Week 7: Object Oriented Programming

Classes and Objects

Instance Fields and Methods

Constructors

Overloading

Scope

Packages

Object Oriented Design

Classes and Objects

Object-Oriented Programming

- Object-oriented programming is centered on creating objects rather than procedures.
- Objects are a melding of data and procedures that manipulate that data.
- Data in an object are known as fields.
- Procedures in an object are known as methods.

Object-Oriented Programming

- Object-oriented programming combines data and behavior via **encapsulation**.
- Data hiding is the ability of an object to hide data from other objects in the program.
- Only an object's methods should be able to directly manipulate its data.
- Other objects are allowed manipulate an object's data via the object's methods.

Why do we hide data?

- Data hiding is very useful.
- Imagine having to understand all of the parts of a car in order to drive a car.
    - Some people do understand all of the parts of their car and I am impressed by this.
- But you don't have to fully understand cars in order to drive them.
- This is a good thing.
    - It makes cars easier to drive as they get more complicated over time. (Your average car today is more complicated than a Ford Model T yet easier to drive.)
    - If an engineer needs to change something, they don't have to tell you. It's properly hidden.

Data Hiding for the Programmer

- Data hiding is important for several reasons.
- It protects the data from accidental corruption by outside objects.
- It hides the details of how an object works, so the programmer can concentrate on using it.
- It allows the maintainer of the object to have the ability to modify the internal functioning of the object without “breaking” someone else’s code.

Code Reusability

- Object-Oriented Programming (OOP) has encouraged object reusability.
- A software object contains data and methods that represents a specific concept or service.
- An object is not a stand-alone program.
- Objects can be used by programs that need the object's service.
- Reuse of code promotes the rapid development of larger software projects.

Example: An Alarm Clock

- Fields define the state that the alarm is currently in.
    - The current second (a value in the range of 0-59)
    - The current minute (a value in the range of 0-59)
    - The current hour (a value in the range of 1-12)
    - The time the alarm is set for (a valid hour and minute)
    - Whether the alarm is on or off (“on” or “off”)

Example: An Alarm Clock

- Methods are used to change a field's value
- Public Methods
  - Set time
  - Set alarm time
  - Turn alarm on (O
  - Turn alarm off
- Private Methods
  - Increment the current second
  - Increment the current minute
  - Increment the current hour
  - Sound alarm

Classes and Objects

- The programmer determines the fields and methods needed, and then creates a class.
- A class can specify the fields and methods that a particular type of object may have.
- A class is a “blueprint” that objects may be created from.
- A class is not an object, but it can be a description of an object.
- An object created from a class is called an instance of the class.

Classes

- From chapter 2, we learned that a reference variable contains the address of an object.

Code.

```
String cityName = "Clarksville";
```

This creates a variable named “cityName” which points to another address in memory within the heap containing our object.

Classes

- The length() method of the String class returns and integer value that is equal to the length of the string.

Code.

```
int stringLength = cityName.length();
```

- Class objects normally have methods that perform useful operations on their data.
- Primitive variables can only store data and have no methods.

Instance Fields and Methods

Classes and Instances

- Many objects can be created from a class.
- Each object is independent of the others.

Code.

```
String person = "Jenny";
String pet = "Fido";
String favoriteColor = "Blue";
```

Classes and Instances

- Each instance of the String class contains different data.
- The instances are all share the same design.
- Each instance has all of the attributes and methods that were defined in the String class.
- Classes are defined to represent a single concept or service.

Building a Rectangle class

- A Rectangle object will have the following fields:
  - length. The length field will hold the rectangle's length.
  - width. The width field will hold the rectangle's width.

Building a Rectangle class

- The Rectangle class will also have the following methods:
  - setLength. The setLength method will store a value in an object's length field.
  - setWidth. The setWidth method will store a value in an object's width field.
  - getLength. The getLength method will return the value in an object's length field.
  - getWidth. The getWidth method will return the value in an object's width field.
  - getArea. The getArea method will return the area of the rectangle, which is the result of the object's length multiplied by its width.

UML Diagram

- Unified Modeling Language (UML) provides a set of standard diagrams for graphically depicting object-oriented systems.
- It's often a way to represent classes and relationships between classes.
- A class diagram is divided into three areas:
  - First box is the name. Usually that's it.
  - Second box is each field.
    - The order of the symbols are the access specifier, the name, and the type
  - Third box is the methods.
    - The order of the symbols are the access specifier, the name, the parameters (if any) and the type
    - Constructors never list a return type. They return themselves and are always named the same as the class.
  - Access specifiers are represented with "-" (for private) and "+" for (for public).

