LEWIS UNIVERSITY
A Catholic and Lasallian University

Object-Oriented Programming

Course Number: CPSC-24500

Week: 2

Instructor: Eric Pogue

Learning Objectives – Week 2

1. Review Polymorphism, Model-View-Controller, and Setters & Getters
2. Understand the root class of all Java classes
3. Distinguish between function overloading and function overriding
4. Understand key components of Java development environment
5. Install* Java development environment and text editor
6. Implement* HelloWorld
7. Understand how to enhance BMI Calculator
   a) Add JavaDocs documentation
   b) Add keyboard input (scanner)
8. Understand implementation of Model portion of Shapes using Model-View-Controller
   a) Declare an abstract class and explain why it is useful
   b) Use Inheritance to build classes and objects that extend base class functionality
   c) Implement Encapsulation and Data Hiding by using setters and getters
   d) Write default and non-default constructors
   e) Override the toString function
   f) Store multiple objects in an ArrayList using generic data types
   g) Distinguish between using an array and an ArrayList
   h) Work with a collection of related objects polymorphically
   i) Explain how Polymorphism is implemented behind the scenes

The asterisk (*) by Install and Implement is a reminder that when it says “Install” or “Implement” (vs “Review” or “Understand”) indicates that you should do this yourself. I will often demonstrate it for us, but you should do it on your local computer as well. In this case I will not be asking for you to submit the code. However, I will be asking you in your weekly assignment if you completed the task.

Learning Objectives – Week 2 / Session 1

1. Review Polymorphism, Model-View-Controller, and Setters & Getters
2. Understand the root class of all Java classes
3. Distinguish between function overloading and function overriding
4. Understand key components of Java development environment
5. Install* Java development environment and text editor

Polymorphism

Polymorphism: Polymorphism enables you to process collections of related things generically. This is particularly useful when you want to use a loop to march through a collection of items.

```
//*******************************************************
// Polymorphism
Shape[] shapes = new Shape[3];
shapes[0] = new Circle();
shapes[1] = new Rectangle();
shapes[2] = new Triangle();
for (Shape s : shapes) {
   System.out.println(s.area());
}
```

Example of polymorphism: a for loop that moves through the entries of a list. The list might be of a collection of related kinds of object. We can refer to each of the objects in the list through a generic variable (whose data type matches the one that all are ultimately related to). But, when we invoke a particular function that all members of the family share, each will respond by performing that function in their own specific way. For example, we could have a collection of Shape objects. We could refer to each entry in the Shape list through a generic Shape variable, even though the actual entries in the list are specific kinds of shapes – Circle, Rectangle, etc. All Shape objects might have the ability to calculate their own area. When we refer to an object in the list through a generic Shape variable and tell it to calculate its area, thanks to polymorphism, the circle version of the area() function will be called when we're dealing with a circle, and the Rectangle version of area() will be called when we're dealing with a rectangle, etc.

The first time through the for loop, we'll call the Circle.area() function – actually, we won't; it will happen automatically. The next time through, we'll call the Rectangle version, and then we'll call the Triangle version.

Polymorphism is implemented behind the scenes using a Virtual Method Table (VMT). The VMT keeps track of where various related classes' same-named functions are located in memory. Using the VMT, the operating system is able to figure out which code to implement when we tell each shape to fire its area() function.

VMT – Virtual Method Table

Model-View-Controller

Model-View-Controller (MVC): MVC is an important pattern, will be a primary focus of this course, and will be an important pattern for you to master in your career.

Segregation of our Model (data) from our View (user interface) is necessary to effectively develop, enhance, and maintain modern software.

```
//*********************************************************
// DeleModel-View-Controller Example
class Student {  //model
    private String studentID;
    private String lastName;
    private String firstName;
}

class StudentDataEntryForm extends JFrame {  //view
    JTextField txtLastName;
    JTextField txtFirstName;
    JTextField txtStudentID;
    public void fillFields(String sid, String lastName,
        String firstName) {

    }
}
class FillDataEntryController
public void fillStudentForm(Student s, StudentDataEntryForm sdef) {
        sdef.fillFields(s.getStudentID(), s.getLastName(),
 s.getFirstName());
    }
}
```

An example would be a system that manages student data. We would want to segregate the Model (data) from the View (UI) for several reasons including that there will likely be many different Views that access the same data including:
student view
faculty view
administrator view,
Web student view,
mobile student view, etc.

Evolution of UI and Data segregation

- Document-View (View was responsible for View-Controller functionality)
- Model-View-Controller
- Model–View–Viewmodel

Learn this Pattern!

