Exercises

SE116 - Introduction to Programming II

Interface: Polymorphism

(**CarbonFootprint**)
Using interfaces, you can specify similar behaviors for possibly disparate classes.

Governments and companies worldwide are becoming increasingly concerned with carbon footprints (annual releases of carbon dioxide into the atmosphere) from buildings burning various types of fuels for heat, vehicles burning fuels for power, and the like. Many scientists blame these greenhouse gases for the phenomenon
called global warming.

- Create three small classes unrelated by inheritance—classes `Building`, `Car` and `Bicycle`. Give each class some unique appropriate attributes and behaviors that it does not have in common with other classes.
- Write an interface `CarbonFootprint` with a `getCarbonFootprint` method. Have each of your classes implement that interface, so that its `getCarbonFootprint` method calculates an appropriate carbon footprint for that class.
- Write an application that creates objects of each of the three classes, places references to those objects in `ArrayList<CarbonFootprint>`, then iterates through the Array-List, polymorphically invoking each object's `getCarbonFootprint` method. For each object, print some identifying information and the object's carbon footprint.

```java
public interface CarbonFootprint {
   void GetCarbonFootprint();
}
```

```java
public class Bicycle implements CarbonFootprint {
   public void GetCarbonFootprint() {
      System.out.println("Bicycle: 0");
   }
}
```

```java
public class Building implements CarbonFootprint {
   private int squareFeet;

   public Building(int squareFeet) {
      this.squareFeet = squareFeet;
   }

   // Simplified formula: Multiply the square footage by 50
   // for the wood frame, by 20 for the basement,
   // by 47 for the concrete, and 17 for the steel
   // Note: The website where we got this information no longer exists.
   public void GetCarbonFootprint() {
      System.out.printf("Building with %d square feet: %d\n",
         squareFeet, squareFeet * (50 + 20 + 47 + 17));
   }
}
```

```java
public class Car implements CarbonFootprint {
   private double gallons;

   public Car(double gallons) {
      this.gallons = gallons;
   }

   // one gallon of gas yields 20 pounds of CO2
   // http://www.enviroduck.com/carbon_footprint_calculations.php
   public void GetCarbonFootprint() {
      System.out.printf("Car that has used %.2f gallons of gas: %.2f\n",
         gallons, gallons * 20);
   }
}
```

```java
import java.util.ArrayList;
public class CarbonFootPrintTest {
   public static void main(String[] args) {
       ArrayList<CarbonFootprint> list = new ArrayList<>();

      // add elements to list
      list.add(new Bicycle());
      list.add(new Building(2500));
      list.add(new Car(10));

      // display carbon footprint of each object
      for (CarbonFootprint item : list) {
         item.GetCarbonFootprint();
      }
   }
}
```

```
Bicycle: 0
Building with 2500 square feet: 335000
Car that has used 10.00 gallons of gas: 200.00
```

Rethrowing Exceptions

Write a program that illustrates rethrowing an exception.

- Define methods `someMethod` and `someMethod2`. Method `someMethod2` should initially throw an exception.
- Method `someMethod` should call `someMethod2`, catch the exception and rethrow it.
- Call `someMethod` from method `main`, and catch the rethrown exception. Print the stack trace of this exception.

```java
public class RethrowException {

   public static void main(String[] args) {
      try { // call someMethod
         System.out.println("In main's try");
         someMethod();
      }
      catch (Exception exception) {
          System.out.println("In main's catch");
          System.err.printf("%s\n\n", exception.getMessage());
          exception.printStackTrace();
      }
   }

   // call someMethod2; rethrow Exceptions back to main
   public static void someMethod() throws Exception {
      try { // call someMethod2
         System.out.println("In someMethod's try");
         someMethod2();
      }
      catch (Exception exception2) {
         System.out.println("In someMethod's catch");
         System.out.println("The caught exception's message: "+ exception2.getMessage());
         throw exception2; // rethrow the Exception
      }
   }

   // throw Exception back to someMethod
   public static void someMethod2() throws Exception {
      System.out.println("In someMethod2's try");
      throw new Exception("Exception thrown in someMethod2");
   }
}
```

```
In main's try
In someMethod's try
In someMethod2's try
In someMethod's catch
The caught exception's message: Exception thrown in someMethod2
In main's catch
Exception thrown in someMethod2

java.lang.Exception: Exception thrown in someMethod2
        at RethrowException.someMethod2(RethrowException.java:32)
        at RethrowException.someMethod(RethrowException.java:20)
        at RethrowException.main(RethrowException.java:7)
```

