用对象的不同方法：

```
A o1 = new A();
A o2 = new B();

o1.method1();          //①
o2.method1();          //②
o1.method1(1);         //③
o2.method1(1);         //④
o1.method1(1L);        //⑤
o2.method1(1L);        //⑥
((B)o1).method2();     //⑦
((B)o2).method2();     //⑧
```

(1) 以上语句能正常运行的有哪些？请给出能正常运行的每条语句的输出结果，并说明原因。

(2) 以上语句编译错误的有哪些？请并说明每个编译出错语句的错误原因。

(3) 以上语句抛出运行是异常的语句有哪些？请说明每个抛出运行时异常语句的为什么编译可以通过但是运行时抛出异常。

答案：

（1）5 分。能正常运行的有：

① 输出结果：A's method1　　1 分（输出结果写错全扣）

② 输出结果：B's method1　　1 分（输出结果写错全扣）

③ 输出结果：A's static method1　　1 分（输出结果写错全扣）

④ 输出结果：A's static method1　　1 分（输出结果写错全扣）

⑧ 输出结果：B's method2　　1 分（输出结果写错全扣）

（2）3 分。编译错误的有：

⑤ o1 的申明类型是 A，A 的静态方法 method1 参数是 int，实参 1L 类型是 long。1.5 分，其中原因说错了全扣。

⑥ o2 的申明类型是 A，A 的静态方法 method1 参数是 int，实参 1L 类型是 long。1.5 分，其中原因说错了全扣。

（3）2 分。抛出运行是异常的语句：

⑦ o1 的运行时类型是 A 类型，运行是强制转换成子类型会抛出异常。2 分，其中原因说错了全扣。

得分	评卷人

二、阅读下面程序，实现未完成的程序并回答问题（共20分）

1: 根据下面BirthDay的定义和Javadoc，请给出该类的构造函数、equals、toString、compareTo方法的实现代码。（构造函数1分，其他它每个方法各2分，共7分）

```
//出生日期类
class BirthDay implements Comparable<BirthDay>{

    private int year;       //年
    private int month;      //月
    private int day;        //日

    //构造函数
    public BirthDay(int year, int month, int day){
        //①请给出实现代码
    }

    //获得年
    public int getYear() { return year; }
    //获得月
    public int getMonth() { return month;}
    //获得日
    public int getDay() { return day; }

    /**
     *
     * @param obj 另外一个出生日期对象
     * @return 如果二个对象的年月日都相等，返回true；否则返回false
     */
    @Override
    public boolean equals(Object obj) {
          //②请给出实现代码
    }

    /**
     *
     * @return 返回年月日的字符串表示
     */
    @Override
    public String toString() {
       //③请给出实现代码
    }

    /**
     * 比较二个Birthday对象的大小
     * @param o 另外一个Birthday对象
     * @return  如果二个Birthday对象日期相等，返回0；如果第一个Birthday对象的日期比
Birthday对象的日期早，返回-1；否则返回1
     */
    @Override
    public int compareTo(BirthDay o) {
        //④请给出实现代码
    }
}
```

答案：

① 1分

```
public BirthDay(int year, int month, int day){
    this.year = year; this.month = month; this.day = day;
}
```

② 2分，其中没有用instanceof 进行类型判断，扣1分

```
public boolean equals(Object obj) {
    BirthDay other = null;
    if( obj instanceof BirthDay){
        other = (BirthDay)obj;
        return this.year == other.year && this.month == other.month &&
                    this.day == other.day;
    }
    else return false;
}
```

③ 2分

```
public String toString() {
    StringBuffer buf = new StringBuffer();
    buf.append(new Integer(year).toString()).append("-").
            append(new Integer(month).toString()).append("-").
            append(new Integer(day).toString());
    return buf.toString();
}
```

④ 2分

```
public int compareTo(BirthDay o) {
    if(this.year > o.year) { return 1; }
    else if(this.year < o.year){ return -1; }
    else {
        if(this.month > o.month){ return 1; }
        else if(this.month < o.month){ return -1; }
        else{
            if(this.day > o.day ){ return 1; }
            else if(this.day < o.day ){ return -1; }
            else return 0;
        }
    }
}
```

