第 2-3 章作业

一、填空题

1. Java 语言中有 4 种基本的整数类型，哪种类型所占的内存空间最小，写出定义该类型的关键字_____**byte**_____。

2. Java 语言中有 4 种基本的整数类型，哪种类型所占的内存空间最大，写出定义该类型的关键字_____**double**_____。

3. Java 中存在一种基本的数据类型，该类型定义的变量不能与其他类型转换，定义该类型用____**boolean**_____。

4. 布尔型定义的成员变量是有默认值的，它的值是___**false**_____。

5. 定义变量保存含有小数的数据时，使用__**double**___定义的变量精度比较高。

6. Java 源程序经过编译后生成被称为____**字节码**_____的特殊机器语言码，然后经过___**JVM**_____解释运行。

7. 声明一个值为 3.14 的 double 型常量 PI 的语句为

_____**final double PI = 3.14;**_____。

8. 下列程序中，首先声明和初始化三个变量 ch = 'a'，变量 d=0.1，变量 l =12L，并将该三个变量打印输出.请将程序补充完整.

```
public class Assign{
    public static void main(String args[]){
        char   ch = 'a';
        double d = 0.1;
        long   l = 12L;
        System.out.println("ch=" + ch );
        System.out.println("d=" + d );
        System.out.println("l=" + l );
    }
}
```

9. 执行下列语句后，变量x，y 的值分别为____**0,1**_____，原因是

对于&&和||符号，如果左式已经决定整个式子的结果，则不再判断右式；对于&和|符号，无论情况如何，都将执行两边的式子。

```
int x = 0, y = 0;
System.out.println( ((x > 1) && (++x == 0)) + " " + x);
System.out.println( ((y < 1) | (y++ == 0)) + " " + y);
```

10. 阅读下面代码

```
System.out.print("Please input your choice[1,2]:");
long i = new Scanner(System.in).nextLong();
switch (i){
    case 1 :
        System.out.println("Your choice is 1");
        break;
    case 2 :
        System.out.println("Your choice is 2");
        break;
    default:
        System.out.println("Wrong choice");

}
```

以上代码错误的地方是:

switch 语句中不能使用 long 类型变量作为判断条件

二、单项选择题

1. 以下说法正确的是__**A**____。

(A) Java 中所有的方法都必须在类内定义

(B) Java 中主方法可以不在类内定义，其他方法都必须定义在类内

(C) Java 中主方法必须定义在类内，其他方法可以不必定义在类内

(D) Java 中所有方法都不必在类内定义

2. Java 源文件和编译后的文件扩展名分别是___**C**____。

(A) .class 和 .java　　(B) .class 和 .class

(C) .java 和 .class　　(D) .java 和 .java

3. 关于布尔类型说法正确的是 ___**A**_____。

(A) boolean 表示布尔类型，它的取值只有 true 和 false

(B) bool 表示布尔类型，它的取值只有 true 和 false

(C) boolean 表示布尔类型，它的取值只有 1 和 0

(D) bool 表示布尔类型，它的取值只有 1 和 0

4．下面表达式错误的是___B_____。

(A) int i = 100;

(B) float f = 100.0;

(C) Object o = “Hello world”;

(D) char c = ‘\u1234’;

5．下面代码的输出是___A_____。

```
class C {
    public static void main (String [] args)
        { int x = 1;
        System.out.print((x > 1) & (x++ > 1));
        System.out.print(“ ”);
        System.out.print((x > 1) && (x++ > 1));
    }
}
```

(A) false true

(B) true false

(C) false false

(D) true true

6．下列叙述中正确的是_____A____。

(A) Java 语言的标识符是区分大小写的

(B) Java 源程序文件名可以任意命名

(C) Java 源程序文件的扩展名为.jar

(D) 一个 Java 源程序文件里public 类的数目不限

7．下列标识符中，合法的是___A_____。

(A) _name　　(B) 4Person

(C) public　　(D) -3.1415

三、编程题

教材第二章编程练习题 2.6 题

```
package course.ch2;
import java.util.Scanner;
/**
 * Calculate the sum of the digits of a number between 0 and
1000
 */
public class sumofDigits {
    /**
     * The entrance of program
     * @param args: command line arguments
     */
    public static void main(String[]
        args){ int num,sum=0;
        System.out.println("Enter  a  number  between  0  and
1000:");
        num=new Scanner(System.in).nextInt();
        for(;num>0;num/=10)
            sum+=num%10;
        System.out.println("The sum of the digits is "+sum);
    }
}
```

第 4-5 章作业

一、填空题

1：假设

```
String s1 = "Welcome to Java";
String s2 = s1;
String s3 = new String("Welcome to Java");
```

那么下面表达式的结果是什么?

(1) s1 == s2 ____true____

(2) s1 == s3 ____false____

(3) s1.equals(s2) ____true____

(4) s2.equals(s3) ____true____

(5) s1.compareTo(s2); ____0____

(6) s2.compareTo(s3); ____0____

(7) s1.charAt(0); ____'W'____

(8) s1.indexOf('j'); ____-1____

(9) s1.indexOf("to"); ____8____

(10) s1.lastIndexOf("o",15) ____9____

(11) s1.substring(3, 11); ____"come to "____

(12) s1.endsWith("Java") ____true____

(13) s1.startsWith("wel"); ____false____

(14) " We come ".trim(); ____"We come"____

(15) s1.toUpperCase(); ____"WELCOME TO JAVA"____

(16) s1.replace('o', 'T'); ____"WelcTme tT Java"____

2. 如果

```
StringBuffer s1 = new StringBuffer("Java");
StringBuffer s2 = new StringBuffer("HTML");
```

假设下列每个语句是独立的，每条语句结束后，写出相应结果

(1) s1.append(" is fun"); s1 为__"Java is fun"____

(2) s1.append(s2); s1 为__"JavaHTML"____

(3) s1.insert(2, "is fun"); s1 为__"Jais funva"____

(4) s1.insert(1,s2); s1 为__"JHTMLava"____

(5) char c = s1.charAt(2); c 为__'v'____

(6) int i = s1.length(); i 为___4____

(7) s1.deleteCharAt(3); s1 为__"Jav"____

(8) s1.delete(1,3); s1 为__"Ja"____

(9) s1.reverse(); s1 为__"avaJ"____

(10) s1.replace(1,3, "Computer"); s1 为__"JComputera"____

(11) String s3 = s1.substring(1,3);

s3 为___"av"____, s1 为____"Java"____

(12) String s4 = s1.substring(2);

S4 为____"va"____, s1 为____"Java"____

3. 假设 StringBuffer s = new StringBuffer("Welcome to JAVA");

将 s 的内容清空的语句是____**s.delete(0,s.length())**__。

4.如果

```
String s1 = "Welcome";
String s2 = new String("Welcome");
String s3 = s2.intern();
String s4 = "Wel" + "come";
String s5 = "Wel";
String s6 = "come";
String s7 = s5 + s6;
String s8 = "Wel" + new String("come");
```

那么下面表达式的结果为:

(1) s1 == s2 __false____

(2) s1 == s3 __true____

(3) s1 == s4 __true____

(4) s1 == s7 __false____

(5) s1 == s8 __false____

(6) s1.equals(s2) ____true____

(7) s1.equals(s3) ____true____

(8) s1.equals(s4) ____true____

(9) s1.equals(s7) ____true____

(10) s1.equals(s8) ____true____

二、单项选择题

1. 可以获取字符串 s 的最后一个字符的表达式是____C____。

(A) s.length()

(B) s[s.length() - 1]

(C) s.charAt(s.length() - 1)

(D) charAt(s, length(s))

2. 下面程序

```
class C {
    public static void main(String[] args) {
       String s = "null";
```

```
        if(s == null)
            System.out.print("a");
        else if(s.length() == 0)
            System.out.print("b");
        else
            System.out.print("c");
    }
}
```

的输出为___C_____。

(A) a　　(B) b

(C) c　　(D) null

3. 下面的程序

```
class C {
    public static void main(String[] args) {
        String s = "Welcome to ";
        concat(s);
        System.out.print(s);
    }
    public static void concat(String s) {
        s += "Java";
    }
}
```

的输出为___A_____。

(A) Welcome to　　(B) Welcome to Java

(C) 编译错误　　(D) 运行时异常

三、编程题

1：编写程序，从控制台或对话框任意输入一个英文字符串，统计字符串中每个英文字母出现的次数并输出到控制台（大小写不敏感）。

```
package course.ch4_5;

import javax.swing.*;

/**
 * Enter an English string from the console or dialog box
 * count the number of times each English letter in the string appears
 * and output it to the console (case-insensitive)
 */
public class CountNumofLetter {
    /**
     * The entrance of program
     *
     * @param args: command line arguments
     */
    public static void main(String[] args) {
        String temp = JOptionPane.showInputDialog(null, "Input an English
String", "InputDialog", JOptionPane.PLAIN_MESSAGE);
        String lowerTemp = temp.toLowerCase();
        System.out.println("Your Input:" + lowerTemp);
        StringBuilder pend = new StringBuilder(lowerTemp);
        int[] count = new int[26];
        for (int i = 0; i < pend.length(); i++)
            count[pend.charAt(i) - 'a']++;
        for (int i = 0; i < 26; i++)
            System.out.printf("%c or %c : %d\n", 'a' + i, 'A' + i, count[i]);
    }
}
```