RegexValidator

- Create a Java program that validates email addresses using regular expressions.
  - **Email Address:** Must follow the standard email format (e.g., user@example.com, user-user@example.com, user.123@example.com.uk, user_2@example.info, etc.).

```java
public class RegexValidator {

    // Validate email address
    public static boolean isValidEmail(String email) {
        // \\w matches any word character (equivalent to [a-zA-Z0-9_]
        // The ^ and $ characters in a regular expression are anchors that specify the start and end of the string, respectively.
        //Together, ^ and $ ensure that the entire string matches the pattern, not just a substring within it.
        String regex = "^[\\w-\\.]+@([\\w-]+\\.)+[a-zA-Z]{2,4}$";
        return email.matches(regex);
    }

    public static void main(String[] args) {
        // Test the validation methods
        String[] emails = {"user@example.com", "user.example.com", "user_us@example.com", "user.name@example.co.uk",
                "user.name@example.infof", "user@example", "user  e@gmail.com", "user?@g.com", "user@___.c", "user@user@com"};


        System.out.println("Email Validation:");
        for (String email : emails) {
            System.out.println(email + ": " + isValidEmail(email));
        }
    }
}
```

```
Email Validation:
user@example.com: true
user.example.com: false
user_us@example.com: true
user.name@example.co.uk: true
user.name@example.infof: false
user@example: false
user  e@gmail.com: false
user?@g.com: false
user@___.c: false
user@user@com: false
```

File Operations

- Create a Java program that reads a text file, counts the number of lines, words, and characters, and writes these statistics to another text file.
- Implement methods for reading a file, counting lines, words, and characters, and writing the results to a file.

```java
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class FileOperations {

    // Method to read a file and return its contents as a String
    public static String readFile(String fileName) throws IOException {
        StringBuilder content = new StringBuilder();
        try (BufferedReader br = new BufferedReader(new FileReader(fileName))) {
            String line;
            while ((line = br.readLine()) != null) {
                content.append(line).append("\n");
            }
        }
        return content.toString();
    }

    // Method to count lines in a String
    public static int countLines(String content) {
        return content.split("\n").length;
    }

    // Method to count words in a String
    public static int countWords(String content) {
        return content.split("\\s+").length;
    }

    // Method to count characters in a String
    public static int countCharacters(String content) {
        return content.length();
    }

    // Method to write results to a file
    public static void writeFile(String fileName, String content) throws IOException {
        try (BufferedWriter bw = new BufferedWriter(new FileWriter(fileName))) {
            bw.write(content);
        }
    }
```

```
    public static void main(String[] args) {
        String inputFileName = "input.txt";
        String outputFileName = "output.txt";

        try {
            // Read the file
            String content = readFile(inputFileName);

            // Count lines, words, and characters
            int lineCount = countLines(content);
            int wordCount = countWords(content);
            int charCount = countCharacters(content);

            // Prepare the result content
            String result = "Lines: " + lineCount + "\n" +
                            "Words: " + wordCount + "\n" +
                            "Characters: " + charCount;

            // Write the results to the output file
            writeFile(outputFileName, result);

            System.out.println("File processed successfully. Check " + outputFileName + " for results.");

        } catch (IOException e) {
            System.err.println("An error occurred: " + e.getMessage());
        }
    }
}
```