BSM 261: Object Oriented Programming
LAB 9: Building a Class Hierarchy

December 8/9, 2022

Date Performed: December 8/9, 2022
Instructor: Orhan özgüner
Teaching Assistant: Yunus Emre Avci

1 Class Material, Lab&Quiz Scores

You can access to the class material from the following link:
https://drive.google.com/drive/folders/1yo4pvaTngQ1L2TkHuWPCzbfapLDyvwUt?usp=sharing

You can check your lab attendance and quiz scores from the following link:
https://docs.google.com/spreadsheets/d/1xKQEqMSGlj3RBo9kLh97UzbTO1fyCvaT/edit?usp=sharing&ouid=104851561367296854968&rtpof=true&sd=true

2 Getting Started

Before you perform the lab please check the following information:
• Make sure that the lab computer you are using has DrJava downloaded into the desktop. Here is the jar file to check:

drjava-beta-20190813-220051.jar

• If you can not locate DrJava, please download it from the following link and place it to your desktop.

https://sourceforge.net/projects/drjava/files/latest/download

• If you have the DrJava jar file in your desktop, you are ready to perform your lab. In order to use jar files in linux, you need to run it from terminal. Run the following commend from the terminal to start DrJava

java -jar /Desktop/drjava-beta-20190813-220051.jar

3 Lab Rules

The purpose of the lab is to get you used to using DrJava, to start experimenting with Java, and to digest the topics you learn over the week.
Here are the lab rules:

- The teaching assistant will make a short recap of the week. Then you will perform the lab in the recitation hour.
- Your grade will be determined by your participation and finishing the assigned work during the recitation. When you get stuck on a question, please ask for help.
- Once you finish all of the tasks, show your work to the TA to receive full credit.
- **If you did not join the lab, or joined more than 10 minutes late or did not show your work to the Teaching Assistant, or did not finish all of the questions in class time you will receive 0.**

4 Lab Instructions and Questions

In this lab, you will create a hierarchy of classes and abstract classes and start to learn some benefits of using the hierarchy.

4.1 A First Class Hierarchy

In many computer games, we have a world filled with items that the game characters can pick up and use. Some items can be food, some items can be weapons, and so on. There can be hundreds of different items, and we want to avoid having to write giant if statements listing all the possible items.

First, get the `Person.java` file with this lab. You can see that currently a person has strength, intelligence, and energy.

Now, let's create items that the person can pickup and carry. For now, we want the following six items:

a. Apple
b. Sword
c. Hammer
d. Steak
e. Bread
f. Statue

We want each item to have a *weight* and a *name* associated with it. We want a person to be able to pick up an item as long as it is not too heavy.

I will start out by showing you one way to do that. Here is such an abstract class. Add this class to your code. Notice that it states that all items have weights and names. It also defines a toString method for the item. Initially the toString just uses the name of the item. The reason these are separate methods is so that later on we can have toString methods that can adjust to game situations.

```
public abstract class GameItem {
      private double weight;
      private String name;

      public GameItem(String name, double weight) {
         this.name = name;
         this.weight = weight;
      }

      public double getWeight() {
         return weight;
      }

      public String getName() {
         return name;
      }

      @Override
      public String toString() {
         return getName();
      }
}
```

Now that we have this class, defining each of the items is easy. Here is the class for Apples. An apple does not weigh very much so I give it a very light weight.

```
public class Apple extends GameItem {
      public Apple() {
         super("Apple", 0.25);
      }
}
```

Create a class for each of the six objects. Then add two new methods to Person:

public boolean pickup(GameItem o) : If the item weighs less than the strength of the person, return true and set a field indicating that the person is carrying this object.

GameItem drop() : Change the field so that the person is no longer carrying an object and return the object that was dropped. Return null if the person was not carrying anything.

To make things easy, right now we will assume a person can carry only one item at a time. Compile and test your code. Make sure some items (like the statue) are too heavy to pickup.

4.2 A First Class Hierarchy

We want to designate some items as food. A food item should have calories that increase the energy of a person who eats it.

Create an abstract class **Food** that extends **GameItem**. Just as how GameItem has a weight, a Food should have a *name*, a *weight* (inherited from GameItem so do not create it), and **calories**.

Change **Apple, Steak, and Bread** to now extends Food intead of GameItem and give each some calories. For example, here is Apple:

```
public class Apple extends Food {
      public Apple() {
         super("Apple", 0.25, 10);
      }
}
```

Just as before, an apple is very light, and I decided that it should provide 10 game "calories".

Now create an **eat** method in Person. You should only be able to eat food and when you eat food, you should increase the energy of the person eating it.

Compile and test your code. You should still be able to carry all items but only eat food.

Demostrate your code to your lab instructor to receive full credit.
LAB is over at this point but I wanted to add a challenge question for you to work on yourself:

4.3 Challenge: Special Food

It is quite common in games for food to give special benefits to the eater besides energy. For example, we could have apples increase intelligence, steak increase strength, and so on. We do not want a giant if statement inside the Person class to cover all the possibilities for the special food benefits.

Add the following method to the Food class:

```
public void eat(Person p) {
}
```

Next, change the eat method in Person to both increase the energy of the person, and to call the eat method of the Food.

Finally, override the eat method in Steak so that increases the strength of the person who ate the steak.

Compile and test your code. Eating steak should now increase a person's strength.
.