2：根据下面Person的定义和Javadoc，请给出该类的构造函数、equals、toString、compareTo方法的实现代码。（构造函数1分，其他它每个方法各2分，共7分）

```
class Person implements Comparable<Person>{

    private String id;      //id
    private BirthDay birthDay; //出生日期

    //构造函数
    public Person(String id, int year, int month, int day){
        //⑤请给出实现代码
    }

    public String getId() { return id;}
```

```
    public BirthDay getBirthDay() { return birthDay;}
    /**
     * 比较二个Person 对象是否相等
     * @return 如果二个Person 对象的id 和出生日期都相等，返回true;否则返回false
     */
    @Override
    public boolean equals(Object obj) {
          //⑥请给出实现代码
    }
    /**
     * @return 返回id 和出生日期的字符串表示
     */
    @Override
    public String toString() {
        //⑦请给出实现代码
    }

    /**
     * 根据出身日期比较大小
     * @param o
     * @return 如果o 的出生日期晚，返回1；如果出生日期相等，返回0；否则返回-1
     */
    @Override
    public int compareTo(Person o) {
        //⑧请给出实现代码
    }
}
```

答案：

⑤ 1分

```
public Person(String id, int year, int month, int day){
    this.id = id;
    this.birthDay = new BirthDay(year,month,day);
}
```

⑥ 2分 如果没有用instanceof 进行类型判断，扣1分，如果没有复用birthday对象的equals方法，扣1分

```
    public boolean equals(Object obj) {
        Person o = null;
        if(obj instanceof Person){
            o = (Person)obj;
            return this.id.equals(o.id) && this.birthDay.equals(o.birthDay);
        }
        else return false;
    }
```

⑦ 2分，如果没有复用birthday对象的toString方法，扣1分

```
public String toString() {
    return "id: " + id + ", birthday: " + birthDay.toString();
}
```

⑧ 2分，如果没有复用birthday对象的compareTo方法，扣1分

```
public int compareTo(Person o) {
    return this.birthDay.compareTo(o.birthDay);
}
```

3：阅读下面的main函数代码，并补全缺失的代码和回答问题。（⑨1分，⑩1分，⑪ 2分，⑫ 2分，共6分）。

```
public class Test1_2 {
    public static void main(String[] args){
        List<Person> list = _____;  //⑨请补=全右边的代码
        Person p1 = new Person("1", 2000, 1, 25);
        Person p2 = new Person("1", 2000, 1, 25);
        Person p3 = new Person("2", 1999, 5, 6);
        Person p4 = new Person("3", 2002, 7, 22);
        Person[] persons = {p1,p2,p3,p4};
        for(Person p: persons){
            if(!list.contains(p)){
                list.add(p);
            }
        }

        System.out.println();
        Collections.sort(list);
        //⑩请在下面补全代码，打印出 list 里的所有元素的字符串表示
        _____
        _____
        _____

    }
}
```

⑪ Collections.sort(list)方法会调用容器里元素的什么方法，对元素进行排序？请给出程序的输出结果。（提示：Collections.sort是按增序排序）。

⑫ 如果Person类没有覆盖equals方法，则上述代码的输出结果和Person类覆盖了equals方法后的输出结果有什么不同？请说明原因。

答案：

⑨ 1分 如果<>里给出类型参数Person也算对

```
        new ArrayList<>();
```

⑩ 1分，用传统的for循环也算对，println(p)也算对

```
for(Person p: list){
    System.out.println(p.toString());
}
```

⑪ 2分。输出结果写错扣1分。注意birthday的toString的结果可能和参考答案不一样。

调用元素的compareTo方法。输出结果为：

id: 2, birthday: 1999-5-6
id: 1, birthday: 2000-1-25
id: 3, birthday: 2002-7-22

⑫ 2分

如果Person用默认的equals方法，比较的是二个Person对象的引用是否相等。因此二个new出来的Person对象的比较结果为false。因此!list.contains(p)始终为true。以上原因解释里有二个关键点：list.contains(p)会调用元素的equals方法，默认的equals方法比较的是二个对象的引用是否相等。

