MIS 768: Advanced Software Concepts

Spring 2024

Exception Handling and Object Serialization

Purpose

- Practice adding exception handler to the program.
- Create customized exceptions.
- Save and read objects from files.

1. Preparation

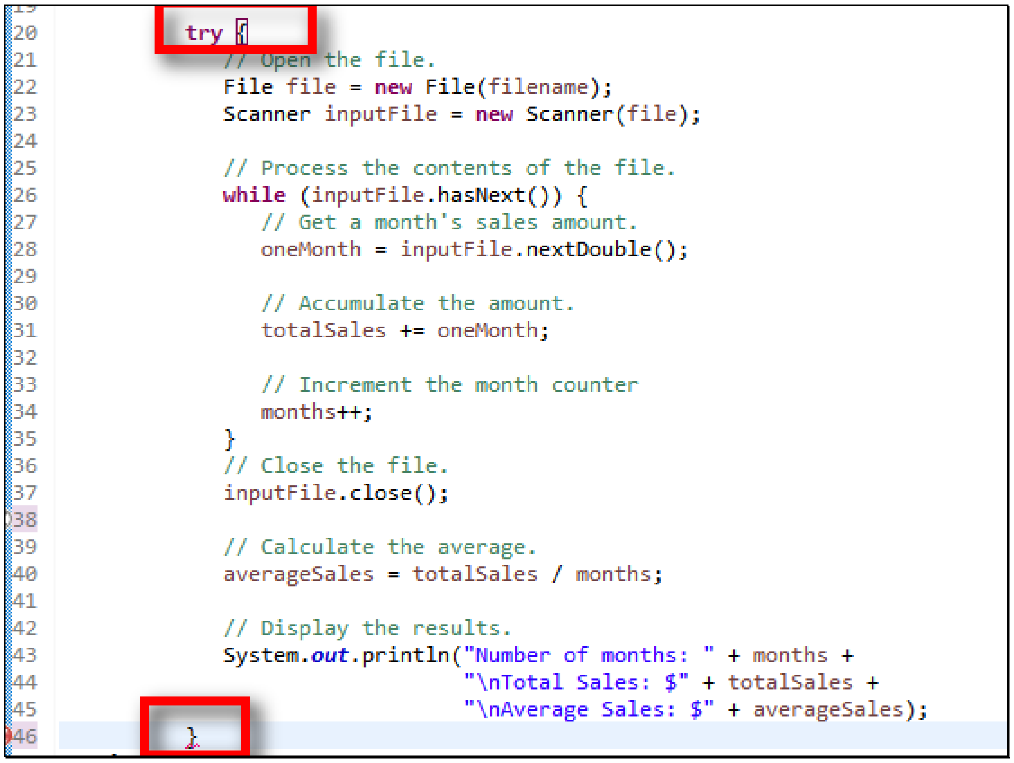
(1) Launch Eclipse. Create a new package to hold our source file. Name the package as **edu.unlv.mis768.labwork13.**

(2) Download **13_lab_files.zip** from WebCampus. Extract the zip file and then import the .java files into the package.

2. Add Exception Handler

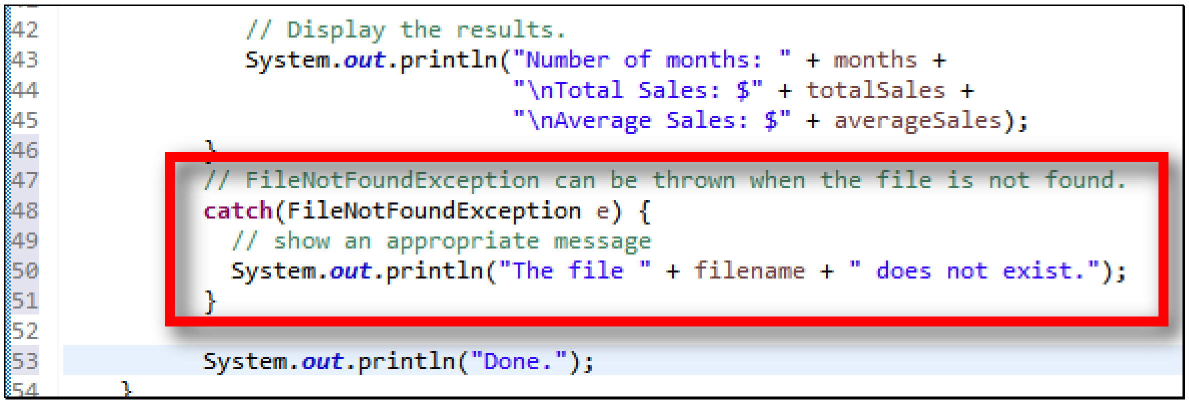
(3) Please open **SalesReport.java**. In this program, a file contains sales data will be read and the sales average will be calculated. However, it does not handle potential exceptions.

Please add the try statement around the file operation statements.



```
        try {
            // Open the file.
            File file = new File(filename);
            Scanner inputFile = new Scanner(file);

            // Process the contents of the file.
            while (inputFile.hasNext()) {
               // Get a month's sales amount.
               oneMonth = inputFile.nextDouble();

               // Accumulate the amount.
               totalSales += oneMonth;

               // Increment the month counter
               months++;
            }
            // Close the file.
            inputFile.close();

            // Calculate the average.
            averageSales = totalSales / months;

            // Display the results.
            System.out.println("Number of months: " + months +
                               "\nTotal Sales: $" + totalSales +
                               "\nAverage Sales: $" + averageSales);
        }
```

(4) Then, add the catch statement to handle the exception. The exception that could be thrown from the try block is a **FileNotFoundException.**



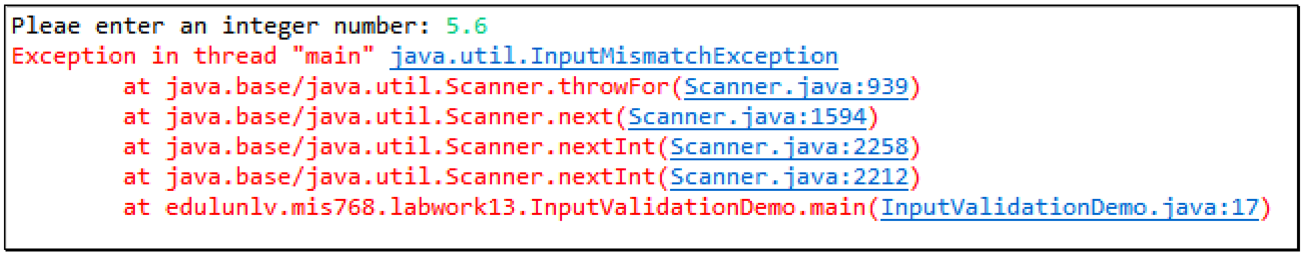
```
42              // Display the results.
43              System.out.println("Number of months: " + months +
44                                 "\nTotal Sales: $" + totalSales +
45                                 "\nAverage Sales: $" + averageSales);
46          }
47          // FileNotFoundException can be thrown when the file is not found.
48          catch(FileNotFoundException e) {
49            // show an appropriate message
50            System.out.println("The file " + filename + " does not exist.");
51          }
52
53          System.out.println("Done.");
54      }
```

(5) When you run and test the program, you will see that the program does not crash even if the file cannot be found.

