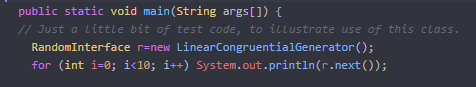
CM2307

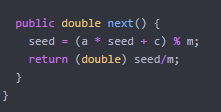
Object Oriented modelling and programming

C1816377 Harry Batchelor

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

I changed the code from IncompatibleRandomInterface to RandomInterface



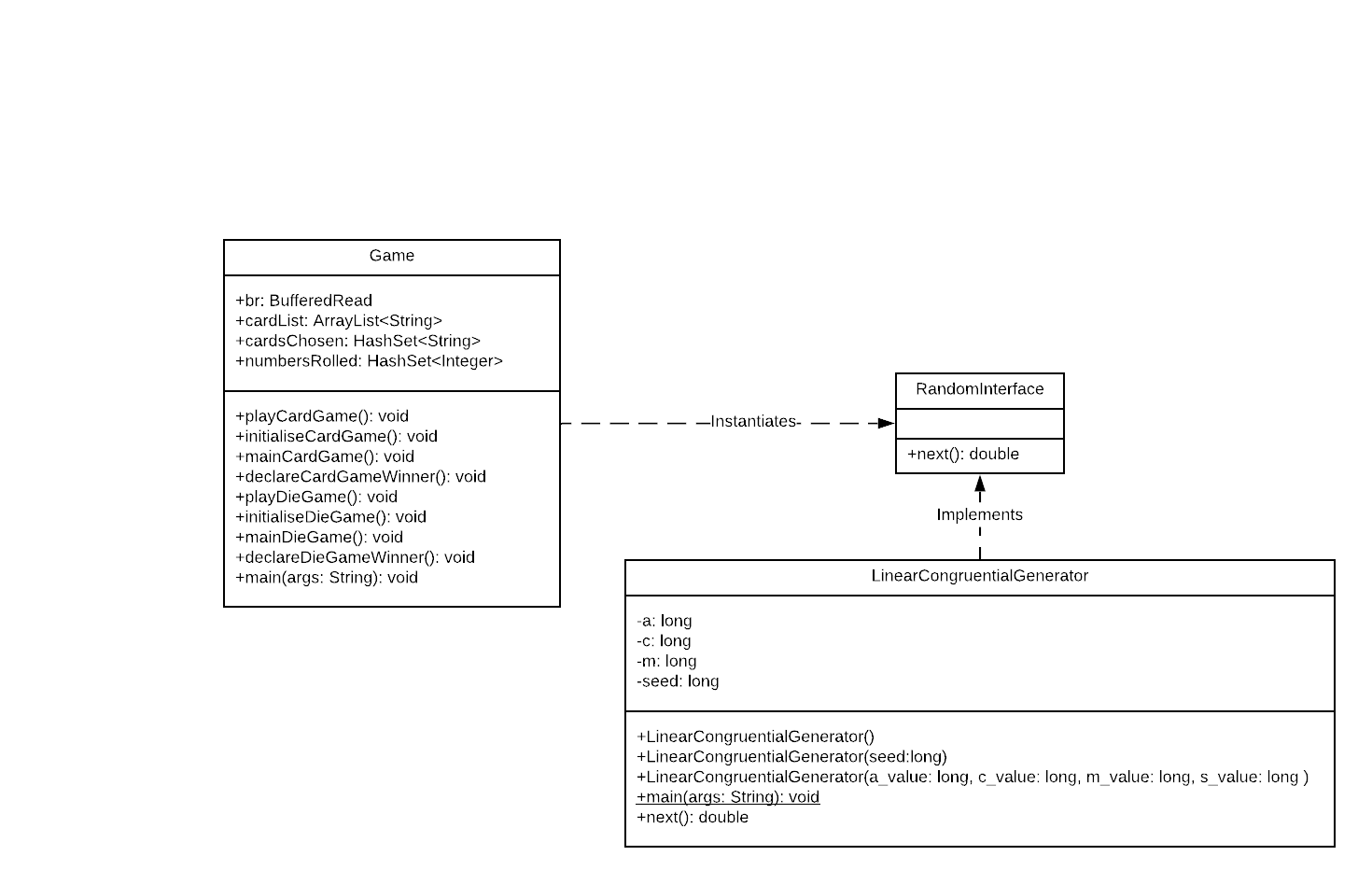
I changed IncompatibleRandomInterface to Randominterface and r.getNextNumber() to r.next()

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.