这时容器里会有四个元素，输出结果为：

id: 2, birthday: 1999-5-6
id: 1, birthday: 2000-1-25
id: 1, birthday: 2000-1-25
id: 3, birthday: 2002-7-22

原因解释正确1分，结果输出正确1分。

得分	评卷人

三、编程题（20 分）

1: 请根据方法的 Javadoc 实现下面代码里的静态方法 reverseInt(5 分)。

```
public class Test1_1 {
    /**
     * 产生一个反转的十进制整数（高位依次反转到低位）
     * 如:  输入 123，返回 321
     *      输入-123，返回-321
     *      输入 0，返回 0
     * 不考虑数值溢出的情况
     * @param n 十进制整数
     * @return  反转的十进制整数
     */
    public static int reverseInt(int n){
        //请给出实现代码
    }

    public static void main(String[] args){
        System.out.println(reverseInt(123456));
        System.out.println(reverseInt(-123456));
        System.out.println(reverseInt(0));
    }
}
```

参考答案：5分

```
public static int reverseInt(int n){
    boolean isNegative = n < 0? true:false;
    String valueString = new Integer(Math.abs(n)).toString();
    String reversedValueString = new StringBuffer(valueString).reverse().toString();
    int reversedInt = Integer.valueOf(reversedValueString);
    return isNegative?-reversedInt:reversedInt;
}
```

2: 请完成下面的子问题(1)-(3)(共 15 分)。

(1) 实现一个字符串解析器 StringParser，该类的静态方法 public static List<String> parse(String lineString)将一个字符串分成一个个单词返回，请根据方法的 Javadoc 给出 parse 的实现(4 分)。提示，可考虑利用 String 类的 split 方法。

```
//字符串解析器
class StringParser{
    /**
     * 将输入的字符串拆分成单词
     * 假定字符串里每个单词之间都是以一个空格分开，例如字符串：Hello World
     * @return 拆分得到的单词列表
     */
    public static List<String> parse(String lineString){
        //TODO：请给出该方法的实现
    }
}
```

参考答案：4分

```
public static List<String> parse(String lineString){
    List<String> words = new ArrayList<>();
    String[] parts = lineString.split(" ");
    for(String s: parts){
        if(!"".equals(s.trim())){
            words.add(s);
        }
    }
    return words;
}
```

(2) 为了对单词过滤，现在定义过滤器接口如下：

```
//单词过滤器
interface Filter{
    /**
     * 过滤单词
     * @param s  输入的单词
     * @return  如果保留该单词，则返回true；如果过滤掉该单词，则返回false
     */
    boolean accept(String s );
}
```

基于该接口，请实现类 NonAlphabetWordFilter，该类实现过滤器接口，过滤掉包含任何非英语字母表里字符的单词，这里英语字母表包括大小写英语字符。例如单词 Hello 保留，但是单词 Hello2 就被过滤掉(5 分)。

参考答案：5分

```
class NonAlphabetWordFilter implements Filter{
    @Override
    public boolean accept(String s) {
        boolean hasNonAlphabetChar = false;
        for(int i = 0; i < s.length();i++){
           char c = s.charAt(i);
           if(!(( c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))){
               hasNonAlphabetChar = true;
           }
        }
        return !hasNonAlphabetChar;
    }
}
```

(3) 在 StringParse 类里，实现一个 parse 方法的重载函数，该函数要求传入一个实现了 Filter 接口的过滤器实例，在这个重载版本的 parse 方法里，对分割得到的单词进行过滤，返回符合要求的单词列表(4 分)。

参考答案：4分

```
public static List<String> parse(String lineString, Filter filter){
    //TODO: 请给出该方法的实现
    List<String> words = new ArrayList<>();
    String[] parts = lineString.split(" ");
```

```
    for(String s: parts){
        if(!"".equals(s.trim())){
            if(filter.accept(s)){
                words.add(s);
            }
        }
    }
    return words;
}
```

(4) 在 main 函数里写出测试代码：定义一个字符串如：

String line = "Hello java and python 123 C++";