2：假设一个车牌号码由三个大写字母和后面的四个数字组成。编写一个程序．随机生成 5 个不重复的车牌号如下。

```
package course.ch4_5;

import java.util.ArrayList;
import java.util.List;

/**
 * Suppose a license plate number consists of three uppercase letters and
four numbers after it.
 * Write a program. Randomly generated into 5 non-duplicate license plate
numbers.
 */
public class VehicleNumberGenerator {
    private static final int UPPER_LETTER_LENGTH = 3;
    private static final int DIGIT_LENGTH = 4;

    /**
     * 产生指定数目的不重复车牌号码，车牌号码由三个大写字母和后面的四个数字组成
     *
     * @param n 车牌个数
     * @return 生成的车牌
     */
    public static String[] generate(int n) {
```

```
        if (n < 1)
            return null;

        List<Object> list = new ArrayList<>();
        while (list.size() < n) {
            //Generate  Uppercase Letter
            char[] letters = new char[UPPER_LETTER_LENGTH];
            for (int j = 0; j < UPPER_LETTER_LENGTH; j++) {
                letters[j] = RandomCharacter.getRandomUpperCaseLetter();
            }
            //Generate Digits
            char[] digits = new char[DIGIT_LENGTH];
            for (int j = 0; j < DIGIT_LENGTH; j++) {
                digits[j] = RandomCharacter.getRandomDigitCharacter();
            }
            String number = String.valueOf(letters) +
String.valueOf(digits);

            if (!list.contains(number)) { //只有不重复，才加入 list
                list.add(number);
            }

        }

        return (String[]) list.toArray(new Object[0]);
    }

    public static void print(String[] numbers)
        { for (String number : numbers) {
            System.out.println(number);
        }
    }

    public static void main(String[] args) {

VehicleNumberGenerator.print(VehicleNumberGenerator.generate(5));
    }
}

class RandomCharacter {
    /**
     * Generate a random character between ch1 and ch2
     */
    public static char getRandomCharacter(char ch1, char ch2) {
        return (char) (ch1 + (int) (Math.random() * (ch2 - ch1 + 1)));
    }

    /**
     * Generate a random lowercase letter
     */
    public static char getRandomLowerCaseLetter()
        { return getRandomCharacter('a', 'z');
    }

    /**
     * Generate a random uppercase letter
     */
    public static char getRandomUpperCaseLetter()
        { return getRandomCharacter('A', 'Z');
    }

    /**
     * Generate a random digit character
     */
    public static char getRandomDigitCharacter()
        { return getRandomCharacter('0', '9');
    }

    /**
     * Generate a random character
     */
    public static char getRandomCharacter() {
        return getRandomCharacter('\u0000', '\uFFFF');
    }
}
```

第 6-7 章作业

一、填空题

1. 函数重载是指在函数名相同，但_____**参数**_____不同

2. 创建大小为 2 行 4 列的二维 char 型数组的语句为 **char [][]a=new char[2][4]**，数组创建后每个元素的值为____ **'\u0000'** _____。

3. 创建一个大小为 10 的整型数组，且数组元素的值分别为 1,2,3,4,5,6,7,8,9,10 的语句为_____**int[] array={1,2,3,4,5,6,7,8,9,10};**_____。

4. 用 final 关键字修饰一个方法形参的含义是

在该方法内部，这个形参的值不能被修改。

5. 下列程序存在的错误是_____**范围内已定义变量 i**_____。

```
public static void m(int i){
        for(int i = 0 ; i < 10;
            i++){ System.out.println(i);
        }
}
```

二、单项选择题

1. 下列语句的输出结果是___**C**_____。

```
String[][] a = {
        {"Beijing","Wuhan"},
        {"Shanghai","Guangzhou","Xian"},
        {"Chongqing","Chengdu"}
};
System.out.println(a[a.length - 1].length);
System.out.println(a[a.length - 1][a[a.length - 1].length - 1].length());
```

A. 2, 5　　B．3, 4　　C．2, 7　　D．3, 8

2. String[]s={"Monday","Tuesday","Wednesday","Thirsday","Friday","Sataday","Sunday"}；，则下列语句正确的是____**C**_____。

A. int a = s.length,　b = s[1].length;

B. int　a = s.length( ), b = s[1].length;

C. int　a = s.length,　b = s[1].length();

D. int　a = s.length( ), b = s[1].length( );

3. 若有下面程序

```
class C {
        public static void main(String[] args)
            { int[] array = new int[10];
            increase(array);
            System.out.print(array[0]);
        }
        public static void increase(int[] array)
            { for(int i = 0; i < array.length; i++) {
                array[i]++;
            }
        }
    }
```

则输出为____**B**_____。

A.0　　B.1　　C.2　　D.10

4. 下面的数组申明和初始化语句不合法的是____**C**_____。

A. int a[ ] = null;

B. int[ ] b = { };

C. int[ ] c = new int{1,2,3,4};

D. int [] d[] = new int[5][ ];

三、判断对错题

1. 局部变量在使用前必须通过初始化或者赋值语句显式地给一个值。（ √ ）

2. 一个方法必须要有一个return 语句。（**X**）

3. 如果定义 int[] nValues={1,2,3,4}; 那么 nValues 为引用类型。（ √ ）

4. 不能基于函数返回类型来重载函数。 （ √ ）

5. 二维数组的行数和列数是相同的。 （ **X** ）

四、阅读下列程序，写出输出结果：

```
public class Test2 {
    public static void main(String[]
        args){ int[] a = {1};
        String[] s = {"Hello"};
        int i = a[0];
        m(s,a,i);
```

```
        for(String v:s){
            System.out.println(v);
        }
        for(int v:a){
            System.out.println(v);
        }
        System.out.println(i);
    }
    public static void m(String[] a1, int[] a2, int
        i){ for(int j =0; j < a1.length;j++){
            a1[j] = "Java";
        }
        for(int j =0; j <
            a2.length;j++){ a2[j]++;
        }
        i++;
    }
}
```

Result:

Java

2

1

第 9 章作业

一、填空题

1. 当希望一个类中的成员不能在类的外部访问时，应使用 ____**private**_____修饰符定义该成员。

2. 使用____**static**_____修饰符定义的类成员，可以通过类直接访问而不需要创建对象后再访问。

3. 类中的一个成员是一个类的对象时，如果该成员没有被初始化，则该对象的初始值是 ____**null**____。

4. 在类的非静态成员函数中，使用 _____**this**_____关键字来表示当前调用该函数的对象。

5. 假设 A 为一个类，则执行 A [] array = new A[10];语句时，一共调用 ____**0**____次 A 的构造函数。

二、单项选择题

1. 下面关于构造函数的说法不正确的是 ____**B**____。

   A. 构造函数的调用时机是实例化对象时

   B. 一个类必须且只能定义一个构造函数

   C. 一个类可以不定义构造函数

   D. 构造函数一定要和类名相同，并且不能有返回值

2. 下列哪个修饰符可以使在一个类中定义的成员变量只能被同一包中的类访问？

   A. private　　**B．无修饰符**　　C．public　　D．protected

3. 给出下列代码，如何使成员变量 m 被方法 fun（）直接访问 ___**C**_____。

```
class Test
{
    private int m;
    public static void fun （）
    {
        …
    }
}
```

   A. 将 private int m 改为 protected int m

   B. 将 private int m 改为 public int m

   C. 将 private int m 改为 static int m

   D. 将 private int m 改为 int m

4. 对于 class A，如果在另一个包中的 class B 中，语句 A a = new A( ); a.m=10;成立，则下列定义正确的是___C___。

A. class A { int m; }　　B．class A {public int m; }

C．public class A{ public int m; }　　D．public class A { int m; }

5. 关于下面程序，说法正确的是____C_____。

```
class AA{
    private long i = 0;
    AA(int i){
        this.i =i;
    }
    String AA(long i)
        { this.i = i;
        return    "i = " + this.i;
    }
}
public class Test_1_6 {
    public static void main(String[] args)
        { System.out.println(new
        AA(10).AA(20));
    }
}
```

A. 以上代码编译出错，一个类的构造函数不能有返回值；

B. 以上代码编译通过，输出结果为 i = 10;

C. 以上代码编译通过，输出结果为 i = 20;

D. 以上代码编译出错，一个类的构造函数不能重载；

6. 对于以下代码，说法正确的是____A_____。

```
package homework.ch9.p1;
public class A {
    private  int  i  =  0;
    protected int j = 0;

}

package  homework.ch9.p2;
import homework.ch9.p1.A;
public class B extends A {
    public void m() {
        new A().j = 10;
        this.j = 10;
    }
```

}A．new A().j = 10; 有编译错误，在方法m 里无法访问 new A().j ;this.j 无编译错误，在方法 m 里可以访问 this.j;

B. new A().j = 10; 无编译错误，在方法 m 里可以访问 new A().j ;this.j 有编译错误，在方法 m 里不可以访问this.j;

C. new A().j = 10; 无编译错误，在方法 m 里可以访问 new A().j ;this.j 无编译错误，在方法 m 里可以访问this.j;

D. new A().j = 10; 有编译错误，在方法 m 里不可以访问 new B().j ;this.j 有编译错误，在方法 m 里不可以访问this.j;

三、判断对错题

1. 若 a 是类 A 的实例化对象，且 a.fun();能顺利执行，则函数 fun 一定是实例方法。(X)

2. protected 修饰的类成员只能被其子类访问。(X)

3. 类的静态变量被该类的所有实例共享。(√)

4. Double 类型的变量是值类型。(X)

5. 当类的实例方法的形参变量与类的实例变量同名时，优先访问形参变量。(√)

四、阅读下列程序，写出输出结果：

```
public class Circle
{
    private double radius;
    public static int count = 0;
    public Circle(double r){
        radius = r;
        count ++;
    }
    public Circle(){
        this(1.0);
    }
    public static void main(String[] args)
    {
        Circle c1 = new Circle ();
        Circle c2 = new Circle (15.0);
        c1. count ++;
        c2. count ++;
```

```
        Circle. count ++;
        System.out.println("count ="+ count);
    }
}
```

输出结果为_____count=5_____。

五、编程题

1. 教材 9.8 题。

```
package code;

public class Fan {
    public final static int SLOW = 1;
    public final static int MEDIUM = 2;
    public final static int FAST = 3;
    private int speed = SLOW;
    private boolean on = false;
    private double radius = 5;
    private String color = "blue";

    public Fan() {
    }

    public Fan(int speed, boolean on, double radius, String color)
        { this.color = color;
        this.on = on;
        this.speed = speed;
        this.radius = radius;
    }

    public int getSpeed()
        { return speed;
    }

    public boolean getOn()
        { return on;
    }

    public double getRadius()
        { return radius;
    }

    public String getColor() {
        return color.toString();
    }

    public void setSpeed(int newSpeed)
        { speed = newSpeed;
    }

    public void setOn(boolean newOn)
        { on = newOn;
    }

    public void setRadius(double newRadius)
        { radius = newRadius;
    }

    public void setColor(String newColor)
        { color = newColor;
    }

    public String toString() {
        StringBuilder speedString = new StringBuilder();
        if (speed == 1)
            speedString = new StringBuilder("SLOW");
        else if (speed == 2)
            speedString = new StringBuilder("MEDIUM");
        else if (speed == 3)
            speedString = new StringBuilder("FAST");
        if (on) {
            return "Speed:" + speedString + ",Radius:" + radius + ",Color:"
+ color.toString();
        } else {
            return "fan is off,Radius:" + radius + ",Color:" +
color.toString();
        }
    }
}
```

