Object-Oriented Applications Assessed Exercise 1
C1620874
"Q1" source code for Q1
"Q2" source code for Q2

Question 1

i) Changes that were made to LinearCongruentialGenerator.java have been highlighted in the four following screenshots which is then followed by one screenshot of the program working.

LinearCongruentialGenerator.java

```
1  public class LinearCongruentialGenerator implements RandomInterface {
2  // Generates pseudo-random numbers using:
3  // X(n+1) = (aX(n) + c) (mod m)
4  // for suitable a, c and m. The numbers are "normalised" to the range
5  // [0, 1) by computing X(n+1) / m.
6
7    private long a, c, m, seed;
8  // Need to be long in order to hold typical values ...
9
10   public LinearCongruentialGenerator(long a_value, long c_value, long m_value, long s_value) {
11     a=a_value; c=c_value; m=m_value; seed=s_value;
12   }
13
14   public LinearCongruentialGenerator(long seed) {
15   // Set a, c and m to values suggested in Press, Teukolsky, et al., "Numberical Recipies"
16     this(1664525, 1013904223, 4294967296l, seed);
17   // NB "l" on the end is the way that a long integer can be specified. The
18   // smaller ones are type-cast silently to longs, but the large number is too
19   // big to fit into an ordinary int, so needs to be defined explicitly
20   }
21
22   public LinearCongruentialGenerator() {
23   // (Re-)set seed to an arbitrary value, having first constructed the object using
24   // zero as the seed. The point is that we don't know what m is until after it has
25   // been initialised.
26
27   this(0);  seed=System.currentTimeMillis() % m;
28
29   }
30
31   public static void main(String args[]) {
32   // Just a little bit of test code, to illustrate use of this class.
33     RandomInterface r=new LinearCongruentialGenerator();
34     for (int i=0; i<10; i++) System.out.println(r.next());
35
36   // Since RandomInterface doesn't know about the instance variables defined in this
37   // particular implementation, LinearCongruentialGenerator, we need to type-cast
38   // in order to print out the parameters (primarily for "debugging" purposes).
39
40     LinearCongruentialGenerator temp=(LinearCongruentialGenerator) r;
41     System.out.println("a: " + temp.a + "  c: " + temp.c + "  m: " + temp.m + "  seed: " + temp.seed);
42
43   }
44
45   public double next() {
46     seed = (a * seed + c) % m;
47     return (double) seed/m;
48   }
49 }
50
```

Object-Oriented Applications Assessed Exercise 1

C1620874

“Q1” source code for Q1

“Q2” source code for Q2

LinearCongruentialGenerator.java

```
1 public class LinearCongruentialGenerator implements RandomInterface {
2 // Generates pseudo-random numbers using:
3 // X(n+1) = (aX(n) + c) (mod m)
4 // for suitable a, c and m. The numbers are "normalised" to the range
5 // [0, 1) by computing X(n+1) / m.
6
7   private long a, c, m, seed;
8 // Need to be long in order to hold typical values ...
9
10  public LinearCongruentialGenerator(long a_value, long c_value, long m_value, long s_value) {
11    a=a_value; c=c_value; m=m_value; seed=s_value;
12  }
13
14  public LinearCongruentialGenerator(long seed) {
15  // Set a, c and m to values suggested in Press, Teukolsky, et al., "Numberical Recipies"
16    this(1664525, 1013904223, 4294967296l, seed);
17  // NB "l" on the end is the way that a long integer can be specified. The
18  // smaller ones are type-cast silently to longs, but the large number is too
19  // big to fit into an ordinary int, so needs to be defined explicitly
20  }
21
22  public LinearCongruentialGenerator() {
23  // (Re-)set seed to an arbitrary value, having first constructed the object using
24  // zero as the seed. The point is that we don't know what m is until after it has
25  // been initialised.
26
27  this(0);  seed=System.currentTimeMillis() % m;
28
29  }
30
31  public static void main(String args[]) {
32  // Just a little bit of test code, to illustrate use of this class.
33    RandomInterface r=new LinearCongruentialGenerator();
34    for (int i=0; i<10; i++) System.out.println(r.next());
35
36  // Since RandomInterface doesn't know about the instance variables defined in this
37  // particular implementation, LinearCongruentialGenerator, we need to type-cast
38  // in order to print out the parameters (primarily for "debugging" purposes).
39
40    LinearCongruentialGenerator temp=(LinearCongruentialGenerator) r;
41    System.out.println("a: " + temp.a + "  c: " + temp.c + "  m: " + temp.m + "  seed: " + temp.seed);
42
43  }
44
45  public double next() {
46    seed = (a * seed + c) % m;
47    return (double) seed/m;
48  }
49 }
50
```

