CM2307

Object Oriented modelling and programming

C1816377 Harry Batchelor

1.

i. To fix the problems discussed in question 1 I altered some lines in LinearCongruentialGgenerator.java

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

I changed the code from IncompatibleRandomInterface to RandomInterface

```
1  public class LinearCongruentialGenerator implements RandomInterface {
2  // Generates pseudo-random numbers using:
```

I changed IncompatibleRandomInterface to Randominterface and r.getNextNumber() to r.next()

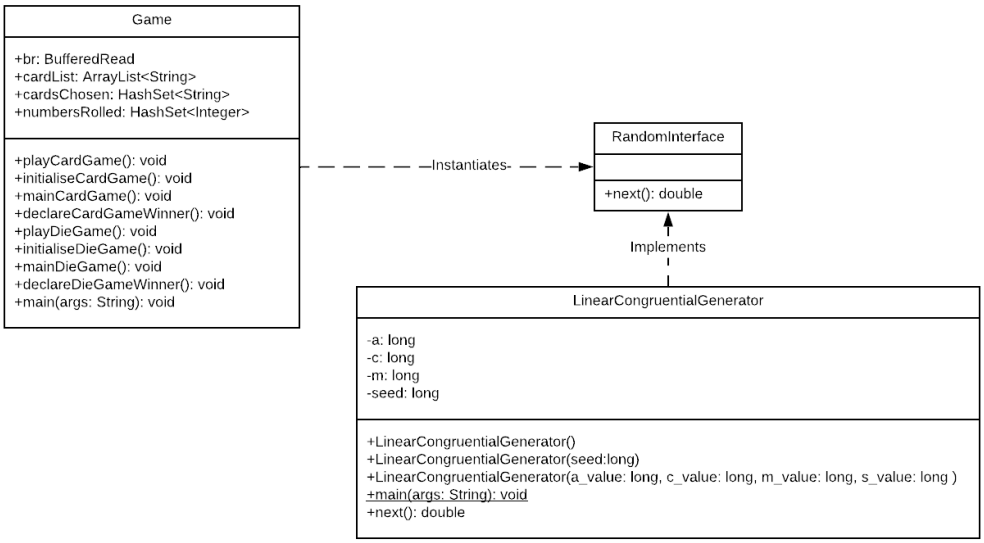
```
  public double next() {
    seed = (a * seed + c) % m;
    return (double) seed/m;
  }
}
```

Changed the name of the operation from getNextNumber() to next()

I also fixed the Game.java file by removing the ";//" which was commenting out the LinearCongruentialGenerator() so it wasn't being implemented correctly.

