Question 1

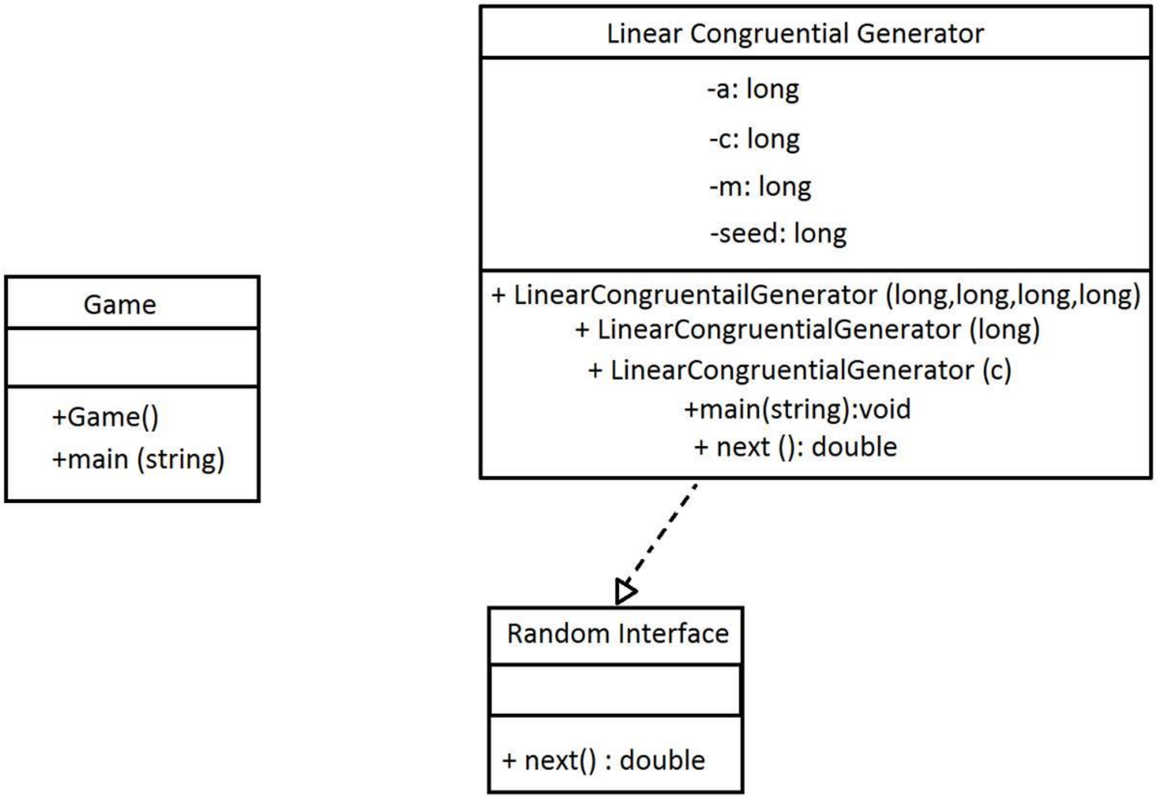
Write a description of the purpose of this program, bearing in mind that you are going to use this description to help you with identifying suitable classes for an improved version of this program in Question 2. [Guide: between 150 and 300 words]

Answer

The program starts by asking the user to choose between playing **cards** (by entering **c**) or **dices** (by entering **d**).

- If the user doesn't enter **c** or **d**, the program breaks and a message informing the user that "Input not understood" appears.
- If the user enters c or d, depending on the answer of the user (using an *if statement*), the program starts the corresponding procedure.
  1. For the game **Cards**, the user chooses two random cards and wins if at least one is an ace and loses if not. In the beginning, the program creates an ArrayList object called "cardlist", which contains the list of cards as *strings*. In advance, the program shuffles the cards-strings of the ArrayList, by selecting two random indices (using the random number from 0 to 1 generated) changing their position and repeats this exact procedure with different random indices, 100 times. When shuffling is complete, the program displays the list of cards and the message "Hit <RETURN> to choose a card". When the user hits return, the program chooses a random card(index), removes this index from the ArrayList, displays the chosen card, stores it at a variable called "cardsChosen" and asks the user again to "Hit <RETURN> to choose a card. The program does the exact same procedure again, and now other than the second chosen card, also displays both chosen cards and the remaining (52-2=) 50 cards. Finally, the program checks if any of the two cards chosen by the user (in cardsChosen) is an ace. If yes, it displays "You won!". If not, it displays "You lost!".
  2. For the game **Dice**, the user rolls one die twice and wins if at least one of the die rolls is 1, or loses if not. In the beginning, the program displays a message "Hit <RETURN> to roll the die". When the user hits return, the program chooses a random die roll from 1 to 6 and stores it at the variable "dieRoll" and at the variable "numbersRolled". After doing it twice, the program checks if any of the "numbersRolled" values is an ace and displays the correlated message.