第 11-13 章作业

一、填空题

1. 使用__**static**_____修饰符定义的类成员，可以通过类直接访问而不需要创建对象后再访问。

2. 用_**abstract**_修饰符定义的方法，没有方法体，使用____**abstract**_____修饰符定义的类不能实例化。

3. 类的中一个成员是一个类的对象时，如果该成员没有被初始化，则该对象的初始值是_____**null**____。

4. 在子类构造函数中使用___**super**____关键字来调用父类的构造函数。

5. Java 接口中可以声明_____**常量**_____和_____**方法**_____。

6. 用 final 关键字修饰一个类表示_____**这个类不可被继承**_____。

7. 在子类的实例方法 m 中要调用父类被覆盖的实例方法 m（方法 m 不带参数且没有返回值）的语句是 ___**super.m()**____。

8. 如有以下类的定义：

```
abstract class A
      {  public void fa ()
      {};
      public abstract void fb();
      public abstract void fc();
}
interface I {
      void fx();
}
abstract class B extends A
                                  {
       public void fb() {};
      public abstract void fd();
}
public class C extends B implements I {
      …
}
```

则在在 class C 中必须要实现的方法为____**fc(),fd(),fx()** ____。

9. 如有下列接口和类的定义：

```
interface I1{}
interface I2 extends I1{    }
class A implements I2{ }
class B extends A{ }
```

则 B 类的一个实例对象 o 的类型可以是_____**A B I1 I2 Object**_____。

10. 下列程序的输出结果是_____**three two one 1 2**_____。

```
class C {
      int x;
      String y;
      public C() {
            this(“1”);
            System.out.print(“one ”);
      }
      public C(String y) {
            this(1, “2”);
            System.out.print(“two ”);
      }
      public C(int x, String y)
            { this.x = x;
            this.y = y;
            System.out.print(“three ”);
      }
      public static void main(String[] args)
            { C c = new C();
            System.out.println(c.x + “ ” + c.y);
      }
}
```

11. 在 Java 中对于程序可能出现的必检异常，要么用 try…catch 语句捕获并处理它，要么使用____**throw**_____语句抛出它，由上一级调用者来处理。

12. 在 Java 中异常分为必检异常和非必检异常二种类型，其中表达式 10/0 会抛出____**非必检**___类型异常，打开一个不存在的文件会抛出___**必检**___类型异常，通过空引用调用实例方法会抛出___**非必检**___类型异常，数组越界访问会抛出___**非必检**_____类型异常，用 throw 语句抛出一个自定义的 Exception 子类异常是____**必检**_____类型异常。

二、单项选择题

1. 接口中的成员变量被隐含地声明为_____**A**_____。

A. public static final　　B. public final

C. public static　　D. public abstract

2. 下列叙述中正确的是___**B**_____。

A. Java 中一个类可以有多个直接父类，可以实现多个接口

B. Java 中一个类只可以有一个直接父类，可以实现多个接口

C. Java 中一个类只可以有一个直接父类，只可以实现一个接口

D. Java 中一个类可以有多个直接父类，只可以实现一个接口

3. 关于子类覆盖（Override）父类的方法，下列说法正确的是__**B**___。

A. 子类方法与父类方法形式参数个数或者类型不同

B. 子类方法与父类方法的形式参数个数、形参类型、返回类型相容

C．子类方法与父类方法的访问权限必须相同

D．子类方法与父类方法形式参数名称必须相同

4. 定义类时不能使用的修饰符是____**D**_____。

A. abstract　　B．final　　C．public　　D．abstract final

5. 下面程序运行后的输出结果为___**D**__。（y 不是 static 类型）

```
class B{
    int y=3;
    static void showy( ){System.out.println("y="+y); }
}
class TestB{
    public static void main(String aaa
        []){ B a1=new B( );
        a1.y++;
        a1.showy( );
    }
}
```

A. y=3;　　B. y=4;

C. y=5;　　D. 程序编译出错

6. 给定以下类的定义

```
public class A{
    public A(){
        System.out.println("Constructor");
    }
    public static int i = 0;
}
```

则下列语句中会在控制台中打印出字符串 Constructor 的是___**C**_____。

A. A a = null;　　B. int k = A.i;

C. int k = new A().i;　　D. A[] array = new A[1];

7. class A extends B implements C，假定 A 和 B 有缺省构造方法，则下面的语句编译和运行正确的是___**C**___。

A．A a = new A( );　B b = a;　C c = b;

B．B b = new B( );　A a = (A) b;

C．A a = new A( );　B b = a; C c1 = a ,c2 = new A( );

D．A a = new A( );　C c = new A( );　B b = new C( );

8. 给定下列程序，程序的输出结果为__**A**____。

```
class Base {
    public Base(String s)
        { System.out.print("B");
    }
}
public class Derived extends Base
    { public Derived (String s) {
        System.out.print("D");
    }
    public static void main(String [] args)
        { new Derived ("C");
    }
}
```

那么结果为?

A. 编译错误　　B. DB　　C. BD　　D. BDC

9. 已知如下目录结构（dira和dirb为目录）

```
dira
|---A.class
```

```
    |---dirb
        |---B.class
```

和如下源代码：

```
import dira.*;
class C {
    A a;
    B b;
}
```

那么要使源代码通过编译，需要在源代码中添加___C___。

A. package dira;
B. package dirb;
C. package dira.dirb;
D. package dirb.dira;

10. 给定下列程序

```
interface I { }
class A implements I { }
class B extends A { }
class C extends B {
    public static void main(String[] args)
        { B b = new B();
        _____C_____
    }
}
```

在横线处添加下面哪条语句运行时会产生异常 。

A. A a = b;
B. I i = b;
C. C c = (C) b;
D. B d = (B) (A) b;

11. 下面程序的输出结果是__C____。

```
class C {
    public static void main(String[] args)
        { try {
            System.out.print(10 + 10 / 0);
        } catch(NullPointerException e1)
            { System.out.print("a");
        } catch(RuntimeException e2)
            { System.out.print("b");
        } finally {
            System.out.print("c");
        }
    }
}
```

A. a　　B. ac　　C. bc　　D. abc

12. 下面程序的执行结果是___D_____。

```
public class MyProgram{
    public static void main (String
        args[]){ try{
            System.out.print("Hello Java.");
        }
        finally{
            System.out.print("Finally Java.");
        }
    }
}
```

A． 无法编译，因为没有 catch 子句　　B． Hello Java.

C． Finally Java.　　D. Hello Java. Finally Java.

13. 下面程序的执行结果是____C_____。

```
public class Test7 {
    public static void main(String[]
        args){ new B().display();
    }
}
class A{
    public void draw() {
        System.out.print("Draw A.");
    }
    public void display()
        { draw();
        System.out.print("Display A.");
    }
}
class B extends A{
    public void draw() {
        System.out.print("Draw B.");
    }
    public void display()
        { super.display();
        System.out.print("Display B.");
    }
}
```

A. Draw A.Display A.Display B.　　B. Draw A.Display B.Display A.

C. Draw B.Display A.Display B.　　D. Draw B.Display B.Display A.

14. 语句 int[] m = new int[5];则 m[5]=10;会有_____**C**____。

A. 编译运行都正确；

B. 编译不正确

C. 会引发 ArrayIndexOutOfBoundsException 异常

D. 会引发 NullPointerException 异常

15. 下面程序执行结果是___**D**_____。

```
public class Test {
    public static void main(String args[])
        { try {
            System.out.print("try.");
            return;
        } catch(Exception
            e){ System.out.print("catch.");
        }finally {
            System.out.print("finally.");
        }
    }
}
```

A. try.catch.finally.　　B. try.

C. try.catch.　　D. try.finally

16. 给定下列程序，下面说法正确的是____**B**_____。

```
public class Test2_16 {
    public void m1() throws
        IOException{ try {
            throw new IOException();
        }
        catch (IOException e){

        }
    }
    public void
        m2(){ m1();
    }
}
```

A. 因 m1 方法里已经捕获了异常，因此 m2 里调用 m1()时不用处理异常，程序编译通过

B. m2 或者用throws 声明异常，或者在方法体里面用 try/catch 块去调用 m1 并捕获异常，否则编译报错

C. m2 方法体里面必须用try/catch 块去调用 m1 并捕获异常，否则编译报错

D. m2 方法必须用 throws 声明异常，否则编译报错

17. 给定下列程序，下面说法正确的是____**A**_____。

```
public class Test2_17 {
    public void m1() throws
        RuntimeException{ throw new
        RuntimeException();
    }


    public void
        m2(){ m1();
    }
}
```

A. 程序编译通过

B. m2 或者用throws 声明异常，或者在方法体里面用 try/catch 块去调用 m1 并捕获异常，否则编译报错

C. m2 方法体里面必须用try/catch 块去调用 m1 并捕获异常，否则编译报错

D. m2 方法必须用 throws 声明异常，否则编译报错

三、判断对错题

1. 包含有抽象方法的类必须是抽象类，抽象类也必须包含有抽象方法。(**X**)

2. 一个接口只能继承一个直接父接口。（**X**）

3. 非抽象类的子类不能是抽象类。（**X**）

4. 接口类型的引用变量能直接指向一个实现了该接口的类的实例而无需强制类型转换。(√)

5. import 语句通常出现在 package 语句之前。(**X**)

6. 抽象类中不能定义构造方法。(**X**)

7. this 关键字可以在类的所有方法里使用。(**X**)

8. 类 A 的所有实例方法都可以在 A 的子类中进行覆盖(Override)。(**X**)

9. 在类的静态初始化块里可以初始化类的静态数据成员和实例数据成员。(**X**)

10. 由于抽象类不能被实例化，因此方法的参数类型和返回类型都不能是抽象类类型。(**X**)

