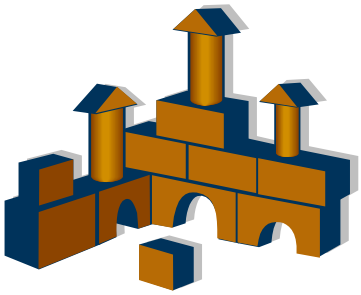


Programming Exercise 11:
Constructible Classes

Purpose: Provides an example of creating a class and using objects constructed from that class.
Background readings from textbook: Liang, Chapter 9.1-9.5, 9.8-9.10, 9.14

Due date for section 001: Monday, April 4 by 10 am
Due date for section 002: Wednesday, April 6 by 10 am

Overview

A strength of object-oriented languages like Java is the ability to *extend* the language by creating classes of your own that can be used by other classes (that you write or that other people write). A class becomes a building block that can be used in other code. This modular approach to programming improves one's ability to design, write, debug, and modify programs.

The concept of a class is that it is a self-contained entity that represents some class of object in the world (whether a physical object like a Car class or a Student class, or an abstract object like a String or a Window). Consider the String class. It stores a string of characters and contains methods that can manipulate and report on the stored string such as length, charAt, toUpperCase, toLowerCase and also given another String, tell us how they compare using compareTo and equals.

Part 1: Writing a Student Class

In order to write your own class, you must define the class' internal data (what it will store) and its methods. The internal data will consist of the variables (which we will call the class' instance data) that you feel are needed to fully describe the class. A Student class for instance might include a name, studentNumber, major, minor, creditHoursEarned, and cumulativeGPA among other items. Notice the spelling here. Class names are capitalized (e.g., String, Student) and variables use the camel notation.

What might we want to do with a Student (that is, what methods do we need)? We might want to learn the Student's major, minor or GPA, update their major, minor, hours earned or GPA, or given their hours earned, learn what class they are in (freshman, sophomore, junior, senior), or given their major, minor and GPA, learn their earning potential.

When we write a class, we define the entities that make up the class (the variables and methods) using *visibility* modifiers. These modifiers dictate who can access the components of the class. Usually we make the instance data *private* so that they can be accessed directly only from within the class but from another program, you cannot directly access them (either obtain their values or change their values) while making the methods *public* (accessible to anyone). Some of the methods will be used to give access to the data. Why make the methods public and the data private? That is, if the methods manipulate the data, what is to prevent another program from changing data to illegal values? We

can employ logic in our methods so that, if the method is asked to do something wrong to a datum, it can deny doing so. For instance, while major is private and cannot be directly accessed, we might have methods getMajor to return the major and setMajor to change the major where setMajor employs proper logic to make sure that the major is only changed to a legitimate major.

Let's consider an example. We have written and compiled our Student class. Now, a programmer uses it by declaring `Student s1;` in their program. At some point, the programmer wants to change their major to RULER, an illegal major. If the major instance datum is public, then the programmer could do this with the statement `s1.major="RULER";` Instead, we force the programmer to use the proper method, for instance, `s1.changeMajor("RULER");` If implemented properly, this method will not change the major because "RULER" is not a legal major (at least at NKU).

When defining our classes, we define three things: the class header, the class instance data and the methods. The header will look like this: `public class Student {…}` The entire class definition (instance data and methods) go inside the { }.

The instance data will be the variables known throughout the class. We define them like we have any of our variables with two exceptions. First, these variables are declared outside of any method. Second, they will require visibility modifiers, typically private.

```
public class Student {
     private String name, major, minor;
     private int creditHoursEarned;
     private double cumulativeGPA;
     …
}
```

The methods are like those we have already written with one exception, we do not include the modifier static. For instance, we might have a method like the following:

```
public String getName() {
     return name;
}
```

What does this method do? It returns the value stored in name. Why do we need it? Because we made name private, so from the outside, the only way to see the value in name is to use this getName method. Such a method that returns one instance variable's value is known as an *accessor* method. A *mutator* method is used to change one or more instance variable's values. For instance, to increase the number of credit hours, we might have the following method.

```
public void changeHours(int newHours) {
     if(newHours>0&&newHours<200) hours=hours+newHours;
}
```

Notice the if statement to ensure that we are not adding an unreasonable numbers to hours.