Also, the code after the try-catch statement (line 53) will continue to be executed.

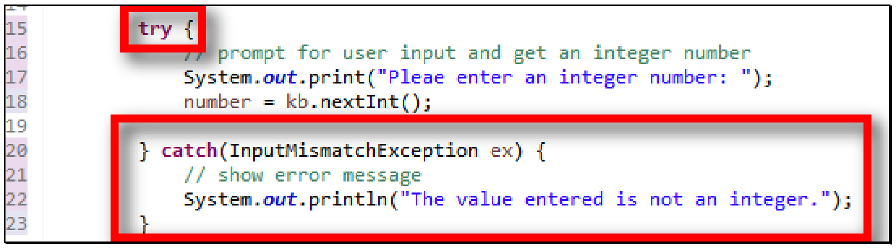
3. Exception Handling for Input Values

(6) The **InputValidationDemo** program needs to get an integer value from the user. If the user enters a float or a string value, it will throw an `InputMismatchException.`



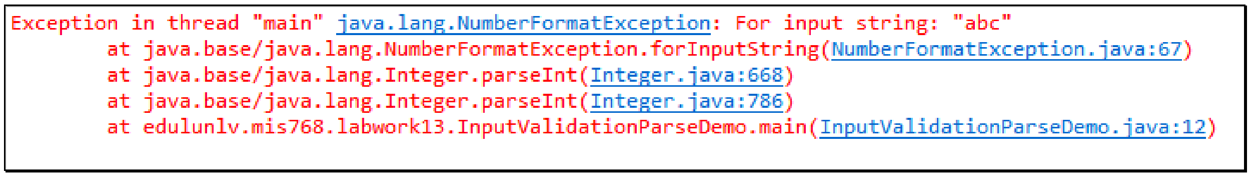
```
Pleae enter an integer number: 5.6
Exception in thread "main" java.util.InputMismatchException
	at java.base/java.util.Scanner.throwFor(Scanner.java:939)
	at java.base/java.util.Scanner.next(Scanner.java:1594)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
	at edulunlv.mis768.labwork13.InputValidationDemo.main(InputValidationDemo.java:17)
```

(7) We can add a try-catch statement to handle it:



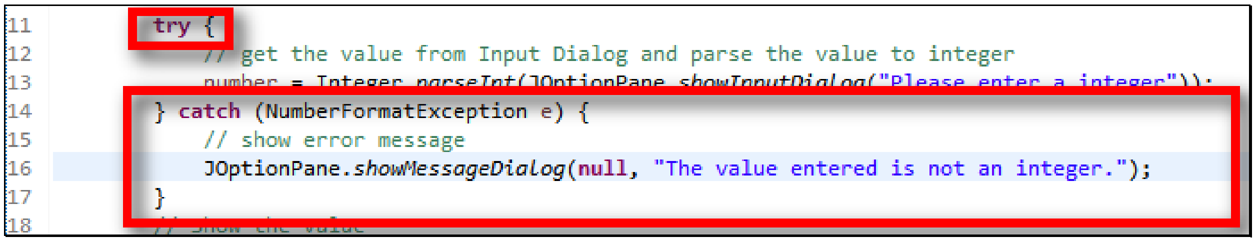
```
        try {
            // prompt for user input and get an integer number
            System.out.print("Pleae enter an integer number: ");
            number = kb.nextInt();

        } catch(InputMismatchException ex) {
            // show error message
            System.out.println("The value entered is not an integer.");
        }
```

(8) Similarly, the **InputValidationParseDemo** program gets a value from an input dialog and parses the string into an integer value. If the user enters a float or a string value, it will throw an `NumberFormatException.`



```
Exception in thread "main" java.lang.NumberFormatException: For input string: "abc"
	at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:67)
	at java.base/java.lang.Integer.parseInt(Integer.java:668)
	at java.base/java.lang.Integer.parseInt(Integer.java:786)
	at edulunlv.mis768.labwork13.InputValidationParseDemo.main(InputValidationParseDemo.java:12)
```

(9) Add a try-catch statement to handle the issue:



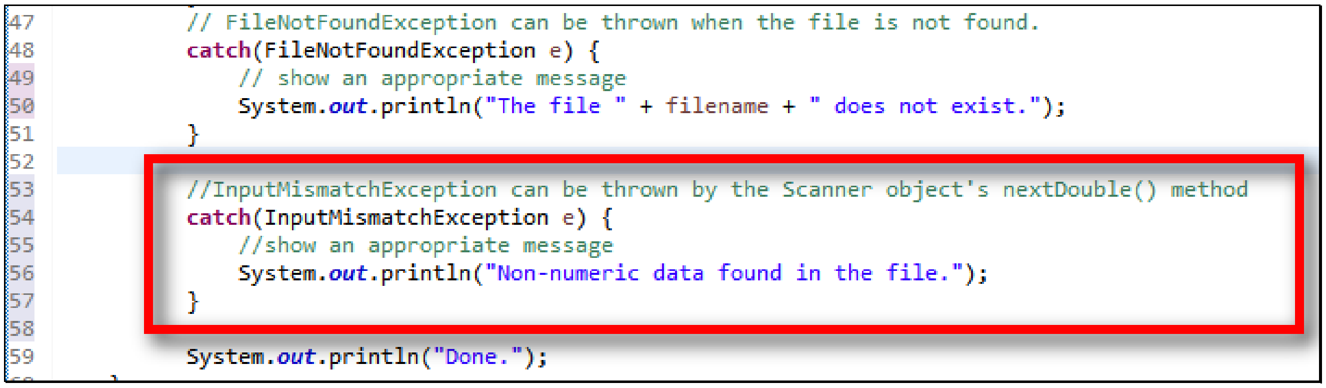
```
        try {
            // get the value from Input Dialog and parse the value to integer
            number = Integer.parseInt(JOptionPane.showInputDialog("Please enter a integer"));
        } catch (NumberFormatException e) {
            // show error message
            JOptionPane.showMessageDialog(null, "The value entered is not an integer.");
        }
        // show the value
```

4. Handle Multiple Exceptions

(10) The **SalesReport** program is subject to another issue: the data in the file are not numeric values. Another type of exception should also be handled.