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

ii. UML diagram for Question 1 after modifying the code for part i



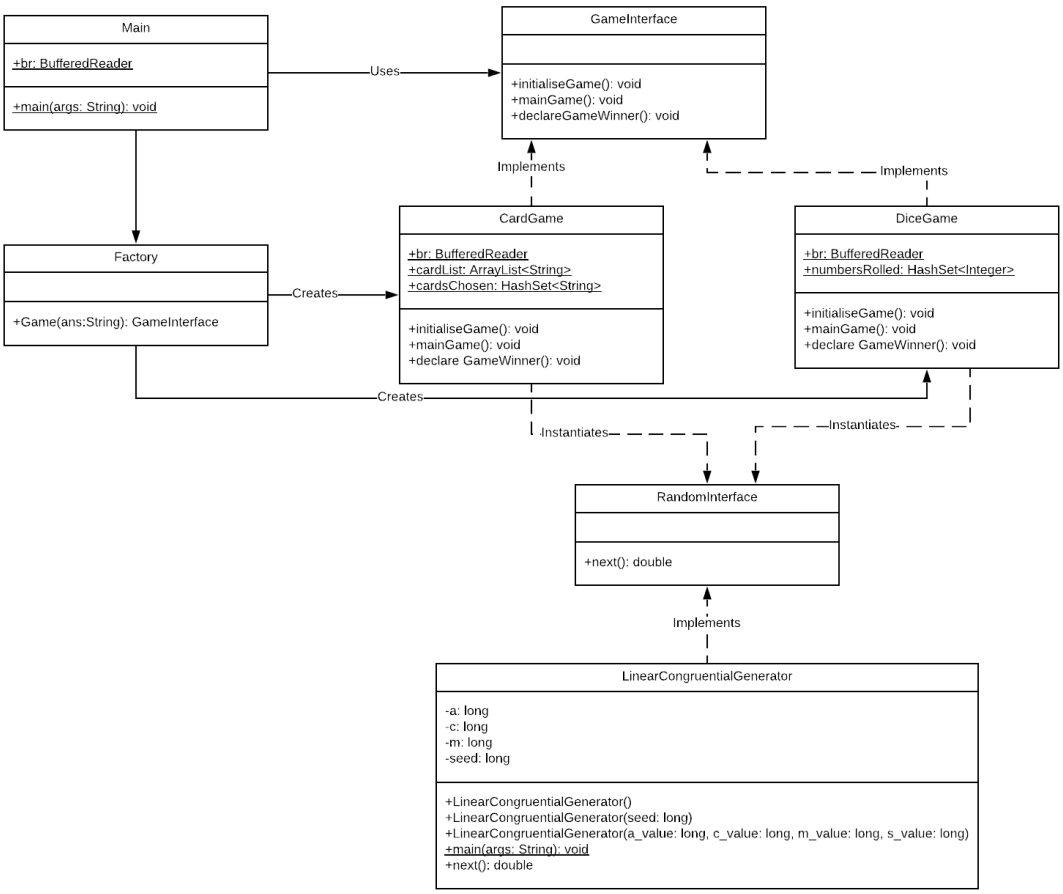

iii. The program contains an interface and 2 classes. The interface, “RandomInterface” is implemented by the class “LinearCongruentialGenerator” which picks a random number normalised from 0 to 1. The value that is given out from the “LinearCongruentialGenerator” is instantiated by the class “Game”; this is then used on the methods that are used to play the different games. The program current contains low cohesion because there are a lot of methods which are all doing lots of different things which all group into just 2 classes. Every class should be responsible for doing one task. So especially for the “Game” class which has 9 methods that should have been split into different classes in order to maximise cohesion. To further optimise the program this class should have been split into 3 more different classes, “Main”, “DiceGame”, “Cardgame” focusing the different methods in relation to the role of the class.
Then again, the classes could be considered loosely coupled because the “Game” class interacts with the “LinearCongruentialGenerator” class, but it also depends on the “RandomInterface” interface. This does make it relatively good for the programme since its maintenance of the code is easier, but there is also from for improvement. They are also not strongly connected as they don’t rely on the internal representation of the other.

2.

i. The instantiation of the classes is restricted by the structural design of the programme. The design follows a Singleton pattern. Several operations and attributes are contained inside the class "Game". These can all be split into 3 different classes. The classes being "Main", "DiceGame" and "CardGame". Each of these will contain their required attributes and methods. As stated in Question 1(iii) the classes have low cohesion by having several non-related attributes and methods packed into one class. To improve the programme, we will need to first change up the design pattern. We can first increase the cohesion by dividing the methods and attributes that only relate to the purpose of the class, this is because they will now only focus on only doing one thing. The next thing we need to do to improve the structure of our programme is to loosen the coupling. This can be done by adding in an interface which implements the methods within the "DiceGame" and

"CardGame" classes. This interface needs to be used directly from the "Main" class. All of this combined makes the loosens the dependency amongst those classes even more in relation to question 1. For all this redesign to work we need to apply a Factory pattern design rather than the original singleton pattern. This will allow us to create an instance of the Game being played and will also inherit the methods used from the connected interface.

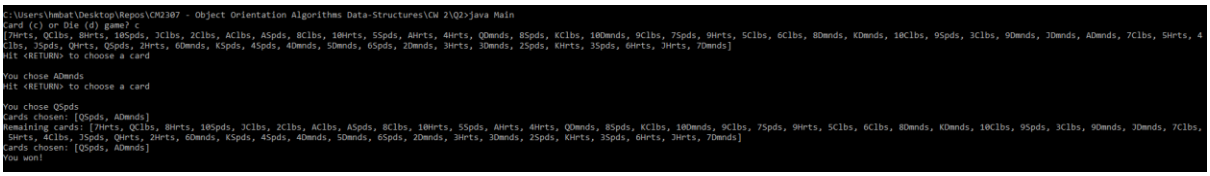
ii. UML Diagram for Question 2 showing the improvements that I will make for part (iii)



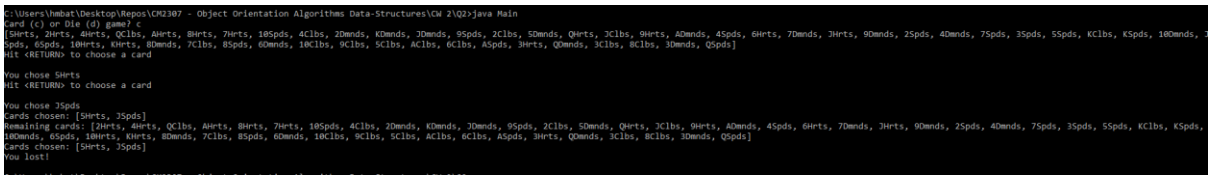

iii. My improvements to the programme include implementing: different interfaces, the UML diagram showing the classes I used and also the factory pattern design. Where the program is initialised, I used a "Main" class which calls the "Factory" class. In the "Factory" class the input that the user chose calls a different object to decide which game is being created. The program will use "GameInterface". This interface holds the operations that need to be created for each game class to tell the program which operations will be used. While the Dice or Card game is being played instances of the "RandomInterface" will be run using the "next()" operation. This operation implements the "LinearCongruentialGenerator" class which creates a random number in normalised form from 0 to 1.

