

**CM2307 – Object Orientation, Algorithms and Data Structures**  
**Coursework 2 – Georgios K. Psevdiotis - C1841824**

---

<b><i>TASK 1</i></b>	<b>2</b>
<b><i>TASK 2</i></b>	<b>4</b>
<b><i>TASK 3</i></b>	<b>5</b>
<b><i>TASK 4</i></b>	<b>5</b>
<b><i>TASK 5</i></b>	<b>8</b>
<b><i>TASK 6</i></b>	<b>14</b>
<b><i>TASK 7</i></b>	<b>15</b>

---

## Task 1

### (i) UML Class Diagram



(ii) The classes above are of correct structure and code but the quality of it is low. To improve the overall factoring of this, I will use a refactoring technique called “Pull-up method” to enhance its elegance and make the software easier to understand. Furthermore, the chances of introducing new bugs will be reduced. To achieve this, two methods must be constructed in the super-class “Pet” for the **getName()** and **setName()** methods which were in the **Canary** and **Hamster** sub-classes.

Superclass **Pet** with the two new methods **setName()** and **getName()**:

```
public class Pet {
    protected String name;
    public void setName(String aName) { name=aName; }
    public String getName() { return name; }
    public String classOfAnimal() { return("Pet"); }
}
```

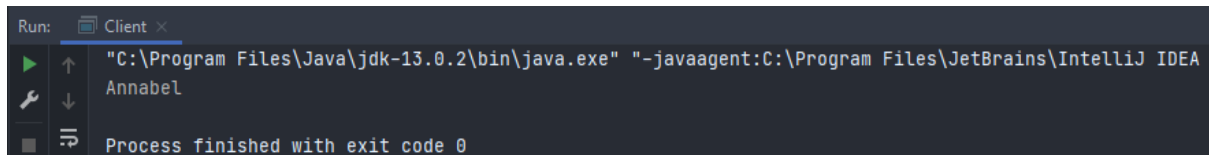
The two subclasses **Hamster** and **Canary** extend the superclass **Pet** without the use of methods **setName()** and **getName()**:

```
public class Hamster extends Pet {
    public String classOfAnimal() { return("Hamster"); }
}

public class Canary extends Pet {
    public String classOfAnimal() { return("Canary"); }
}
```

Class **Client** with the result of the code output:

```
public class Client {  
    public static void main(String[] args) {  
        Pet p = new Canary();  
        p.setName("Annabel");  
        System.out.println(p.getName());  
    }  
}
```



Run: Client x

"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA" Annabel

Process finished with exit code 0

(iii) Interface **Vegetarian**:

```
public interface Vegetarian { public String food(); }
```

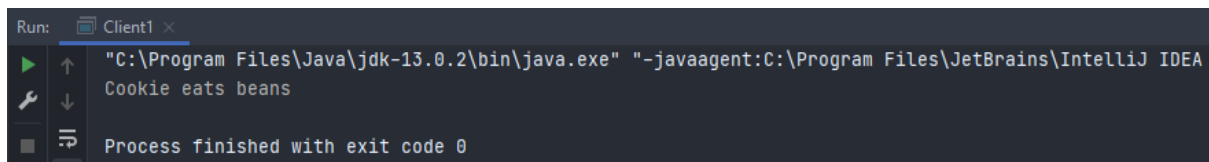
The class **Hamster** was revised to implement the interface **Vegetarian** using the **food** method:

```
public class Hamster extends Pet implements Vegetarian {  
    public String classOfAnimal() { return("Hamster"); }  
  
    @Override  
    public String food() {  
        return("beans");  
    }  
}
```

Class **Client1.java**:

```
public class Client1 {  
    public static void main(String[] args) {  
        Hamster h = new Hamster();  
        Pet p = h;  
        Vegetarian v = h;  
        h.setName("Cookie");  
        System.out.println(p.getName() + " eats " + v.food());  
    }  
}
```

The result of code given in **Client1.java**:



```
Run: Client1 x
"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Cookie eats beans
Process finished with exit code 0
```

(iii) The **Hamster** class uses a polymorphic reference variable that can be assigned to more than one kind of value and allows to perform a single action in different ways. This gives the object the ability to take on many forms and evolve. Since the Hamster class is a child class, all the reference variables h, p, v refer to the same Hamster object in the collection. The polymorphism of it allows it to redefine already defined behaviour inside parent class. It increases the program's speed and quality while it also reduces the programming effort of it. The reference variable calls on the objects, are pre-determined by the form of the reference variable.