(11) We can add another handler for **InputMismatchException**.

Add the following lines of code to the program:



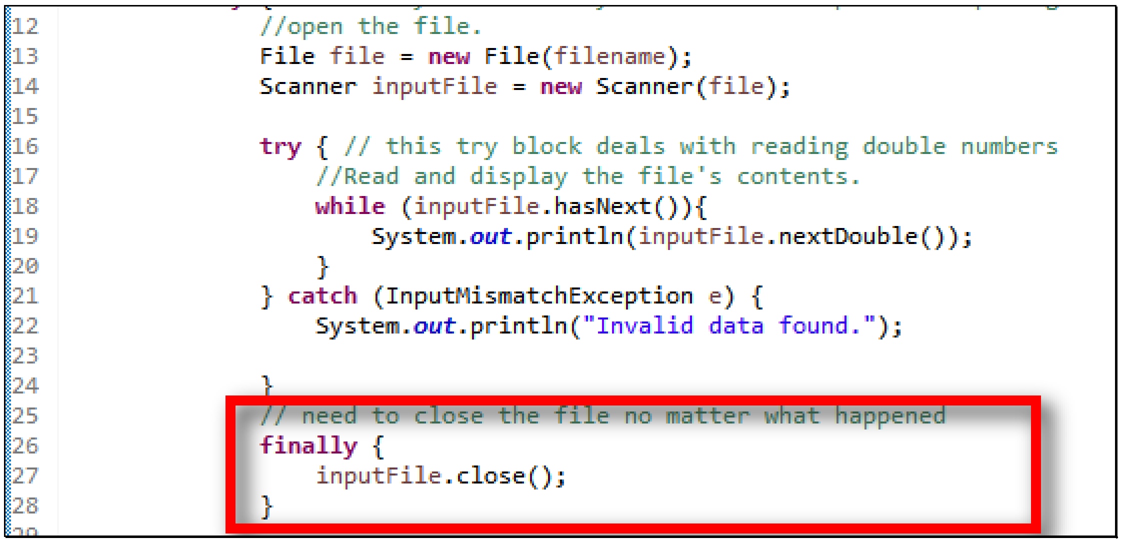
```
        // FileNotFoundException can be thrown when the file is not found.
        catch(FileNotFoundException e) {
            // show an appropriate message
            System.out.println("The file " + filename + " does not exist.");
        }

        //InputMismatchException can be thrown by the Scanner object's nextDouble() method
        catch(InputMismatchException e) {
            //show an appropriate message
            System.out.println("Non-numeric data found in the file.");
        }

        System.out.println("Done.");
```

(12) Change the file path of **SalesData.txt** (line 13) and run the program again.

5. finally Clause

(13) Open **SalesReportFinally.java**. In the program, we will use a **finally** clause to close the file.

Add the following lines of code to the program.



```
        //open the file.
        File file = new File(filename);
        Scanner inputFile = new Scanner(file);

        try { // this try block deals with reading double numbers
            //Read and display the file's contents.
            while (inputFile.hasNext()){
                System.out.println(inputFile.nextDouble());
            }
        } catch (InputMismatchException e) {
            System.out.println("Invalid data found.");

        }
        // need to close the file no matter what happened
        finally {
            inputFile.close();
        }
```

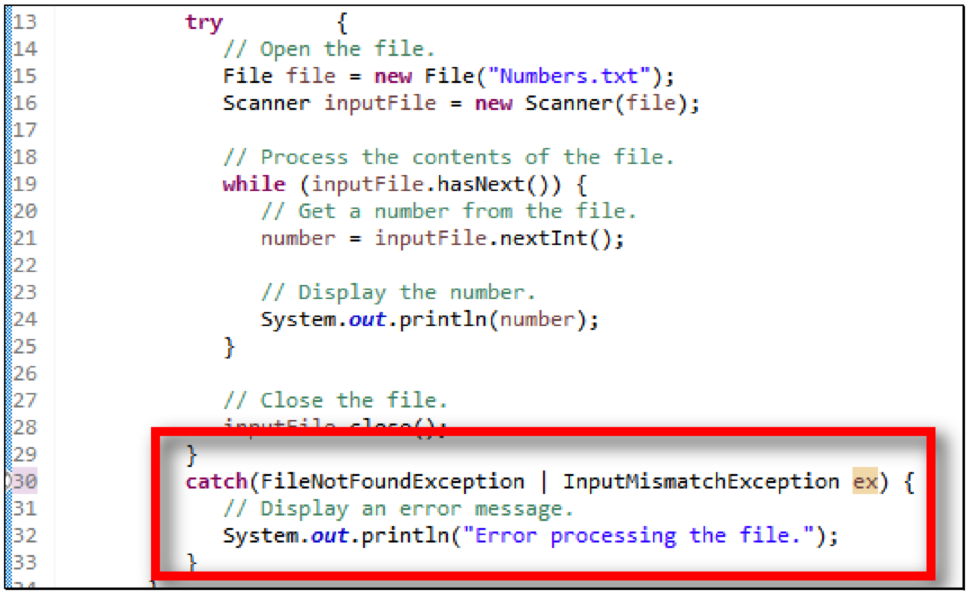
(14) Now run and test the program.

6. Multi-Catch

(15) Open **SalesReportMultiCatch.java**.

In this program, we intend to catch both **FileNotFoundException** and **InputMismatchException** with one catch clause.

(16) Please locate the **catch** statement and add the exceptions.