UML Diagram for Rectangle class

```
|------------------------------------|
| Rectangle                          |
|------------------------------------|
| width                              |
| length                             |
|------------------------------------|
| setWidth                           |
| getWidth                           |
| setLength                          |
| getLength                          |
| getArea                            |
|------------------------------------|
```

Writing the Code for the Class Fields

```
public class Rectangle {
    private double length;
    private double width;
}
```

Access Specifiers

- An access specifier is a Java keyword that indicates how a field or method can be accessed.
- public
  - When the public access specifier is applied to a class member, the member can be accessed by code inside the class or outside.
  - In UML, this is a “+” symbol.
- private
  - When the private access specifier is applied to a class member, the member cannot be accessed by code outside the class. The member can be accessed only by methods that are members of the same class.
  - In UML, this is a “-” symbol.

Header for the setLength Method

```
public void setLength (double len)
```

1. Access specifier
2. Return type
3. Name
4. Parameters.

Notice that we don't use the **static** keyword. In OOP, you typically never use **static**.

Writing and Demonstrating the setLength Method

```
/**
   The setLength method stores a value in the
   length field.
   @param len The value to store in length.
*/
public void setLength(double len) {
   length = len;
}
```

Creating a Rectangle object

```
Rectangle box = new Rectangle ();
box.setLength(10.0);
```

Likewise in order to set the width, we should create a method for "setWidth".

Writing the getLength Method

```
/**
   The getLength method returns a Rectangle
   object's length.
   @return The value in the length field.
*/
public double getLength() {
      return length;
}
```

Similarly, the setWidth and getWidth methods can be created.

Writing and Demonstrating the getArea Method

```
/**
   The getArea method returns a Rectangle
   object's area.
   @return The product of length times width.
*/
public double getArea() {
    return length * width;
}
```

Accessor and Mutator Methods

- Because of the concept of data hiding, fields in a class are private.
- The methods that retrieve the data of fields are called accessors (or getter methods).
- The methods that modify the data of fields are called mutators (or setter methods).
- Each field that the programmer wishes to be viewed by other classes needs an accessor.
- Each field that the programmer wishes to be modified by other classes needs a mutator.

Accessors and Mutators

- For the Rectangle example, the accessors and mutators are:
  - setLength: Sets the value of the length field.
  - setWidth: Sets the value of the width field.
  - getLength: Returns the value of the length field.
  - getWidth: Returns the value of the width field.

Accessors and Mutators

```
public void setLength(double len)
public void setLength(double wid)
public double getLength()
public double getWidth()
```

- Other names for these methods are getters and setters.

Stale Data

- Some data is the result of a calculation.
- Consider the area of a rectangle.
  - length times width
- It would be impractical to use an area variable here.
- Data that requires the calculation of various factors has the potential to become stale.
- To avoid stale data, it is best to calculate the value of that data within a method rather than store it in a variable.

Stale Data

Rather than use an area variable in a Rectangle class:

```
public double getArea() {
        return length * width;
}
```

- This dynamically calculates the value of the rectangle's area when the method is called.
- Now, any change to the length or width variables will not leave the area of the rectangle stale.

UML Data Type and Parameter Notation

- UML diagrams are language independent.
- UML diagrams use an independent notation to show return types, access modifiers, etc.

UML fields

In Java, to declare a private field, we would write this:

```
private double width;
```

In UML, this is written as:

```
-width: double
```

- The minus sign at the beginning means that something is private.
- Fields and their types are always listed with “name: type”.

UML Methods

In Java, to write a public setter method for the width, we would write this:

```
public void setWidth(double wid)
```

In UML, we would write this:

```
+setWidth(wid: double): void
```

- The plus sign at the beginning means that something is public.
- Like fields, return types are listed with a colon and then the return type.