四、阅读题

阅读下列程序，写出输出结果：

```
class
    SuperClass{ stati
    c int i = 10; static{
        System.out.println(" static in SuperClass");
    }
    {
        System.out.println("SupuerClass is called");
    }
}
class SubClass extends
    SuperClass{ static int i = 15;
    static{
        System.out.println(" static in SubClass");
    }
    SubClass( ){
        System.out.println("SubClass is called");
    }
    public static void main(String[]
        args){ int i = SubClass.i;
        new SubClass( );
        new SuperClass( );
    }
}
```

运行结果：

static in SuperClass
static in SubClass
SupuerClass is called
SubClass is called
SupuerClass is called

五、阅读题

下面程序，写出指定语句的输出结果，并解释原因。

```
    public class Test5 {
        public static void main(String...
            args){ C o1 = new D();
            o1.m(1,1);              //①
            o1.m(1.0,1.0);          //②
            o1.m(1.0f, 1.0f);       //③

            D o2 = new D();
            o2.m(1,1);              //④
            o2.m(1.0,1.0);          //⑤
            o2.m(1.0f, 1.0f);       //⑥
        }
    }

    class C{
        public void m(int x, int y)
            { System.out.println("C's
            m(int,int)");
        }
        public void m(double x, double y)
            { System.out.println("C's
            m(double,double)");
        }
    }

    class D extends C{
        public void m(float x, float y)
            { System.out.println("D's
            m(float,float)");
        }
        public void m(int x, int y)
                                { System.out.
            println("D's m(int,int)");
        }
    }
```

语句①的输出结果为 D's m(int, int) ，原因是
o1 运行时指向 D 类对象，且 D 中的 m(int x, int y)覆盖了 C 的该方法
语句②的输出结果为 C's m(double, double) ，原因是
D 没有覆盖 C 中的该方法，故调用 C 的 m(double, double)方法 。
语句③的输出结果为 C' m(double, double) ，原因是
重载是早期绑定，由声明类型决定选择哪个方法 。
语句④的输出结果为 D's m(int, int) ，原因是
o2 运行时指向 D 类对象，D 中的 m(int x, int y)覆盖了 C 的该方法
语句⑤的输出结果为 C' m(double, double) ，原因是
D 没有方法覆盖 C 中的该方法，故调用 C 的 m(double, double)方法 。
语句⑥的输出结果为 D's m(float, float) ，原因是
o2 运行时指向 D 类对象，调用 D 中的 m(float, float) 。

六、阅读题

阅读下面程序，写出指定语句的输出结果，并解释原因。

```
public class Test_Hide_Override {
    public static void main(String...
        args){ A o = new C();
        o.m1();          //①
        o.m2();          //②
        ((B)o).m1();     //③
        ((A)(B)o).m1();  //④
        ((A)(B)o).m2();  //⑤
    }
}

class A{
    public static void m1(){   System.out.println("A's m1"); }
    public void m2(){ System.out.println("A's m2"); }
}

class B extends A{
    public static void m1(){   System.out.println("B's m1"); }
    public void m2(){ System.out.println("B's m2"); }
}

class C extends B{
    public static void m1(){   System.out.println("C's m1"); }
    public void m2(){ System.out.println("C's m2"); }
}
```

语句①的输出结果为_____A's m1_____，原因是

m1 为静态方法，被隐藏可以再发现，o 是 A 类引用，故访问 A 类的静态方法。

语句②的输出结果为__C's m2__，原因是

m2 为实例方法，父类的对应方法被覆盖，通过父类引用访问的方法是子类重新定义的方法。

语句③的输出结果为___B's m1___，原因是

m1 为静态方法，被隐藏可以再发现，((B)o)是 B 类引用，故访问 A 类的静态方法。

语句④的输出结果为___A's m1_____，原因是

m1 为静态方法，被隐藏可以再发现，((A5)(B5)o)是 A 类引用，故访问 A 类的静态方法。

语句⑤的输出结果为_C's m2_，原因是

m2 为实例方法，父类的对应方法被覆盖，通过父类引用访问的方法是子类重新定义的方法。

第 19 章作业

一、填空题

1. 语句 Class clz = null；的含义是**定义了一个名为 clz 的 Class 类的对象实例，并初始化为空对象。**

2. 给定下列类的定义：

```
class GeometricObject {}
class PolyGon extends GeometricObject {}
class Rectangle extends PolyGon {}
GeometricObject o = new Rectangle ();
Class clz1 = o. getClass ();
```

(1) 声明一个指向PolyGon 及其子类的类型信息的引用变量 clz 的语句应该是

_____**Class<? extends PolyGon> clz=null;**_____;

(2) System.out.println(o.getClass().getSimpleName());的输出结果是_**Rectangle**_____;

(3) 下列语句中有错误的是_____;

```
Class<PolyGon> clz3 = null;
clz3 = PolyGon.class;          ①
clz3 = Rectangle.class;        ②

Class<? extends PolyGon> clz4 = null;
clz4  =  GeometricObject.class;  ③
clz4 = PolyGon.class;          ④
clz4 = Rectangle.class;        ⑤
```

错误原因是（按错误题号解释）

②clz3 的类型为PolyGon，泛型中的类型可以持有其子类的引用；

③clz4 的通配符表示可以持有PolyGon 及其子类类型的引用（extends PolyGo）。

3. 下面五条语句中，错误的有_____**(2),(3)**_____。

(1) ArrayList<String> lists = new ArrayList<String>();

(2) ArrayList<Object> lists = new ArrayList<String>();

(3) ArrayList<String> lists = new ArrayList<Object>();

(4) ArrayList<String> lists = new ArrayList();

(5) ArrayList lists = new ArrayList<String>();

错误原因是**泛型不具有协变性，ArrayList<Object>与 ArrayList<String>不具有父子类关系。**

使用泛型通配符?将错误的语句修改正确的方法是

(2) ArrayList<? extends Object> lists = new ArrayList<String>();

(3) ArrayList<? super String> lists = new ArrayList<Object>();

4. 下面代码给出了泛型类和非泛型类的定义：

```
class Holder<T>
      { T value;
      public Holder (T value) {this.value = value;}
      public T getValue () {return value;}
}

class RawHolder
      { Object value;
      public RawHolder (Object value) {this.value = value;}
      public Object getValue () {return value;}
}
```

基于上面二个类的定义，有下面四段代码：

① Holder<String> h1 = new Holder<>("aaa"); String s1 = h1. getValue (); System.out.println(s1);	② RawHolder h1 = new RawHolder("aaa"); String s1 = (String)h1. getValue (); System.out.println(s1);
③ Holder<String> h1 = new Holder<> (Integer.valueOf(111)); String s1 = h1. getValue (); System.out.println(s1);	④ RawHolder h1 = new RawHolder (Integer.valueOf(111)); String s1 = (String)h1. getValue (); System.out.println(s1);

上面四段代码中编译通过运行不出错的是__**①②**___。

上面四段代码中编译通过运行出错的是_**④**_，原因是__**Integer 不能转换为 String**____，

上面四段代码中编译不通过是_**③**_，原因是_____**无法推断出构造方法的实参**____，

这个例子说明泛型的作用是_____**规范集合中的数据，保持类型一致性**_____。

二、单项选择题

1. 泛型参数<T>代表的是____**D**____。

A. 任意类型

B. 某类型的子类型

C. 某类型的父类型

D. 固定指代某种类型

2. 泛型通配符<?>代表的是___**A**____。

A. 任意类型

B. 某类型的子类型

C. 某类型的父类型

D. 固定指代某种类型

3. 下面泛型定义中不正确的是___**D**_____。

A. class Test1<T> {}

B. interface Test2<T> {}

C. class Test3{<T> void test () {}}

D. class Test4{void <T> test () {}}

4. 泛型通配符<? extends T>代表的是_____**B**____。

A. 任意类型

B. 某类型T 的子类型

C. 某类型T 的父类型

D. 固定指代某种类型

5. 泛型通配符<? super T>代表的是____**C**_____。

A. 任意类型

B. 某类型T 的子类型

C. 某类型T 的父类型

D. 固定指代某种类型

6. 关于下面代码，描述正确的是_____**C**_____。

```
List<String> list = new ArrayList<String>();
list.add("test");
list.add("red");
list.add (100);
System.out.println(list. size ());
```

A. 输出 2

B. 输出 3

C. 编译错误

D. 运行时报异常

7. 关于下面代码，描述正确的是_____**B**_____。

```
List<Integer> ex_int= new ArrayList<Integer> ();
List<Number> ex_num = ex_int;
System.out.println(ex_num. size ());
```

A. 0

B. 编译错误

C. 运行时报异常

D. 1

8. 下列语句编译时不出错的是____**D**_____。

A. List<?> c1 = new ArrayList<String> (); c1.add (new Object ());

B. List<?> c2 = new ArrayList<String> (); c2.add (new String ("1"));

C. List<?> c3 = new ArrayList<String> (); c3.add ("1");

D. List<?> c4 = new ArrayList<String> (); c4.add(null);

9. 给定下列代码：

```
class Shape {}
class Circle extends Shape {}
class Triangle extends Shape {}
public class Test2_9 {
    public static void main (String [] args) {
        List<? extends Shape> list1 = new ArrayList< Triangle> (); List<?
        extends Shape> list2 = new ArrayList<Circle> ();

        System.out.println(list1 instanceof List< Triangle>);          ①
        System.out.println(list2 instanceof List);                     ②
        System.out.println(list1.getClass() == list2.getClass());      ③
    }
}
```