One other type of method that we will want for our class is known as a *constructor*. The constructor's role is to initialize any instance data when the class is instantiated into an object. The constructor is called when the `new` command is used. For instance, if we do `Student s1=new Student(…);`, then the Student constructor is called. The … will be any parameters we want to pass to Student to initialize it. Our constructor might expect just a name (as new Students have no major, minor, hours or GPA yet). So our constructor might look like this:

```
public Student(String newStudentName) {
     name=newStudentName;
     major="";
     minor="";
     hours=0;
     cumulativeGPA=0.0;
}
```

Now, to create some Students, we will do the following in another program:

```
Student s1, s2, s3;
s1=new Student("Frank Zappa");
s2=new Student("George Duke");
s3=new Student("Ruth Underwood");
s1.changeHours(15);
s2.changeHours(35);
…
```

Aside from accessors, mutators and construcors, we might have other methods that tell us useful information about the class. For instance, we might have a type of accessor is to get the Student's class rank given their hours. It would either output a String or return a String. Here is such a method which will return the proper class rank as a String. Notice that class rank is not an instance datum of Student, so we are not just returning rank like we earlier returned major.

```
public String getRank( ) {
     if(hours>=90) return "senior";
     else if(hours>=60) return "junior";
     else if(hours>=30) return "sophomore";
     else return "freshman";
}
```

There is one last method of note for a class known as a `toString` method. If, in the above code, we did `System.out.println(s1);` the output would not be what you expect. Instead of outputting information about s1, we would get s1's memory address. It would look like garbage to us as it would appear in hexadecimal notation. If a class has a toString method, it will automatically be called when we attempt to output an object of that class using println. We define the toString to return a String of the relevant parts of the object. For instance, given our Student class, we might define a toString as follows.

```
public String toString() {
     String tempMajor, tempMinor;
     if(!major.equals("")) tempMajor=major
          else tempMajor="undeclared";
     if(minor!="") tempMinor=minor else tempMinor="none";
     return name + "\t" + tempMajor + "\t" + tempMinor
               + getRank();
}
```

Once we have defined our Student class, we compile it and fix any syntax errors. Once it compiles, we can use it in other classes. But since Student does not have a main method, we cannot run Student directly. Instead, we must create variables of type Student, instantiate them, and then use those

variables. This makes classes somewhat trickier to debug – you cannot debug a class until you write both the class itself and then a user class (also sometimes called a driver class).

Part 2: Pitfalls

```
public class SomeClass
{
       public int var1, var2;
       ... // methods go here
}
```

Logic Error

Making the instance data public leaves the class open to being misused.

```
public class SomeClass
{
       public static void main(String[] args) {
              ...
       }
}
```

Possible Syntax Error

Unless this class will contain the code that "starts" the program, do not include a main method.

```
public class SomeClass
{
       ...
       private int someMethod( ) {...}
}
```

Possible Logic Error

Usually we want to make our methods public. There are occasions for having private methods – if the method is only going to be accessed from other methods of this class but otherwise make it public.

Other pitfalls include:

1. Not defining the accessors and mutators needed so that the class can be useful.
2. Not protecting your instance data in your mutators by testing to make sure that the value that an instance datum is changing to is legal.
3. Not providing one or more constructors. Although constructors are optionally, its good to offer one or more. You can offer multiple constructors if there are different patterns of parameters you might expect. For instance, a transfer student may already have a major, minor, hours and GPA. So a second constructor to the one shown in part 1 will be:

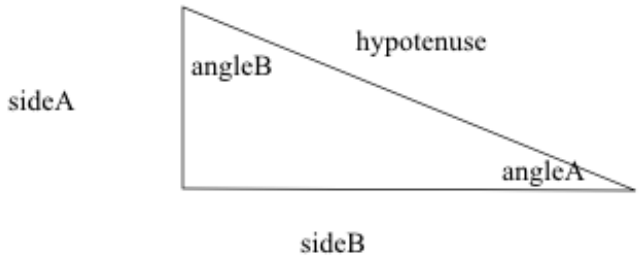
```
public Student(String transferName, String transferMajor,
     String transferMinor, int transferHours,
     int transferGPA) {
          name=transferName; major=transferMajor;
          minor=transferMinor;
          hours=transferHours; cumulativeGPA=transferGPA;
}
```

4. Not compiling your class before you write a user class to use it.
5. Placing a return type in your constructor's header. Constructors have no return type or void, instead they would look like: `public ClassName(…) {…}`

Part 3: Problem

We want to define a class called RightTriangle. The class will represent different RightTriangles. After writing and compiling this class, we will define a *user* class to create and use various

RightTriangle objects. Our RightTriangle class requires two instance data: sideA and sideB. Make them both doubles. See the triangle below.



Both instance data should be private. You will not have instance data for the hypotenuse, angleA or angleB, these will be computed via methods. Define each of the following methods, all of which should be public (and none should be static):

- A constructor which receives two doubles, say a and b, and sets sideA and sideB appropriately
- A constructor which receives no parameters and sets sideA and sideB to 0 (having multiple constructors is allowed as long as each one has a different number and type of parameters, this method has 0 parameters).
- A method to compute and return the hypotenuse (a double), computed as
  $$\sqrt{sideA^2 + sideB^2}$$
  Use Math.sqrt for the square root, you can use Math.pow or sideA*sideA
- A method to compute and return angleA (a double), computed as arcsin(sideB/hypotenuse), use Math.asin for arcsin
- A method to compute and return angleB (a double), arcsin(sideA/hypotenuse)

NOTE: The `Math.asin(…)` method returns the value in *radians*, not *degrees*. We want degrees. So take the result and pass it to `Math.toDegrees`.

- A method to compute and return the perimeter of the triangle (a double), computed as sideA + sideB + hypotenusel this will require that you call getHypotenuse in this method
- A method to compute and return the area of the triangle, which is ½ * sideA * sideB (a double)
- A toString method to return the 3 sides of the triangle, its perimeter and area as a String
- Two mutators called changeSideA and changeSideB to set sideA and sideB to new values, making sure that the parameter passed (a double) to the method is greater than 0 or else do not change the value

Once completed, compile your RightTriangle class and fix any syntax errors. Now, write a user class which will create the following RightTriangle objects and do the following operations.

- Create triangle1: 5, 20, triangle2: 3, 4, triangle3 (no parameters), triangle4: 16.3, 4.889
- Obtain and output the perimeter of triangle1
- Obtain and output angleA of triangle1
- Obtain and output angleB of triangle1
- Obtain and output the area of triangle2
- Set triangle3's sideA to 10.1
- Set triangle3's sideB to 12.4

- Obtain and output the angleA of triangle3
- Obtain and output the angleB of triangle3
- Output triangle4 (use System.out.println which will call your toString)
- Set triangle4's sideA to -6 (this should not impact traingle4)
- Output triangle4 (use System.out.println which will call your toString)
- Set triangle 2's sideA to 5.0
- Output triangle2 (using System.out.println)

Part 4: Enhancement and Submitting Your Assignment

Notice that the output of many of the double values has many decimal points of accuracy. Let's use DecimalFormat to reduce such output to no more than 2 decimal points (but 0 if the number does not need have any fractional part such as the hypotenuse for triangle2). You will have to do this formatting in both the user class when outputting any double value and in your toString method of your RightTriangle class. Therefore, import the DecimalFormat class in both classes.

Compile both classes, fixing any syntax errors and then run your user program again. Print out and hand in or email both classes (RightTriangle and the user class) and your output.