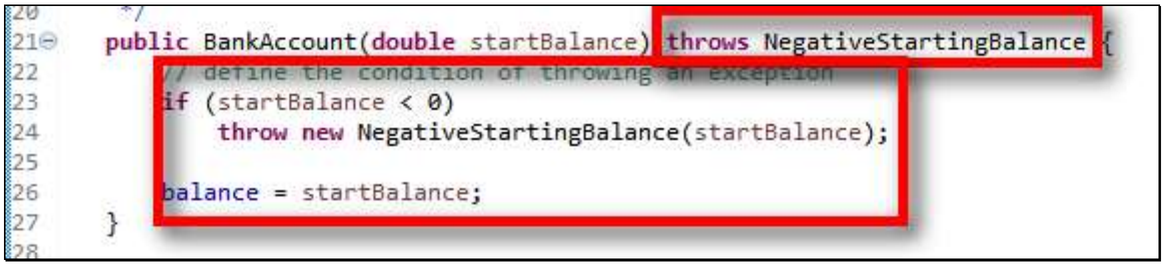
```
        try         {
           // Open the file.
           File file = new File("Numbers.txt");
           Scanner inputFile = new Scanner(file);

           // Process the contents of the file.
           while (inputFile.hasNext()) {
              // Get a number from the file.
              number = inputFile.nextInt();

              // Display the number.
              System.out.println(number);
           }

           // Close the file.
           inputFile.close();
        }
        catch(FileNotFoundException | InputMismatchException ex) {
           // Display an error message.
           System.out.println("Error processing the file.");
        }
```

7. Exception Classes

(17) **BankAccount** class stores and handles the data of a bank account.

The starting balance of the account cannot be negative. We need to throw and exception if the starting balance is negative.

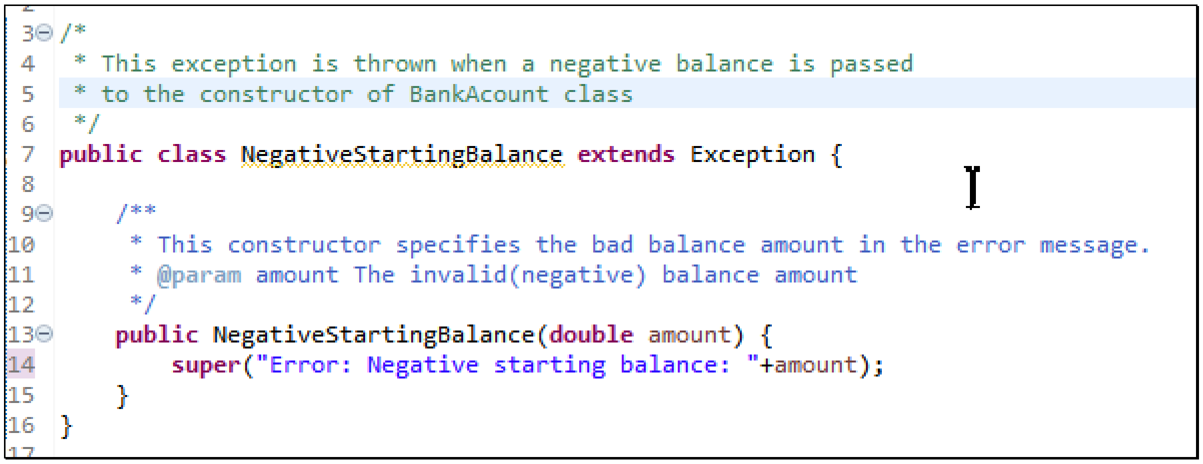
(18) Add the following clause to the constructor **BankAccount(double startBalance).**



```
     */
   public BankAccount(double startBalance) throws NegativeStartingBalance {
      // define the condition of throwing an exception
      if (startBalance < 0)
         throw new NegativeStartingBalance(startBalance);

      balance = startBalance;
   }
```

(19) We then need to create the **NegativeStaringBalence** class.

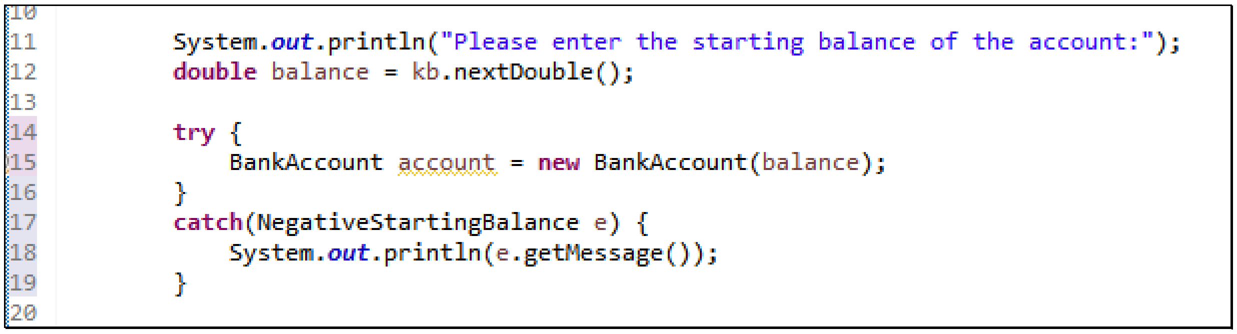
In the constructor, we assign the value of the message for this exception, so that it can be printed with the **getMessage**() method.



```
/*
 * This exception is thrown when a negative balance is passed
 * to the constructor of BankAcount class
 */
public class NegativeStartingBalance extends Exception {

    /**
     * This constructor specifies the bad balance amount in the error message.
     * @param amount The invalid(negative) balance amount
     */
    public NegativeStartingBalance(double amount) {
        super("Error: Negative starting balance: "+amount);
    }
}
```

(20) Now we can test the **BankAccount** class. Please open **BankAccountDemo** program.