则关于语句①②③说法正确的是：_____**D**_____。

A. ①②③输出结果为 true、false、false

B. ①②③输出结果为 true、true、true

C. ①编译出错，②③输出结果为 false、false

D. ①编译出错，②③输出结果为 true、true

三、多项选择题（一个或多个正确选项）

1. 对于泛型类 class A<T> { … }，T 在 A 类里可以用作不同的地方，在 A 类类体内，下面语句正确的有_____**ABDG**_____。

A. T x;

B. T m1() {return null;}

C. static T y;

D. void m2(T i) {}

E. static T s1() {return null;}

F. static void s2(T i) {}

G. static <T1> void s3(T1 i, T1 j){}

2. 下列语句编译时不出错的是___**AEGH**_____。

A. List<? super Integer> x1 = new ArrayList<Number> ();

B. List<? super Number> x2 = new ArrayList<Integer> ();

C. List<? super Number> x3 = new ArrayList<Short> ();

D. List<? super Integer> x4 = new ArrayList<Short> ();

E. List<? extends Number> x5 = new ArrayList<Integer> ();

F. List<? extends Number> x6 = new ArrayList<Object> ();

G. List<Number> x7 = new ArrayList<> ();

H. List<? extends Comparable<Double>> x8 = new ArrayList<Double> ();

I. List<? extends Number> x9 = new ArrayList<int> ();

3. 下面泛型类是 List<?>的子类的是_____**ABCD**_____。

A. List<String>

B. List<Object>

C. List<Integer>

D. List<FLOAT>

4. 泛型参数应该写在的位置是_____**BD**_____。

A. 类名前

B. 类名后

C. 方法名前

D. 方法返回值类型前

5. 关于 java 泛型，下面描述正确的是_____**ABCD**_____。

A. 泛型的类型参数只能是类类型（包括自定义类），不能是基本类型

B. 泛型的类型参数可以有多个

C. 不能对泛型的具体实例类型使用 instanceof 操作，如 o instanceof ArrayList<String>，否则编译时会出错。

D. 不能创建一个泛型的具体实例类型的数组，如 new ArrayList<String>[10]，否则编译时会出错。

6. 给定下列类和泛型方法的定义：

```
class A {}
class  B  extends  A  {}
class  C  extends  B  {}
class  D  extends  C  {}
public class Test2_9{
      public static <T> void m (List<? super T> list1, List<? extends T> list2) {}
}
```

则下面 6 段代码编译出错的是_____**CEF**_____。

A.
```
List<B> l1 = new ArrayList<> ();
List<B> l2 = new ArrayList<> ();
Test2_9.m (l1, l2);
```
B.
```
List<B> l3 = new ArrayList<> ();
List<D> l4 = new ArrayList<> ();
Test2_9.m (l3, l4);
```
C.
```
List<B> l5 = new ArrayList<> ();
List<A> l6 = new ArrayList<> ();
Test2_9.m (l5, l6);
```
D.
```
List<C> l7 = new ArrayList<> ();
List<D> l8 = new ArrayList<> ();
Test2_9.m (l7, l8);
```
E.
```
List<C> l7 = new ArrayList<> ();
List<D> l8 = new ArrayList<> ();
Test2_9. <B>m (l7, l8);
```
F.
```
List<D> l9 = new ArrayList<> ();
List<C> l10 = new ArrayList<> ();
Test2_9.m (l9, l10);
```

四、问答题

阅读下列程序，并填写表格

```
import java.util.*;
class A {}
class B extends A {}
class Test {
        public static void m1(List<? extends A> list) {}
        public static void m2(List<A> list) {}
        public static void m3(List<? super A> list) { }
        public static void main (String [] args) {
              List<A> listA = new ArrayList<A> ();
              List<B> listB = new ArrayList<B> ();
              List<Object> listO = new ArrayList<Object> ();

            // insert code here
      }
}
```

在上面代码插入点插入的代码	结果（从下面结果选项中选择）
m1(listA);	**C**
m2(listA);	**C**
m3(listA);	**C**
m1(listB);	**C**
m2(listB);	**A**
m3(listB);	**A**
m1(listO);	**A**
m2(listO);	**A**
m3(listO);	**C**
结果选项	
A. 编译出错	
B. 编译正确，运行出错	
C.编译正确，运行正确	

第 19 章编程作业

一、编程题 1

下列代码定义了迭代器接口，数组迭代器和容器类：

```
/**
 * 迭代器接口，用于遍历组件树里的每一个组件. 注意这不是 java.util.Iterator 接口
 */
interface Iterator {
    /**
     *  是否还有元素
     * @return 如果元素还没有迭代完，返回 true;否则返回 false
     */
    boolean hasNext();

    /**
     * 获取下一个元素
     * @return  下一个元素
     */
    Object next();
}

/**
 * 数组迭代器
 */
class  ArrayIterator implements
    Iterator{ private int pos = 0;
    private Object[] a = null;

    public ArrayIterator(Object[]
        array){ a = array;
    }

    @Override
    public boolean hasNext()
        { return !(pos >= a.length);
    }

    @Override
    public Object next()
        { if(hasNext()){
            Object c = a[pos];
            pos ++;
            return c;
        }
        else
            return null;
    }
}
```

```
/**
 * 容器类，内部用 Object[]保存元素
 */
class Container {
    private Object[] elements;
    private int elementsCount = 0;
    private int size = 0;

    public Container(int
        size){ elements = new
        Object[size]; this.size =
        size;
    }

    public boolean add(Object
        e){ if(elementsCount <
        size){
            elements[elementsCount ++] = e;
            return true;
        }
        else{
            return  false;
        }
    }

    /**
     * 返回容器的迭代器
     * @return
     */
    public Iterator iterator(){
        return new ArrayIterator(elements);
    }
}
public class Test{
    public static void main(String[]
        args){ Container container = new
        Container(6); container.add("12");
        container.add("34");
        container.add("56");
        container.add("78");
        container.add("9");
        container.add(10);
        Iterator it = container.iterator();
        while (it.hasNext()){
            String s = (String)it.next();
            if( s != null)
                System.out.println(s);
        }
```

1. 上述代码存在什么问题？请分析存在问题的原因。

container.add(10);一行存在问题，由于 container 中存在 int 类型数据 10，不属于 String 类型数据，会在遍历时抛出类型转换异常。

2. 请将迭代器接口 Iterator、数组迭代器ArrayIterator、容器 Container 分别改成泛型迭代器接口 Iterator<T>、泛型数组迭代器 ArrayIterator<T>、泛型容器 Container<T>，并写出和上面一样的测试代码。要求泛型代码实现和非泛型接口/类同样的方法，另外泛型容器Container<T>也必须实现iterator 方法。

```
/**
  * 迭代器接口，用于遍历组件树里的每一个组件．注意这不是 java.util.Iterator 接口
 */
interface Iterator<T> {
    /**
     * 是否还有元素
     *
     * @return 如果元素还没有迭代完，返回 true;否则返回 false
     */
    boolean hasNext();

    /**
     * 获取下一个元素
     *
     * @return 下一个元素
     */
    T next();
}

/**
 * 数组迭代器
 */
class ArrayIterator<T> implements Iterator<T>
    { private int pos = 0;
    private T[] a = null;

    public ArrayIterator(T[] array)
        { a = array;
    }

    @Override
    public boolean hasNext()
        { return !(pos >=
        a.length);
    }

    @Override
    public T next()
        { if (hasNext())
            pos++;
            return c;
        } else
            return null;
    }
}
import java.lang.reflect.Array;

/**
 * 容器类，内部用 T[]保存元素
 */
class Container<T> {
    private Object[] elements;
    private int elementsCount = 0;
    private int size = 0;

    public Container(int
        size){ elements = new
        Object[size]; this.size =
        size;
    }

    public boolean add(T
        e){ if(elementsCount < size){
            elements[elementsCount ++] = e;
            return true;
        }
        else{
            return  false;
        }
    }

    /**
     * 返回容器的迭代器
     * @return new ArrayIterator<T>((T[])elements)
     */
    public Iterator<T> iterator(){
        return new ArrayIterator<T>((T[])elements);
    }
}
public class Test {

    public static void main(String[] args) {
        testInteger();
        testString();
```

```
    }

    public static void testInteger()
        { System.out.println("-----Test Integer --");
        Container<Integer> container = new Container<>(6);
        container.add(1);
        container.add(1);
        container.add(3);
        container.add(4);
        container.add(5);
        container.add(10);
        Iterator<Integer> it = container.iterator();
        while (it.hasNext()) {
            System.out.println(it.next());
        }
    }

    public       static       void       testString()
        {    System.out.println("-----Test    String
        ");
        Container<Object> container = new Container<>(6);
        container.add("12");
        container.add("34");
        container.add("56");
        container.add("78");
        // container.add(10);
        Iterator<Object> it = container.iterator();
        while (it.hasNext()) {
            String s = (String) it.next();
            if (s != null)
                System.out.println(s);
        }
}
```

