

BankTransaction.java

Part 1

```
1  /** Application Purpose: This is the class for BankTransaction blueprint.
2  *   Author: Alex Dorodko
3  *   Date: 31/OCT/2020
4  *   Time: 07:44 PM
5  */
6
7  public class BankTransaction
8  {
9      //Setting up the variables.
10     private boolean canadianFunds;
11     private double exchangeRate;
12     private int transactionNumber;
13     private long transactionReferenceNumber;
14
15     //This is the counter for the records.
16     public static int recordNumber = 0;
17
18     //Making the constructor with the state.
19     public BankTransaction(boolean canadianFunds, double exchangeRate, int transactionNumber, long transactionReferenceNumber)
20     {
21         this.canadianFunds = canadianFunds;
22         this.exchangeRate = exchangeRate;
23         this.transactionNumber = transactionNumber;
24         this.transactionReferenceNumber = transactionReferenceNumber;
25     }
26
27
28
29     //Setting up the getters.
30     public void setCanadianFunds(boolean canadianFunds)
31     {
32         this.canadianFunds = canadianFunds;
33     }
34
35     public void setExchangeRate(double exchangeRate)
36     {
37         this.exchangeRate = exchangeRate;
38     }
39
40     public void setTransactionNumber(int transactionNumber)
41     {
42         this.transactionNumber = transactionNumber;
43     }
44
45     public void setTransactionReferenceNumber(long transactionReferenceNumber)
46     {
47         this.transactionReferenceNumber = transactionReferenceNumber;
48     }
49 }
```

Part 2

```
25 }
26
27 //Setting up the getters.
28 public void setCanadianFunds(boolean canadianFunds)
29 {
30     this.canadianFunds = canadianFunds;
31 }
32
33 public void setExchangeRate(double exchangeRate)
34 {
35     this.exchangeRate = exchangeRate;
36 }
37
38 public void setTransactionNumber(int transactionNumber)
39 {
40     this.transactionNumber = transactionNumber;
41 }
42
43 public void setTransactionReferenceNumber(long transactionReferenceNumber)
44 {
45     this.transactionReferenceNumber = transactionReferenceNumber;
46 }
47
48
49
50
51 //Setting up the setters.
52 public boolean getCanadianFunds()
53 {
54     return canadianFunds;
55 }
56
57 public double getExchangeRate()
58 {
59     return exchangeRate;
60 }
61
62 public int getTransactionNumber()
63 {
64     return transactionNumber;
65 }
66
67 public long getTransactionReferenceNumber()
68 {
69     return transactionReferenceNumber;
70 }
71
72 }
```

BankTransactionTestHarness.java

Part 1

```
BankTransactionTestHarness.java
week7 > BankTransactionTestHarness.java ...
1  /** Application Purpose: This is the test harness for the Bank Transaction class with the main method.
2   * Author: Alex Donohue
3   * Date: 31/OCT/2020
4   * Time: 09:34 PM
5   */
6
7
8  //Importing Scanner and Random.
9  import java.util.Random;
10 import java.util.Scanner;
11
12 //Declaring the class
13 public class BankTransactionTestHarness
14 {
15     //This is the main method.
16     Run[Debug
17     public static void main(String[] args)
18     {
19         //Instantiating the scanner and random.
20         Scanner sc = new Scanner(System.in);
21         Random random = new Random();
22
23         //Instantiating the BankTransaction constructor, which has 500 records.
24         BankTransaction[] mainArray = new BankTransaction[500];
25
26         //This will come later on for the for loop, just declaring it at this time.
27
28         //This will fill all the 500 records with random data, and print it out.
29         for (int i = 0; i < 500; i++, BankTransaction.recordNumber++)
30         {
31             mainArray[i] = new BankTransaction(random.nextBoolean(), random.nextDouble() * (10000 - 0 + 1), random.nextInt(), random.nextLong());
32
33             System.out.println("\n" + mainArray[i].getCanadianFunds() + "\n" + mainArray[i].getExchangeRate() + "\n" + mainArray[i].getTransactionNumber() + "\n" + mainArray[i].getTransactionReferenceNumber() +
34                             "\n" + (BankTransaction.recordNumber + 1) + "\n\n-----");
35         }
36
37         //Asking the user if he would like to run this section again.
38         System.out.println("\nWould you like to run this code again? (true/false)\n");
39         boolean answer = sc.nextBoolean();
40
41         //This section will run the code again, filling the 500 records, overriding them.
42         while (answer == true)
43         {
44             //Resetting the counter for the records.
45             BankTransaction.recordNumber = 0;
46
47             for (int i = 0; i < 500; i++, BankTransaction.recordNumber++)
48             {
49
50             }
51         }
52     }
53 }
```

Part 2

```
BankTransactionTestHarness.java
week7 > BankTransactionTestHarness.java > BankTransactionTestHarness > main(String[])
14 {
15     //This is the main method.
16     Run[Debug
17     public static void main(String[] args)
18     {
19         //Instantiating the scanner and random.
20         Scanner sc = new Scanner(System.in);
21         Random random = new Random();
22
23         //Instantiating the BankTransaction constructor, which has 500 records.
24         BankTransaction[] mainArray = new BankTransaction[500];
25
26         //This will come later on for the for loop, just declaring it at this time.
27
28         //This will fill all the 500 records with random data, and print it out.
29         for (int i = 0; i < 500; i++, BankTransaction.recordNumber++)
30         {
31             mainArray[i] = new BankTransaction(random.nextBoolean(), random.nextDouble() * (10000 - 0 + 1), random.nextInt(), random.nextLong());
32
33             System.out.println("\n" + mainArray[i].getCanadianFunds() + "\n" + mainArray[i].getExchangeRate() + "\n" + mainArray[i].getTransactionNumber() + "\n" + mainArray[i].getTransactionReferenceNumber() +
34                             "\n" + (BankTransaction.recordNumber + 1) + "\n\n-----");
35         }
36
37         //Asking the user if he would like to run this section again.
38         System.out.println("\nWould you like to run this code again? (true/false)\n");
39         boolean answer = sc.nextBoolean();
40
41         //This section will run the code again, filling the 500 records, overriding them.
42         while (answer == true)
43         {
44             //Resetting the counter for the records.
45             BankTransaction.recordNumber = 0;
46
47             for (int i = 0; i < 500; i++, BankTransaction.recordNumber++)
48             {
49                 mainArray[i] = new BankTransaction(true, 999.999, 12345, 948346355);
50
51                 System.out.println("\n" + mainArray[i].getCanadianFunds() + "\n" + mainArray[i].getExchangeRate() + "\n" + mainArray[i].getTransactionNumber() + "\n" + mainArray[i].getTransactionReferenceNumber() +
52                                     "\n" + (BankTransaction.recordNumber + 1) + "\n\n-----");
53             }
54
55             //Asking them once again, if they would like to keep running it. They can run it as many times as they wish.
56             System.out.println("\nWould you like to run this code again? (true/false)\n");
57             answer = sc.nextBoolean();
58         }
59     }
60 }
```