Encapsulation… and Setters & Getters

Setters and Getters:

Setters and Getters are a practice where public Methods are put in place to control how private Attributes are updated.

They can be beneficial in:

- Validation
- Optimization
- Converting types (English to metric)
- Debugging breakpoints
- Some libraries expect setters and getters

Shapes with Setters and Getters:

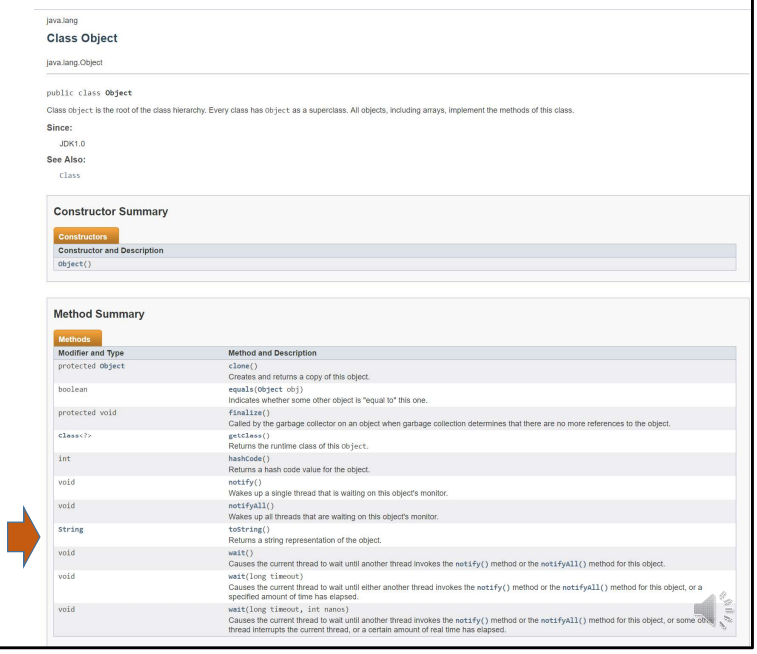
```
//*******************************************************
// Shapes Step: Setters and Getters
abstract class Shape {
    private int positionX;
    private int positionY;

    public int getPositionX() {
        return positionX;
    }

    public void setPositionX(int positionXIn) {
        positionX = positionXIn;
    }

    public int getPositionY() {
        return positionY;
    }

    public void setPositionY(int positionYIn) {
        positionY = positionYIn;
    }

    public Shape() {
        positionX = 0;
        positionY = 0;
    }
}
```

Why use Setters & Getters? Because 2 weeks (months, years) from now when you realize that your setter needs to do more than just set the value, you'll also realize that the property has been used directly in 238 other classes. (Internet quote)

Java Root Class

java.lang.Object:

- The Class named Object is the base class for ALL classes
- It contains numerous methods including toString()
- A little foreshadowing… the screen to the right was automatically generated using JavaDoc



java.lang

Class Object

java.lang.Object

public class **Object**

Class object is the root of the class hierarchy. Every class has object as a superclass. All objects, including arrays, implement the methods of this class.

Since:

JDK1.0

See Also:

Class

Constructor Summary

Constructors

Constructor and Description
Object()

Method Summary

Methods

Modifier and Type	Method and Description
protected Object	clone() Creates and returns a copy of this object.
boolean	equals(Object obj) Indicates whether some other object is "equal to" this one.
protected void	finalize() Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
Class<?>	getClass() Returns the runtime class of this object.
int	hashCode() Returns a hash code value for the object.
void	notify() Wakes up a single thread that is waiting on this object's monitor.
void	notifyAll() Wakes up all threads that are waiting on this object's monitor.
String	toString() Returns a string representation of the object.
void	wait() Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.
void	wait(long timeout) Causes the current thread to wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.
void	wait(long timeout, int nanos) Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.

A package is a logical grouping of Java library classes. The java.lang package comes "for free" for all Java applications.

All java classes inherit from Object.

Note: Yes, I know, the terminology of having a base class called "Object" can be challenging. For our purposes objects are instances of classes. However we also recognize that the base Java class is also regrettably called "Object". I will generally call it "the base java class" and not refer to it as "Object".

Take note of the "toString()" method and recognize that EVERY java class can be expected to have a "toString" method. We will be overriding this method regularly in this week's examples.

Overriding vs. Overloading

Overriding Method: Subclass implementing same method name and same parameters

Overloading Method: Same function name with different number of parameters. Only use this in specific situations like Constructors.