二、编程题 2

实现一个泛型二元组类 TwoTuple<T1, T2>，其中T1、T2 分别是二元组第 1 个、第 2 个元素的类型参数，要求如下：

（1） T1、T2 是实现了 Comparable 接口的类型；

（2） TwoTuple<T1, T2>必须实现 Comparable 接口；

（3） TwoTuple<T1, T2>必须覆盖 equals 方法和 toString 方法；

（4） TwoTuple<T1, T2>二个私有数据成员变量名分别为 first, second；

（5） 必须实现私有数据成员 first, second 的公有 getter 和 setter 方法；

（6）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；

（7） 覆盖 equals 方法的实现语义是：二个元组对象的 first 和 second 分别都相等时，这个二个元组对象相等。

测试代码的输出结果应为：

```
true
(1, aaa)
(1, bbb)
(1, ccc)
(2, aaa)
(2, bbb)
(2, ccc)
-1
((1, aaa), (1, bbb))
```

```
package Exercise2;

import java.util.Objects;

public class TwoTuple<T1 extends Comparable<T1>, T2 extends Comparable<T2>>
implements Comparable<TwoTuple<T1, T2>> {
    private T1 first;
    private T2 second;

    public TwoTuple(T1 first, T2 second)
        { this.first = first;
        this.second = second;
    }
```

```
    @Override
    public boolean equals(Object o)
        { if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        TwoTuple<T1, T2> T= (TwoTuple<T1, T2>) o;
        return Objects.equals(first, T.first) && Objects.equals(second, T.second);
    }

    public T2 getSecond()
        { return second;
    }

    public void setSecond(T2 second)
        { this.second = second;
    }

    public T1 getFirst()
        { return first;
    }

    public void setFirst(T1 first)
        { this.first = first;
    }

    @Override
    public String toString() {
        return "(" + first + "," + second + ")";
    }

    @Override
    public int compareTo(TwoTuple<T1, T2> o)
        { if (first.equals(o.first))
            return second.compareTo(o.second);
        else
            return first.compareTo(o.first);
    }
}
```

```
package Exercise2;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class TestTwoTuple {
    public static void main(String[] args) {

        TwoTuple<Integer, String> twoTuple1 = new TwoTuple<>(1, "ccc");
        TwoTuple<Integer, String> twoTuple2 = new TwoTuple<>(1, "bbb");
        TwoTuple<Integer, String> twoTuple3 = new TwoTuple<>(1, "aaa");
        TwoTuple<Integer, String> twoTuple4 = new TwoTuple<>(2, "ccc");
        TwoTuple<Integer, String> twoTuple5 = new TwoTuple<>(2, "bbb");
        TwoTuple<Integer, String> twoTuple6 = new TwoTuple<>(2, "aaa");
        List<TwoTuple<Integer, String>> list = new ArrayList<>();
        list.add(twoTuple1);
        list.add(twoTuple2);
        list.add(twoTuple3);
        list.add(twoTuple4);
        list.add(twoTuple5);
        list.add(twoTuple6);

        //测试 equals，contains 方法是基于 equals 方法结果来判断
        TwoTuple<Integer, String> twoTuple10 = new TwoTuple<>(1, "ccc"); //内容
=twoTuple1
        System.out.println(twoTuple1.equals(twoTuple10)); //应该为 true
        if (!list.contains(twoTuple10)) {
            list.add(twoTuple10);  //这时不应该重复加入
        }

        //sort 方法是根据元素的 compareTo 方法结果进行排序，课测试 compareTo 方法是否实现
正确
        Collections.sort(list);


        for (TwoTuple<Integer, String> t : list)
            { System.out.println(t);
        }

        TwoTuple<TwoTuple<Integer, String>, TwoTuple<Integer, String>> tt1 =
                new TwoTuple<>(new TwoTuple<>(1, "aaa"), new TwoTuple<>(1, "bbb"));
        TwoTuple<TwoTuple<Integer, String>, TwoTuple<Integer, String>> tt2 =
                new TwoTuple<>(new TwoTuple<>(1, "aaa"), new TwoTuple<>(2, "bbb"));
        System.out.println(tt1.compareTo(tt2));  //输出-1
        System.out.println(tt1);
    }
}
```

第 30 章作业

一、填空题

1. 创建线程的方式有**继承 Thread 类并覆盖 run 方法**和**实现接口 Runnable 的 run 方法。**

2. 程序中可能出现一种情况：多个线种互相等待对方持有的锁，而在得到对方的锁之前都不会释放自己的锁，这就是**死锁。**

3. 若在线程的执行代码中调用 yield 方法后，则该线程将**从运行态变为就绪态，允许其他线程执行，同时该线程也可能立即执行。**

4. 线程程序可以调用 **sleep()**方法，使线程进入睡眠状态，可以通过调用 **setPriority()**方法设置线程的优先级。

5. 获得当前线程 id 的语句是 **currentThread().getId()。**

二、单项选择题

1. 能够是线程进入死亡状态的是 **C。**

A. 调用 Thread 类的 yield 方法

B. 调用 Thread 类的 sleep 方法

C. 线程任务的run 方法结束

D. 线程死锁

2. 给定下列程序：

```
public class Holder
    { private int data = 0;
    public int getData () {return data;}
    public synchronized void inc (int amount)
        { int newValue = data + amount;
        try {Thread.sleep(5);
        } catch (InterruptedException e) {}
        data = newValue;
    }
    public void dec (int amount) {
        int newValue = data - amount;
        try {Thread.sleep(1);
        } catch (InterruptedException e) {}
        data = newValue;
    }
}
public static void main (String [] args) {
    ExecutorService es = Executors.newCachedThreadPool();
    Holder holder = new Holder ();
    int incAmount = 10, decAmount = 5, loops = 100;
    Runnable incTask = () -> holder.inc(incAmount);
    Runnable decTask = () -> holder.dec(decAmount);
    for (int i = 0; i < loops; i++) {
        es. execute(incTask);
        es. execute(decTask);
    }
    es. shutdown ();
    while (! es. isTerminated ()) {}
}
```

下列说法正确的是 **B。**

A. 当一个线程进入 holder 对象的 inc 方法后，holder 对象被锁住，因此其他线程不能进入 inc 方法和 dec 方法

B. 当一个线程进入 holder 对象的 inc 方法后，holder 对象被锁住，因此其他线程不能进入 inc 方法，但可以进入 dec 方法

C. 当一个线程进入 holder 对象的 dec 方法后，holder 对象被锁住，因此其他线程不能进入 dec 方法和 inc 方法

D. 当一个线程进入 holder 对象的 dec 方法后，holder 对象被锁住，因此其他线程不能进入 dec 方法，但可以进入 inc 方法

3. 给定下列程序：

```
public class Test2_3 {
    private static Object lockObject = new Object ();
    /**
     * 计数器
     */
    public static class Counter
        { private int count = 0;
        public int getCount () {return count;}
        public void inc () {
```

```
            synchronized (lockObject)
                { int temp = count + 1;
                try {Thread.sleep(5);} catch (InterruptedException e) {}
                count = temp;
            }
        }
        public void dec ()
            {  synchronized
            (lockObject) {
                int temp = count - 1;
                try {Thread.sleep(5);} catch (InterruptedException e) {}
                count = temp;
            }
        }
    }

    public static void main (String [] args) {
        ExecutorService es = Executors.newCachedThreadPool();
        Counter counter1 = new Counter ();
        Counter counter2 = new Counter ();
        int loops1 = 10, loops2 = 5;
        Runnable incTask = () -> counter1.inc ();
        Runnable decTask = () -> counter2.dec ();
        for (int i = 0; i < loops1; i++) {es. execute(incTask);}
        for (int i = 0; i < loops2; i++) {es. execute(decTask);}
        es. shutdown ();
        while (! es. isTerminated ()) {}
    }
}
```

下面说法正确的是 **C。**

A. incTask 的执行线程进入 counter1 对象的 inc 方法后，counter1 对象被上锁，会阻塞 decTask 的执行线程进入 counter2 对象的 dec 方法

B. incTask 的执行线程进入 counter1 对象的 inc 方法后，counter1 对象被上锁，不会阻塞 decTask 的执行线程进入 counter2 对象的 dec 方法

C. incTask 的执行线程进入对象counter1 的inc 方法后，lockObject 对象被上锁，会阻塞decTask 执行线程进入 counter2 对象的方法 dec

D. incTask 的执行线程进入对象 counter1 的 inc 方法后，lockObject 对象被上锁，不会阻塞 decTask 执行线程进入 counter2 对象的方法 dec

4. 给定下列程序：

```
public class Test2_4 {
    public static class Resource
        { private int value = 0;
        public int sum (int amount) {
            int newValue = value + amount;
            try {Thread.sleep(5);} catch (InterruptedException e) {}
            return newValue;
        }
        public int sub (int amount) {
            int newValue = value - amount;
            try {Thread.sleep(5);} catch (InterruptedException e) {}
            return newValue;
        }
    }

    public static void main (String [] args) {
        ExecutorService es = Executors.newCachedThreadPool();
        Resource r = new Resource ();
        int loops1 = 10, loops2 = 5, amount = 5;
        Runnable sumTask = () -> r.sum(amount);
        Runnable subTask = () -> r.sub(amount);
        for (int i = 0; i < loops1; i++) {es. execute(sumTask);}
        for (int i = 0; i < loops2; i++) {es. execute(subTask);}
        es. shutdown ();
        while (! es. isTerminated ()) {}
    }
}
```

下面说法正确的是 **C。**

A. 由于方法 sum 和 sub 都没有采取任何同步措施，所以 sumTask 和 subTask 的执行线程都可以同时进入共享资源对象 r 的 sum 方法或 sub 方法，造成对象r 的实例成员 value 的值不一致；

B. 由于方法 sum 和 sub 都没有采取任何同步措施，所以 sumTask 和 subTask 的执行线程都可以同时进入共享资源对象 r 的 sum 方法或 sub 方法，造成方法内局部变量 newValue 和形参 amount 的值不一致；

C. 虽然方法 sum 和 sub 都没有采取任何同步措施，但 Resource 类的 sum 和 sub 里的局部变量 newValue 和形参 amount 位于每个线程各自的堆栈里互不干扰，同时多个线程进入共享资源对象r 的 sum 方法或sub 方法后，对实例数据成员 value 都只有读操作，因此 Resource 类是线程安全的

D. 以上说法都不正确

5. 给定下列程序：

```
public class Test2_5 {
    public static class Resource
        { private static int value = 0;
        public static int getValue () {return value;}
        public static void inc (int amount) {
            synchronized (Resource. Class) {
                int newValue = value + amount;
                try {Thread.sleep(5);} catch (InterruptedException e) {}
                value = newValue;
            }
        }
        public synchronized static void dec (int amount)
            { int newValue = value - amount;
            try {Thread.sleep(2);} catch (InterruptedException e) {}
            value = newValue;
        }
    }
    public static void main (String [] args) {
        ExecutorService es = Executors.newCachedThreadPool();
        int incAmount = 10, decAmount = 5, loops = 100;
        Resource r1 = new Resource ();
        Resource r2 = new Resource ();
        Runnable incTask = () -> r1.inc(incAmount);
        Runnable decTask = () -> r2.dec(decAmount);
        for (int i = 0; i < loops; i++) {es. execute(incTask); es. execute(decTask);}
        es. shutdown ();
        while (! es. isTerminated ()) {}
    }
}
```

下面说法**错误的**的是 **B。**

A. 同步的静态方法 public synchronized static void dec (int amount) {} 等价于 public static void dec (int amount) {synchronized (Resource. class) {}}

B. incTask 的执行线程访问的对象 r1，decTask 访问的是对象r2，由于访问的是不同对象，因此 incTask 的执行线程和 decTask 的执行线程之间不会同步

C. 虽然 incTask 的执行线程和 decTask 的执行线程访问的是Resource 类不同对象r1 和 r2，但由于调用的是Resource 类的同步静态方法，因此 incTask 的执行线程和 decTask 的执行线程之间是被同步的

D. 一个线程进入Resource 类的同步静态方法后，这个类的所有静态同步方法都被上锁，而且上的是对象锁，被锁的对象是 Resource.class。但是这个锁的作用范围是 Resource 类的所有实例，即不管线程通过Resource 类的哪个实例调用静态同步方法，都将被阻塞

