CARDIFF SCHOOL OF COMPUTER SCIENCE AND INFORMATICS
Coursework Assessment Pro-forma

Module Code: CM2307
Module Title: Object Orientation, Algorithms and Data Structures
Lecturer: Dr Nervo Verdezoto
Assessment Title: Object-Oriented Modelling and Programming Assignment
Assessment Number: 2
Date Set: Friday 26th February 2021 (Week 4)
Submission Date & Time: Friday 30th April 2021 at 9:30am
Return Date: Monday 28th May 2021

This coursework is worth 50% of the total marks available for this module. If coursework is submitted late (and where there are no extenuating circumstances):

1. If the assessment is submitted no later than 24 hours after the deadline, the mark for the assessment will be capped at the minimum pass mark;
2. If the assessment is submitted more than 24 hours after the deadline, a mark of 0 will be given for the assessment.

Your submission must include the official Coursework Submission Cover sheet, which can be found here: https://docs.cs.cf.ac.uk/downloads/coursework/Coversheet.pdf

INSTRUCTIONS

Follow the instructions in the attached coursework specification.

SUBMISSION INSTRUCTIONS

Your coursework should be submitted through Learning Central. Your submission should include:

Description		Type	Name
Cover sheet	**Compulsory**	PDF (.pdf) file	[student number].pdf
ONE PDF file (and no more than one) which contains the diagrams, explanations, discussion, program listings (extracts from the code which show clearly what you did in order to complete the TASKS), and evidence that each of these programs "works", in particular for TASKS 4 and 5 that includes a card game and a die-rolling game	**Compulsory**	PDF (.pdf) file	CM2307-[student number].pdf
ONE ZIP archive file (and no more than one) containing all the source code files for your answers to TASKS 1 to 7	**Compulsory**	ZIP (.zip) archive file	CM2307Source-[student number].zip

Any code submitted will be run on a University machine and must be submitted as stipulated in the instructions above.

Any deviation from the submission instructions above (including the number and types of files submitted) will result in a reduction of 20% of the mark.

Staff reserve the right to invite students to a meeting to discuss coursework submissions

ASSIGNMENT

See the description on Page 3 for details.

LEARNING OUTCOMES ASSESSED

This assignment particularly addresses the following module learning outcomes:

- Apply principles of good OO software design to the creation of robust, elegant, maintainable code
- Explain and utilise a range of design patterns and synchronization mechanisms
- Design and implement fully OO Java programs
- Model systems using UML class diagrams

CRITERIA FOR ASSESSMENT

Credit will be awarded against the following specific criteria. Ability to:

- Apply principles of good object-oriented design in order to create a critique of a supplied program (check the source code files provided),
- Describe the purpose of a program and analyse this description for particular classes.
- Use UML class diagrams as a means of documentation,
- Apply design patterns and synchronization mechanisms that pertain to a specified programming task,
- Refactor a program in order to improve its elegance, and
- Reflect on the refactoring process.

Feedback on your performance will address each of the above criteria. The total mark is a weighted sum of marks for the different tasks.

The following will help you to interpret the marks awarded for each task:
0-39%: The submitted answer only addresses the tasks to a very limited extent.
40-49%: Some attempt has been made to address the tasks, and the code mostly works.
50-59%: The tasks have been completed, the code “works”, but minimal insight has been shown.
60-69%: The tasks have been completed, the code “works”, and some clear insight has been shown.
70-100%: An excellent implementation, showing a good degree of insight and reflection, and the code is of good quality.

FURTHER DETAILS

Feedback on your coursework will address the above criteria and will be returned in approximately **4 weeks from submission deadline (28th May 2021)** via Learning Central. This will be supplemented with oral feedback via Revision Lecture in Week 13 (to be arranged).

Object-Oriented Modelling and Programming Assignment

Note that no improvement to the user interface presented by the program is required! This exercise is worth 50% of the total marks available for CM2307. For each of the following tasks, a list of the files is included in the present coursework specification at the end for your convenience.

TASK 1. This is worth 15% of the total marks available for the present coursework.

Download the zip archive file from Learning Central that contains the source code files for this task: **Pet.java, Hamster.java, Canary.java**.