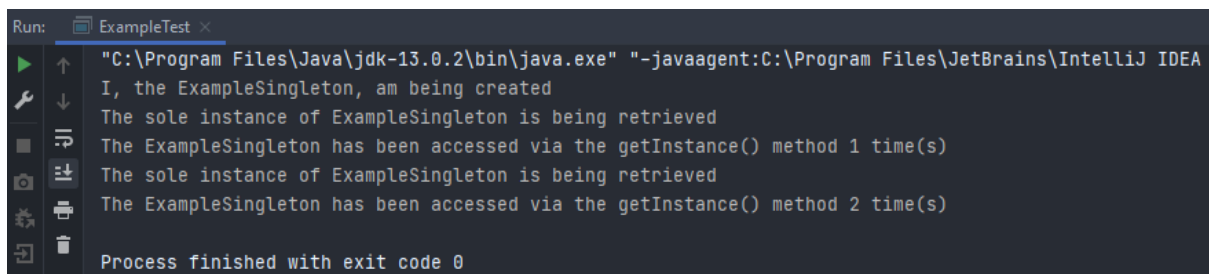
## Task 2

(i) The purpose of the Singleton design pattern is to make sure that a class has only one instance while still having a global reference point to it. It allows control over object creation by limiting the number of objects to only one entry point to create a new instance of a class.

(ii) Implementation of the class **ExampleSingleton** and its output:

```
public class ExampleSingleton {
    private int accessCount;
    private static final ExampleSingleton exInstance = new ExampleSingleton();

    private ExampleSingleton() { System.out.println("I, the ExampleSingleton, am being created"); }
    public static ExampleSingleton getInstance(){
        System.out.println("The sole instance of ExampleSingleton is being retrieved");
        return exInstance;
    }
    public int accessCount() { return accessCount = accessCount + 1 ; }
}
```



```
Run: ExampleTest x
"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
I, the ExampleSingleton, am being created
The sole instance of ExampleSingleton is being retrieved
The ExampleSingleton has been accessed via the getInstance() method 1 time(s)
The sole instance of ExampleSingleton is being retrieved
The ExampleSingleton has been accessed via the getInstance() method 2 time(s)
Process finished with exit code 0
```

### Task 3

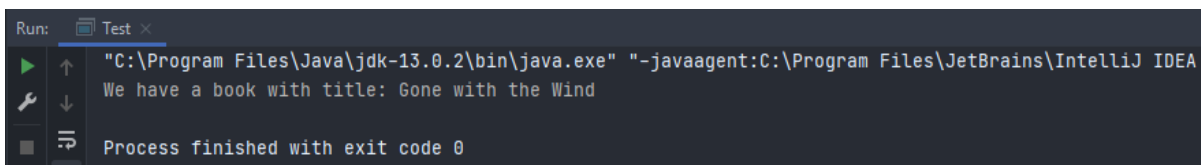
(i) The Adapter design patterns has a variety of purposes. Firstly, it allows objects with incompatible interfaces to collaborate by converting data into various formats making it understandable. It works as a bridge and uses a single class called the adapter class to join functionalities of independent or incompatible interfaces to allow them to work with each other without modifying their source code.

(ii) Implementation of the class **IncompatibleBook**:

```
public class IncompatibleBook {
    private String titleString;
    public String getTitle() { return titleString; }
    public void setTitle(String aString) { this.titleString = aString; }
}
```

(iii) Implementation of the class **BookAdapter** and its output:

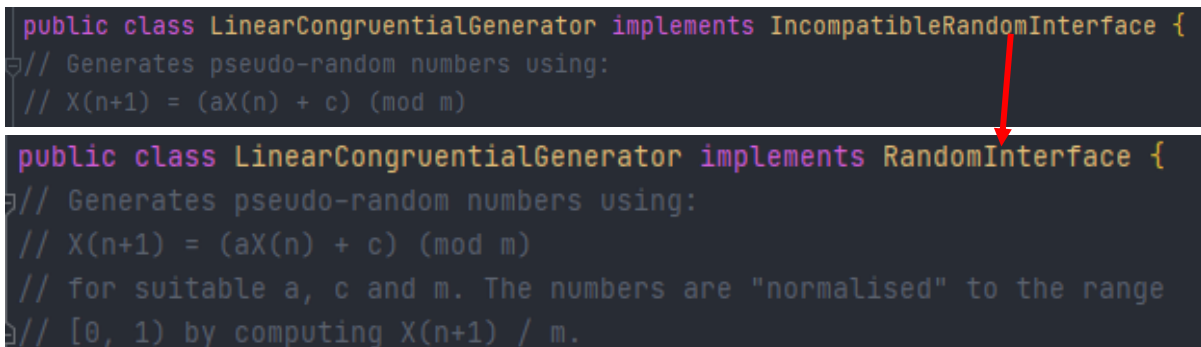
```
public class BookAdapter extends Book{
    private IncompatibleBook aString = new IncompatibleBook();
    public void setTitleString(String aString) { this.aString.setTitle(aString); }
    public String getTitleString() { return this.aString.getTitle(); }
}
```



```
Run: Test x
"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
We have a book with title: Gone with the Wind
Process finished with exit code 0
```