6. 假设一个临界区通过 Lock 锁进行同步控制，当一个线程拿到一个临界区的 Lock 锁，进入该临界区后，该临界区被上锁。这时下面的说法正确的是 **D。**

A. 如果在临界区里线程执行 Thread.sleep 方法，将导致线程进入阻塞状态，同时临界区的锁会被释放；如果在临界区里线程执行 Lock 锁的条件对象的 await 方法，将导致线程进入阻塞状态，同时临界区的锁会被释放

B. 如果在临界区里线程执行 Thread.sleep 方法，将导致线程进入阻塞状态，同时临界区的锁不会被释放；如果在临界区里线程执行 Lock 锁的条件对象的 await 方法，将导致线程进入阻塞状态，同时临界区的锁不会被释放

C. 如果在临界区里线程执行 Thread.sleep 方法，将导致线程进入阻塞状态，同时临界区的锁会被释放；如果在临界区里线程执行 Lock 锁的条件对象的 await 方法，将导致线程进入阻塞状态，同时临界区的锁不会被释放

D. 如果在临界区里线程执行 Thread.sleep 方法，将导致线程进入阻塞状态，同时临界区的锁不会被释放；如果在临界区里线程执行 Lock 锁的条件对象的await 方法，将导致线程进入阻塞状态，同时临界区的锁会被释放

三、问答题

1：有三个线程 T1，T2，T3，怎么确保它们按指定顺序执行：首先执行 T1，T1 结束后执行 T2，T2 结束后执行T3，T3 结束后主线程才结束。请给出示意代码。

```
T1.start();
T1.join();
T2.start();
T2.join();
T3.start();
T3.join();
```

第 30 章编程作业

一、编程题 1

下面程序是一个泛型容器 Container<T>的定义，该容器是对 ArrayList 的一个封装，实现了四个公有的方法 add、remove、size、get。

```
class Container<T>{
    private List<T> elements = new ArrayList<>();

    /**
     * 添加元素
     * @param e 要添加的元素
     */
    public  void add(T
        e){ elements.add(e);
    }

    /**
     * 删除指定下标的元素
     * @param index 指定元素下标
     * @return 被删除的元素
     */
    public T remove(int
        index){  return
        elements.remove(index);
    }

    /**
     * 获取容器里元素的个数
     * @return 元素个数
     */
    public int size(){
        return elements.size();
    }

    /**
     * 获取指定下标的元素
     * @param index 指定下标
     * @return 指定下标的元素
     */
    public T get(int
        index){  return
        elements.get(index);
```

现在用如下程序对泛型容器进行测试。

```
public class Test {
    public static void testAdd(){
        Container<Integer> container = new Container<>();
        int addLoops = 10;  //addTask 内的循环次数
     Runnable addTask = new Runnable() {
           @Override
           public void run() {
               for(int i = 0; i < addLoops;
                   i++){ container.add(i);
               }
           }
       };

       int addTaskCount = 100; //addTask 线程个数
     ExecutorService es = Executors.newCachedThreadPool();
       for(int i = 0; i < addTaskCount; i++){
           es.execute(addTask);
       }

       es.shutdown();
       while (!es.isTerminated()){}
       System.out.println("Test add " + (addLoops * addTaskCount) +
               " elements to container");
       System.out.println("Container size = " + container.size() +
               ", correct size = " + (addLoops * addTaskCount));

    }

    public static void testRemove(){
        Container<Integer> container = new Container<>();
        int removeLoops = 10; //removeTask 内的循环次数
     int removeTaskCount = 100; //removeTask 线程个数

     //首先添加 removeLoops * removeTask 个元素到容器
     for(int i = 0; i < removeLoops * removeTaskCount;
           i++){ container.add(i);
       }

       Runnable removeTask = new Runnable()
           { @Override
           public void run() {
               for(int i = 0 ; i < removeLoops;
                   i++){ container.remove(0);
               }
           }
       };

       ExecutorService es = Executors.newCachedThreadPool();
       for(int i = 0; i < removeTaskCount; i++){
           es.execute(removeTask);
       }

       es.shutdown();
       while (!es.isTerminated()){}

       System.out.println("Test remove " + (removeLoops * removeTaskCount) +
               " elements from container");
       System.out.println("Container size = " + container.size() +
               ", correct size = 0");
    }


    public static void main(String[] args){
        testAdd();
```

```
        testRemove();
    }
}
```

1） 请运行上面的测试程序来测试泛型 Container 是否是线程安全的，请分析 ArrayList 的 add 方法与 remove 方法的源代码，简单分析导致泛型 Container 不是线程安全的原因；

不是线程安全的，当多个线程同时调用 add 和 remove 方法时，线程间执行的顺序不同，会导致结果不同，因此是非线程安全的。

2）请实现泛型 Container<T>的一个线程安全的版本 SynchronizedContainer<T>，只需要实现与 Container<T>一样的 4 个公有方法。要求必须用 synchronized 同步语句块或 Lock 锁实现

3）用上面同样的测试代码，来测试 SynchronizedContainer<T>的线程安全性；

```
package Excercise1;

import java.util.ArrayList;
import java.util.List;

class Container<T> {
    private List<T> elements = new ArrayList<>();

    public synchronized void add(T e)
        { elements.add(e);
    }

    public synchronized T remove(int index)
        { return elements.remove(index);
    }

    public int size() {
        return elements.size();
    }

    public T get(int index)
        { return
        elements.get(index);
    }
}
package Excercise1;

import java.util.concurrent.Executors;

public class Test {
    public static void testAdd() {
        Container<Integer> container = new Container<>();
        int addLoops = 10;  //addTask 内的循环次数
        Runnable addTask = new Runnable() {
            @Override
            public synchronized void run() {
                for (int i = 0; i < addLoops; i++)
                    { container.add(i);
                }
            }
        };

        int addTaskCount = 100; //addTask 线程个数
        ExecutorService es = Executors.newCachedThreadPool();
        for (int i = 0; i < addTaskCount; i++) {
            es.execute(addTask);
        }

        es.shutdown();
        while (!es.isTerminated()) {
        }
        System.out.println("Test add " + (addLoops * addTaskCount) +
                " elements to container");
        System.out.println("Container size = " + container.size() +
                ", correct size = " + (addLoops * addTaskCount));

    }

    public static void testRemove() {
        Container<Integer> container = new Container<>();
        int removeLoops = 10; //removeTask 内的循环次数
        int removeTaskCount = 100; //removeTask 线程个数

        //首先添加 removeLoops * removeTask 个元素到容器
        for (int i = 0; i < removeLoops * removeTaskCount; i++)
            { container.add(i);
        }

        Runnable removeTask = new Runnable()
            { @Override
            public synchronized void run() {
```

```
            for (int i = 0; i < removeLoops; i++)
                { container.remove(0);
                }
            }
        };

        ExecutorService es = Executors.newCachedThreadPool();
        for (int i = 0; i < removeTaskCount; i++) {
            es.execute(removeTask);
        }

        es.shutdown();
        while (!es.isTerminated()) {
        }

        System.out.println("Test remove " + (removeLoops * removeTaskCount) +
                " elements from container");
        System.out.println("Container size = " + container.size() +
                ", correct size = 0");
    }

    public static void main(String[] args) {
        testAdd();
        testRemove();
    }
}
```

二、编程题 2

实现一个线程安全的同步队列 SyncQueue<T>，模拟多线程环境下的生产者消费者机制，SyncQueue<T>的定义如下：

```
/**
 * 一个线程安全同步队列，模拟多线程环境下的生产者消费者机制
 * 一个生产者线程通过 produce 方法向队列里产生元素
 * 一个消费者线程通过 consume 方法从队列里消费元素
 * @param <T> 元素类型
 */
public class SyncQueue<T> {
    /**
     * 保存队列元素
     */
    private ArrayList<T> list = new ArrayList<>();

    //TODO 这里加入需要的数据成员


    /**
     * 生产数据
     * @param elements  生产出的元素列表，需要将该列表元素放入队列
     * @throws InterruptedException
     */
    public void produce(List<T> elements) {
        //TODO 这里需要实现代码
    }

    /**
     * 消费数据
     * @return 从队列中取出的数据
     * @throws InterruptedException
     */
    public List<T> consume(){
        //TODO 这里需要实现代码
    }
}
```

SyncQueue<T>的测试代码为：创建一个生产者线程不断地向队列生产数据，创建一个消费者线程不断地从队列消费数据，**要求生产出来的数据和消费的数据次序完全一样。**测试代码如下所示：

```
public class TestSyncQueue {
    public static void main(String[]
        args){ SyncQueue<Integer> syncQueue = new
        SyncQueue<>(); Runnable produceTask = ()->{
            while(true){ t
                ry {
                    List<Integer> list = new ArrayList<>();
                    int elementsCount = (int)(Math.random() * 10) + 1;
                    for(int i = 0; i < elementsCount; i++){
                        int r = (int)(Math.random() * 10) + 1;
                        list.add(r);
                    }
                    syncQueue.produce(list);
                    Thread.sleep((int)(Math.random() * 5) + 1);
                }
                catch (InterruptedException e) { e.printStackTrace(); }
            }
        };

        Runnable consumeTask = ()-
            >{ while (true){
                try{
                    List<Integer> list = syncQueue.consume();
                    Thread.sleep((int)(Math.random() * 10) + 1);
                }
                catch (InterruptedException e) { e.printStackTrace(); }
            }
        };

        ExecutorService es = Executors.newFixedThreadPool(2);
        es.execute(produceTask);
        es.execute(consumeTask);
```

```
        es.shutdown();
        while (!es.isTerminated()){}
    }
}
```

要求实现基于二个版本的 SyncQueue<T>，第 1 个版本为类名为 SyncQueue1<T>，第 1 个版本为类名为SyncQueue2<T>，其功能要求分别为：

1）SyncQueue1<T>实现生产者线程和消费者线程**轮流**生产数据和消费数据，即如果队列不为空，则生产者线程必须等到消费者线程将队列里的数据消费完后才能向队列生产数据；如果队列为空，则消费者线程必须等待生产者线程向队列生产数据后才能消费数据。这个版本的测试结果应该如下所示：