(i) Draw a UML class diagram illustrating these three classes, their attributes, their methods and their relationships to each other

(ii) You will notice that the **setName()** and **getName()** methods in Canary do the same as the corresponding methods in **Hamster**. Modify the supplied program to improve and restructure (refactoring) the code and explain the steps and decisions taken, so that your client code (create a Client.java) could include lines such as:

```
Pet p=new Canary();
p.setName("Annabel");
System.out.println(p.getName());
```

(iii) Define an *interface* Vegetarian () and add a food () method to it that returns a String so that your revised program should be capable of being tested using the following lines of code (create a Client1.java):

```
Hamster h=new Hamster();
Pet p = h;
Vegetarian v = h;
h.setName("Cookie");
System.out.println(p.getName() + " eats " + v.food());
```

Which generates the following output on the standard output stream: "*Cookie eats beans*". Carefully explain the polymorphic behaviour of the Hamster class.

TASK 2. This is worth 5% of the total marks available for the present coursework

Download the **ExampleTest.java** file from Learning Central that contains the code for this task. Consider the following UML class diagram, describing a particular kind of *singleton*, where **accessCount()** returns the number of times the singleton has been accessed via the **getInstance()** method:

ExampleSingleton
-accessCount: int -singletonInstance: ExampleSingleton
-ExampleSingleton() +getInstance(): ExampleSingleton +accessCount(): int

(i) State the purpose of the *Singleton* design pattern

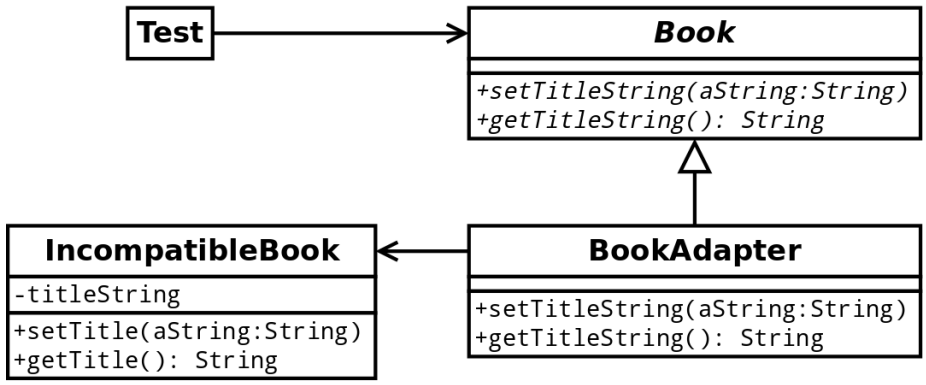
(ii) Write an implementation of **ExampleSingleton.java** such that, when the **main()** method in **ExampleTest.java** is executed, it will generate the following output on the standard output stream:

```
I, the ExampleSingleton, am being created
```

```
The sole instance of ExampleSingleton is being retrieved
The ExampleSingleton has been accessed via the getInstance() method 1 time(s)
The sole instance of ExampleSingleton is being retrieved
The ExampleSingleton has been accessed via the getInstance() method 2 time(s)
```

TASK 3. This is worth 10% of the total marks available for the present coursework

Download the ZIP archive file from Learning Central that contains the code for this task (Book.java, and Test.java). Consider the following UML class diagram, describing a particular kind of *adapter*:



(i) State the purpose of the *Adapter* design pattern

(ii) Write an implementation of the class **IncompatibleBook**. The **getTitle**() and **setTitle**() methods should be written as *accessors* which get and set the **titleString** instance variable, without any side-effects.

(iii) Write an implementation of the class **BookAdapter**. When the **BookAdapter** is created, it should create the **IncompatibleBook** to which it refers. The **getTitleString**() and **setTitleString**() methods should call the corresponding accessors in the **IncompatibleBook** in order to get and set the book's title string

Note: You are not required to make the code thread-safe for this task.

TASK 4. This is worth 15% of the total marks available for the present coursework.

Download the ZIP archive file from Learning Central that contains the source code files for this task: **Game.java**, **LinearCongruentialGenerator.java, RandomInterface.java** and **IncompatibleRandomInterface.java**.

In principle, this program will not work, as **Game.java** is expecting a **LinearCongruentialGenerator** defined differently from the way it is defined in the ZIP archive file.

i) "Fix" this problem by altering **LinearCongruentialGenerator** to implement **RandomInterface**.

ii) Draw a UML class diagram representing the program which results from the above modifications. You can select any of the tools we have reviewed in class for creating UML Diagrams. **Note:** *if you use yUML then you will not be able to underline static (class) methods*

or attributes. In that case, indicate in an explanatory note which ones you would have underlined if you could!

iii) Write a description of the purpose of this program, bearing in mind that you are going to use this description to help you with identifying suitable classes for an improved version of this program in TASK 5. [*Guide:* between 150 and 300 words]

TASK 5. This is worth 35% of the total marks available for the present coursework.

