SCHOOL OF COMPUTER SCIENCE
COURSEWORK ASSESSMENT PROFORMA

MODULE & LECTURER: CM2201, Andrew Jones

DATE SET: Friday 27th October 2017 (Week 4)

SUBMISSION DATE AND TIME: 09:30, Friday 24th November 2017 (Week 8)

SUBMISSION ARRANGEMENTS:
Upload the files specified in the submission instructions to Learning Central by 09:30, Friday 24th November, 2017.

TITLE: Exercise 1
This coursework is worth 30% of the total marks available for this module. The penalty for late or non-submission is an award of zero marks. **You are reminded of the need to comply with Cardiff University's Student Guide to Academic Integrity.** Your work should be submitted using the official Coursework Submission Cover Sheet.

INSTRUCTIONS
Follow the instructions in the attached coursework specification.

SUBMISSION INSTRUCTIONS

You should upload the following to Learning Central. Note that any deviation from this will result in a mark of zero for the question(s) affected.

Description		Type	Name
Cover sheet	**Compulsory**	PDF (.pdf) file	[student number].pdf
ONE PDF file (and no more than one) which contains the diagrams, discussion, program listings (extracts from the code which show clearly what you did in order to answer Question 1; full listing for Question 2), and evidence that each of these programs "works" for a card game and a die-rolling game	**Compulsory**	PDF (.pdf) file	CM2201_[student number].pdf
ONE ZIP archive (and no more than one) of all the source code of your answers to Questions 1 and 2	**Compulsory**	ZIP (.zip) archive	CM2201Source_[student number].zip

CRITERIA FOR ASSESSMENT
This exercise 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
- Design and implement fully OO Java programs
- Model systems using UML class diagrams

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,
- describe the purpose of a program and analyse this description for candidate classes.
- use UML class diagrams as a means of documentation,
- apply design patterns 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 following will help you to interpret the marks awarded for each question:
0-39%: The submitted answer only addresses the question to a very limited extent.
40-49%: Some attempt has been made to address the question, and the code mostly works.
50-59%: The question has been answered, the code "works", but minimal insight has been shown.
60-69%: The question has been answered, the code "works", and some clear insight has been shown.
70-100%: An excellent answer, showing a good degree of insight and reflection, and the code is of good quality.

FURTHER DETAILS

Use the laboratory classes to get guidance and assistance in doing this exercise.

You are encouraged to contact me by e-mail if you have any queries; it may well be possible for me to answer them almost immediately. If necessary you are welcome to arrange to come and see me individually about the coursework. There will also be the opportunity to ask me about the coursework when we have lectures.

Feedback on your coursework will address the above criteria and will be returned in approximately: **4 weeks from submission deadline (22nd December 2017).**
This will be supplemented with oral feedback via… Revision Lecture in Week 12 (to be arranged).

Object-Oriented Applications
Assessed Exercise 1

Note that no improvement to the user interface presented by the program is required! This exercise is worth 30% of the total marks available for CM2201, including those for the examination.

Question 1

(This is worth 30% of the total marks available for the present exercise.)

Download the ZIP archive containing the files **Game.java**, **LinearCongruentialGenerator.java, RandomInterface.java** and **IncompatibleRandomInterface.java** from Learning Central. A listing of these files is included in the present coursework specification for your convenience.

This program will not work, as **Game.java** is expecting a **LinearCongruentialGenerator** defined differently from the way it is defined in the download files.

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.
(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 Question 2. [*Guide:* between 150 and 300 words]

Question 2

(This is worth 70% of the total marks available for the present exercise.)

The program considered in Question 1 above is deliberately inelegant. The purpose of Question 2 is to design and implement an improved version which:

- Shows evidence of your having applied principles relating to concepts such as coupling and cohesion
- Makes use of the description created in answer to Question 1 to help your identification of suitable classes
- Separates out the two game implementations. (You should apply some kind of Factory pattern, or a related pattern, in order to achieve this.[1])
- Includes brief comments explaining the code

i) Identify suitable classes for an improved version of the program, taking into account the points listed above, and explain your choice. [*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!

Andrew Jones. 27th October 2017

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

Program listing of Game.java

```
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");
  }
}
```

Program listing of RandomInterface.java

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

Program listing of IncompatibleRandomInterface.java

```
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();
}
```

Program listing of LinearCongruentialGenerator.java

```
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;
  }
}
```