Produce elements: 4 5 3 1 9 6 10 6 7 6

Consume elements: 4 5 3 1 9 6 10 6 7 6

Produce elements: 2 5 9 9 7 10

Consume elements: 2 5 9 9 7 10

Produce elements: 8 8 9 2

Consume elements: 8 8 9 2

```
package Excercise2;

import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class SyncQueue1<T> {
    /**
     * 保存队列元素
     */
    private ArrayList<T> list = new ArrayList<>();
    private static Lock lock = new ReentrantLock();
    private static Condition condition1 = lock.newCondition();
    private static Condition condition2 = lock.newCondition();

    /**
     * 生产数据
     *
     * @param elements 生产出的元素列表，需要将该列表元素放入队列
     */
public void produce(List<T> elements) {
    lock.lock();
    while (!list.isEmpty())
        { try {
            condition1.await();
        } catch (InterruptedException e1)
            { e1.printStackTrace();
        }
    }
    list.addAll(elements);
    condition2.signalAll();
    System.out.print("Produce elements: ");
    for (T e : elements) {
        System.out.print(e + " ");
    }
    System.out.println();
    lock.unlock();
}

/**
 * 消费数据
 *
 * @return 从队列中取出的数据
 */
public List<T> consume() {
    lock.lock();
    while (list.isEmpty())
        { try {
            condition2.await();
        } catch (InterruptedException e1)
            { e1.printStackTrace();
        }
    }
    System.out.print("Consume elements: ");
    for (T e : list) {
        System.out.print(e + " ");
    }
    System.out.println();
    List<T> l = list;
    list.removeAll(list);
    condition1.signalAll();
    lock.unlock();
    return l;
```

```
    }
}
```

2）SyncQueue2<T>要求有些区别：即**生产者线程不管队列是否为空，随时可以向队列生产数据**；消费者线程则在队列为空时，必须等待生产者线程向队列生产数据后才能消费数据。

这个版本的测试结果应该如下所示：

Produce elements: 7 9 2 7 6 8 9 1 7 3

Produce elements: 7 5 9 8 8

Consume elements: 7 9 2 7 6 8 9 1 7 3 7 5 9 8 8

Produce elements: 6 2 8

Produce elements: 3 2 5 10 3 4 5 1 6

Produce elements: 5 7

Consume elements: 6 2 8 3 2 5 10 3 4 5 1 6 5 7

Produce elements: 3

Consume elements: 3

Produce elements: 10 3 4

Consume elements: 10 3 4

```
package Excercise2;

import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

/**
 * 一个线程安全同步队列，模拟多线程环境下的生产者消费者机制
 * 一个生产者线程通过 produce 方法向队列里产生元素
 * 一个消费者线程通过 consume 方法从队列里消费元素
 *
 * @param <T> 元素类型
 */
public class SyncQueue2<T> {
    /**
     * 保存队列元素
     */
    private ArrayList<T> list = new ArrayList<>();
    private static Lock lock = new ReentrantLock();
    private static Condition condition = lock.newCondition();

    /**
     * 生产数据
     *
     * @param elements 生产出的元素列表，需要将该列表元素放入队列
     */
    public void produce(List<T> elements) {
        lock.lock();
        list.addAll(elements);
        condition.signalAll();
        System.out.print("Produce elements: ");
        for (T e : elements) {
            System.out.print(e + " ");
        }
        System.out.println();
        lock.unlock();
    }

    /**
     * 消费数据
     *
     * @return 从队列中取出的数据
     */
    public List<T> consume() {
        lock.lock();
        while (list.isEmpty())
            { try {
                condition.await();
            } catch (InterruptedException e1)
                { e1.printStackTrace();
            }
        }
        System.out.print("Consume elements: ");
        for (T e : list) {
            System.out.print(e + " ");
        }
        System.out.println();
        List<T> l = list;
        list.removeAll(list);
        lock.unlock();
        return l;
```

```
    }
}
```

三、编程题 3

我们知道 JDK 提供了线程池的支持，线程池可以通过重复利用已创建的线程降低线程创建和销毁造成的消耗。但是 Thread 一旦启动，执行完线程任务后，就不可再次启动。请看下面示例代码：

```
public class ThreadTest {
    public static void main(String[] args){
        Runnable task = ()->{ System.out.println("task is running"); };
        Thread t = new Thread(task);
        t.start();
        try { Thread.sLeep(1000); }
        catch (InterruptedException e) { e.printStackTrace(); }
        System.out.println(t.isAlive()); //当线程任务结束后，isAlive 返回 false
        t.start(); //一旦线程结束，不可再 start，否则抛出异常
    }
}
```

上面运行结果如下：

```
D:\jdk-13.0.2-64bit\bin\java.exe --module-path D:\javafx-sdk-11.0.2\lib --add-mo
task is running
false
Exception in thread "main" java.lang.IllegalThreadStateException
    at java.base/java.lang.Thread.start(Thread.java:790)
    at hust.cs.javacourse.ch30.thread.pool.ThreadTest.main(ThreadTest.java:17)

Process finished with exit code 1
```

那么 JDK 线程池是如何做到线程复用的？这是一个值得思考的问题。这里的线程复用指的是一个 Thread 线程对象一旦启动，可以运行多个线程任务。Thread 类的构造函数可以传入 Runnable 对象，然后在 start 方法里启动子线程后，会调用传入的 Runnable 对象的 run 方法。

Thread 类在内部通过实例变量target 保存构造函数传入的 Runnable 方法：

```
/* What will be run. */
private Runnable target;
```

同时 Thread 类实现了 Runnable 接口，它的run 方法实现为：

```
@Override
public void run() {
    if (target != null) {
        target.run();
    }
}
```

可以看到，当子线程启动以后，会自动调用 Thread 的 run 方法，run 方法里检查任务对象 target 是否为空，如果不为空则执行 target.run。因此当 target.run 方法结束以后，线程执行结束，这个时候是无法复用线程对象的。

因此，要想复用线程对象，必须要让线程的 run 方法永远不结束，也就是一个 while(true) 循环，在循环里等待新的线程任务到来，大致的逻辑应该是这样的：

```
while(true){
    等待新任务
    执行新任务
}
```

基于以上思路，请实现一个可复用的线程类ReusableThread，类的定义为：

```
public class ReusableThread extends Thread{
    private Runnable runTask = null;  //保存接受的线程任务
  //TODO 加入需要的数据成员


  //只定义不带参数的构造函数
  public ReusableThread(){
      super();
  }

  /**
   * 覆盖 Thread 类的 run 方法
   */
  @Override
  public void run() {
      //这里必须是永远不结束的循环
  }

  /**
   * 提交新的任务
   * @param task 要提交的任务
   */
  public void submit(Runnable task){

  }
}
```

提示：可以使用条件对象的 await/singalAll 机制：在 run 方法里，如果没有线程任务就等待；一旦有线程任务通过 submit 提交，就唤醒线程执行提交的任务。

实现好 ReusableThread 类，可用下面的测试代码进行测试：

```
public static void test3(){
    Runnable task1 = new Runnable()
        { @Override
        public void run() {
            System.out.println("Thread " + Thread.currentThread().getId() +
                    ": is running " + toString());
            try { Thread.sleep(200); }
            catch (InterruptedException e) { e.printStackTrace(); }
        }
        @Override
        public String toString()
            { return "task1";
        }
    };

    Runnable task2 = new Runnable()
        { @Override
        public void run() {
            System.out.println("Thread " + Thread.currentThread().getId() +
                    " is running " + toString());
            try { Thread.sleep(100); }
            catch (InterruptedException e) { e.printStackTrace(); }
        }
        @Override
        public String toString()
            { return "task2";
        }
    };

    ReusableThread t =new ReusableThread();
    t.start(); //主线程启动子线程
  for(int i = 0; i < 5; i++){
        t.submit(task1);
        t.submit(task2);
    }
}
```

上述测试代码的执行结果为:

```
Thread 13: is running task1
Thread 13 is running task2
Thread 13: is running task1
Thread 13 is running task2
Thread 13: is running task1
Thread 13 is running task2
Thread 13: is running task1
Thread 13 is running task2
Thread 13: is running task1
Thread 13 is running task2
```

注意由于ReusableThread 线程对象会等待新的任务，因此上面的程序永远不会结束。通过这个编程题大家就清楚了在 Java 里如何复用一个线程对象。这个原理大家清楚后，以后若去看 JDK 线程池的源代码，就很好理解了。

```
package Excercise3;

import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class ReusableThread extends Thread
    { private Runnable runTask = null;

    //TODO 加入需要的数据成员
    private static Lock lock = new ReentrantLock();
    private static Condition condition1 = lock.newCondition();
    private static Condition condition2 = lock.newCondition();

    //只定义不带参数的构造函数
    public ReusableThread() {
        super();
    }

    /**
     * 覆盖 Thread 类的 run 方法
     */
    @Override
    public void run() {
        //这里必须是永远不结束的循环
        while (true) {
            lock.lock();
            while (runTask == null)
                { try {
                    condition1.await();
                } catch (InterruptedException e)
                    { e.printStackTrace();
                }
            }
            runTask.run();
```

```
            runTask = null;
            condition2.signalAll();
            lock.unlock();
        }
    }

    /**
     * 提交新的任务
     *
     * @param task 要提交的任务
     */
    public void submit(Runnable task) {
        lock.lock();
        if (runTask != null)
            { try {
                condition2.await();
            } catch (InterruptedException e)
                { e.printStackTrace();
            }
        }
        if (task == null) {
        } else {
            runTask = task;
            condition1.signalAll();
        }
        lock.unlock();
    }
}
```


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：

⑦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）

⑦ myBalance = balance; myInterestRate = rate;
② getBalance() = balance; myInterestRate = rate;
③ super(); myInterestRate = rate;
④ super(balance); myInterestRate = rate;

答案：

⑦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）

⑦ super(balance);
② super(); deposit(balance);
③ deposit(balance);

答案：

⑦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）

⑦ 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：

⑦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）

⑦ s.addInterest();

② s.withDraw(500);

③ b.deposit(200);

答案：

⑦ 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，⑪2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



    }
}
```

⑫ 2分，共6分）。

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(123
        456)); 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
    }

    /**
```



```
     */
    @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;
```