调用上面定义的类和方法，得到经分割并过滤后产生的单词列表(2 分)。

参考答案：2分

```
public static void main(String[] args){
        String line = "Hello java and    python 123 C++";
//        List<String> words = StringParser.parse(line);
        List<String> words = StringParser.parse(line,new NonAlphabetWordFilter());
        for(String s : words){
            System.out.println(s);
        }
    }
```

得分	评卷人

四、编程题。(15 分)

下面的代码定义了泛型接口

```
/**
 * 泛型接口，可以进行add和subtract计算的接口
 * @param <T> 类型参数
 */
interface Computable<T> extends Comparable<T>{
    /**
     * 计算二个T类型对象的和
     * @param y 另外一个T类型对象
     * @return 二个T类型对象的和
     */
    T add(T y);
    /**
     * 计算二个T类型对象的差
     * @param y 另外一个T类型对象
     * @return 二个T类型对象的差
     */
    T subtract(T y);
}
```

根据以上接口定义，定义了一个现了Computable和Comparable泛型接口的二元组类ComputableComparableTuple，代码如下：

```
/**
 * 定义一个实现了Computable和Comparable泛型接口的二元组类ComputableComparableTuple
 * @param <T1> first的类型参数,T1必须是实现了Computable和Comparable的类型
 * @param <T2> second的类型参数, T2必须是实现了Computable和Comparable的类型
 */

_1:请给出类的声明代码_____________________________________________________
_____________________________________________________________________________
_____________________________________________________________________________
{
    private T1 first;       //二元组的第1部分
    private T2 second;      //二元组的第2部分

    /**
     * 构造函数
     * @param first
     * @param second
     */
    public ComputableComparableTuple(T1 first, T2 second){
        this.first = first;
        this.second = second;
    }

    /**
     *
     * 二个元组对象相加的语义：二个对象的first和second部分分别相加
     */
    @Override
    public ComputableComparableTuple<T1, T2> add(ComputableComparableTuple<T1, T2> y)
    {
       2: //TODO:请给出实现代码
    }
```

```
    /**
     *
     * 二个元组对象相减的语义：二个对象的first 和second 部分分别相减
     */
    @Override
    public ComputableComparableTuple<T1,T2> subtract(ComputableComparableTuple<T1,T2>
y) {
        3: //TODO:请给出实现代码

    }

    /**
    *Comparable 接口的compareTo 方法实现的语义是：
    *如果二个元组对象的first 部分不相等，以二个对象first 成员比较结果作为最终比较结果；
    * 如果二个元组对象的first 部分相等，则以二个对象second 成员比较结果作为最终比较结果；
    */
    @Override
    public int compareTo(ComputableComparableTuple<T1, T2> o) {
        4: //TODO:请给出实现代码
    }
}
```

1: 请给出ComputableComparableTuple类的声明代码。(4分)

答案：4分，如有任何错误，全扣

```
class ComputableComparableTuple<T1 extends Computable<T1> , T2 extends Computable<T2>>
        implements Computable<ComputableComparableTuple<T1,T2>>,
                   Comparable<ComputableComparableTuple<T1,T2>>
```

2-4: 请给出三个接口方法add、subtract、compareTo的实现。(每个方法2分)

参考答案：每个方法2分，共6分

```
@Override
public ComputableComparableTuple<T1, T2> add(ComputableComparableTuple<T1, T2> y) {
    return new ComputableComparableTuple(this.first.add(y.first),
                   this.second.add(y.second));
}

@Override
public ComputableComparableTuple<T1, T2> subtract(ComputableComparableTuple<T1, T2> y)
{
    return new ComputableComparableTuple(this.first.subtract(y.first),
                   this.second.subtract(y.second));
}

@Override
public int compareTo(ComputableComparableTuple<T1, T2> o) {
    int firstCompare = first.compareTo(o.first);
    return (firstCompare != 0)?firstCompare:second.compareTo(o.second);
}
```