Object-Oriented Applications Assessed Exercise 1

C1620874

“Q1” source code for Q1

“Q2” source code for Q2

LinearCongruentialGenerator.java

```
1  public class LinearCongruentialGenerator implements RandomInterface {
2  // Generates pseudo-random numbers using:
3  // X(n+1) = (aX(n) + c) (mod m)
4  // for suitable a, c and m. The numbers are "normalised" to the range
5  // [0, 1) by computing X(n+1) / m.
6
7    private long a, c, m, seed;
8  // Need to be long in order to hold typical values ...
9
10   public LinearCongruentialGenerator(long a_value, long c_value, long m_value, long s_value) {
11     a=a_value; c=c_value; m=m_value; seed=s_value;
12   }
13
14   public LinearCongruentialGenerator(long seed) {
15   // Set a, c and m to values suggested in Press, Teukolsky, et al., "Numberical Recipies"
16     this(1664525, 1013904223, 4294967296l, seed);
17   // NB "l" on the end is the way that a long integer can be specified. The
18   // smaller ones are type-cast silently to longs, but the large number is too
19   // big to fit into an ordinary int, so needs to be defined explicitly
20   }
21
22   public LinearCongruentialGenerator() {
23   // (Re-)set seed to an arbitrary value, having first constructed the object using
24   // zero as the seed. The point is that we don't know what m is until after it has
25   // been initialised.
26
27   this(0);  seed=System.currentTimeMillis() % m;
28
29   }
30
31   public static void main(String args[]) {
32   // Just a little bit of test code, to illustrate use of this class.
33     RandomInterface r=new LinearCongruentialGenerator();
34     for (int i=0; i<10; i++) System.out.println(r.next());
35
36   // Since RandomInterface doesn't know about the instance variables defined in this
37   // particular implementation, LinearCongruentialGenerator, we need to type-cast
38   // in order to print out the parameters (primarily for "debugging" purposes).
39
40     LinearCongruentialGenerator temp=(LinearCongruentialGenerator) r;
41     System.out.println("a: " + temp.a + "  c: " + temp.c + "  m: " + temp.m + "  seed: " + temp.seed);
42
43   }
44
45   public double next() {
46     seed = (a * seed + c) % m;
47     return (double) seed/m;
48   }
49 }
50
```

Object-Oriented Applications Assessed Exercise 1

C1620874

“Q1” source code for Q1

“Q2” source code for Q2

LinearCongruentialGenerator.java

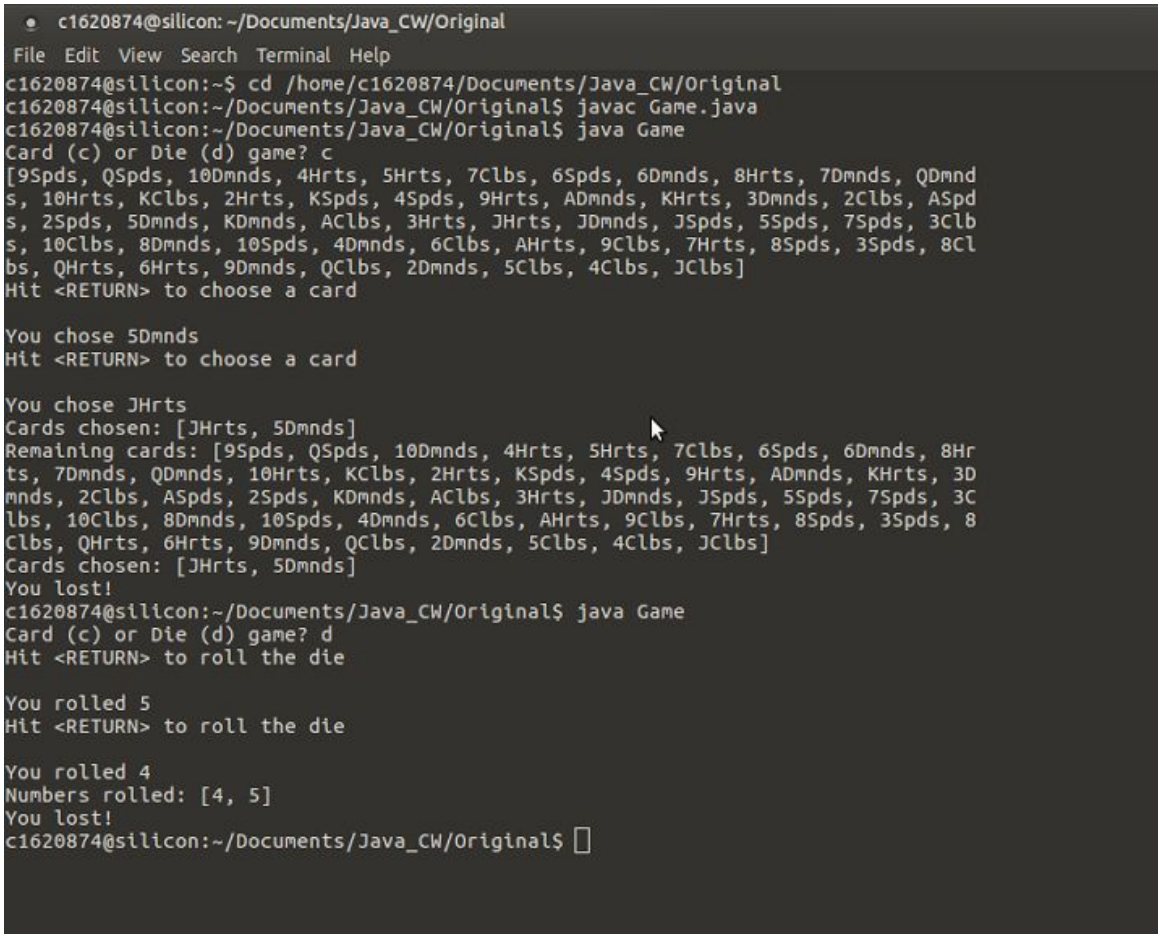
```
public class LinearCongruentialGenerator implements RandomInterface {
// Generates pseudo-random numbers using:
// X(n+1) = (aX(n) + c) (mod m)
// for suitable a, c and m. The numbers are "normalised" to the range
// [0, 1) by computing X(n+1) / m.

  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;
  }
}
```