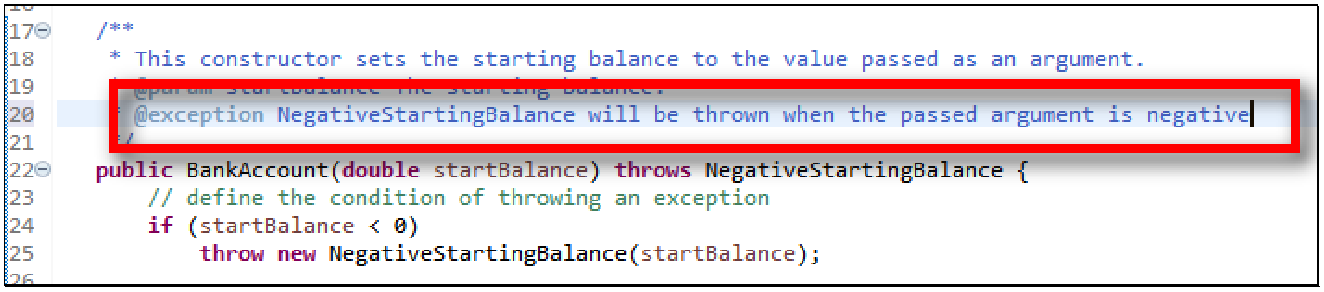
This program gets input from the user and use it to instantiate a **BankAccount** object. Because the constructor of **BankAcount** would throw an exception, we need to add try-catch statement.



```
        System.out.println("Please enter the starting balance of the account:");
        double balance = kb.nextDouble();

        try {
            BankAccount account = new BankAccount(balance);
        }
        catch(NegativeStartingBalance e) {
            System.out.println(e.getMessage());
        }
```

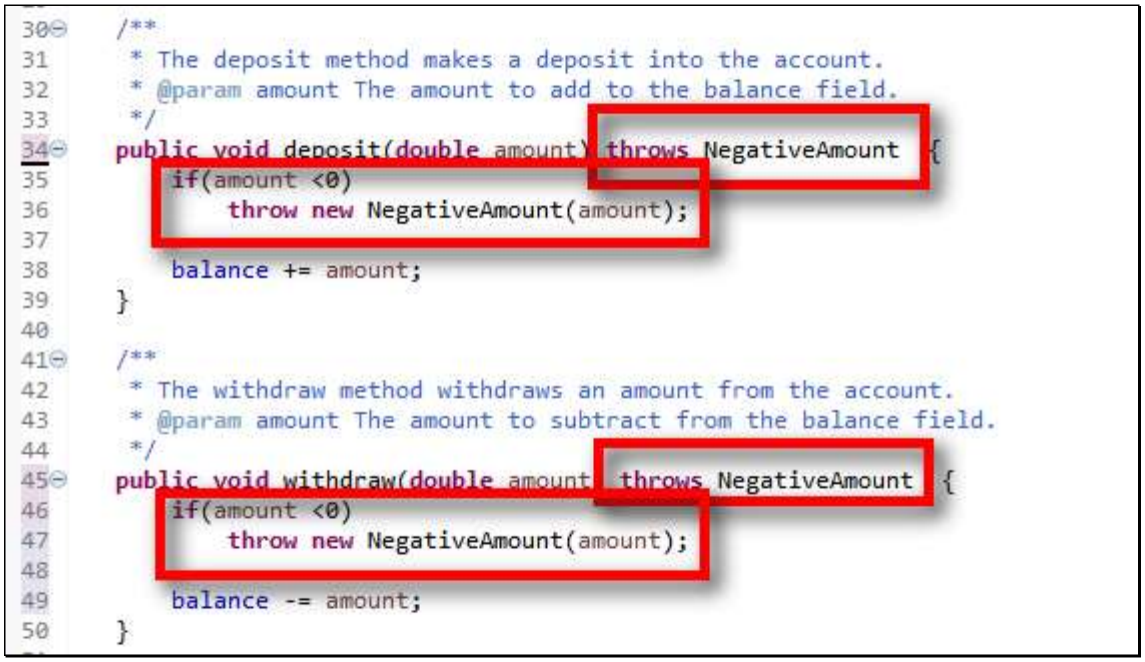
(21) You can now run and test the program.

(22) Switch back to **BankAccount.java** and add the description of the exception handling to the documentation of the constructor **BankAccount**.



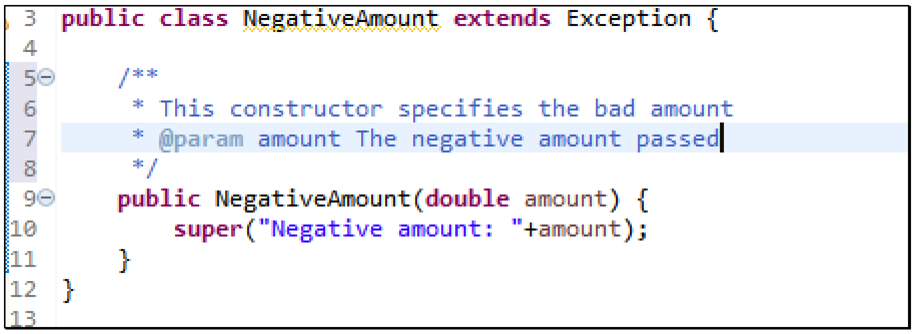
```
    /**
     * This constructor sets the starting balance to the value passed as an argument.
     * @param startBalance The starting balance.
     * @exception NegativeStartingBalance will be thrown when the passed argument is negative
     */
    public BankAccount(double startBalance) throws NegativeStartingBalance {
        // define the condition of throwing an exception
        if (startBalance < 0)
            throw new NegativeStartingBalance(startBalance);
```

(23) Similarly, we can add the **throws** clause to the **deposit**() and **withdraw**() method.