5: 定义一个实现了Computable泛型接口的类IntComputable，该类包含一个私有的、int类型数据成员value。该类应该使得下面的测试代码通过。（5分）

```
public class Test1_3 {
    public static void main(String[] args){
        ComputableComparableTuple<IntComputable, IntComputable> tuple1=
            new ComputableComparableTuple(new IntComputable(2),new IntComputable(3));
        ComputableComparableTuple<IntComputable, IntComputable> tuple2=
            new ComputableComparableTuple(new IntComputable(4),new IntComputable(5));

        ComputableComparableTuple<IntComputable, IntComputable> tuple3 =
                            tuple1.add(tuple2);
        ComputableComparableTuple<IntComputable, IntComputable> tuple4 =
                            tuple1.subtract(tuple2);
        System.out.println(tuple1.compareTo(tuple2));
        System.out.println();
    }
}
```

参考答案： 5分

```
class IntComputable  implements Computable<IntComputable>{
    private int value;

    public IntComputable(int value){
        this.value = value;
    }

    @Override
    public IntComputable add(IntComputable y) {
        return new IntComputable(this.value + y.value);
    }

    @Override
    public IntComputable subtract(IntComputable y) {
        return new IntComputable(this.value - y.value);
    }

    @Override
    public int compareTo(IntComputable o) {
        return this.value - o.value;
    }
}
```

得分	评卷人

五、编程题。(20 分)

下面的代码定义了抽象类Item和迭代器接口ItemIterator：

```
//抽象类Item ，代表商品
abstract class Item implements Cloneable{
    protected String name;    //商品名称
    public Item(String name){ this.name = name; }

    //计算商品价格并返回
    public abstract double salePrice();

    @Override
    public Object clone() throws CloneNotSupportedException {
        Item item = (Item)super.clone();
        item.name = new String(this.name);
        return item;
    }
}

//商品迭代器接口
interface ItemIterator{
    //返回下一个商品
    Item next();
    //是否还有下一个商品
    boolean hasNext();
}
```

1：Item的二个子类的部分定义如下：

```
//全价商品，不打折
class FullPriceItem extends Item{
    protected double price;   //商品价格
    public FullPriceItem(String name, double price){
        ①//TODO: 请实现该方法
    }

    @Override
    public double salePrice() { return price; }

    @Override
    public Object clone() throws CloneNotSupportedException {
        ②//TODO: 请实现该方法
    }
}

//打折商品
class DiscountItem extends FullPriceItem{
    private double discount; //商品折扣
    public DiscountItem(String name, double price,double discount){
        ③//TODO: 请实现该方法
    }

    /**
     * 返回打折后的价格：注意中文的打9折，discount 就是10%
     * @return 打折后的价格
```

```
     */
    @Override
    public double salePrice() {
        ④//TODO: 请实现该方法
    }

    @Override
    public Object clone() throws CloneNotSupportedException {
        ⑤//TODO: 请实现该方法
    }
}
```

请实现子类FullPriceItem的构造函数(1分)、子类FullPriceItem的clone方法(2分)、子类DiscountItem的构造函数(1分)、子类DiscountItem的salePrice方法(1分)、子类DiscountItem的clone方法(2分)。(共7分)

参考答案：

① 1分。如果不是调用super，扣0.5

```
public FullPriceItem(String name, double price){
    super(name);
    this.price = price;
}
```

② 2分，没有调用super.clone扣1分

```
public Object clone() throws CloneNotSupportedException {
    //首先 clone 父类部分
    FullPriceItem fullPriceItem = (FullPriceItem)super.clone();
    //再 Clone 子类部分
    fullPriceItem.price = this.price;
    return fullPriceItem;
}
```

③ 1分. 如果不是调用super，扣0.5

```
public DiscountItem(String name, double price,double discount){
    super(name,price);
    this.discount = discount;
}
```

④ 1分. 如果写成price * discount，不扣分

```
public double salePrice() {
    return price * (1- discount);
}
```

⑤ 2分，没有调用super.clone扣1分

```
public Object clone() throws CloneNotSupportedException {
    //首先 clone 父类部分
    DiscountItem discountItem = (DiscountItem)super.clone();
    //在 Clone 子类部分
    discountItem.discount = this.discount;
    return discountItem;
}
```