Overriding

```
//*********************************************************
// BMI Calculator (OOP Java)
// BMI = weight over height squared

abstract class BMI {
    abstract public float CalcBMI(float height, float weight);
}

class BMIMetric extends BMI {
    public float CalcBMI(float height, float weight) {
         return weight / (height * height);
    }
}

class BMIEnglish extends BMI {
    public float CalcBMI(float height, float weight) {
        // Convert to meters and kg.
        height = height * (float)0.025;
        weight = weight * (float)0.45;
        return weight / (height * height);
    }
}

class CalcBMI {
    public static void main(String[] args) {
        BMI myBMI = new BMIEnglish();
        float BMIresult = myBMI.CalcBMI(
            (float)((6.0 * 12.0) + 1.0) /*height*/,
            (float)190.0 /*weight*/);

        System.out.println(BMIresult);
    }
}
```

It's regrettable that the naming is so close on these two concepts.

Method Overriding is what we have been doing as we create supperclasses and subclasses. We Override methods in the parent class to add functionality. For example, we overrode (override, overriding, overrode) the CalcBMI method of BMI when we implemented CalcBMI in BMIEnglish.

We will be Overriding methods constantly in this class. Overloading will be less common and less important, but we do need to understand what it is.

Overriding MUST have exactly the SAME parameters and return types.

Overriding MUST have at least one parameter or return type that is differenct.

Overriding vs. Overloading

Overriding Method: Subclass implementing same method name and same parameters

Overloading Method: Same function name with different number of parameters. Only use this in specific situations like Constructors.

Overloading

```
//*******************************************************
// Method Overriding

class AddTwoNums {
    public int Add(int n1, int n2) {
        return n1 + n2;
    }
}

class AddThreeNums extends AddTwoNums {
    public int Add(int n1, int n2, int n3) {
      return n1 + n2 + n3;
    }
}

public class Override {
    public static void main(String[] args) {
        AddTwoNums myTwoNums = new AddTwoNums();
        AddThreeNums myThreeNums = new AddThreeNums();

        int TwoNums = myTwoNums.Add(1,2);
        int ThreeNums = myThreeNums.Add(1,2,3);

        System.out.format("TwoNum = %d\n", TwoNums);
        System.out.format("ThreeNum = %d\n", ThreeNums);
    }
}
```

AddThreeNums overloads the Add method. It is important to understand Overloading; however, it is not as important or powerful as Overriding. Overloading is also only loosely related to object-oriented programming.

Note that we could have overloaded the Add method in AddTwoNums (without creating a subclass) and it still would be considered method overloading.

Java Environment Overview [link]

The Java Application Platform SDK includes:

- Java API
- Java Compiler (javac)
- Java Runtime Environment (java)
- Java Doc (Javadoc)

… And much much more.

Java code (.java) –> Java compiler –> Java bytecodes – Java runtime Environment (JRE)

Java API is made up of LOTS of classes. Those classes are organized into Packages which are simply libraries of related classes.

JavaDoc example:

```
/**
   This class does lots of good things.
   @author Eric Pogue
*/
```

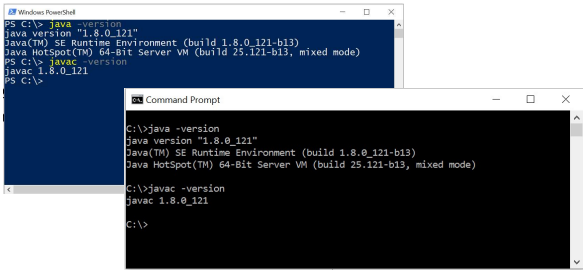
@author
@param
@return

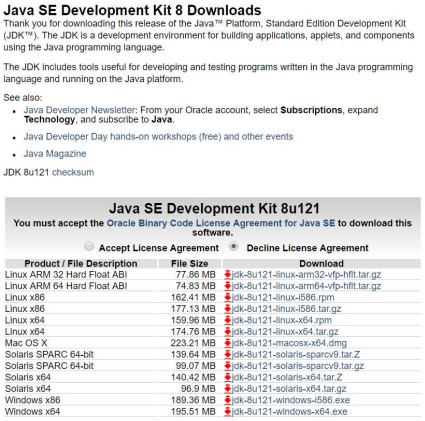
Example: javadoc –d .\docs Shares.java

Install* Java Development Environment [link]

Standard Java tools for this class include:

- Java SE Development Kit (SDK) 8 from Oracle
- Text Editor





Java SE Development Kit 8 Downloads

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications, applets, and components using the Java programming language.