### Task 4

(i) By running the program, a few exceptions are thrown while trying to play either a card game or a die game. To fix these problems, at first, the class' **LinearCongruentialGenerator** interface must be changed to **RandomInterface**.



```
public class LinearCongruentialGenerator implements IncompatibleRandomInterface {
    // Generates pseudo-random numbers using:
    // X(n+1) = (aX(n) + c) (mod m)

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.
```

Furthermore, the **IncompatibleRandomInterface** method must be replaced with **RandomInterface**.

```
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.next());  
}
```

```
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());  
}
```

Then on, the variable **getNextnumber()** which is **IncompatibleRandomInterface** 's needs to be replaced with **next()** which is **RandomInterface** 's in the main function of **LinearCongruentialGenerator** as well as the function in the end of the class.

```
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.getNextNumber());  
}
```

```
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());  
}
```

```
public double getNextNumber() {  
    seed = (a * seed + c) % m;  
    return (double) seed/m;  
}
```

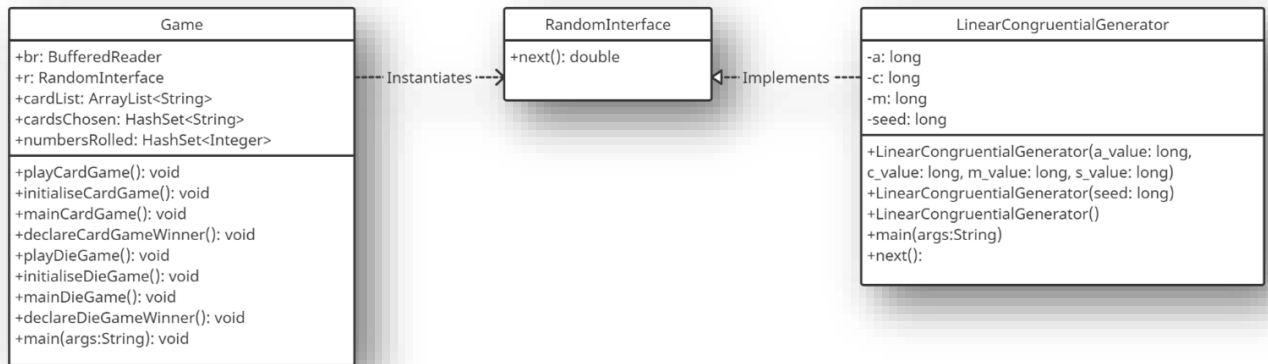
```
public double next() {  
    seed = (a * seed + c) % m;  
    return (double) seed/m;  
}
```

Finally, the **commented** part has to be removed so that a new random number can be generated and be used correctly.

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

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

## (ii) UML Class Diagram



(iii) The program is comprehended by two classes and an interface, and by observation it can be acknowledged that it follows a Singleton pattern design restricting the instantiation of classes. Firstly, the **RandomInterface** interface is implemented by the **LinearCongruentialGenerator** class and it generates a random normalized number from 0 to 1. Secondly, the **RandomInterface** interface is instantiated by the **Game** class by using the value created by the **LinearCongruentialGenerator** throughout the game. The **Game** class contains various operations and attributes that could have been divided between other different classes. Due to this, the program is said to have low cohesion because it includes plenty of methods and attributes used, and the tasks perform between only two classes. In order to maximize the cohesion and optimization of the program, the responsibility of each class should be broken down into smaller bits, divided accordingly to different classes and connect them to increase efficiency and the quality of the program. A way to approach a new design to this solution would be to break down the **Game** class to 3 classes. One being the **Main** class and one class for each game, i.e., **GameDie** and **GameCard**. Also, a new class can be created that includes the user's choice on to which game to play separately from the Main class, and this can help with error recognisability. Another interface could be created that would be responsible to create the operations of the chosen game and declare the game winner. This way the coupling is loosen as well as the dependencies between these classes.

## Task 5

**(i) Main →** The main class is responsible to call the UserChoice class and initialise the program after the user has selected which game to play.

**GameInterface →** The methods of the **GameCard** and **GameDie** are implemented, as well as the operation created by the two methods, and are directed to the **Main** class. By creating this interface, it improves the overall structure of the program and its code and it assists in decreasing the dependencies between all the classes and loosens the coupling.