The program considered in TASK 4 (see above) is deliberately inelegant (e.g., different bad code smells). The purpose of TASK 5 is to design and implement an improved version which:

- Shows evidence of the application of good software design strategies to reduce coupling and maximize cohesion
- Makes use of the description created in TASK 4 to help you identifying suitable classes
- Separates out the two game implementations. For this, you should apply some kind of design pattern, in order to achieve this[1].
- Includes brief comments explaining the code and decisions

In summary, you should:

i) Identify suitable classes for an improved version of the program, taking into account the points listed above, and explain your design choices.
[*Guide:* you may well choose to present these results in tabular form, with just a very small number of words explaining the role of each class, plus a small number of additional sentences explaining your overall choice]

ii) Draw a UML class diagram representing the improved program you are going to implement.

iii) Implement the improved program!

TASK 6. This is worth 5% of the total marks available for the present coursework.

Download the **TestThread.java** file from Learning Central that contains the source code for this task including the **TestThread** class and the **Example** class.

(i) After running the code of **TestThread.java** file several times, provide two reasons for why this code is not thread-safe.

(ii) Modify the supplied code and implement one or more strategies to make this code thread-safe

(iii) Carefully explain why your code is now thread-safe

TASK 7. This is worth 15% of the total marks available for the present coursework.

Download the ZIP file from Learning Central that contains a typical solution for the dinning philosophers problem: **DiningPhilosophers.java** and the **Philosopher.java**

(i) Explain the use of the *synchronized* keyword in the Philosopher class on Fork objects

(ii) After running the code for several times, you will notice that a *deadlock* is taking place. List and describe in your own words the conditions that must be satisfied for a deadlock to occur and which of these are causing a deadlock in the code

(iii) Modify the code and explain how your minor changes help avoid a deadlock situation

[1]Remember to look up http://www.oodesign.com/ for relevant information.

Program listing for Task 1

```
public class Pet {
        public String classOfAnimal() {
                return("Pet");
        }
}
public class Hamster extends Pet {
        protected String name;
        public void setName(String aName) { name=aName; }
        public String getName() { return name; }
        public String classOfAnimal() { return("Hamster"); }
}
public class Canary extends Pet {
        protected String name;
        public void setName(String aName) { name=aName; }
        public String getName() { return name; }
        public String classOfAnimal() { return("Canary"); }
}
```

Program listing for Task 2

```
public class ExampleTest {
        public static void main(String[] args) {
                ExampleSingleton s = ExampleSingleton.getInstance();
                System.out.println("The ExampleSingleton has been "
                + "accessed via the getInstance() method "
                + s.accessCount()
                + " time(s)");
                s = ExampleSingleton.getInstance();
                System.out.println("The ExampleSingleton has been "
                + "accessed via the getInstance() method "
                + s.accessCount()
                + " time(s)");
        }
}
```

Program listing for Task 3

```
public abstract class Book {
        public abstract void setTitleString(String aString);
         public abstract String getTitleString();
}

public class Test {
          public static void main(String[] theArguments) {
                  Book b=new BookAdapter();
                 b.setTitleString("Gone with the Wind");
                System.out.println("We have a book with title: "
                      + b.getTitleString());
  }
}
```

