5COSC001W - Tutorial 4 Exercises

1 Interfaces

Define an interface `Printable` which provides a `void print()` method.

Write the code for two classes `Letter`, `Student` which implement interface `Printable`. Class `Letter` contains a `String text` field, while class `Student` has a `String` field corresponding to the name of a student, and an array of five `String` elements corresponding to the grades of the student.

Both `Letter` and `Student` should implement the `print()` method of the `Printable` interface, so that it prints all the relevant details for an object of the corresponding class (i.e. field `text` for `Letter`, and fields `name`, `grades` for `Student`).

2 Members (data and methods) of Interfaces

What is wrong with the following program? Justify why you get the specific compilation errors.

```
interface Equation {
    int numberOfVariables = 1;

    private double solve();

    public void printFormula() {
        System.out.println("x^2 + 3x + 1");
    }
}

class LinearEquation implements Equation {
    public double solve() {
        numberOfVariables = 2;
        // ... details omitted
    }

    public static void main(String[] args) {

    }
}
```

Hint: What kind of *fields* and *methods* can exist in an interface (look up the relevant section of the lecture notes).

3 The `Comparable` Interface

The `Comparable` interface is a commonly used interface in Java. The interface defines how objects of a class which implement the interface are ordered.

Look up the `Comparable` interface in the Java API documentation.

The `compareTo` method of the `Comparable` interface compares two parameters, the implicit and explicit parameter. The call `a.compareTo(b)` returns:

- 1, if a is larger than b
- -1, if a is smaller than b
- 0, if a and b are the same

1. Implement the `compareTo` method of a `BankAccount` class which implements the `Comparable` interface, so that it compares the balances of the accounts. Some of the code is provided to you:

```
public class BankAccount implements Comparable<BankAccount>
{
    private double balance;

    . . .

    /**
        Compares two bank accounts.
        @param other the other BankAccount
        @return 1 if this bank account has a greater balance than the other one,
        -1 if this bank account is has a smaller balance than the other one,
        and 0 if both bank accounts have the same balance
    */
    public int compareTo(BankAccount other)
    {
        . . .
    }
}
```

2. The `sort` method of the `Collections` class can sort a list of objects of classes that implement the `Comparable` interface.

   Here is the outline of the required code for sorting a number of `BankAccount` objects:

```
import java.util.ArrayList;
import java.util.Collections;
. . .

// put bank accounts into a list
ArrayList<BankAccount> list = new ArrayList<BankAccount>();
list.add(ba1);
```

```
list.add(ba2);
list.add(ba3);

// call the library sort method
Collections.sort(list);

// print out the sorted list
for (BankAccount b : list)
   System.out.println(b.getBalance());
```

Using this outline, write a test program that sorts a list of five bank accounts.

4 The `Comparator` Interface

The `java.util.Comparator` interface can also be implemented in order to be able to do sorting according to different criteria.

You can implement the interface in a class by implementing its method `compare(T object1, T object2` to return a negative number, zero and a positive number if the first argument is less than, equal to, or greater than the second respectively.

Re-implement the previous example by using `Comparator` instead of `Comparable`. In this case, the `Collections.sort(list, comparator)` method should be used for sorting.

5 Abstract Classes

Consider the following abstract class:

```
public abstract class Card {
    private String name;

    public Card() {
        name = "";
    }

    public Card(String n) {
        name = n;
    }

    public String getName() {
        return name;
    }

    public abstract boolean isExpired();


    public String format() {
```

```
        return "Card holder: " + name;
    }
}
```

1. Use this class as a superclass to implement a hierarchy of related classes:

   - `CreditCard` with fields `int pinNumber`, `int number`.
   - `DriverLicense` with field `int expirationYear`.
   - `Passport` with fields `String birthLocation`, `int expirationYear`.

   Write definitions for each of the subclasses. For each subclass, the instance variables should be declared private. Leave the bodies of the constructors blank for now.

   Make the assumption that a credit card does not expire.

2. Implement constructors for each of the three subclasses. Each constructor should call the superclass constructor to set the name. Here is one example:

   ```
   public CreditCard(String n, int pin, int num)
   {
       super(n);
       pinNumber = pin;
       number = num;
   }
   ```

   Note that the `super()` call invokes the constructor of the parent class. Thus, `super(n)` in the code above, calls the constructor `Card(String)` of class `Card`, and passes `n` as an argument to it.

6 Overriding Methods

Implement method `format()` in classes `CreditCard`, `DriverLicense`, `Passport` of Exercise 4, so as to redefine (override) the implementation of the parent class `Card`.