**UsersChoice →** The main goal of this interface is to create an instance of the user's decision as to which game to play. The methods used are inherited by the interface and it is a main structure object to divide the two **Game** classes.

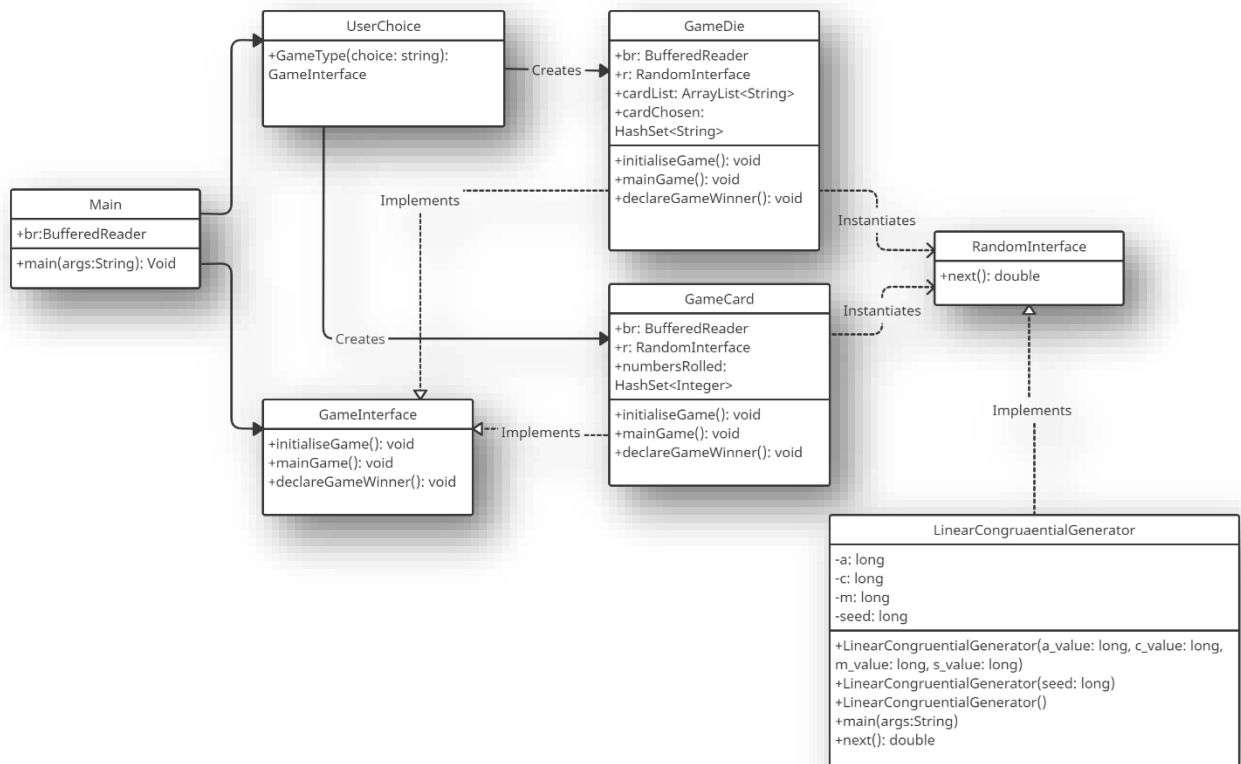
**GameCard & GameDie →** The two classes include their related methods and attributes obtained from the previous **Game** class.

**RandomInterface →** The **GameCard** and **GameDie** classes instantiates this interface to retrieve the next random number.

**LinearCongruentialGenerator →** Implements the **RandomInterface** interface.



## (ii) UML Class Diagram



## (iii) Implementation

### Main Class:

```
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class Main {
    public static BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    public static void main(String[] args) throws Exception {
        System.out.print("Card (c) or Die (d) game? ");
        String ans = br.readLine();
        UserChoice userChoice = new UserChoice();
        GameInterface game = userChoice.GameType(ans);
        game.initialiseGame();
        game.mainGame();
        game.declareGameWinner();
    }
}
```

### UsersChoice Class:

```
public class UserChoice {  
    public GameInterface GameType(String choice) throws Exception {  
        if (choice.equals("c")) {  
            return new GameCard();  
        } else if (choice.equals("d")) {  
            return new GameDie();  
        } else  
            System.out.println("Invalid choice!");  
            System.exit(status: 1);  
            return null;  
    }  
}
```

### GameInterface Interface:

```
public interface GameInterface {  
    public void initialiseGame() throws Exception;  
    public void mainGame() throws Exception;  
    public void declareGameWinner() throws Exception;  
}
```

