附录　部分习题答案

第 1 章

1-4 不能编译

第 2 章

2-1　解析：

第 1 步：通过命令行输入圆球半径；

第 2 步：使用公式 $4/3\pi r^3$，计算圆球体积；

第 3 步：输出结果。

参考代码如下：

```
public class SquareVolume {
    public static void main(String args[]){
        double r = 0 , v = 0 ;                          // 初始化半径、体积
        r = Double.parseDouble(args[0]);                // 命令行输入半径值
        v = 4*3.1415/3*r*r*r ;                          // 计算圆球体积
        System.out.println(" 圆球体积为：" + v);
    }
}
```

2-2 参考代码如下：

```
if (grade == 6 || grade == 7) {
    a = 11 ;
    b = 22 ;
}else if(grade == 5){
    a = 33 ;
    b = 44 ;
}else{
    a = 55 ;
}
```

2-3 解析：要判断一个数是否完数，首先要求这个数的所有因子，接着求所有因子之和，再将所有因子的和与该数比较，如果所有因子之和与该数相等，则它是完数，否则不是完数。

参考代码如下：

```
public class ComputerNumber {
    public static void main(String args[]) {
        int i, j, sum;
        for (i = 1; i <= 100; i++) {
            sum = 0;
            for (j = 1; j < i; j++) {
                if (i%j==0) {
```

```
                    sum = sum + j;
                }
            }
            if (i == sum) {
                System.out.println(i + "");
            }
        }
    }
}
```

第 3 章

3-1 D 3-2 B 3-3 B 3-4 1,3 3-5 99

第 4 章

4-1 略 4-1 略 4-3 A 4-4 B 4-5 C

第 5 章 略

第 6 章

6-1 Iterator 6-2 entrySet 6-3 ensureCapacity 6-4 trimToSize 6-5 values
6-6 D 6-7 B 6-8 C 6-9 A 6-10 A

第 7 章

7-1 解析：Java 使用 I/O 类进行文件操作，通过使用 FileInputStream 和 BufferedReader 等类来实现此目的。使用 BufferedReader 类，有助于提高读取文件数据的性能。所以此处建议使用 BufferedReader 类。参考代码：

```
import java.io.File;
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;
public class ReadTextFileData {
    public ReadTextFileData() {
    }
    public static void main(String[] args) {
        String userid = "";
        String pwd = "";
        try {
            File file = new File("/demo.txt");
            if (file.exists()) {
                String value;
                FileReader filereader = new FileReader(file);
                BufferedReader reader = new BufferedReader(filereader);
                value = reader.readLine();
                if (value != null) {
```

```
                    userid = value;
                }
                value = reader.readLine();
                if (value != null) {
                    pwd = value;
                }
                filereader.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.println("用户名:" + userid);
        System.out.println("密码:" + pwd);
    }
}
```

7-2 解析：首先需要把从键盘输入的数据接收下来，保存为字符串，然后把字符串写到文本文件 demoinput.txt 中。参考代码：

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.BufferedWriter;
import java.io.FileWriter;
public class WriteInputContent {
    public WriteInputContent() {
    }
    public static void main(String[] args) {
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        try {
            FileWriter fw = new FileWriter("/demoinput.txt");
            BufferedWriter out = new BufferedWriter(fw);
            while (true) {
                System.out.print(">");
                String line = in.readLine();
                if (line != null && line.equals("quit")) {
                    break;
                }
                out.write(line);
                out.newLine();
            }
            out.flush();
            out.close();
            fw.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

第 8 章

8-1 C 8-2 A 8-3 D 8-4 B

第 9 章

9-1 B 9-2 A 9-3 C 9-4 D 9-5 B

第 10 章

10-1 A 10-2 A 10-3 C 10-4 A 10-5 B 10-6 D
10-7 B 10-8 B 10-9 C 10-10 A 10-11 B

第 11 章

11-1 C 11-2 A 11-3 B 11-4 B 11-5 C

第 12 章

12-1

（1）解决元素存储的安全性问题。

其主要原理是在类声明时通过一个标识表示类中某个属性的类型或者是某个方法的返回值及参数类型。这样在类声明或实例化时只要指定好需要的具体的类型即可。

（2）解决获取数据元素时，需要类型强制转换的问题。

Java泛型可以保证如果程序在编译时没有发出警告，运行时就不会产生 ClassCastException 异常。同时，代码更加简洁、健壮。

12-2

（1）对象实例化时不指定泛型，默认为 Object。

（2）泛型不同的引用不能相互赋值。

（3）加入集合中的对象类型必须与指定的泛型类型一致。

（4）静态方法中不能使用类的泛型。

（5）如果泛型类是一个接口或抽象类，则不可以创建泛型类的对象。

（6）不能在 catch 中使用泛型。

（7）从泛型类派生子类，泛型类型需具体化。

12-3

（1）values()方法：返回枚举类型的对象数组。该方法可以很方便地遍历所有的枚举值。

（2）valueOf(String str)：可以把一个字符串转为对应的枚举类对象。要求字符串必须是枚举类对象的“名字”。如不是，会出现运行时异常。详细的方法参见 Java API 文档。

第 13 章

13-1

这些类都位于 java.lang.reflect 包中：

（1）Class 类：代表一个类。

（2）Field 类：代表类的成员变量（成员变量也称为类的属性）。

（3）Method 类：代表类的方法。

（4）Constructor 类：代表类的构造方法。

（5）Array 类：提供了动态创建数组，以及访问数组元素的静态方法。

13-2

（1）获得对象的类型：

getName()：获得类的完整名字。

getFields()：获得类的 public 类型的属性。

getDeclaredFields()：获得类的所有属性。

getMethods()：获得类的 public 类型的方法。

getDeclaredMethods()：获得类的所有方法。

getMethod(String name,Class[]parameterTypes)：获得类的特定方法，name 参数指定方法的名字，parameterTypes 参数指定方法的参数类型。

getConstructors()：获得类的所有 public 类型的构造方法。

getConstructor(Class[]parameterTypes)：获得类的特定构造方法，parameterTypes 参数指定构造方法的参数类型。

newInstance()：通过类的不带参数的构造方法创建这个类的一个对象。

（2）通过默认构造方法创建一个新的对象：

Object objectCopy=classType.getConstructor(newClass[]{}).newInstance(newObject[]{});

以上代码先调用 Class 类的 getConstructor()方法获得一个 Constructor 对象，它代表默认的构造方法，然后调用 Constructor 对象的 newInstance()方法构造一个实例。

（3）获得对象的所有属性：

Fieldfields[]=classType.getDeclaredFields();

Class 类的 getDeclaredFields()方法返回类的所有属性，包括 public、protected、默认和 private 访问权限的属性。

（4）获得每个属性相应的 getXXX()和 setXXX()方法，然后执行这些方法，把原来对象的属性复制到新的对象中。