CS 501B – Introduction to JAVA Programming

Fall 2022 Semester

Due: 10/28/2022 Friday at 11:59 PM

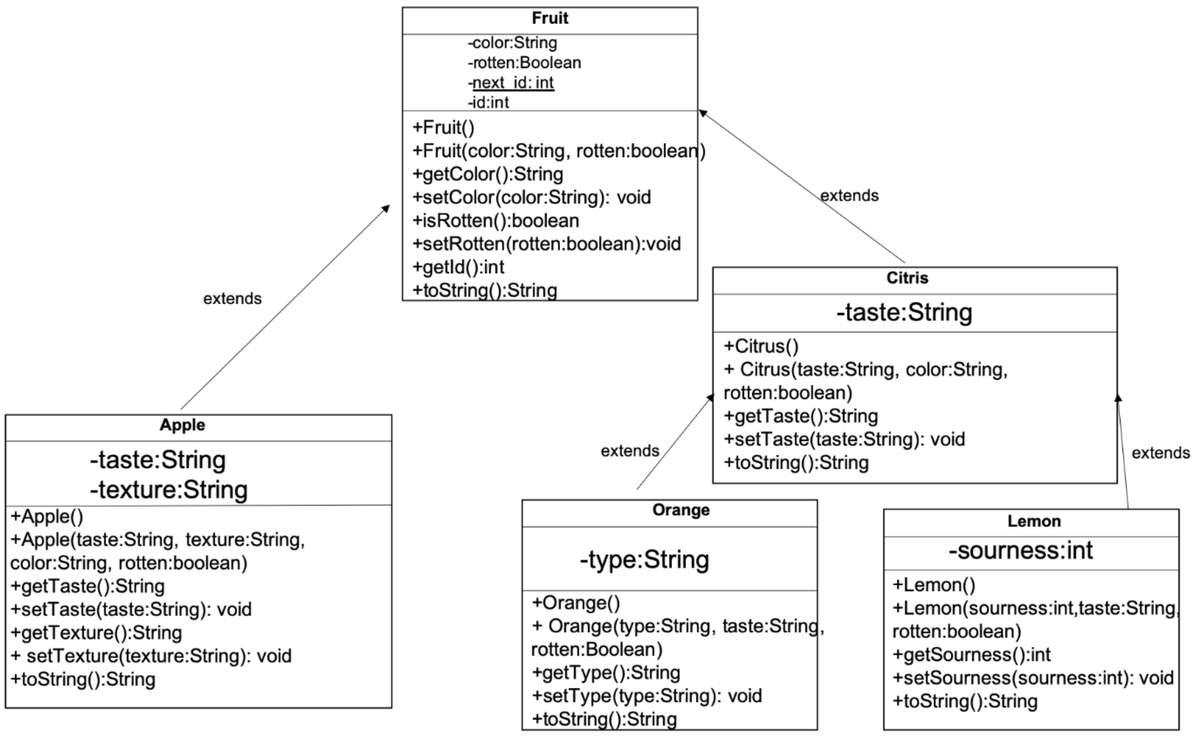
Instructions:

1) You are not allowed to use any package/library unless stated otherwise.
2) Please comment your name and CWID in the first two lines of every .java file.
3) You are required to zip the all the .java files as FirstName_LastName_Assignment#.zip (Ex: Suhas_HS_Assignment3.zip) with no sub-directories. There should be only text files with a .java extension in the zip file.
4) Do not include packages statements in your code. Some IDEs automatically structure the project into packages (usually folders or directories). If your IDE uses package statement remove or comment them out before bundliing the files into the .zip submission file.
5) This assignment covers topics from weeks 6.
6) Students are not allowed to collaborate with classmates and any other people outside. All work must be done individually. Any work having evidence of showing academic dishonesty violation are subjected to give zero for the assignment.

Part I: Inheritance and Polymorphism (40 Points)

1. Below is a UML hierarchy of Fruits. Your first step is to implement this hierarchy and each class should be in separate java file (ex - Fruit.java, Apple.java and so on). In constructors with args, you should use super() constructor with args to set the values in superclasses.

   toString() should contain the name of the class and all of the fields/data for that class. You can decide whether this will work by a subclass calling the toString method of the superclass and including that in the subclass toString result or if the toString method accesses all the data/fields in the object itself.


Fruit
-color:String
-rotten:Boolean
-next_id: int
-id:int
+Fruit()
+Fruit(color:String, rotten:boolean)
+getColor():String
+setColor(color:String): void
+isRotten():boolean
+setRotten(rotten:boolean):void
+getId():int
+toString():String
extends
extends
Citris
-taste:String
+Citrus()
+ Citrus(taste:String, color:String, rotten:boolean)
+getTaste():String
+setTaste(taste:String): void
+toString():String
Apple
-taste:String
-texture:String
+Apple()
+Apple(taste:String, texture:String, color:String, rotten:boolean)
+getTaste():String
+setTaste(taste:String): void
+getTexture():String
+ setTexture(texture:String): void
+toString():String
extends
extends
Orange
-type:String
+Orange()
+ Orange(type:String, taste:String, rotten:Boolean)
+getType():String
+setType(type:String): void
+toString():String
Lemon
-sourness:int
+Lemon()
+Lemon(sourness:int,taste:String, rotten:boolean)
+getSourness():int
+setSourness(sourness:int): void
+toString():String

Orange objects are always "orange" in color. Lemon objects are always "yellow" in color.

This uses the static "id" pattern. Every creation of a new Fruit object (of any kind) should generate a new and unique id from the static next_id field, which is stored in the id field and accessible by getId

2. Now that you've implemented this, write equals methods for all the classes. The equals method should take an Object as an argument and return true if the field values of the class and its superclass(es) are equal.