## GameCard Class:

```
import ...

public class GameCard implements GameInterface{
    // The BufferedReader used throughout
    public static BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
    // The random number generator used throughout
    public static RandomInterface r=new LinearCongruentialGenerator();
    // Variable(s) used in the card game methods
    public static ArrayList<String> cardList;
    public static HashSet<String> cardsChosen=new HashSet<>();

    @Override
    public void initialiseGame() throws Exception {
        // The initialisation phase:
        // Create a list of cards ... and shuffle them
        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"};
        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);
    }

    @Override
    public void mainGame() throws Exception {
        // The main game:
        // Let user select two cards from the pack
        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);
    }

    @Override
    public void declareGameWinner() throws Exception {
        // Declare the winner:
        // User wins if one of them is an Ace
        System.out.println("Cards chosen: " + cardsChosen);
        if (cardsChosen.contains("AHrts") || cardsChosen.contains("ADmnds") ||
            cardsChosen.contains("ASpds") || cardsChosen.contains("AClbs")) {
            System.out.println("You won!");
        }
        else System.out.println("You lost!");
    }
}
```

## GameDie Class:

```
import ...

public class GameDie implements GameInterface {
    // The BufferedReader used throughout
    public static BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

    //Variable(s) used in the die game methods
    public static HashSet<Integer> numbersRolled=new HashSet<>();

    public static RandomInterface r=new LinearCongruentialGenerator();

    @Override
    public void initialiseGame() throws Exception {}

    @Override
    public void mainGame() throws Exception{
        // The main game:

        // Let the user roll the die twice
        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));
        }

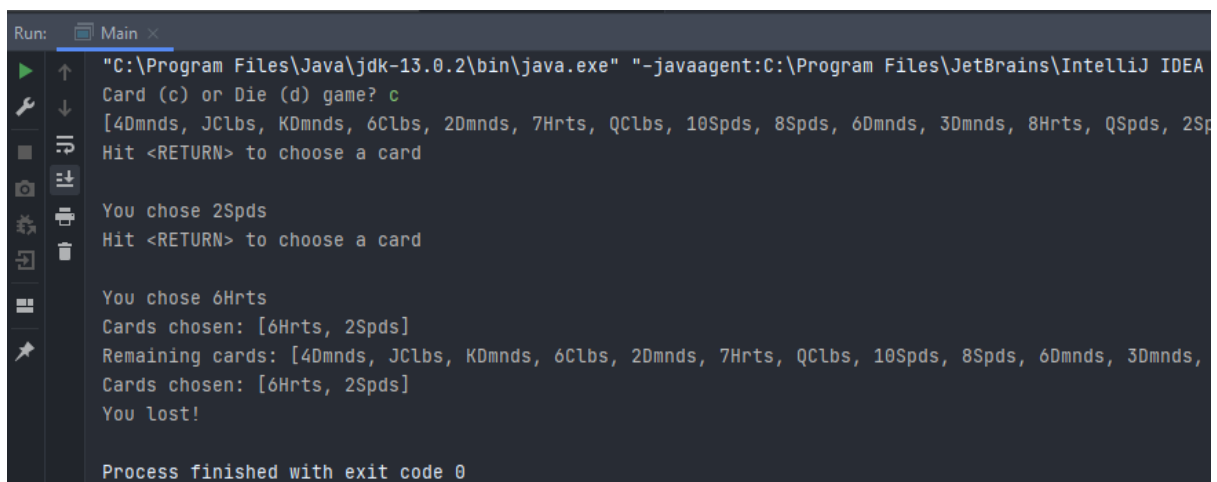
        // Display the numbers rolled
        System.out.println("Numbers rolled: " + numbersRolled);
    }

    @Override
    public void declareGameWinner() throws Exception {
        // Declare the winner:

        // User wins if at least one of the die rolls is a 1
        if (numbersRolled.contains(new Integer(value: 1))) {
            System.out.println("You won!");
        }
        else System.out.println("You lost!");
    }
}
```