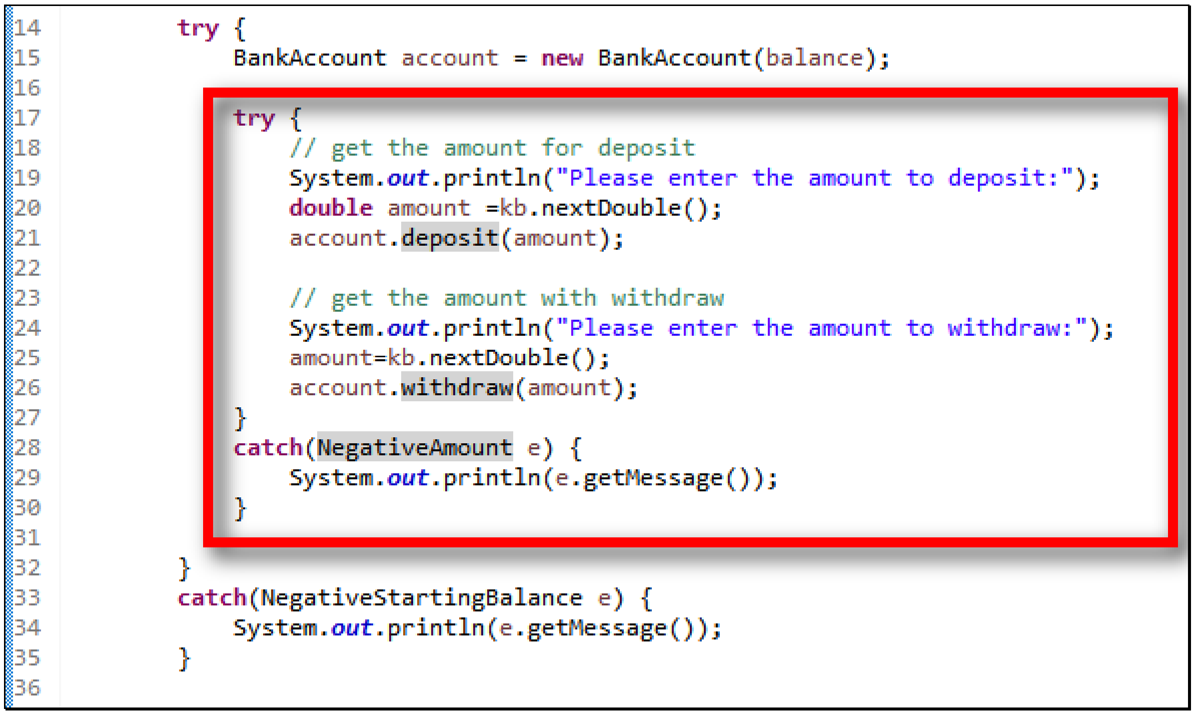
```
    /**
     * The deposit method makes a deposit into the account.
     * @param amount The amount to add to the balance field.
     */
    public void deposit(double amount) throws NegativeAmount {
        if(amount <0)
            throw new NegativeAmount(amount);

        balance += amount;
    }

    /**
     * The withdraw method withdraws an amount from the account.
     * @param amount The amount to subtract from the balance field.
     */
    public void withdraw(double amount) throws NegativeAmount {
        if(amount <0)
            throw new NegativeAmount(amount);

        balance -= amount;
    }
```

(24) Now, create NegativeAmount.java



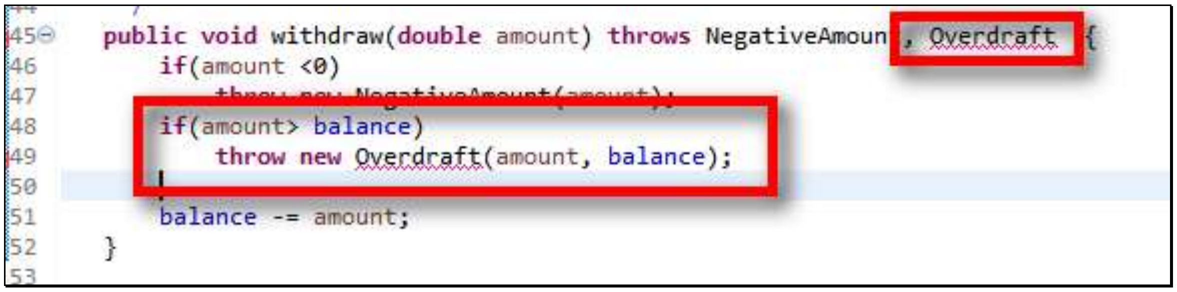
```
public class NegativeAmount extends Exception {

    /**
     * This constructor specifies the bad amount
     * @param amount The negative amount passed
     */
    public NegativeAmount(double amount) {
        super("Negative amount: "+amount);
    }
}
```

(25) Please open **BankAccountDemo2**. This program ask use to enter the amount for deposit and with draw. Please add the try-catch statement.



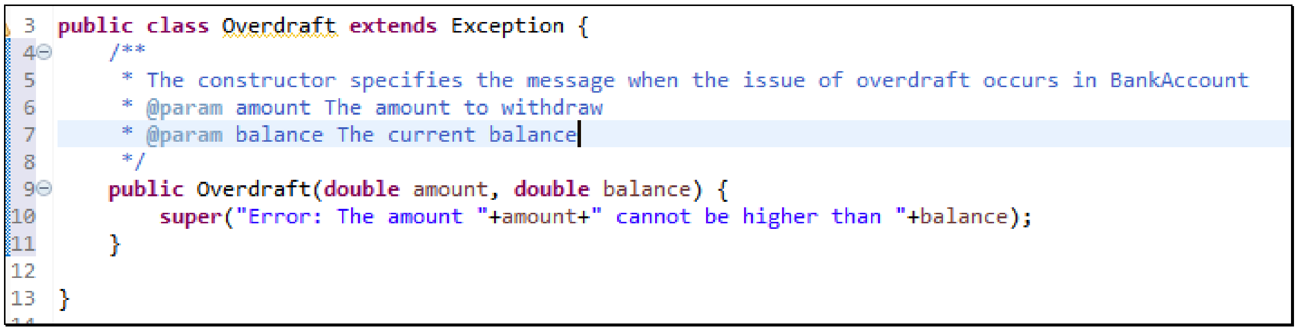
```java
        try {
            BankAccount account = new BankAccount(balance);

            try {
                // get the amount for deposit
                System.out.println("Please enter the amount to deposit:");
                double amount =kb.nextDouble();
                account.deposit(amount);

                // get the amount with withdraw
                System.out.println("Please enter the amount to withdraw:");
                amount=kb.nextDouble();
                account.withdraw(amount);
            }
            catch(NegativeAmount e) {
                System.out.println(e.getMessage());
            }

        }
        catch(NegativeStartingBalance e) {
            System.out.println(e.getMessage());
        }
```

(26) We can also add the **Overdraft** exception to the **withdraw()** method.