3. Create an Arraylist of Fruit objects.

    a. **Create 8 Fruit objects:**

    1 non-rotten red Apple with a crisp texture and sweet taste
    2 rotten green Apple objects with a soft texture and tart taste
    3 non-rotten Lemon objects with a sour taste. "sourness" should be a random integer from 0-100 for each object
    2 rotten Orange objects of type "mandarin" with a sweet taste.

    Put those objects in the Arraylist. They should all be able to be added to the same Arraylist, regardless of their subclass. The Arraylist MUST be parameterized to accept Fruit objects. It should accept all Fruit objects but should not accept non-fruit objects. For example, this:

    fruitArrayList.add(**new** Object());

should not compile

b. Print out the average sourness of all the Lemon objects in the Arraylist. You have to do this by looping through the array list, finding the Lemon objects, and their sourness

c. **Remove the matching objects:** Retain the 1st rotten green Apple object in an Apple variable. The goal is ultimately to remove all of the Apple objects in the Arraylist that match this variable:

   To start: Loop through the Arraylist and print out (using toString) which objects in the Apple object is equal (in value) to the Apple object in your variable.

   Also print out which object in the Arraylist is the **same object** as the one in your variable.

   You must figure out how to remove all the matching objects from the Arraylist. There is no one correct way to do this. But there are incorrect ways.

d. **Print out the remaining objects:** Loop through the Arraylist again and print out (using toString) all the remaining objects in the Arraylist.
e. Perform all the above operations in **public static void main(String[] args)** of class **FruitTest** of file name (**FruitTest.java**).

   FruitTest.java

```
class FruitTest {

   public static void main(String[] args) {

      // All the above operations

   }

}
```

Part 2: Exception Handling and Text 1/0 (60 Points)

1. Take the following code, ListOfNumbers.java:

```
import java.io.*;

import java.util.List;

import java.util.ArrayList;
```

```
public class ListOfNumbers {
    private List<Integer> list;
    private static final int SIZE = 10;

    public ListOfNumbers () {
        list = new ArrayList<Integer>(SIZE);
        for (int i = 0; i < SIZE; i++)
            list.add(new Integer(i));
    }
    public void writeList() {
        PrintWriter out = null;
        try {
            System.out.println("Entering try statement");
            out = new PrintWriter(new FileWriter("outFile.txt"));
            for (int i = 0; i < SIZE; i++)
                out.println("Value at: " + i + " = " + list.get(i));
        } catch (IndexOutOfBoundsException e) {
            System.err.println("Caught IndexOutOfBoundsException: " +
                                 e.getMessage());
        } catch (IOException e) {
            System.err.println("Caught IOException: " + e.getMessage());
        } finally {
            if (out != null) {
                System.out.println("Closing PrintWriter");
                out.close();
            } else {
                System.out.println("PrintWriter not open");
            }
        }
    }
}
```

Add a readList method to ListOfNumbers.java. This method should read in int values from a file, print each value, and append them to the end of the ArrayList called list. You should catch all appropriate errors. You will read from the text file numberfile.txt.

The writeList method writes out the contents of the ArrayList to outFile.txt.

2. (Write/read data) Write a program to create a file named Java_Program.txt if it does not exist. Write 100 integers created randomly into the file using text I/O. Integers are separated by spaces in the file. Read the data back from the file and display the data in increasing order.

   WRData.java

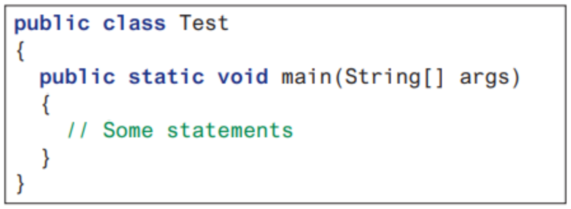
   ```
   class WRData {
      public static void main(String[] args) {
            //  operations
      } }
   ```

3. (ArrayIndexOutOfBoundsException) Write a program that meets the following requirements:
   ■ Creates an array with 100 randomly chosen integers.
   ■ Prompts the user to enter the index of the array, then displays the corresponding element value. If the specified index is out of bounds, display the message "Out of Bounds"

   AIOBETest.java

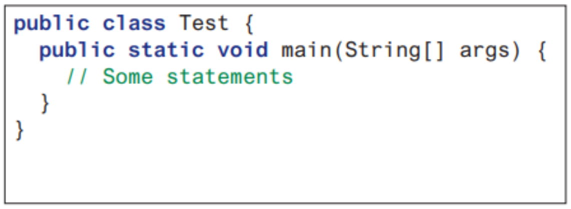
   ```
   class AIOBETest {
      public static void main(String[] args) {
            //  operations
      } }
   ```

4. (Reformat Java source code) Write a program that converts the Java source code from the next-line brace style to the end-of-line brace style. For example, the following Java source in (a) uses the next-line brace style. Your program converts it to the end-of-line brace style in (b).



```java
public class Test
{
  public static void main(String[] args)
  {
    // Some statements
  }
}
```

(a) Next-line brace style



```java
public class Test {
  public static void main(String[] args) {
    // Some statements
  }
}
```

(b) End-of-line brace style

Your program can be invoked from the command line with the Java source-code file as the argument. It converts the Java source code to a new format. For example, the following command converts the Java source-code file **Test.java** to the end-of-line brace style.

java ReformatJavaSourceCode Test.java

ReformatJavaSourceCode.java

```java
class ReformatJavaSourceCode {
   public static void main(String[] args) {
         //  operations
   } }
```