The overridden implementations should produce a formatted description of the card details (e.g. for a `Passport` object it should print the `birthLocation`, `name`, `expirationYear` fields). The subclass methods should call the superclass `format()` method to get the formatted name of the cardholder.

7 Polymorphism

1. Guess what will be the output of the following program without running it. Verify your guess by running the program.

   ```
   class A {
       void print() {
           System.out.println("print A!");
       }
   ```

```
}

class B extends A {
    void print() {
        System.out.println("print B!");
    }
}

class C extends A {
    void print() {
        System.out.println("print C!");
    }
}

class D extends B {
    void print() {
        System.out.println("print D!");
    }
}

public class PTester {
    public static void main(String[] args) {
        A a1 = new A();
        a1.print();

        a1 = new B();
        a1.print();

        a1 = new C();
        a1.print();

        a1 = new D();
        a1.print();

        B b1 = new D();
        b1.print();

        D d1 = (D) b1;  // cast line 1

        A a2 = new A();
        d1 = (D) a2;  // cast line 2
    }
}
```

2. Why is the cast required, in the line accompanied by the comment "cast line 1"?

3. What is wrong in the line accompanied by the comment "cast line 2"?

8 Access Specifiers

Guess (without attempting to compile) which of the lines of the following program will not compile. Verify your guess (after justifying it), by compiling the program and make sure you understand what is happening in each individual line.

The program consists of three separate files, `P.java`, `Program1.java`, `Program2.java`.

File `P.java` contains:

```
package myfirst;

public class P {
    public double x;
    protected double y;
    private double z;
    double w;

    public void foo() {
        x = 2.1;  // line 11
        y = 3.1;  // line 12
        z = 4.1;  // line 13
        w = 5.1;  // line 14
    }
}
```

File `Program1.java` contains:

```
package myfirst;

class Q extends P {
    public void bar() {
        x = 2.2;  // line 20
        y = 3.2;  // line 21
        z = 4.2;  // line 22
        w = 5.2;  // line 23

        P p1 = new P();
        p1.x = 2.2;  // line 26
        p1.y = 3.2;  // line 27
        p1.z = 4.2;  // line 28
        p1.w = 5.2;  // line 29
    }
}

class R extends Q {
    public void bar2() {
        x = 2.3;  // line 35
        y = 3.3;  // line 36
        z = 4.3;  // line 37
```

```
        w = 5.3;  // line 38

        P p2 = new P();
        p2.x = 2.3;  // line 41
        p2.y = 3.3;  // line 42
        p2.z = 4.3;  // line 43
        p2.w = 5.3;  // line 44
    }
}

public class Program1 {
    public static void main(String[] args) {
        P p3 = new P();
        p3.x = 2.4;  // line 51
        p3.y = 3.4;  // line 52
        p3.z = 4.4;  // line 53
        p3.w = 5.4;  // line 54
    }
}
```

File `Program2.java` contains:

```
package mysecond;

import myfirst.P;

class S extends P {
    public void bar() {
        x = 2.5;  // line 8
        y = 3.5;  // line 9
        z = 4.5;  // line 10
        w = 5.5;  // line 11

        P p4 = new P();
        p4.x = 2.5;  // line 14
        p4.y = 3.5;  // line 15
        p4.z = 4.5;  // line 16
        p4.w = 5.5;  // line 17
    }
}

public class Program2 {
    public static void main(String[] args) {
        P p5 = new P();
        p5.x = 2.6;  // line 24
        p5.y = 3.6;  // line 25
        p5.z = 4.6;  // line 26
        p5.w = 5.6;  // line 27
    }
}
```

9 Challenge: A Program for Appointments

Implement a superclass `Appointment` and subclasses `Onetime`, `Daily`, and `Monthly`. An appointment has a description (for example, "see the dentist") and a date.

Implement methods `occursOn(int year, int month, int day)` for each of the subclasses that checks whether the appointment occurs on that date or not and return true or false respectively. For example, for a monthly appointment, you must check whether the day of the month matches.

Create a tester class that creates various objects from all the implemented classes and call their `occursOn` method to see if you have implemented the classes to do the right calculations.

Give the user the option to add new appointments. The user must specify the type of the appointment, the description, and the date. You do NOT need to save these in a file, once the program terminates all appointments are lost.

Add another method `dayOfTheWeek` in class `Appointment` which returns a string corresponding to the day of the week that the appointment occurs, e.g if the appointment occurs on 17 October 2015, the method returns "Sunday". (*Hint:* use the `GregorianCalendar` library class).

Implement a method `displayAll()` which displays all the appointments sorted by date in ascending order, i.e. appointments occurring at an earlier date should be displayed first.