CSCI 202: Object-Oriented Programming

Homework 3 Part 1 - Due along with Part 2, which will be assigned at a future point. The due date for this assignment will be given along with Part 2 when it is assigned.

Resources:

- Java API index: `https://docs.oracle.com/en/java/javase/17/docs/api/index-files/index-1.html`

Part 1

Your task in this homework is to create a program that takes various fruit (represented as objects) and adds them to a juicer to make juice. In this part you will create several classes for the various different kinds of fruit and test out these classes. In the next part (which will be assigned later) you will finish the program.

Download and unzip Hw3.zip onto your local computer. Then open up the unzipped project with IntelliJ.

At this point you will see errors in the the project. This is because the program is not complete. It will be your task to complete the program.

Under src you will see several packages.

- Package `copy` contains interface `Copyable`, which defines an interface to be used for creating a deep copy of objects using the alternative approach to the clone method described in class.
- Package `juicer` will contains facilities for the juicer and fruit.
- Package `test` contains TestFruit.java, which will be used for testing the various fruit classes.

Note: There may be tasks in Part 1 that we haven't gone over yet in class.
Skip those parts for now and add them once we go over them.

Part 1a: Creating the `Fruit` class

In Fruit.java add an abstract class called `Fruit` that implements `Cloneable` and `Copyable`. In this class add the following:

- The private data fields
```
private double mass;
private boolean isJuiceRemoved;
```
- The protected constructor `Fruit(double theMass)` which initializes `mass` to `theMass` and initializes `isJuiceRemoved` to `false`. If `theMass` is less than or equal to zero, then `mass` must be set to 1.
- A protected copy constructor. This will be called from subclasses of `Fruit`.
- The public get method `getMass()` for `mass`.
- The protected set method `setMass(double value)` for `mass`. If `value` is less than or equal to zero, then `mass` must be set to 1.
- The public get method `isJuiceExtracted()` for `isJuiceRemoved`.
- The method `protected abstract double juiceRatio();`

- The method

  ```
  public double extractJuice()
  {
      double liquidMass = amountJuice();

      if (!isJuiceRemoved)
      {
          isJuiceRemoved = true;
          mass -= liquidMass;
      }

      return liquidMass;
  }
  ```

- The method

  ```
  public double amountJuice()
  {
      if (isJuiceRemoved) return 0.0;

      return mass * juiceRatio();
  }
  ```

- Override the `equals` method to do a deep comparison.
- Override the `hashCode` method using the procedure described in lecture.
- Override the `clone` method to make a deep copy.
- Override the `toString` method as follows:

  ```
  @Override
  public String toString()
  {
      return "\tmass = " + mass +
             "\n\tisJuiceExtracted = " + isJuiceRemoved + "\n";
  }
  ```

The abstract method `juiceRatio()` is to be overridden in subclasses. Its purpose is to return the ratio of mass that can be extracted as juice. For example: if 75% of the fruit can be extracted as juice, then this method would return 0.75.

The method `extractJuice()` extracts the juice from the fruit. The amount of extracted juice is returned as a `double`. The remaining mass that cannot be extracted as juice is set as the new mass of the fruit.

Method `amountJuice()` returns the amount of juice (rather than the ratio) that can be extracted without actually extracting it.

Part 1b: Creating subclasses of class `Fruit`

The `Apple` class (file Apple.java): Do not modify this file.

The `Apple` class extends class `Fruit`. An `Apple` object can be dried, but by default is is not. Notice that `Apple` overrides method `juiceRatio()` to return different values depending on if the apple is dry or not.

The `Orange` class (file Orange.java): Do not modify this file.

The `Orange` class also extends class `Fruit`. It overrides `juiceRatio()` to return 0.87 (i.e., 87% of an orange can be turned into juice).

The `Banana` class (file Banana.java):

In package `juicer` create a class called `Banana` similar to class `Orange`. Override method `juiceRatio()` to return 0.79. (i.e., 79% of a banana can be turned into juice). Do **not** change the access modifier of this method to public, but keep it protected. Add a constructor for initializing its mass. Also, add a copy constructor and overload method `copy()` to be a wrapper method around the copy constructor. Overload the `equals()` method to do a deep comparison. Finally, overload the `toString()` method as follows:

```
@Override
public String toString()
{
    return "Banana:\n" + super.toString();
}
```

The `Strawberry` class (file Strawberry.java):

In package `juicer` create a class called `Strawberry` similar to class `Orange`. Override method `juiceRatio()` to return 0.92. (i.e., 92% of a strawberry can be turned into juice). Do **not** change the access modifier of this method to public, but keep it protected. Add a constructor for initializing its mass. Also, add a copy constructor and overload method `copy()` to be a wrapper method around the copy constructor. Overload the `equals()` method to do a deep comparison. Finally, overload the `toString()` method as follows:

```
@Override
public String toString()
{
    return "Strawberry:\n" + super.toString();
}
```

Part 1c: Testing the fruit classes.

In package `test` you will find TestFruit.java. In here add code to the `main` method to test the various fruit classes (`Apple`, `Orange`, `Banana` and `Strawberry`). Create several instances of these classes and test each of their methods. Make sure these methods work as expected. To run this file, right click on the green run triangle to the left of the `main` method and then select Run 'TestFruit.main()'.

When you are done, chill. When Part 2 is assigned you will be given instructions on what to do to finish Homework 3 and a due date for Homework 3 will be given then.