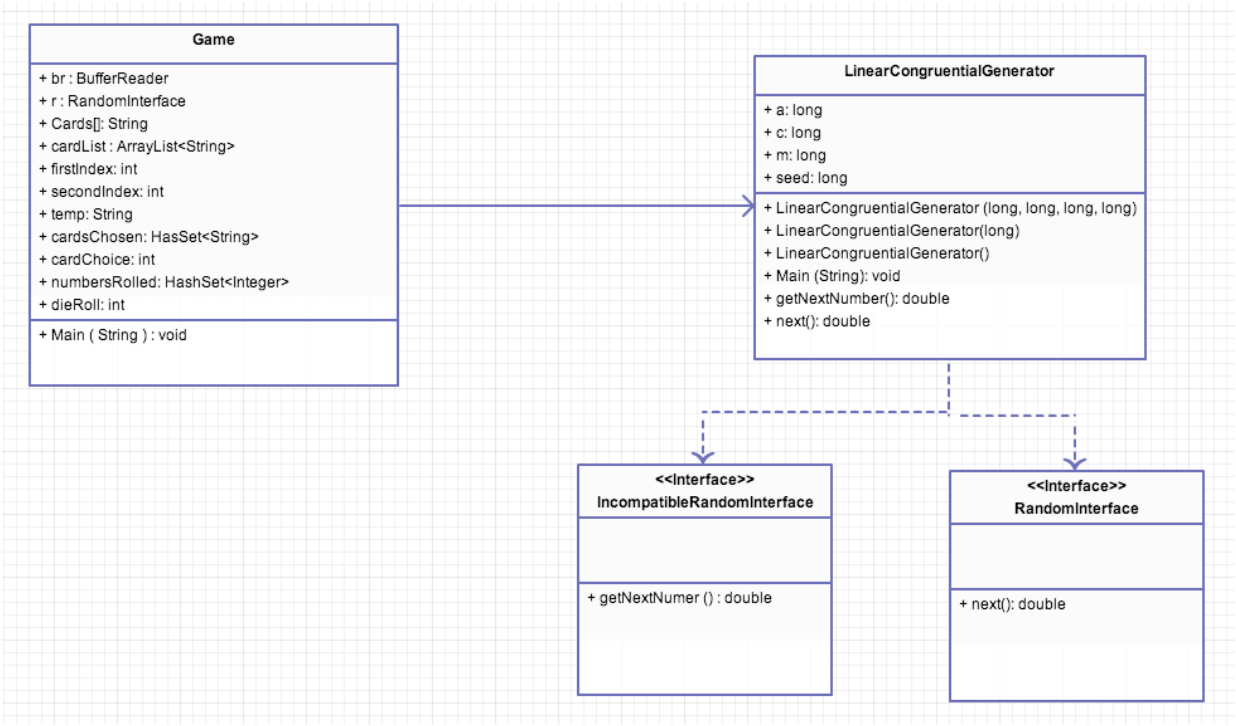
OOA Coursework

Question 1:

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

Code in appendix and attached in ZIP file.

ii) *DrawaUMLclassdiagramrepresentingtheprogramwhichresultsfromtheabove modifications.*



iii) This program is still deliberately inelegant. Write a critique of the program.

Both Card and Dice game are dealt with in one class (Game). This is inelegant as it breaches the Single Responsibility Principle "A class should have one, and only one, reason to change." (butunclebob)

No Factory Pattern. The Game is doing too much, it needs to know how to create the objects only.