Current UML Diagram

```
|--------------------------------------|
| Rectangle                            |
|--------------------------------------|
| -width: double                       |
| -length: double                      |
|--------------------------------------|
| +setWidth(wid: double): void         |
| +getWidth(): double                  |
| +setLength(len: double): void        |
| +getLength(): double                 |
| +getArea(): double                   |
|--------------------------------------|
```

Current Rectangle Code

```
public class Rectangle {
    private double width;
    private double height;
    public void setWidth(double wid) { width = wid; }
    public double getWidth() { return width; }
    public void setLength(double len) { length = len; }
    public double getLength() { return length; }
    public double getArea() { return width * height; }
}
```

Class Layout Conventions

- The layout of a source code file can vary by employer or instructor.
- A common layout is:
  - Fields listed first
  - Methods listed second
    - Accessors and mutators are typically grouped.
- There are tools that can help in formatting layout to specific standards.
  - We will explore these tools in NetBeans.

Instance Fields and Methods

- Fields and methods that are declared as previously shown are called instance fields and instance methods.
- Objects created from a class each have their own copy of instance fields.
- Instance methods are methods that are not declared with a special keyword, **static**.

Instance Fields and Methods

- Instance fields and instance methods require an object to be created in order to be used.

Code.

```
Rectangle kitchen = new Rectangle();
Rectangle bedroom = new Rectangle();
Rectangle den = new Rectangle();
```

Each of these rooms probably have different dimensions. It is rare for a house to have every room use the same dimensions. We need something to allow us to customize these dimensions.

Constructors

Constructors

- Classes can have special methods called constructors.
- A constructor is a method that is automatically called when an object is created.
- Constructors are used to perform operations at the time an object is created.
- Constructors typically initialize instance fields and perform other object initialization tasks.

Constructors

- Constructors have a few special properties that set them apart from normal methods.
  - Constructors have the same name as the class.
  - Constructors have no return type (not even void).
  - Constructors may not return any values.
  - Constructors are typically public.

Constructor for Rectangle Class

```
/**
   Constructor
   @param len The length of the rectangle.
   @param wid The width of the rectangle.
*/
public Rectangle(double len, double wid) {
   length = len;
   width = wid;
}
```

Final UML Diagram

```
|--------------------------------------|
| Rectangle                            |
|--------------------------------------|
| -width: double                       |
| -length: double                      |
|--------------------------------------|
| +Rectangle(len: double, wid: double) |
| +setWidth(wid: double): void         |
| +getWidth(): double                  |
| +getArea(): double                   |
|--------------------------------------|
```

Uninitialized Local Reference Variables

- Reference variables can be declared without being initialized. A varaible which is not initialized is set to `null`.

Code.

```
Rectangle box;
```

- This statement does not create a Rectangle object, so it is an uninitialized local reference variable.
- This is the price we pay for not having pointers in Java.

Uninitialized Local Reference Variables

A local reference variable must reference an object before it can be used, otherwise a compiler error will occur.
Code. box = new Rectangle(7.0, 14.0);

- **box** will now reference a Rectangle object of length 7.0 and width 14.0.

The Default Constructor

- When an object is created, its constructor is always called.
- If you do not write a constructor, Java provides one when the class is compiled. The constructor that Java provides is known as the default constructor.
    - It sets all of the object's numeric fields to 0.
    - It sets all of the object's boolean fields to false.
    - It sets all of the object's reference variables to the special value null.
- If this satisfies the needs of your application, it's okay to use Java's default constructor. If not, you'll have to write your own.

The Default Constructor

- The default constructor is a constructor with no parameters, used to initialize an object in a default configuration.
- The only time that Java provides a default constructor is when you do not write any constructor for a class.
- A default constructor is not provided by Java if a constructor is already written.

Default, Initialization, and Copy Constructor.

I like to use the “cookie analogy” when describing constructors. Imagine a cookie stand that sells cookies. The person at the stand asks what you’d like to order:

- “I want a cookie.” You are calling the default constructor. You have no choice over the cookie you get.
- “I want a chocolate chip cookie.” You are calling the initialization constructor. You get a chocolate chip cookie.
- “I want the same cookie my friend has.” You are calling the copy constructor. You get a cookie identical to the one your friend ordered.

Knowing which features you’d like to have in your class will help you to determine which constructors you need to write.

Writing Your Own No-Arg Constructor

- A constructor that does not accept arguments is known as a no-arg constructor.
- The default constructor (provided by Java) is a no-arg constructor.
- We can write our own no-arg constructor

Code.

```
public Rectangle() {
    length = 1.0;
    width = 1.0;
}
```

The String Class Constructor

- One of the String class constructors accepts a string literal as an argument.
- This string literal is used to initialize a String object.