2：在以上类的基础上，实现一个商品仓库类WareHouse，该类存放不打折商品和打折商品，二种类型的商品分别放在不同的容器对象里。该类实现ItemIterator接口，对外以一致的方式遍历每个商品。WareHouse类的部分定义如下：

```
/**
 * 商品的仓库，里面存放不打折商品和打折商品
 * 二种类型的商品分别放在不同的的容器对象里
 * 商品的仓库 实现 ItemIterator 接口，对外以一致的方式遍历每个商品
 */
class WareHouse implements ItemIterator{
    private List<Item> fullPriceItems = new ArrayList<>();  //FullPriceItem 放这里
    private List<Item> discountPriceItems = new ArrayList<>();  //DiscountItem 放这里
    //可以添加自己的私有数据成员和自己的私有方法

    /**
     * 构造函数，传入已经放好二种商品的 list
     * @param fullPriceItems
     * @param discountPriceItems
     */
    public WareHouse(List<Item> fullPriceItems, List<Item> discountPriceItems){
        ⑥//TODO: 请实现该方法
    }
    @Override
    public Item next() {
        ⑦//TODO: 请实现该方法
    }
    @Override
    public boolean hasNext() {
        ⑧//TODO: 请实现该方法
    }

    //返回迭代器对象
    public ItemIterator iterator(){
        ⑨//TODO: 请实现该方法
    }
}
```

请实现类WareHouse的构造函数(1分)、ItemIterator接口方法next(4分)、ItemIterator接口方法hasNext(3分)、WareHouse的iterator方法(2分)。(共10分)

参考答案：

首先添加私有成员

```
private int totalItemCount = 0;
private int pos = 0;
```

⑥ 1分

```
public WareHouse(List<Item> fullPriceItems, List<Item> discountPriceItems){
    this.fullPriceItems = fullPriceItems;
    this.discountPriceItems = discountPriceItems;
    totalItemCount = fullPriceItems.size() + discountPriceItems.size();
}
```

⑦ 4分

```
public Item next() {
    if(pos < fullPriceItems.size()){
        Item item = fullPriceItems.get(pos);
        pos ++;
        return item;
    }
    else if(pos - fullPriceItems.size() < discountPriceItems.size()){
        Item item = discountPriceItems.get(pos - fullPriceItems.size());
        pos ++;
        return item;
    }
    else
        return null;
}
```

⑧ 3分

```
public boolean hasNext() {
    if(pos >= totalItemCount)
        return false;
    else
        return true;
}
```

⑨ 2分

```
public ItemIterator iterator(){
    return this;
}
```

3. 编写测试代码，实现一个静态方法public static double totalPrice(ItemIterator it)，计算仓库里所有商品的总价格(3分)。以上的代码要能够得main方法里的代码正确运行。

```
public class Test1_4 {
    /**
     * 计算商品价格的总和
     * @param it 商品迭代器
     */
    public static double totalPrice(ItemIterator it){
        ⑩//TODO: 请实现该方法
    }

    public static void main(String[] args) throws CloneNotSupportedException {
        Item fullPriceItem1 = new FullPriceItem("洗衣机",1000);
        Item fullPriceItem2 = (Item)fullPriceItem1.clone();
        List<Item> fullPriceItems = new ArrayList();
        fullPriceItems.add(fullPriceItem1);
        fullPriceItems.add(fullPriceItem2);

        Item discountPriceItem1 = new DiscountItem("冰箱",1000, 0.5);
        Item discountPriceItem2 = (Item)discountPriceItem1.clone();
        List<Item> disCountPriceItems = new ArrayList();
```

```
        disCountPriceItems.add(discountPriceItem1);
        disCountPriceItems.add(discountPriceItem2);

        WareHouse wareHouse = new WareHouse(fullPriceItems,disCountPriceItems);
        totalPrice(wareHouse.iterator());

    }
}
```

参考答案：3分

```
public static double totalPrice(ItemIterator it){
    double totalPrice = 0.0;
    while (it.hasNext()){
        Item item = it.next();
        totalPrice += item.salePrice();
    }
    return totalPrice;
}
```