Object-Oriented Applications Assessed Exercise 1

C1620874

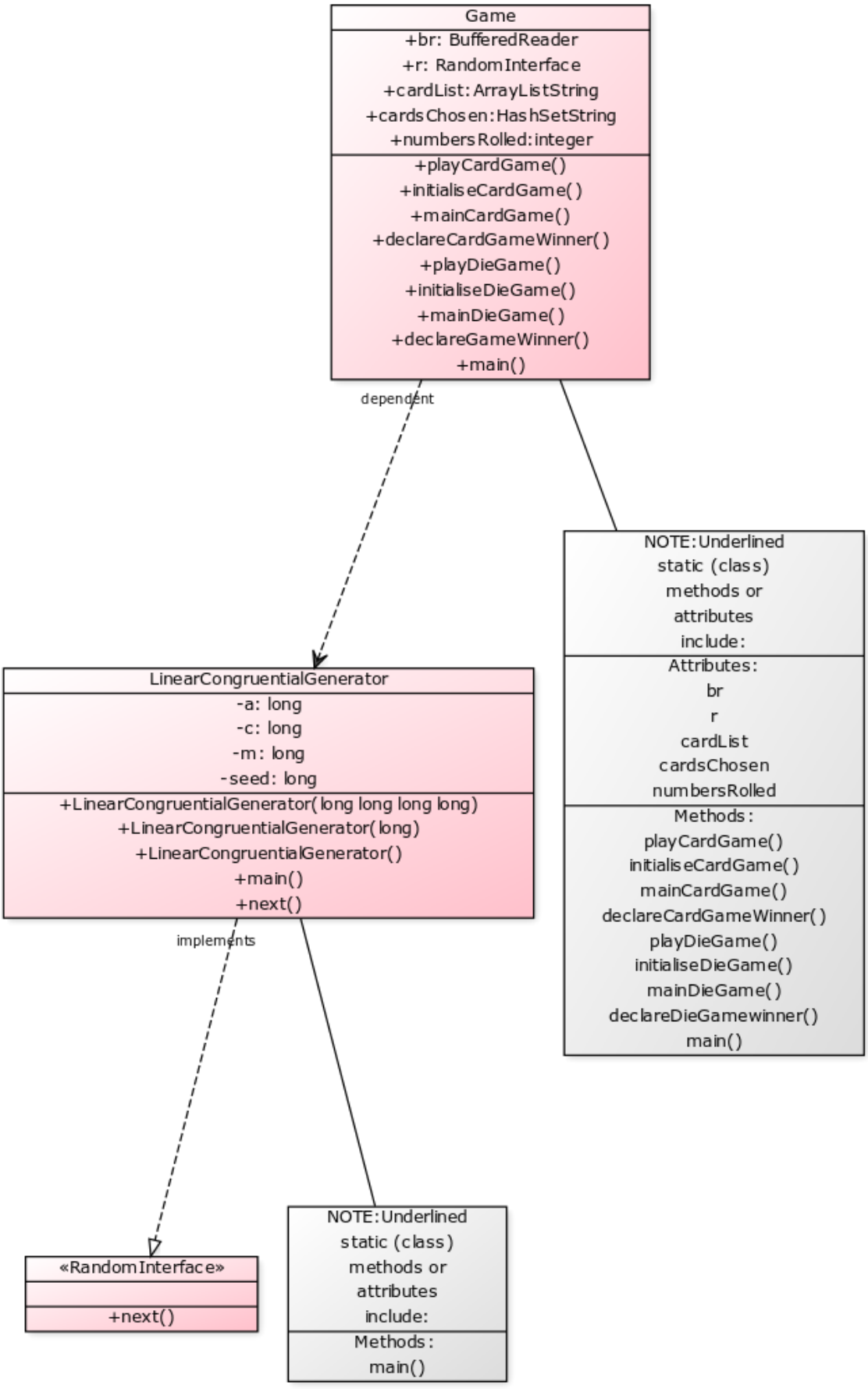
“Q1” source code for Q1

“Q2” source code for Q2



Object-Oriented Applications Assessed Exercise 1
C1620874
"Q1" source code for Q1
"Q2" source code for Q2

ii)



iii) The purpose of the Game program is to allow the user to play either a card or die game via a simple command line interface. Once the game type has been chosen the user must in both cases press return twice. In the card game the program chooses a card at random (from a generated list of possible cards) each time return is pressed and the user 'wins' if the one of the selected cards is an ace. In the die game the program simulates a roll of the die each time return is pressed and the user 'wins' if one of the rolls is a one.

The Game class uses a BufferedReader to handle user input, configures the random number generator used in either game using the LinearCongurentialGenerator class and contains the code to prompt the user to choose a game (via main()) and depending on input runs either a card game, playCardGame() or a die game, playDieGame().

The Game class is dependent on the LinearCongurentialGenerator class to implement the random number generator by implementing the RandomInterface interface (once corrected). The next() method implementation generates a random number in the range [0, 1] which the Game class can scale to choose of the 52 possible card choices or 6 possible die roll outcomes.

The LinearCongurentialGenerator class offers 3 constructors allowing the random number generator to be self-seeded using the current timestamp and sensible defaults, provided with a seed or fully specified. The Game class uses only the first method.

Question 2

i)

Class	Role
GameRunner	This class is uses a BufferedReader to handle the user input and stores the answer in a String variable called "ans".
GameFactory	This class is used to create the appropriate Game class. If the user inputted "c", the class will return a new CardGame(). If the user inputted "d", the class will return a new DieGame(). If the user has inputted neither of these, the class will return null.
Game	This class is an interface which CardGame and DieGame implements. Here the RandomInterface "r" is used by the CardGame and DieGame to get a random number used by both games.