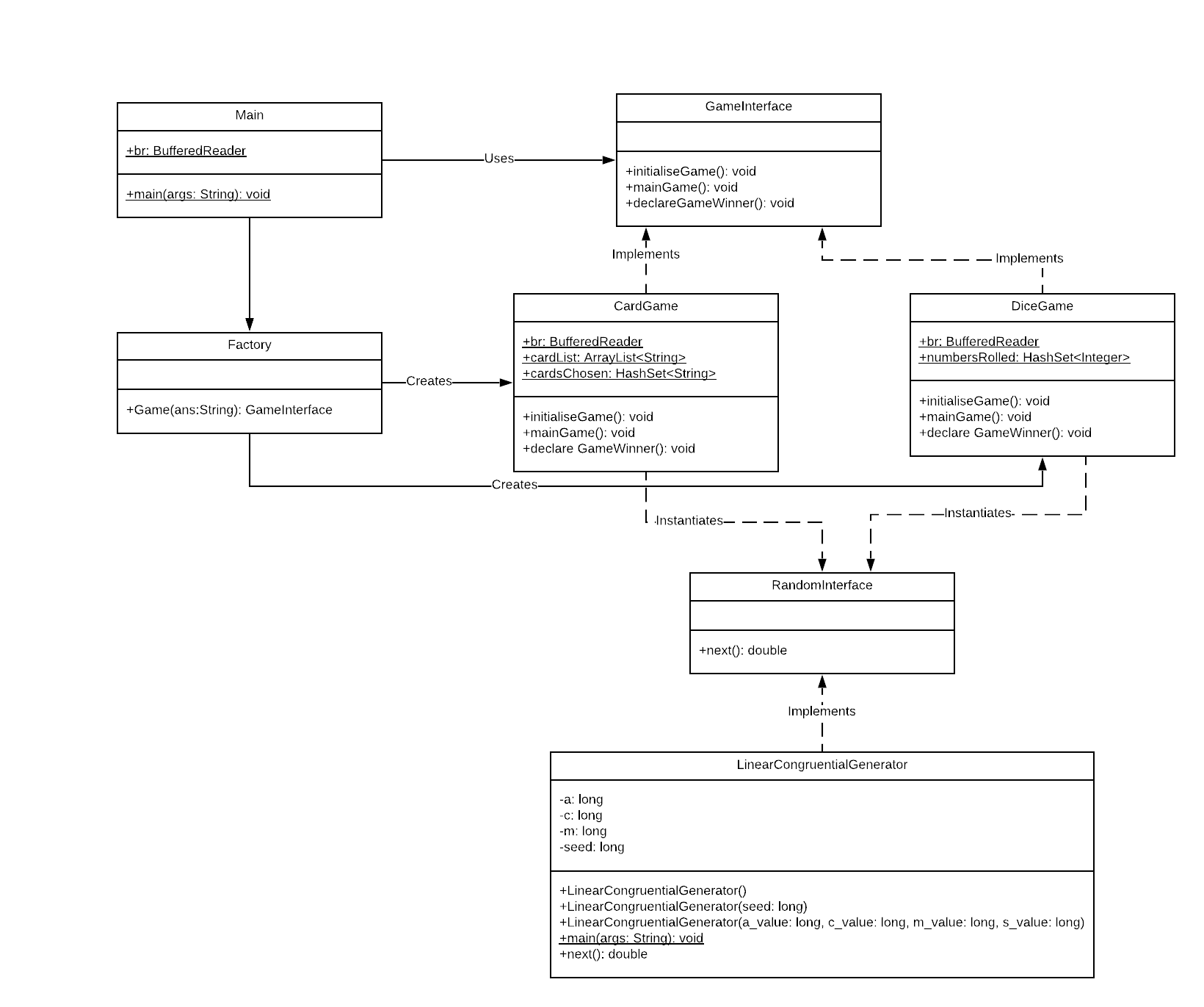


* + 1. UML diagram for Question 1 after modifying the code for part i

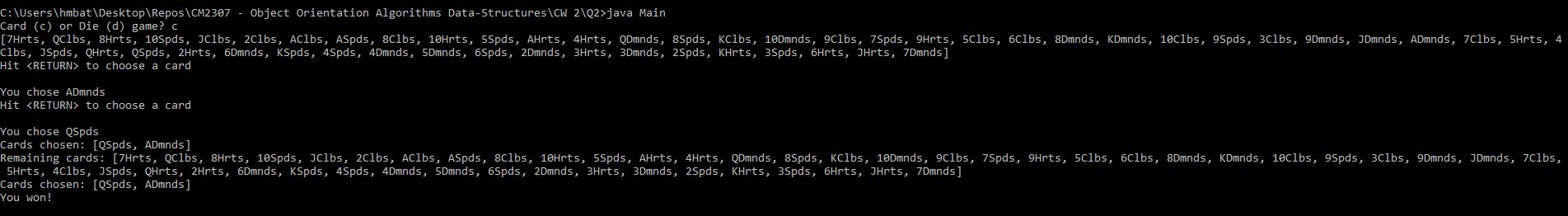


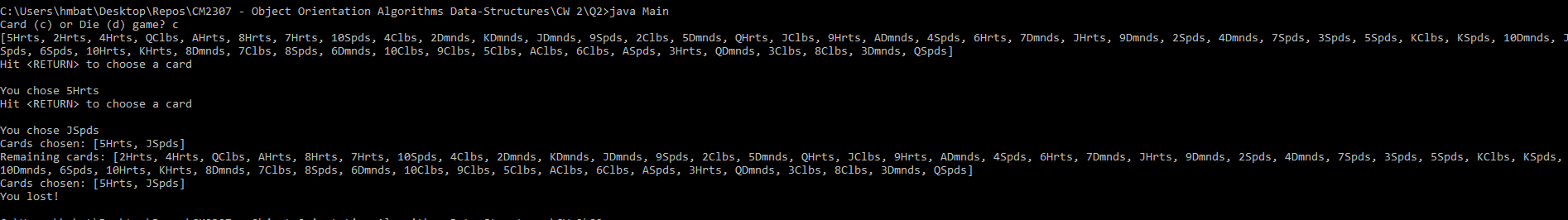
* + 1. The program contains an interface and 2 classes. The interface, “RandomInterface” is implemented by the class “LinearCongruentialGenerator” which picks a random number normalised from 0 to 1. The value that is given out from the “LinearCongruentialGenerator” is instantiated by the class “Game”; 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.

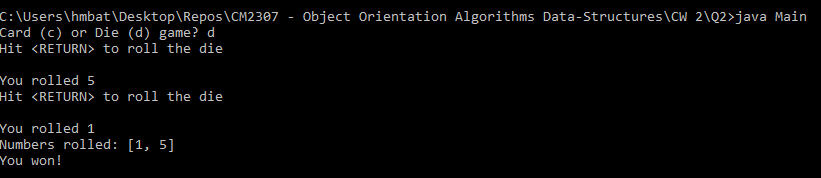
1. * 1. The instantiation of the classes is restricted by the structural design of the programme. The design