**\*\* The LinearCongruentialGenerator class  
and the RandomInterface interface remain the same \*\***

## Card Game Outputs:



```
Run: Main x
"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Card (c) or Die (d) game? c
[40mnds, JClbs, KDmnds, 6Clbs, 2Dmnds, 7Hrts, QClbs, 10Spds, 8Spds, 6Dmnds, 3Dmnds, 8Hrts, QSpds, 2Sp
Hit <RETURN> to choose a card

You chose 2Spds
Hit <RETURN> to choose a card

You chose 6Hrts
Cards chosen: [6Hrts, 2Spds]
Remaining cards: [40mnds, JClbs, KDmnds, 6Clbs, 2Dmnds, 7Hrts, QClbs, 10Spds, 8Spds, 6Dmnds, 3Dmnds,
Cards chosen: [6Hrts, 2Spds]
You lost!

Process finished with exit code 0
```

```
Run: Main x
"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Card (c) or Die (d) game? c
[QHrts, 8Spds, 10Dmnds, 6Hrts, ACLbs, QClbs, KDmnds, 7Hrts, 2Clbs, 7Spds, ASpds, KSpds, 3Spds, JDmnds]
Hit <RETURN> to choose a card

You chose AHrts
Hit <RETURN> to choose a card

You chose 3Clbs
Cards chosen: [3Clbs, AHrts]
Remaining cards: [QHrts, 8Spds, 10Dmnds, 6Hrts, ACLbs, QClbs, KDmnds, 7Hrts, 2Clbs, 7Spds, ASpds, KSpds, 3Spds, JDmnds]
Cards chosen: [3Clbs, AHrts]
You won!

Process finished with exit code 0
```

## Die Game Output:

```
Run: Main x
"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Card (c) or Die (d) game? d
Hit <RETURN> to roll the die

You rolled 6
Hit <RETURN> to roll the die

You rolled 6
Numbers rolled: [6]
You lost!

Process finished with exit code 0
```

```
Run: Main x
"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Card (c) or Die (d) game? d
Hit <RETURN> to roll the die

You rolled 3
Hit <RETURN> to roll the die

You rolled 1
Numbers rolled: [1, 3]
You won!

Process finished with exit code 0
```

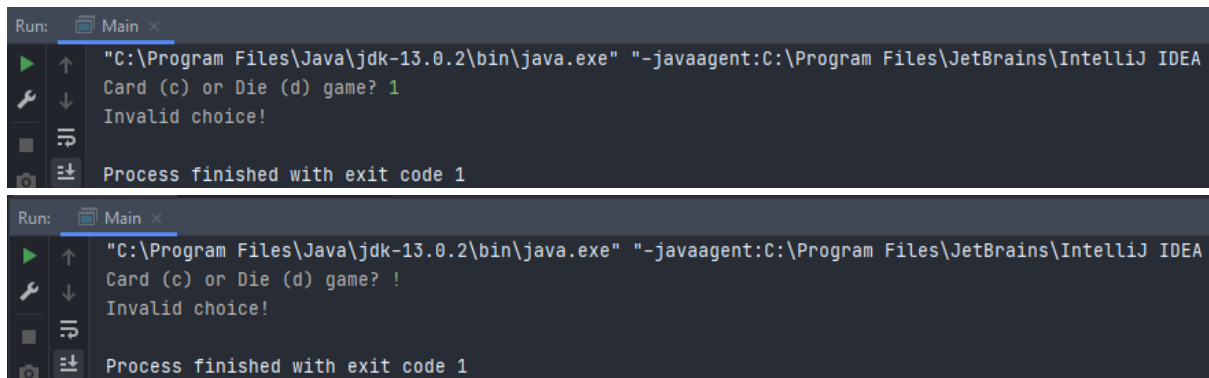
## Erroneous Inputs Output:

```
Run: Main x
"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Card (c) or Die (d) game?
Invalid choice!

Process finished with exit code 1
```

```
Run: Main x
"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Card (c) or Die (d) game? a
Invalid choice!

Process finished with exit code 1
```



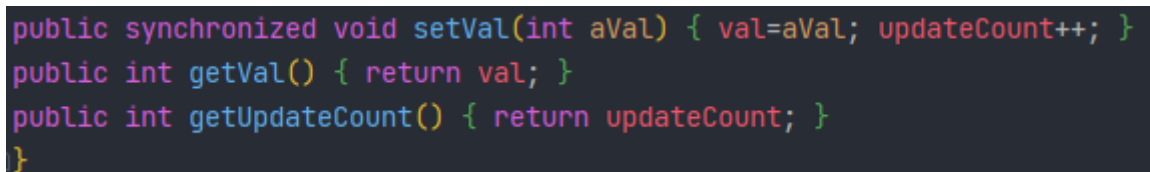
```
Run: Main x
"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Card (c) or Die (d) game? 1
Invalid choice!
Process finished with exit code 1

Run: Main x
"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Card (c) or Die (d) game? !
Invalid choice!
Process finished with exit code 1
```

## Task 6

(i) To begin with, after running the code numerous times, the output of it was 9997,9999,9998 and 10000. This is because the threads use the same tests/generators, and this results in different outputs each time it runs instead of reaching 1000. The problem arises because the threads try to overwrite a shared resource simultaneously and this causes a race condition because the threads share memory space. Each thread attempts to finish accessing the shared resource first but each time they try, they overlap each other in execution which causes the final outcome to be unpredictable. The overlap is caused by the threads interfering each other when a thread executes the setVal() method. Firstly, the updateCount value is retrieved. Then the value is incremented by 1 and finally it is stored back in updateCount.