UML CLASS DIAGRAM


Linear Congruential Generator
-a: long
-c: long
-m: long
-seed: long
+ LinearCongruentailGenerator (long,long,long,long)
+ LinearCongruentialGenerator (long)
+ LinearCongruentialGenerator (c)
+main(string):void
+ next (): double
Game
+Game()
+main (string)
Random Interface
+ next() : double

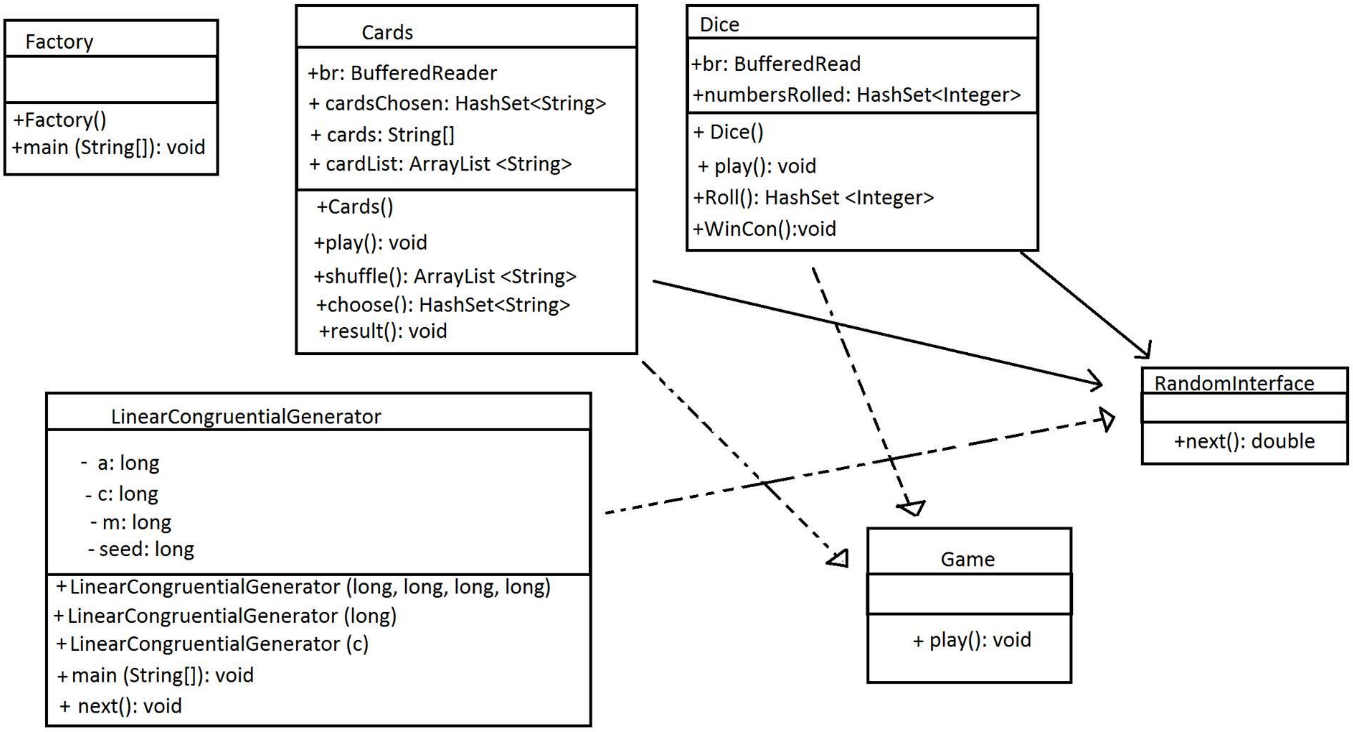
Question 2

Identify suitable classes for an improved version of the program, taking into account the points listed above, and explain your choice. [Guide: you may well choose to present these results in tabular form, with just a very small number of words explaining the role of each class, plus a small number of additional sentences explaining your overall choice].

Answer

It is acceptable for a small program to be consisted of one class but when it comes to a big program then having only one class makes the coding procedure difficult. Having more classes means code can be reused and methods can be called in other instances as well.
There should be four classes to have an improved version of the program.
One should be the card game. The card game contains all code needed to run the card game and is called by the Factory Class.
Another should be the dice game. The dice game contains all code needed to run the dice game and is called by the Factory Class.
The LinearCongruentialGenerator class was given to us which generates random numbers by implementing the RandomInterface interface.
The Factory class I created is needed to run the game and call the specific functions when needed. Using the Factory pattern is much better than having the program in a single file.
Two interfaces where needed, one to create random numbers and another one to run the games.

UML CLASS DIAGRAM


Factory
+Factory()
+main (String[]): void
Cards
+br: BufferedReader
+ cardsChosen: HashSet<String>
+ cards: String[]
+ cardList: ArrayList <String>
+Cards()
+play(): void
+shuffle(): ArrayList <String>
+choose(): HashSet<String>
+result(): void
Dice
+br: BufferedRead
+numbersRolled: HashSet<Integer>
+ Dice()
+ play(): void
+Roll(): HashSet <Integer>
+WinCon():void
RandomInterface
+next(): double
LinearCongruentialGenerator
- a: long
- c: long
- m: long
- seed: long
+LinearCongruentialGenerator (long, long, long, long)
+LinearCongruentialGenerator (long)
+LinearCongruentialGenerator (c)
+main (String[]): void
+ next(): void
Game
+ play(): void