Object-Oriented Applications Assessed Exercise 1
C1620874
"Q1" source code for Q1
"Q2" source code for Q2

CardGame	This class is where the program chooses a card at random (from a generated list of possible cards) each time return is pressed and the user 'wins' if the one of the selected cards is an ace.
DieGame	In this class the program simulates a roll of the die each time return is pressed, and the user 'wins' if one of the rolls is a one.
LinearCongruentialGenerator (unchanged from Q1)	The LinearCongurentialGenerator class offers 3 constructors allowing the random number generator to be self-seeded using the current timestamp and sensible defaults, provided with a seed or fully specified.
RandomInterface (unchanged from Q1)	The LinearCongurentialGenerator implements the random number generator by implementing the RandomInterface interface. The next() method implementation generates a random number in the range [0, 1] which can be scaled to choose of the 52 possible card choices or 6 possible die roll outcomes.

The improved version of the program applies principles relating to a number of concepts. Firstly, is the use of a Factory Pattern. This is a creational pattern that uses factory methods to deal with the problem of creating objects without having to specify the exact class of the object that will be created. In this case, the factory pattern returns a card game, a die game or null depending on the user's input.
Other issues that have been addressed include cohesion and coupling. I have made the classes highly cohesive and loosely coupled by reducing the dependency on the classes. This has been done by ensuring that each class represents one type of object and one type only. For example, by splitting up the original class Game which dealt with the user input and the resulting card game or die game so that each class had a specific responsibility also with the use of an interface.
Lastly, is encapsulation which is done by hiding information within a structure. I have done this by making most methods and attributes private in all of the classes. In the CardGame and DieGame classes, there is only one method that is not private called game() which is used by the Game interface.

Object-Oriented Applications Assessed Exercise 1
C1620874
"Q1" source code for Q1
"Q2" source code for Q2

ii)