Program listing for Task 4 and Task 5

```
import java.io.*;
import java.util.*;

public class Game {
  // The BufferedReader used throughout
  public static BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

  // The random number generator used throughout
  public static RandomInterface r=new LinearCongruentialGenerator();

  // Variable(s) used in the card game methods
  public static ArrayList<String> cardList;
  public static HashSet<String> cardsChosen=new HashSet<String>();

  public static void playCardGame() throws Exception {
     // Play card game:

     // Initialise the game
     initialiseCardGame();

     // Play the main game phase
     mainCardGame();

     // Now see if (s)he has won!
     declareCardGameWinner();
  }

  public static void initialiseCardGame() throws Exception {
     // The initialisation phase:

     // Create a list of cards ... and shuffle them
     String cards[]={"AHrts", "2Hrts", "3Hrts", "4Hrts", "5Hrts", "6Hrts",
                     "7Hrts", "8Hrts", "9Hrts", "10Hrts", "JHrts",
                     "QHrts", "KHrts",
                     "ADmnds", "2Dmnds", "3Dmnds", "4Dmnds", "5Dmnds",
                     "6Dmnds", "7Dmnds", "8Dmnds", "9Dmnds", "10Dmnds",
                     "JDmnds", "QDmnds", "KDmnds",
                     "ASpds", "2Spds", "3Spds", "4Spds", "5Spds", "6Spds",
                     "7Spds", "8Spds", "9Spds", "10Spds", "JSpds",
                     "QSpds", "KSpds",
                     "AClbs", "2Clbs", "3Clbs", "4Clbs", "5Clbs", "6Clbs",
                     "7Clbs", "8Clbs", "9Clbs", "10Clbs", "JClbs",
                     "QClbs", "KClbs"};
     cardList=new ArrayList<String> (Arrays.asList(cards));
     // Taking advantage of "generics" to tell the compiler all the elements will be Strings

     // Shuffle them
     for (int i=0; i<100; i++) {
       // choose two random cards at random and swap them, 100 times
       int firstIndex=((int) (r.next() * 52));
       int secondIndex=((int) (r.next() * 52));
       String temp=(String) cardList.get(firstIndex);
       cardList.set(firstIndex, cardList.get(secondIndex));
       cardList.set(secondIndex, temp);
     }

     // Print out the result
     System.out.println(cardList);
  }
```

```
  public static void mainCardGame() throws Exception {
     // The main game:

     // Let user select two cards from the pack
     for (int i=0; i<2; i++) {
       System.out.println("Hit <RETURN> to choose a card");
       br.readLine();

       int cardChoice=(int) (r.next() * cardList.size());
       System.out.println("You chose " + cardList.get(cardChoice));
       cardsChosen.add(cardList.remove(cardChoice));
     }

     // Display the cards chosen and remaining cards
     System.out.println("Cards chosen: " + cardsChosen);
     System.out.println("Remaining cards: " + cardList);

  }

  public static void declareCardGameWinner() throws Exception {
     // Declare the winner:

     // User wins if one of them is an Ace
     System.out.println("Cards chosen: " + cardsChosen);
     if (cardsChosen.contains("AHrts") || cardsChosen.contains("ADmnds") ||
        cardsChosen.contains("ASpds") || cardsChosen.contains("AClbs")) {
       System.out.println("You won!");
     }
     else System.out.println("You lost!");
  }

//Variable(s) used in the die game methods
  public static HashSet<Integer> numbersRolled=new HashSet<Integer>();

  public static void playDieGame() throws Exception {
     // Play die game:

     // Initialise the game
     initialiseDieGame();

     // Play the main game phase
     mainDieGame();

     // Now see if (s)he has won!
     declareDieGameWinner();
  }

  public static void initialiseDieGame() throws Exception {
     // The initialisation phase:

     // Actually there isn't anything to do here
  }
```

```
public static void mainDieGame() throws Exception {
      // The main game:

      // Let the user roll the die twice
      for (int i=0; i<2; i++) {
        System.out.println("Hit <RETURN> to roll the die");
        br.readLine();
        int dieRoll=(int)(r.next() * 6) + 1;

        System.out.println("You rolled " + dieRoll);
        numbersRolled.add(new Integer(dieRoll));
      }

      // Display the numbers rolled
      System.out.println("Numbers rolled: " + numbersRolled);
  }

  public static void declareDieGameWinner() throws Exception {
      // Declare the winner:

      // User wins if at least one of the die rolls is a 1
      if (numbersRolled.contains(new Integer(1))) {
        System.out.println("You won!");
      }
      else System.out.println("You lost!");
  }

  public static void main(String[] args) throws Exception {
    // Ask whether to play a card game or a die game

    System.out.print("Card (c) or Die (d) game? ");
    String ans=br.readLine();

    if (ans.equals("c")) {
      playCardGame();
    }

    else if (ans.equals("d")) {
      playDieGame();
    }

    else System.out.println("Input not understood");
  }
}

public interface RandomInterface {
// Simply defines a method for retrieving the next random number
  public double next();
}

public interface IncompatibleRandomInterface {
// Simply defines a method for retrieving the next random number
// This version won't work with the Game class defined
  public double getNextNumber();
}


public class LinearCongruentialGenerator implements IncompatibleRandomInterface {
// Generates pseudo-random numbers using:
// X(n+1) = (aX(n) + c) (mod m)
// for suitable a, c and m. The numbers are "normalised" to the range
// [0, 1) by computing X(n+1) / m.

  private long a, c, m, seed;
// Need to be long in order to hold typical values ...
```

```
  public LinearCongruentialGenerator(long a_value, long c_value, long m_value, long s_value) {
    a=a_value; c=c_value; m=m_value; seed=s_value;
  }

  public LinearCongruentialGenerator(long seed) {
  // Set a, c and m to values suggested in Press, Teukolsky, et al., "Numberical Recipies"

   this(1664525, 1013904223, 4294967296l, seed);
  // NB "l" on the end is the way that a long integer can be specified. The
  // smaller ones are type-cast silently to longs, but the large number is too
  // big to fit into an ordinary int, so needs to be defined explicitly
  }

  public LinearCongruentialGenerator() {
  // (Re-)set seed to an arbitrary value, having first constructed the object using
  // zero as the seed. The point is that we don't know what m is until after it has
  // been initialised.

  this(0);  seed=System.currentTimeMillis() % m;

  }

  public static void main(String args[]) {
  // Just a little bit of test code, to illustrate use of this class.
    IncompatibleRandomInterface r=new LinearCongruentialGenerator();
    for (int i=0; i<10; i++) System.out.println(r.getNextNumber());

  // Since RandomInterface doesn't know about the instance variables defined in this
  // particular implementation, LinearCongruentialGenerator, we need to type-cast
  // in order to print out the parameters (primarily for "debugging" purposes).

    LinearCongruentialGenerator temp=(LinearCongruentialGenerator) r;
    System.out.println("a: " + temp.a + "  c: " + temp.c + "  m: " + temp.m + "  seed: " + temp.seed);

  }

  public double getNextNumber() {
    seed = (a * seed + c) % m;
    return (double) seed/m;
  }
}
```

