FruitNinja Design Section Mini-Assignment

Due Date: Monday, September 30th, 11:59pm

Instructions

- Read the assignment handout carefully and play around with the demo before you start these questions.
- Hand in your answers via email to your section TAs before the due date. Any handins after the due date will not be accepted.
- Please bring either a printed or electronic copy to your scheduled section time. You can print course material for free in the Sunlab!

Questions

1. Name the objects that need to be modeled in this program. Which classes will you **not** need to write because they are provided in the support code? Which classes will you need to write yourself?

2. Read the following charts on containment and inheritance diagrams. We will discuss this in more detail in section, but for now, just understand the following: containment and inheritance diagrams are a great way of detailing program design before you even begin to code. By creating these diagrams, you can have a firm understanding of the relationships between classes in your program and avoid rewrites and unwanted complexity later. The following two tables describe how to create these diagrams:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpleClass</td>
<td>Concrete Class Box</td>
<td>Classes are represented by a box with its name. “Concrete classes” are classes that can be instantiated.</td>
</tr>
<tr>
<td>AbstractClass</td>
<td>Abstract Class Box</td>
<td>Abstract classes are represented by a box with its name in italics.</td>
</tr>
<tr>
<td>ClassB</td>
<td>Unfilled Diamond Arrow</td>
<td>Shows that a <strong>single</strong> instance of a given class (ClassA) exists in the class the arrow points to (ClassB).</td>
</tr>
</tbody>
</table>
Inheritance Diagrams: Symbols and Meanings

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Inheritance Arrow" /></td>
<td>Inheritance Arrow</td>
<td>This means that ClassA extends or inherits from the ClassB.</td>
</tr>
<tr>
<td><img src="image" alt="Interface Circle" /></td>
<td>Interface Circle</td>
<td>ClassA implements InterfaceB.</td>
</tr>
</tbody>
</table>

To get you started, we have included some diagrams based on Andy’s examples in lecture. Here is what a containment diagram would look like for the Race between two Racers. Note that a filled diamond arrow is used between Race and Racer as the Race class contains multiple instances of Racer. The Racer class also has references to (i.e. is associated with) Car and Bike, which is indicated with the one-way arrows between these classes.

![Inheritance Diagram](image)

In this next diagram, we have illustrated an interface relationship. Both Car and Bike are classes that model forms of transportation, and they implement the Transporter interface.
Note that interfaces are included in inheritance diagrams rather than containment diagrams. In this example, however, there were no inheritance relationships between classes, so the inheritance diagram only includes the Transporter interface. If Car or Bike inherited from a parent class, that parent-child relationship would be depicted in this diagram as well.

Lastly, here is an example of how we depict inheritance: the Dog class inherits from the Mammal class (a dog “is-a” mammal), and the Poodle and Labrador classes inherit from Dog (a poodle or a labrador “is-a” dog).

Create two diagrams, one illustrating association and containment, and the other illustrating the inheritance relationships among your classes. In your diagrams, you must include both the classes you write yourself and the support classes. We recommend using a digital program such as PowerPoint or Google Slides. This question must be completed digitally. Please submit a photo or a screenshot of your diagram, inserted into your mini assignment handin. We will be providing feedback on your diagrams so please put some time and thought into them. This will be greatly beneficial for when you start building your FruitNinja project.

3. Will you ever need to instantiate a cs015.prj.FruitNinjaSupport.CS15Game directly? Why or why not?

4. How do interfaces and superclasses help save you work in this project? Think about what happens if you’re adding more fruits that have different points. What if, hypothetically, we told you there were 100 different types of fruit? In addition to a lemon, apple, pear, and peach, we would want you to model a watermelon, kiwi, banana, etc. What classes would you need to write to do this? Which classes would you not have to alter because you have utilized the power of polymorphism?
CS Responsibility

Coding gives us the power to create things such as Fruit-Ninja. As technology evolves as a rapid rate, it is important to take a step back and consider the consequences of the software that we create.

The GPT-2 is a language model built by OpenAI that generates human-like text. It is a smaller version of a more powerful model. The model is generative, meaning that it is trained on a giant text-corpus (8-Billion Web-Pages) and then the statistical features of the inputted text are used to generate more text. Read the first article to learn more about OpenAI’s decision to release the model, and the second article about how researchers are using the model.

First Article:
Second Article:

Upon reading the articles:

1. Why might some people view OpenAI’s decision as an ethically responsible one? (list two reasons)
2. Why might some people view OpenAI’s decision as a harmful one? (list two reasons)