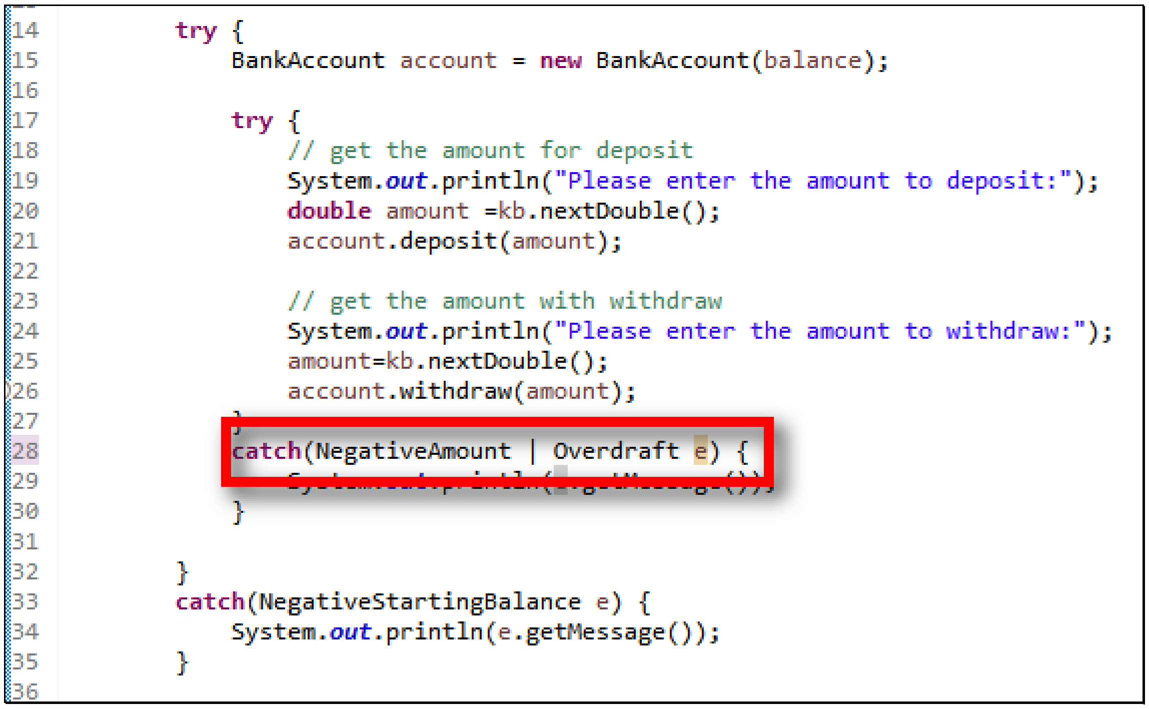
```java
    public void withdraw(double amount) throws NegativeAmoun, Overdraft {
        if(amount <0)
            throw new NegativeAmount(amount);
        if(amount> balance)
            throw new Overdraft(amount, balance);

        balance -= amount;
    }
```

(27) Create **Overdraft.java**



```java
public class Overdraft extends Exception {
    /**
     * The constructor specifies the message when the issue of overdraft occurs in BankAccount
     * @param amount The amount to withdraw
     * @param balance The current balance
     */
    public Overdraft(double amount, double balance) {
        super("Error: The amount "+amount+" cannot be higher than "+balance);
    }

}
```

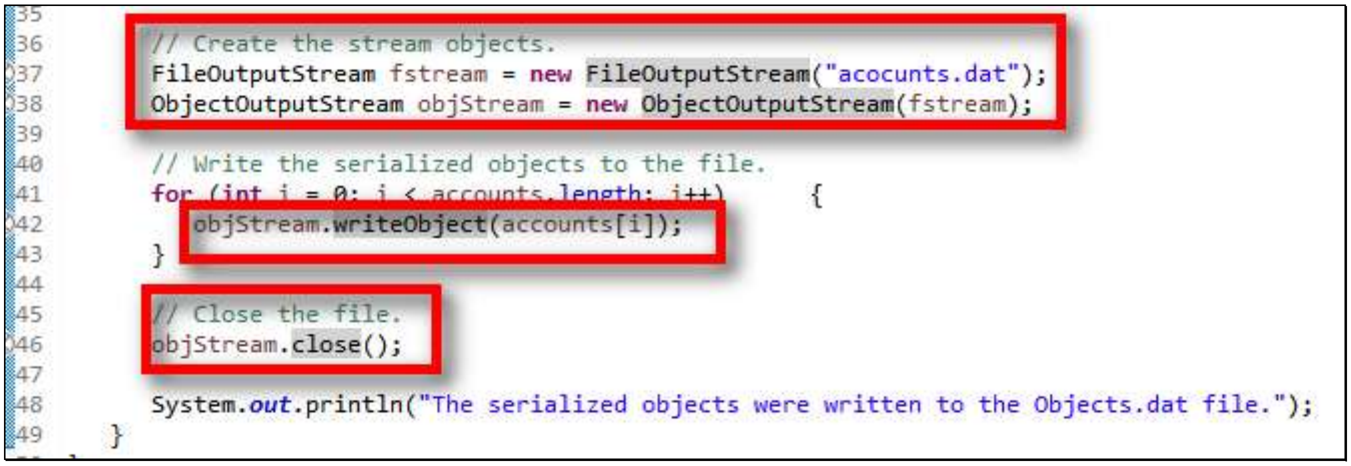
(28) Now back to the **AccountDemo2.java** to add the **Overdraft** exception.



```
        try {
            BankAccount account = new BankAccount(balance);

            try {
                // get the amount for deposit
                System.out.println("Please enter the amount to deposit:");
                double amount =kb.nextDouble();
                account.deposit(amount);

                // get the amount with withdraw
                System.out.println("Please enter the amount to withdraw:");
                amount=kb.nextDouble();
                account.withdraw(amount);
            }
            catch(NegativeAmount | Overdraft e) {
                System.out.println(e.getMessage());
            }

        }
        catch(NegativeStartingBalance e) {
            System.out.println(e.getMessage());
        }
```

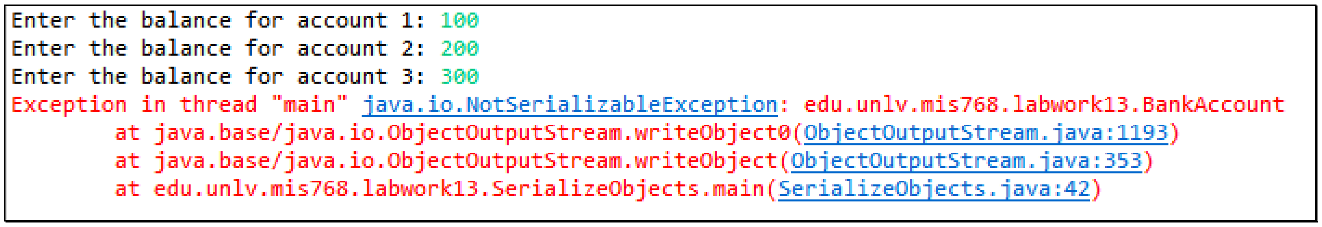
8. Serialize Objects

(29) We will try to create an array on **BankAccount** and write the objects to a file. Open **SerializeObjects.java** and complete the program.