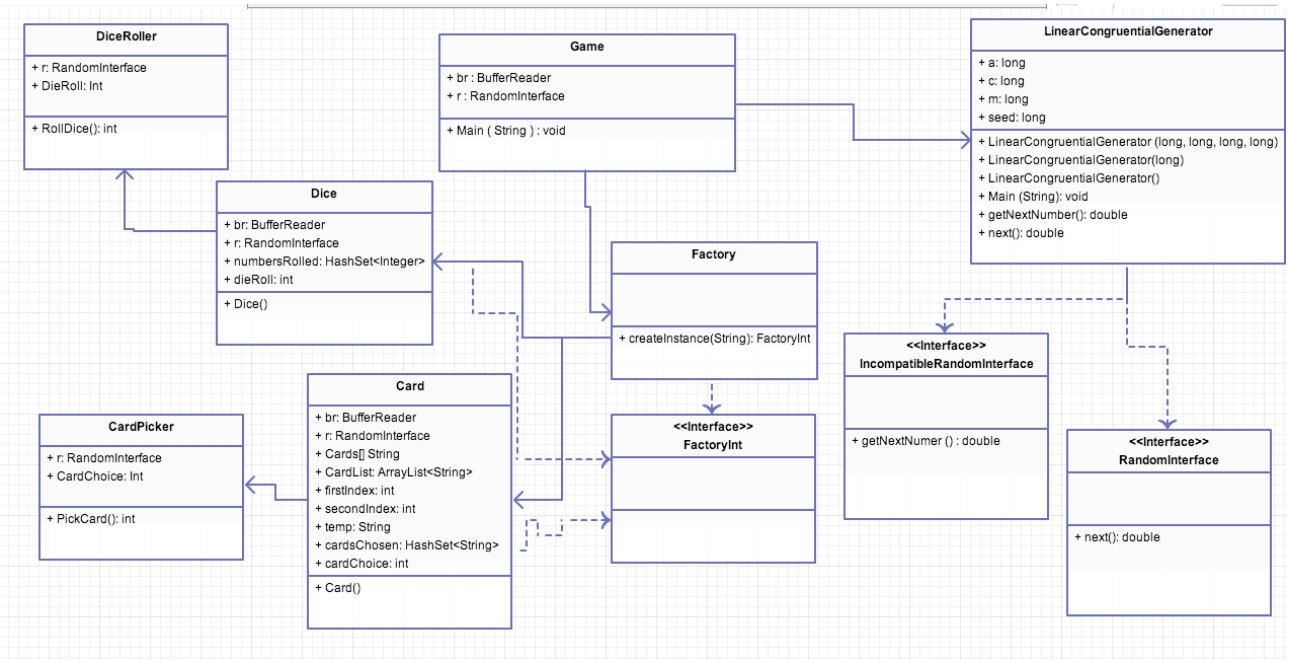
Main Throws exceptions and therefore means you will not know if any exceptions are happening and therefore won't be able to ratify them.

Question 2:

i) *Modify the program resulting from Question 1 above in order to improve its elegance.*

Code in appendix and attached in ZIP file.

ii) Draw a UML class diagram representing the program which results from the above modifications.



iii) Briefly explain your choice of pattern for separating out the two game implementations. Very briefly state the reasons for the other changes which you have made in answering this question.

I chose the procedural factory pattern as it is the most used design pattern in modern programming. It creates objects without exposing the logic to the client and refers to the newly created object through a common interface.

I divided the Dice and Card implementation into separate classes, this is consistent with the Single Responsibility Principle (REF). I added error handling through try and catch loops, so an exception is thrown. The dice rolling and card picking are in separate classes, again consistent with the Single Responsibility Principle. I have rebuilt it so there is no now loose coupling and high cohesion within the program e.g. the dice class doesn't rely on the internal workings of the DiceRolled class, just the output.

References:

ArticleS.UncleBob.PrinciplesOfOod. 2013.ArticleS.UncleBob.PrinciplesOfOod. [ONLINE] Available at:http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod. [Accessed 19 April 2013].

Appendix:

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

public class Game {
```

```
public static void main(String[] args) throws Exception {
  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

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

  // 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")) {
    // Play card game
    // Create a list of cards
    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"};
    ArrayList<String> 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);

    // The game: let user select two cards from the pack;
    // User wins if one of them is an Ace.