(ii)



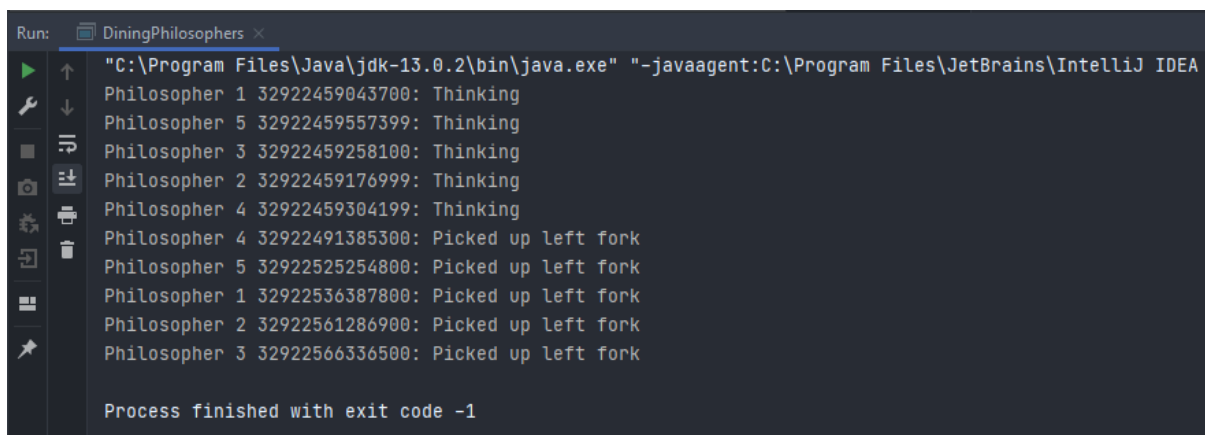
```
public synchronized void setVal(int aVal) { val=aVal; updateCount++; }
public int getVal() { return val; }
public int getUpdateCount() { return updateCount; }
}
```

(iii) In order to turn the code to be thread-safe, the setVal() method was changed to be synchronized to ensure that the threads cannot call setVal() on the same sequential instance at the same time. This causes the threads to have a mutual exclusive access when incrementing the updateCount 's value and avoids interleaving.

## Task 7

(i) The use of the **synchronized** keyword in the Philosopher class on Fork objects, acts as a lock to ensure that two threads cannot use it at the same time but only one can be executed at a time. To do that, it obtains the fork object's internal monitor and blocks other threads from doing the same even though all the threads start at the same time.

(ii) A Deadlock occurs when a system's progress is halted while each process is waiting for a resource held by another process. This can be caused when threads make requests for new threads constantly or when threads that already hold locks request for new ones. Finally, it can also occur when threads form a circular chain in which each thread waits for a lock which is already acquired by the next thread in the chain, and cannot be provided therefore, it hangs, like in our case. Each Philosopher picks up his left fork but halts because another Philosopher holds the right fork, and they cannot proceed.



```
Run: DiningPhilosophers x
"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Philosopher 1 32922459043700: Thinking
Philosopher 5 32922459557399: Thinking
Philosopher 3 32922459258100: Thinking
Philosopher 2 32922459176999: Thinking
Philosopher 4 32922459304199: Thinking
Philosopher 4 32922491385300: Picked up left fork
Philosopher 5 32922525254800: Picked up left fork
Philosopher 1 32922536387800: Picked up left fork
Philosopher 2 32922561286900: Picked up left fork
Philosopher 3 32922566336500: Picked up left fork

Process finished with exit code -1
```

(iii) In order to prevent a deadlock situation from occurring, some adjustments were made. At first, the philosopher's variable is made final which means that once it is assigned a value, it cannot be changed.

```
final Philosopher[] philosophers = new Philosopher[5];
Object[] forks = new Object[philosophers.length];
```

