Classes: Documentation, Method Overloading, Scope, Packages, and "Finding the Classes"

20A05302T-OOPS THROUGH JAVA

S BABU, AP/CSE, KEC

Review

- What is data kept in an object is known as?
  - Attributes
- What are procedures kept in an object are known as?
  - Methods
- What is the concept of combining of data and code into a single entity is called?
  - Encapsulation
- What is the ability for an object to hide its data from outside entities is called?
  - Data Hiding
- What specifies what attributes and methods an object can have?
  - A Class

Review

- What do access specifiers do?
  - Determine where an entity can be used
- What are methods that change the data in the class are called?
  - Mutator methods
- What are methods that access private attributes of a class are called?
  - Accessor methods
- What are methods and attributes that are part of objects created from a class and not part of the class itself are called?
  - Instance variables and instance methods.
- What is the method that is called when you create and object?
  - The constructor

Documentation

- You should document your .java files that contain your classes just like any other .java file you have written before
  - This includes:
    - A top block with the required information (Author, course, etc.)
    - In-line comments describing what the code does
    - Comments indicating what the attributes are
    - Block comments above methods
  - The only exception is that you probably don't need to have a block comment above the setters and getters.
- Or you can create a javadoc!
  - CarDoc.java

Method Overloading

- Sometimes you want to have a multiple methods with the same name be able to do different operations on different parameters.
  - Java allows this through a process called <u>overloading</u>.
    - Overloading is having multiple methods in the same class with the same name, but accept different types of parameters.
  - For instance:

```
public double add(double num1, double num2) {
        return num1 + num2;
}

public String add(String str1, String str2) {
        return str1 + str2;
}
```

  - Even though both of these methods are named `add`, they perform different operations on different parameters.

Method Overloading

- When we call a method, the compiler must determine which of the methods to use through a process called binding.
  - Java binds methods by matching a method's signature to how it is called.
    - A method's signature consists of its name and the data types of its parameters.
    - The signatures of the two previous methods are:
      - add(double, double)
      - add(String, String)
    - So the java compiler can tell which method to used based on how it was called.
    - Note, that you cannot have methods with the same name and same data types for parameters EVEN IF THEY HAVE A DIFFERENT return type.

Constructor Overloading

- One of the more useful uses of method overloading is to overload constructors.
  - For example, you want to give programmers the option to create a rectangle without worrying about the length and width or the option to set the length and width when they create the object.
  - You can create two constructors, one with no parameters and one with two that initialize the attributes.
    - Example: MultipleConstructorsRectangle.java

Scope in Classes

- Data members of a class are in scope for the entirety of the class definition including all of its methods.
  - Knowing this, what would happen here?

```
private double length;

public void setLength(double inLength) {
   double length;
   length = Math.abs(inLength);
}
```

  - Since `length` is declared in the `setLength` method, it is used for the `length = Math.abs(inLength);` line. Even though both the local variable `length` and the attribute `length` are in scope, the local variable `length` is what is used.
    - The identifier with the lowest (most recently opened) scope is always used unless specifically told otherwise.
    - Hiding an attribute with a local variable is known as shadowing.

Packages

- All of the Java API classes are organized into packages.
  - A package is a group of related classes.
- Most Java API packages are available for use without importing them, although we have seen some that need to be explicitly imported:

```
import java.util.Scanner;
```

  - This statement tells the compiler to look in the `java.util` package for the `Scanner` class.
    - More specifically this tells the compiler to look in a directory called `util` that is in a directory called `java` for a class definition called `Scanner`.
  - `java.util` has many other classes to use.
  - The above import statement explicitly imports one class, however, if you wanted to import all of the classes in the java.util package, you can use a wildcard.

```
import java.util.*;
```

  - The * tells the compiler to import all of the classes in `java.util`.
- There are many other packages in the Java API and you can create your own for others to import (we may get to this).

Finding the Classes

- Object-Oriented Programming is different in many ways to the procedural programming we've done so far.
  - One interesting difference is in design.
    - Remember the focus of OOP is making objects and in Java that means making classes that define objects.
      - This means that you should focus design on what classes should be made and what attributes and methods should be inside of the classes.
  - Object-Oriented Design is focused around finding the classes that make up a problem.
  - This is done by following these steps:
    - Get a written description of the problem domain
    - Identify all of the nouns (including pronouns and noun phrases) in the description. Each of these is a potential class.
    - Refine the list to include only the classes that are relevant to the problem
      - Identify duplicates (nouns that mean the same thing)
      - Identify the nouns that do not concern us
      - Identify the nouns that are objects, not classes.
      - Identify nouns that can be stored as a simple variable and do not need a class.
  - After you have found the classes, you can focus on the data that is to be held in each class (attributes), and how outside sources use this data (the methods).
  - Also, think about how the objects need to interact.