    HashSet<String> cardsChosen=new HashSet<String>();

    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);

    // Now see if (s)he has won!
    if (cardsChosen.contains("AHrts") || cardsChosen.contains("ADmnds") ||
        cardsChosen.contains("ASpds") || cardsChosen.contains("AClbs")) {
      System.out.println("You won!");
    }
    else System.out.println("You lost!");
  }

  else if (ans.equals("d")) {
    // Play die game. User wins if at least one of the die rolls is a 1
    HashSet<Integer> numbersRolled=new HashSet<Integer>();

    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);

    // Now see if (s)he has won!
    if (numbersRolled.contains(new Integer(1))) {
      System.out.println("You won!");
    }
    else System.out.println("You lost!");
  }
  else System.out.println("Input not understood");


 }
}
```

Appendix 1: *Game.java - For part 1*

```
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();
}
```

Appendix 2: *IncompatibleRandomInterface.java - For part 1*

```
public class LinearCongruentialGenerator implements IncompatibleRandomInterface,
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.
    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;
  }

  public double next(){
  return getNextNumber();
  }
}
```

Appendix 3: LinearCongruentialGenerator.java - For part 1

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

Appendix 4: RandomInterface.java - For part 1

```
public class LinearCongruentialGenerator implements IncompatibleRandomInterface,
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.
```

```
    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;
  }

  public double next() {
    return getNextNumber();
  }
}
```

Appendix 5: LinearCongruentialGenerator.java - For Part 2

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

Appendix 6: RandomInterface.java - For Part 2

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

public class Game {

  public static void main(String[] args) throws Exception {
          start();
    }
    public static void start(){
      BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

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

        // Ask whether to play a card game or a die game

       System.out.print("Card (c) or Die (d) game? ");

       String ans = "";
```

```
      try{
      ans=br.readLine();
     }catch(IOException e){System.out.println(e);}
       Factory.createInstance(ans);

   }
}
```

Appendix 7: Game.java - For part 2

```
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();
}
```

Appendix 8: IncompatibleRandomInterface.java - For part 2

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

public class Card implements FactoryInt{

        public  Card(){

        RandomInterface r=new LinearCongruentialGenerator();
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

                // Play card game
                // Create a list of cards
        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"};
        ArrayList<String> 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);

          // The game: let user select two cards from the pack;
          // User wins if one of them is an Ace.

          HashSet<String> cardsChosen=new HashSet<String>();

          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 = CardPicker.PickCard();

           // 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);

          // Now see if (s)he has won!
          if (cardsChosen.contains("AHrts") || cardsChosen.contains("ADmnds") ||
                  cardsChosen.contains("ASpds") || cardsChosen.contains("AClbs")) {
            System.out.println("You won!");
          }
          else System.out.println("You lost!");
}
}
```

Appendix 9: Card.java - For part 2

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

public class Dice implements FactoryInt{

          public Dice(){

          RandomInterface r=new LinearCongruentialGenerator();
          BufferedReader br=new BufferedReader(new InputStreamReader(System.in));


          // Play die game. User wins if at least one of the die rolls is a 1
```

```
        HashSet<Integer> numbersRolled=new HashSet<Integer>();

        for (int i=0; i<2; i++) {
                System.out.println("Hit <RETURN> to roll the die");
                try{
                br.readLine();
        }catch(IOException e){System.out.println(e);}

                int dieRoll = DiceRoller.RollDice();

                //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);

        // Now see if (s)he has won!
        if (numbersRolled.contains(new Integer(1))) {
                System.out.println("You won!");
        }else {System.out.println("You lost!");}
        }
}
```

Appendix 10: Dice.java - For part 2

```
public class Factory implements FactoryInt{

        // Creates an object dependent on input
        public static FactoryInt createInstance(String id){
                if(id.equals("c")){
                        return new Card();
                }else if(id.equals("d")){
                        return new Dice();
                }else{
                        System.out.println("Input not understood");
                        return null;
                }
        }
}
```

Appendix 11: Factory.java - For part 2

```
//Required to sustain factory
interface FactoryInt{

}
```

Appendix 12: FactoryInt.java - For part 2

```
public class DiceRoller{
```

```
	public static int RollDice(){
		// Simply rolls dice by geberating random number
		RandomInterface r=new LinearCongruentialGenerator();
		int dieRoll=(int)(r.next() * 6) + 1;

		return dieRoll;

	}

}
```

Appendix 13: DiceRoller.java - For part 2

```
public class CardPicker{

	public static int PickCard(){
		// Simply picks card by generating random number
		RandomInterface r=new LinearCongruentialGenerator();
		int cardChoice=(int) (r.next() * cardList.size());

		return cardChoice;

	}

}
```

Appendix 14: CardPicker.java - For part 2