The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java platform.

See also:

- Java Developer Newsletter: From your Oracle account, select **Subscriptions**, expand **Technology**, and subscribe to **Java**.
- Java Developer Day hands-on workshops (free) and other events
- Java Magazine

JDK 8u121 checksum

Java SE Development Kit 8u121

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Accept License Agreement Decline License Agreement

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.86 MB	jdk-8u121-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	74.83 MB	jdk-8u121-linux-arm64-vfp-hflt.tar.gz
Linux x86	162.41 MB	jdk-8u121-linux-i586.rpm
Linux x86	177.13 MB	jdk-8u121-linux-i586.tar.gz
Linux x64	159.96 MB	jdk-8u121-linux-x64.rpm
Linux x64	174.76 MB	jdk-8u121-linux-x64.tar.gz
Mac OS X	223.21 MB	jdk-8u121-macosx-x64.dmg
Solaris SPARC 64-bit	139.64 MB	jdk-8u121-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.07 MB	jdk-8u121-solaris-sparcv9.tar.gz
Solaris x64	140.42 MB	jdk-8u121-solaris-x64.tar.Z
Solaris x64	96.9 MB	jdk-8u121-solaris-x64.tar.gz
Windows x86	189.36 MB	jdk-8u121-windows-i586.exe
Windows x64	195.51 MB	jdk-8u121-windows-x64.exe

We want to develop our understanding of object-oriented programming concepts, patterns, and principles in a way that is independent of any particular language or platform. However, in order to practice those concepts, patterns, and principles, we need to utilize at least one (and preferably several) language and environment.

If you don't already have an appropriate Java development environment installed, you will need to install it. I would like you to actually set this up on your local computer and validate that it is installed properly. This will likely take quite a while depending on your internet download speed and if you run into any environment issues.

Steps:
#1: Select the appropriate Java SE install from the Oracle site:
http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
For example, I am using "Windows 10" and selected the "Windows x64" version.

#2: Validate your installation is installed properly and that you have access to the key tools. For examples, I am using the Windows 10 cmd prompt (and PowerShell) so I opened a command window and executed "java -version" and "javac -version" to verify the tools were in place.

Note that I did initially run into a "Path" challenge in Windows 10 where the prompt could not find the "javac" compiler. I found a pretty good YouTube video that showed how to update the Path to add the location of the "javac" compiler. It is located at: https://www.youtube.com/watch?v=Wp6uS7CmivE

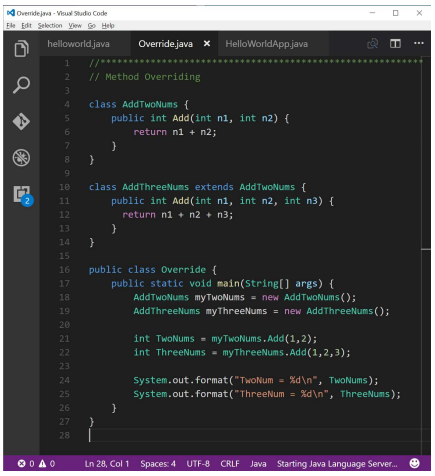
You will want to verify you environment before implanting HelloWorld.

Install* Java Development Environment (continued)

Standard Java tools for this class include:

- Java SE Development Kit (SDK) 8 from Oracle
- Text Editor… Use any text editor you desire

- Example: Microsoft Code [link]



#3: Verify that your favorite text editor is working, or download the one you want to use.

For example, I have recently switched to using the Microsoft Code editor (note that it is cross platform and available for Mac and Linux). I have had mixed results, but if you want to use it, it is available for free download at:
http://code.visualstudio.com/

I will be using the command line Java JDK to review and grade your assignments. Make CERTAIN that the assignments you submit compile an run in this environments.

Java Integrated Development Environment

- Source Code Editor (syntax highlighting, code completion, etc.)
- Compiler
- JRE
- Debugger**
- JavaDoc

Eclipse

Visual Studio

We will likely take a look at Eclipse in the coming weeks; however, for now let's focus on using just a text editor and command line tools.

Some editors have started introducing syntax highlighting, code completion, compilation integration, etc.

If you do want to install a Integrated Java Development Environment (Java IDE). It is best to install the JDK before installing IDE to be safe.

LEWIS UNIVERSITY
A Catholic and Lasallian University

End of Session

Course Number: CPSC-24500

Week: 2

Session: 1

Instructor: Eric Pogue