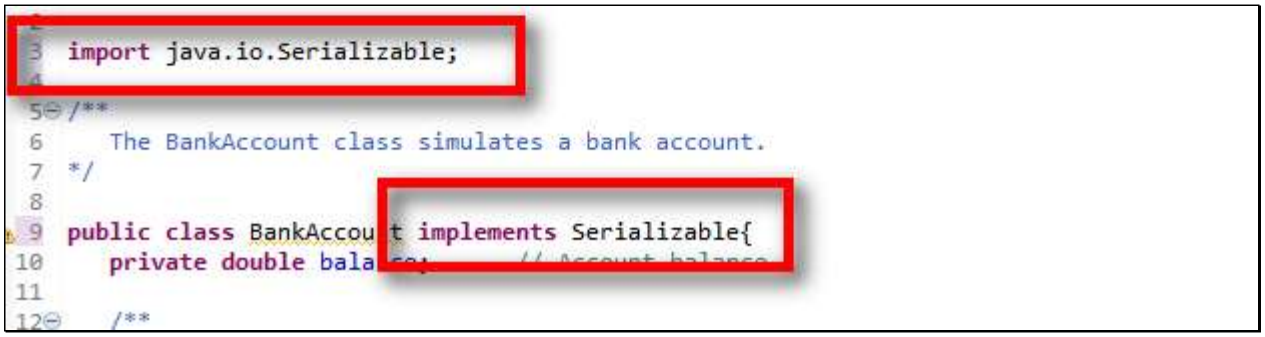
```
        // Create the stream objects.
        FileOutputStream fstream = new FileOutputStream("acocunts.dat");
        ObjectOutputStream objStream = new ObjectOutputStream(fstream);

        // Write the serialized objects to the file.
        for (int i = 0; i < accounts.length; i++)      {
            objStream.writeObject(accounts[i]);
        }

        // Close the file.
        objStream.close();

        System.out.println("The serialized objects were written to the Objects.dat file.");
    }
```

(30) When you run this program, you will see errors, because **BankAccount** did not implement the **Serializable** interface.



```
Enter the balance for account 1: 100
Enter the balance for account 2: 200
Enter the balance for account 3: 300
Exception in thread "main" java.io.NotSerializableException: edu.unlv.mis768.labwork13.BankAccount
	at java.base/java.io.ObjectOutputStream.writeObject0(ObjectOutputStream.java:1193)
	at java.base/java.io.ObjectOutputStream.writeObject(ObjectOutputStream.java:353)
	at edu.unlv.mis768.labwork13.SerializeObjects.main(SerializeObjects.java:42)
```

(31) To resolve that problem, open **BankAccount** and add implementation to the class header (and import the Serializable interface as well).



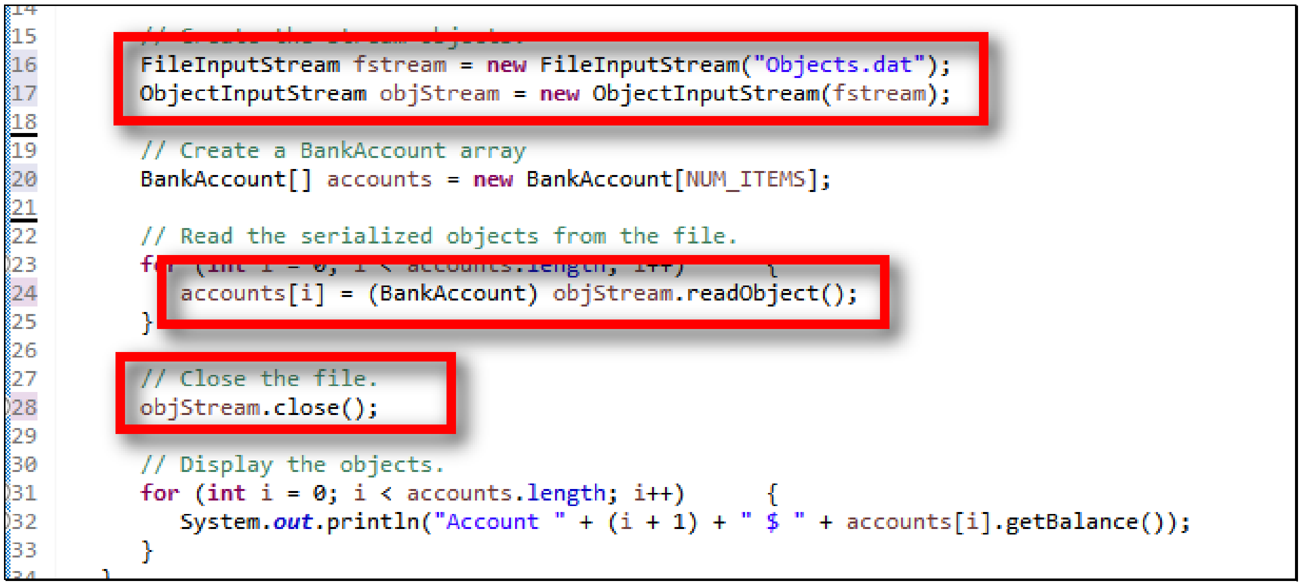
```
import java.io.Serializable;

/**
   The BankAccount class simulates a bank account.
*/

public class BankAccount implements Serializable{
   private double balance;       // Account balance

   /**
```

(32) Now when you run **SerializeObjects.java**, an array of **BankAccount** is written to the file.

9. Deserialize Objects

(33) On the other hand, **DeserializeObjects.java** reads the object data from a file.

Please complete the program.



```
      // Create the stream objects.
      FileInputStream fstream = new FileInputStream("Objects.dat");
      ObjectInputStream objStream = new ObjectInputStream(fstream);

      // Create a BankAccount array
      BankAccount[] accounts = new BankAccount[NUM_ITEMS];

      // Read the serialized objects from the file.
      for (int i = 0; i < accounts.length; i++)      {
         accounts[i] = (BankAccount) objStream.readObject();
      }

      // Close the file.
      objStream.close();

      // Display the objects.
      for (int i = 0; i < accounts.length; i++)      {
         System.out.println("Account " + (i + 1) + " $ " + accounts[i].getBalance());
      }
   }
```