**Object Oriented Applications Coursework**

Q1 i) LinearCongruentialGenerator.java

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);

}

// NEW CODE

public double next() {

return this.getNextNumber();

}

public double getNextNumber() {

seed = (a \* seed + c) % m;

return (double) seed/m;

}

// END NEW CODE

}

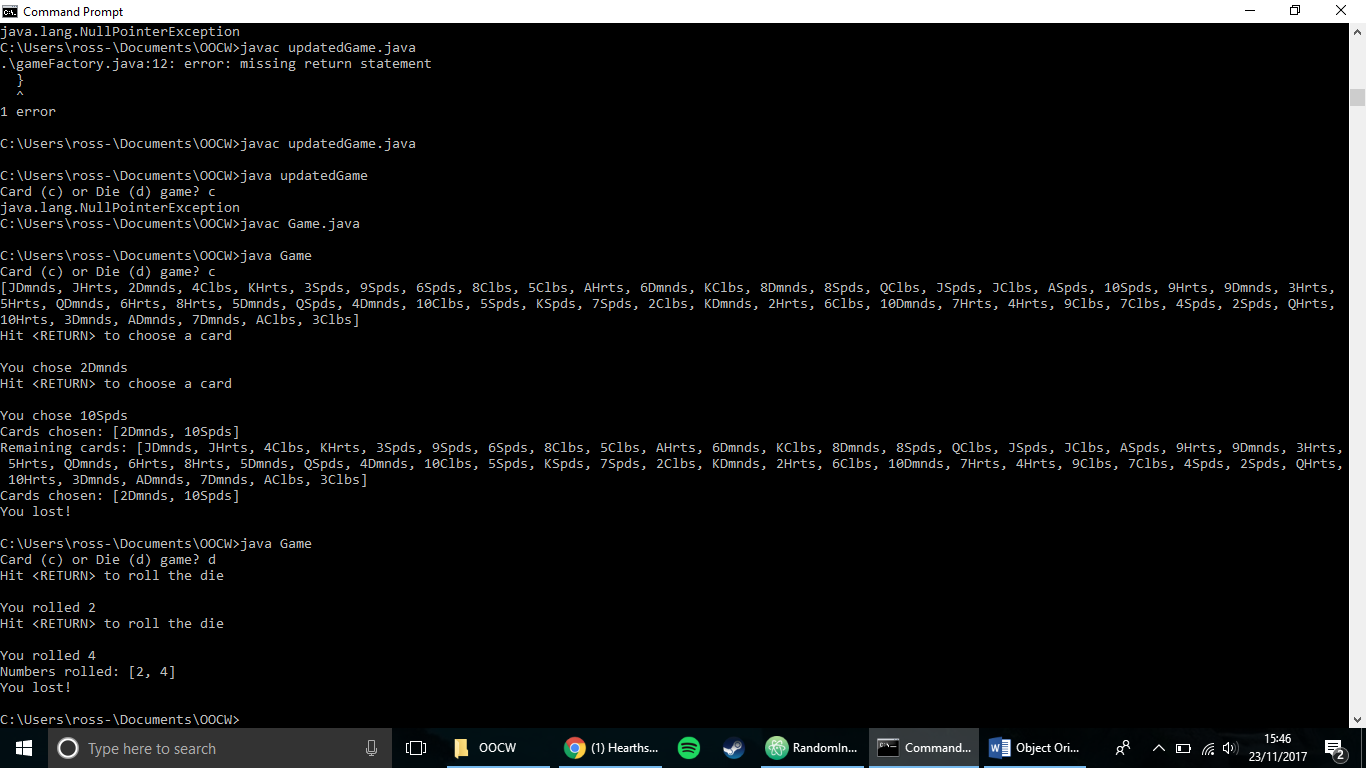
RandomInterface.java

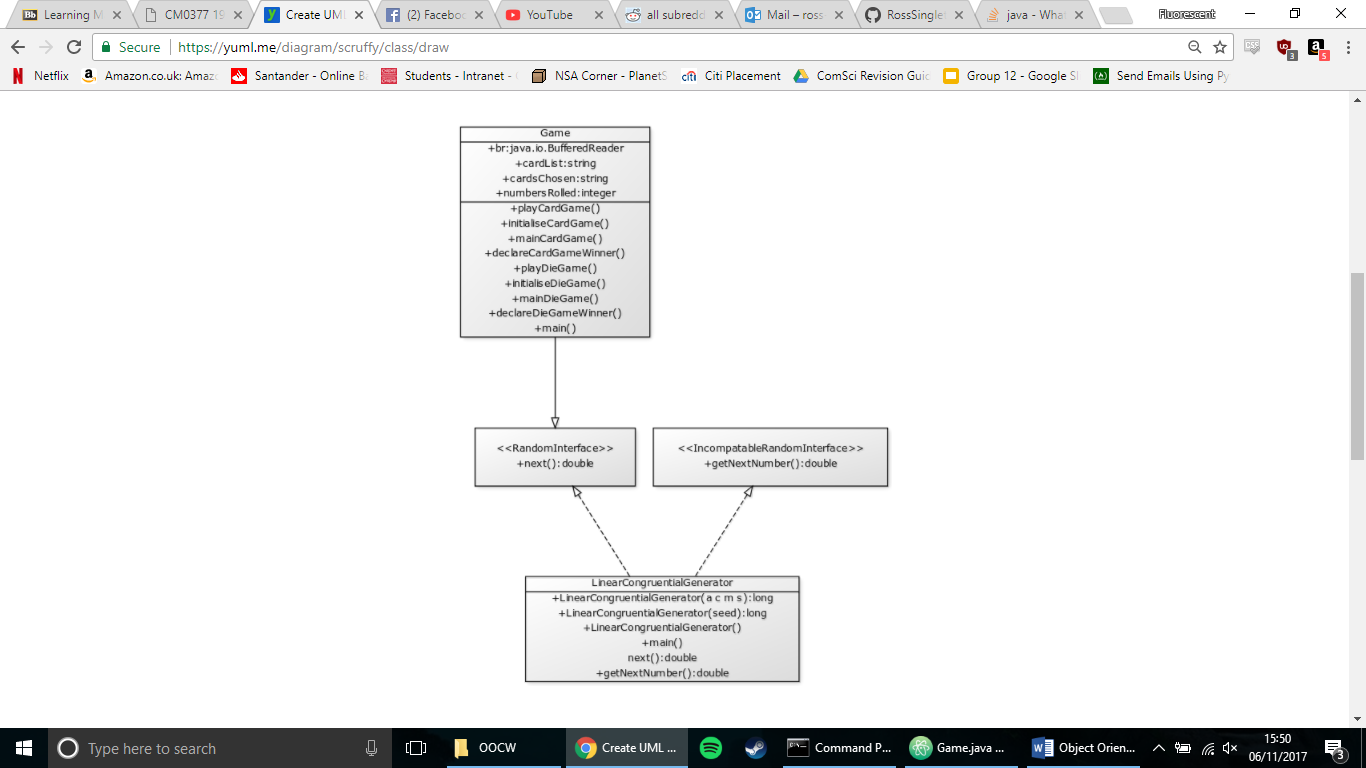
public interface RandomInterface {

// Simply defines a method for retrieving the next random number

public double next();

}



Q1 ii)

Q1 iii) The purpose of the program is to simulate either a card picking or a dice rolling game for the user. The aim of the card game being to pick any of the 4 aces in a standard deck of cards, giving you two chances to do so. The aim of the dice game is to roll the dice and get a number 1, which again you have two rolls of the dice to achieve.

Initially, the user is presented with an option “Card (c) or die (d) game?” with the response “c” or “d” deciding which game is executed. Both of the games make use of interfaces through LinearCongruentialInterface and RandomInterface to choose the random numbers that define which card you have picked and what number was on the die.

Q2 i)

|  |  |  |
| --- | --- | --- |
| **Class Name** | **Class Role** | **Explanation/Justification** |
| updatedGame.java | Class that holds the main menu. | User will run this class, and will decide which game they want to play. |
| gameFactory.java | Factory for creating new games. | updatedGame will use the factory to create new instances of games, depending on the answer given by the user. |
| gameInterface.java | Interface used by Factory. | Defines the methods that dieGame and cardGame are going to override. |
| cardGame.java | Specific methods for the card game. | This class contains the specific methods for the card game, and overrides the methods set out in gameInterface. |
| dieGame.java | Specific methods for die game. | This class contains the specific methods for the die game, and overrides the methods set out in gameInterface. |