follows a Singleton pattern. Several operations and attributes are contained inside the class "Game". These can all be split into 3 different classes. The classes being "Main", "DiceGame" and "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.
     2.  UML Diagram for Question 2 showing the improvements that I will make for part (iii)
     3. 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.

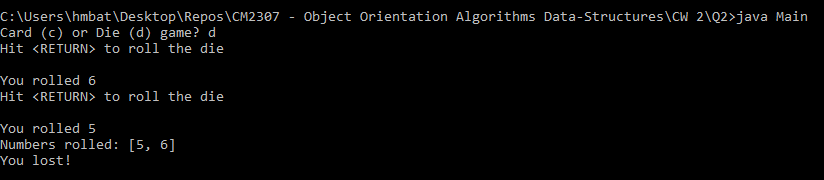
**Card Game**

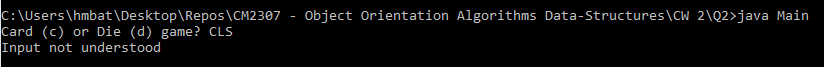
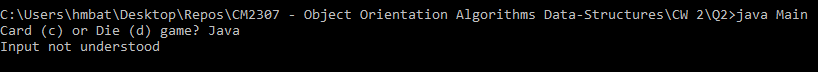
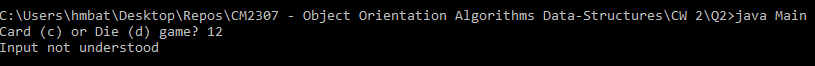
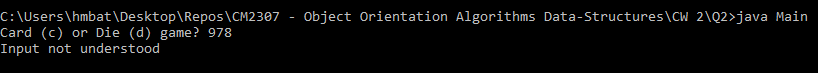
Winning game

Losing game

**Dice Game**

Winning game

Losing game

**Wrong inputs**