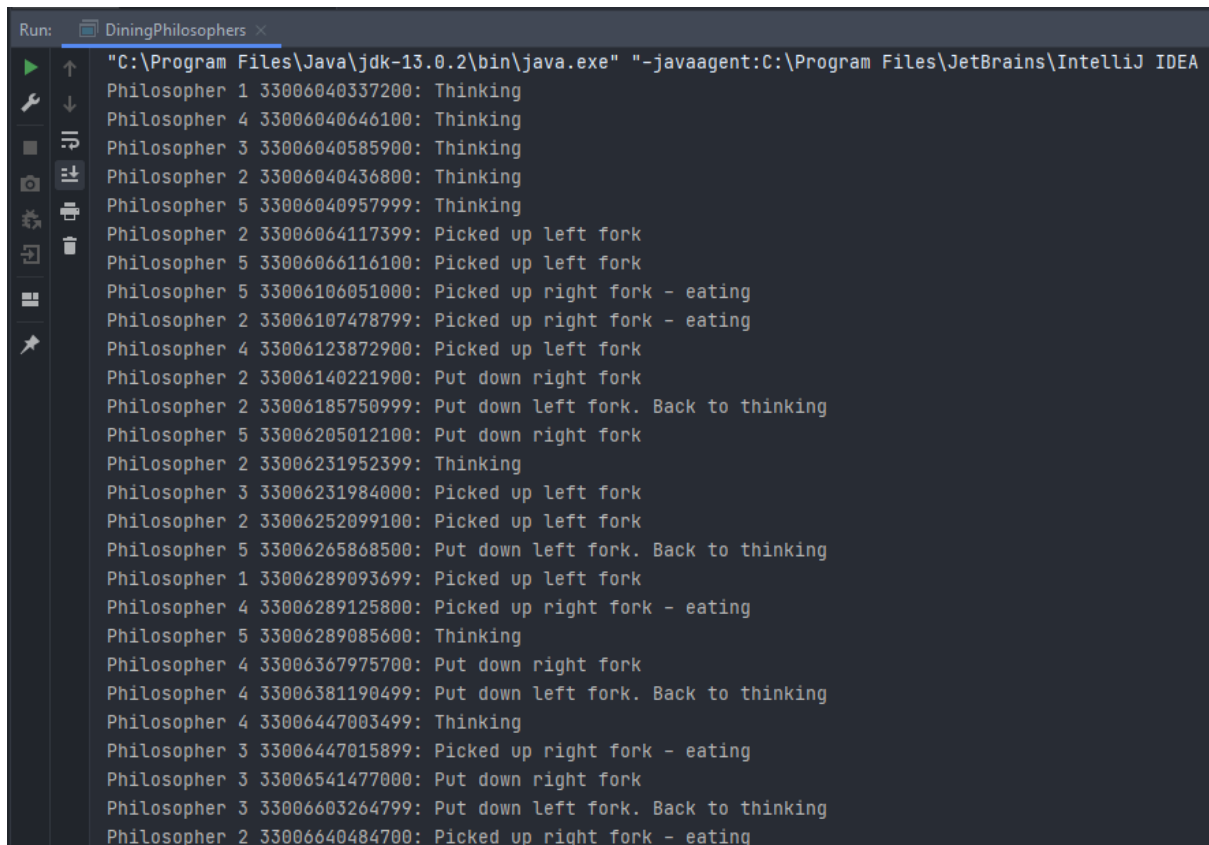
Moreover, to break the circular chain and avoid the deadlock, an if statement is introduced that forces the last Philosopher to pick up the right fork in the beginning instead of the left.

```

if (i == philosophers.length - 1) {
    philosophers[i] = new Philosopher(rightFork, leftFork);
} else {
    philosophers[i] = new Philosopher(leftFork, rightFork);
}

```

Result without the deadlock situation:



```

Run: DiningPhilosophers
"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Philosopher 1 33006040337200: Thinking
Philosopher 4 33006040646100: Thinking
Philosopher 3 33006040585900: Thinking
Philosopher 2 33006040436800: Thinking
Philosopher 5 33006040957999: Thinking
Philosopher 2 33006064117399: Picked up left fork
Philosopher 5 33006066116100: Picked up left fork
Philosopher 5 33006106051000: Picked up right fork - eating
Philosopher 2 33006107478799: Picked up right fork - eating
Philosopher 4 33006123872900: Picked up left fork
Philosopher 2 33006140221900: Put down right fork
Philosopher 2 33006185750999: Put down left fork. Back to thinking
Philosopher 5 33006205012100: Put down right fork
Philosopher 2 33006231952399: Thinking
Philosopher 3 33006231984000: Picked up left fork
Philosopher 2 33006252099100: Picked up left fork
Philosopher 5 33006265868500: Put down left fork. Back to thinking
Philosopher 1 33006289093699: Picked up left fork
Philosopher 4 33006289125800: Picked up right fork - eating
Philosopher 5 33006289085600: Thinking
Philosopher 4 33006367975700: Put down right fork
Philosopher 4 33006381190499: Put down left fork. Back to thinking
Philosopher 4 33006447003499: Thinking
Philosopher 3 33006447015899: Picked up right fork - eating
Philosopher 3 33006541477000: Put down right fork
Philosopher 3 33006603264799: Put down left fork. Back to thinking
Philosopher 2 33006640484700: Picked up right fork - eating

```