Code.

```
String name = new String("George Washington");
```

The String Class Constructor

- This creates a new reference variable name that points to a String object that represents the name “Michael Long”
- Because they are used so often, String objects can be created with a shorthand:

Code.

```
String name = "George Washington";
```

Overloading

Overloading Methods and Constructors

- Two or more methods in a class may have the same name as long as their parameter lists are different.
- When this occurs, it is called **method overloading**. This also applies to constructors.
- Method overloading is important because sometimes you need several different ways to perform the same operation.

Overloaded Method add

```
public int add(int num1, int num2)
{
        int sum = num1 + num2;
        return sum;
}
public String add (String str1, String str2)
{
        String combined = str1 + str2;
        return combined;
}
```

Method Signature and Binding

A method signature consists of the method's name and the data types of the method's parameters, in the order that they appear. The return type is not part of the signature.

```
add(int, int)
add(String, String)
```

The process of matching a method call with the correct method is known as binding. The compiler uses the method signature to determine which version of the overloaded method to bind the call to.

Rectangle Class Constructor Overload

If we were to add the no-arg constructor we wrote previously to our Rectangle class in addition to the original constructor we wrote, what would happen when we execute the following calls?

```
Rectangle box1 = new Rectangle();
Rectangle box2 = new Rectangle(5.0, 10.0);
```

Rectangle Class Constructor Overload

If we were to add the no-arg constructor we wrote previously to our Rectangle class in addition to the original constructor we wrote, what would happen when we execute the following calls?

```
Rectangle box1 = new Rectangle();
Rectangle box2 = new Rectangle(5.0, 10.0);
```

The first call would use the no-arg constructor and box1 would have a length of 1.0 and width of 1.0. The second call would use the original constructor and box2 would have a length of 5.0 and a width of 10.0.

Overloading

- In summary, in order to know which method is being called in a class, you must know the full name of the method, the number of arguments, and each type of argument being called.
- The method that matches this exactly is the method which will be called.
- Note: You do not need to know the **return type**. The **return type** is not part of a method's signature.

Scope

Scope of Instance Fields

- Variables declared as instance fields in a class can be accessed by any instance method in the same class as the field.
- If an instance field is declared with the **public** access specifier, it can also be accessed by code outside the class, as long as an instance of the class exists.
- I consider this to be a Very Bad Idea (tm).
- It is recommended that you only use **public** along with **final**.

Shadowing

- A parameter variable is, in effect, a local variable.
- Within a method, variable names must be unique.
- A method may have a local variable with the same name as an instance field.
- This is called **shadowing**.
- **The local variable will hide the value of the instance field.**
- You can always reference a local variable using the name directly and the instance field using “this.” and then the name.

Shadowing Example

```
public class Rectangle {
    private double width;
    private double height;
    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }
}
```

Here, the fields **width** and **height** are being shadowed by the local variables in the constructor. We can still access them using “this.” and their name. I use this trick often.

Packages

Packages and import Statements

- Classes in the Java API are organized into packages.
- Explicit and Wildcard import statements
  - Explicit imports name a specific class
    - import java.util.Scanner;
  - Wildcard imports name a package, followed by an *
    - import java.util.*;
- The java.lang package is automatically made available to any Java class.

Do I need to import that?

- If the class you wish to use is not private and is in the same directory as the class you are working, Java will automatically be able to access it.
- If the class is in a select package named **java.lang**, then you will automatically be able to access it.
- If the class is in another directory, then you'll need an import statement.

Some Java Standard Packages

- java.io: Used for various types of input and output
- java.lang: General classes for the Java language. Automatically imported.
- java.net: Network communication
- java.security: Security features
- java.sql: Databases access using the structured query language
- java.text: Text formatting libraries
- java.util: Utility classes (like **Scanner**)
- javax.swing: Graphical User Interfaces

Object Oriented Design

Finding Classes and Their Responsibilities

- Finding the classes
  - Get written description of the problem domain
  - Identify all nouns, each is a potential class
  - Refine list to include only classes relevant to the problem
- Identify the responsibilities
  - Things a class is responsible for knowing
  - Things a class is responsible for doing
  - Refine list to include only classes relevant to the problem

Finding Classes and Their Responsibilities

- Identify the responsibilities
    - Things a class is responsible for knowing
    - Things a class is responsible for doing
    - Refine list to include only classes relevant to the problem