## Card Game

Winning game




Losing game




## Dice Game

Winning game

```
C:\Users\hmbat\Desktop\Repos\CM2307 - Object Orientation Algorithms Data-Structures\CW 2\Q2>java Main
Card (c) or Die (d) game? d
Hit <RETURN> to roll the die

You rolled 5
Hit <RETURN> to roll the die

You rolled 1
Numbers rolled: [1, 5]
You won!
```

Losing game

```
C:\Users\hmbat\Desktop\Repos\CM2307 - Object Orientation Algorithms Data-Structures\CW 2\Q2>java Main
Card (c) or Die (d) game? d
Hit <RETURN> to roll the die

You rolled 6
Hit <RETURN> to roll the die

You rolled 5
Numbers rolled: [5, 6]
You lost!
```

## Wrong inputs

```
C:\Users\hmbat\Desktop\Repos\CM2307 - Object Orientation Algorithms Data-Structures\CW 2\Q2>java Main
Card (c) or Die (d) game? CLS
Input not understood
```

```
C:\Users\hmbat\Desktop\Repos\CM2307 - Object Orientation Algorithms Data-Structures\CW 2\Q2>java Main
Card (c) or Die (d) game? Java
Input not understood
```

```
C:\Users\hmbat\Desktop\Repos\CM2307 - Object Orientation Algorithms Data-Structures\CW 2\Q2>java Main
Card (c) or Die (d) game? 12
Input not understood
```

```
C:\Users\hmbat\Desktop\Repos\CM2307 - Object Orientation Algorithms Data-Structures\CW 2\Q2>java Main
Card (c) or Die (d) game? 978
Input not understood
```

## Full Code for Q2

**Program listing of CardGame.java**

```
import java.io.*;
import java.util.*;
public class CardGame implements GameInterface {
 // 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 void initialiseGame() {
     // 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 void mainGame() {
    // 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");

        try {br.readLine();}
        catch (IOException e){System.out.println(e);}

        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 void declareGameWinner(){
    // 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!");
   }
}
```

**Program listing of DiceGame.java**

```
import java.io.*;
import java.util.*;
public class diceGame implements GameInterface {
 // 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 dice game methods
 public static HashSet<Integer> numbersRolled=new HashSet<Integer>();
  public void initialiseGame() {
     // The initialisation phase:
     // Actually there isn't anything to do here
  }
  public void mainGame() {
     // The main game:
     // Let the user roll the dice twice
     for (int i=0; i<2; i++) {
        System.out.println("Hit <RETURN> to roll the dice");

        try {br.readLine();}
        catch (IOException e){System.out.println(e);}

        int diceRoll=(int)(r.next() * 6) + 1;
```

```
            System.out.println("You rolled " + diceRoll);
            numbersRolled.add(new Integer(diceRoll));
        }
        // Display the numbers rolled
        System.out.println("Numbers rolled: " + numbersRolled);
    }
    public void declareGameWinner() {
        // Declare the winner:
        // User wins if at least one of the dice rolls is a 1
        if (numbersRolled.contains(new Integer(1))) {
            System.out.println("You won!");
        }
        else System.out.println("You lost!");
    }
}
```

**Program listing of Factory.java**

```
public class Factory {
    public GameInterface Game(String ans) {
        if (ans.equals("c")) {
            return new CardGame();
        } else if (ans.equals("d")) {
            return new DiceGame();
        } else {
            System.out.println("Input not understood");
            System.exit(1);
            return null;
        }
    }
}
```

**Program listing of GameInterface.java**

```
public interface GameInterface {
  void initialiseGame() throws Exception;
  void mainGame() throws Exception;
  void declareGameWinner() throws Exception;
}
```

**Program listing of LinearCongruentialGenerator.java**

```
public class LinearCongruentialGenerator implements RandomInterface {
// 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.
   RandomInterface r=new LinearCongruentialGenerator();
   for (int i=0; i<10; i++) System.out.println(r.next());

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

**Program listing of Main.java**

```
import java.io.*;
public class Main {

   public static BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

   public static void main(String[] args) throws Exception {
      Factory Factory = new Factory();

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

      GameInterface Game = Factory.Game(ans);
```

```
        Game.initialiseGame();
        Game.mainGame();
        Game.declareGameWinner();
    }
}
```

**Program listing of RandomInterface.java**

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