Program listing for Task 6

```
public class TestThread extends Thread {
private int theValue;
public TestThread(int aValue) { theValue=aValue; }

public void run() {
            Example e = Example.getInstance();
            e.setVal(theValue);
            }
public static void main(String[] args) throws java.lang.InterruptedException {
            TestThread[] tests=new TestThread[10000];
            for (int i=0; i<10000; i++) { tests[i]=new TestThread(i); }
            for (int i=0; i<10000; i++) { tests[i].start(); }
            for (int i=0; i<10000; i++) { tests[i].join(); }
            Example e = Example.getInstance();
            System.out.println("The Example has value " + e.getVal() + " and has been updated " +
                                    e.getUpdateCount() + " time(s)");
            }
}
class Example {
```

```
private static Example myInstance;
private int updateCount=0;
private int val=0;

private Example() { }
public static Example getInstance() {
            if (myInstance == null) {myInstance = new Example();}
            return myInstance;
}
public void setVal(int aVal) { val=aVal; updateCount++; }
public int getVal() { return val; }
public int getUpdateCount() { return updateCount; }
}
```

Program listing for Task 7

```
public class Philosopher implements Runnable {

// The forks on either side of this Philosopher
    private Object leftFork;
    private Object rightFork;

    public Philosopher(Object leftFork, Object rightFork) {
        this.leftFork = leftFork;
        this.rightFork = rightFork;
    }

            private void doAction(String action) throws InterruptedException {
        System.out.println(Thread.currentThread().getName() + " " + action);
        Thread.sleep(((int) (Math.random() * 100)));
    }

    public void run() {
        try {
            while (true) {

                // thinking
                doAction(System.nanoTime() + ": Thinking");
                synchronized (leftFork) {
                    doAction(System.nanoTime() + ": Picked up left fork");
                    synchronized (rightFork) {
                        // eating
                        doAction(System.nanoTime() + ": Picked up right fork - eating");
                        doAction(System.nanoTime() + ": Put down right fork");
                    }
                    // Back to thinking
                    doAction(System.nanoTime() + ": Put down left fork. Back to thinking");
                }
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            return;
        }
    }
}


public class DiningPhilosophers {
    public static void main(String[] args) throws Exception
        Philosopher[] philosophers = new Philosopher[5];
        Object[] forks = new Object[philosophers.length];

        for (int i = 0; i < forks.length; i++) {
            forks[i] = new Object();
        }

        for (int i = 0; i < philosophers.length; i++) {
            Object leftFork = forks[i];
```

```
            Object rightFork = forks[(i + 1) % forks.length];

            philosophers[i] = new Philosopher(leftFork, rightFork);

            Thread t = new Thread(philosophers[i], "Philosopher " + (i + 1));
            t.start();
        }
    }
}
```