Oracle Academy
Java for AP Computer Science A

10-1

Inheritance

ORACLE
Academy

Objectives

- This lesson covers the following objectives:
  - Demonstrate and explain UML (Unified Modeling Language) class diagrams
  - Use the extends keyword to inherit a class
  - Compare and contrast superclasses and subclasses
  - Describe how inheritance affects member access
  - Use super to call a superclass constructor
  - Use super to access superclass members
  - Create a multilevel class hierarchy

Overview

- This lesson covers the following topics:
  - Recognize when constructors are called in a class hierarchy
  - Demonstrate understanding of inheritance through the use of applets
  - Recognize correct parameter changes in an existing applet

What is Inheritance?

- Inheritance is a simple but powerful tool of object-oriented languages that allows classes to inherit methods and fields from other classes

- Inherit means to receive or obtain something from your predecessor or parent

- In Java, the concept of inheritance is similar to genetics
  - Genes and genetic traits are passed down from parent to child
  - Children often look and act like their parents as a result

More Information about Inheritance

- For more information about inheritance, visit:
  - http://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html

ORACLE
Academy

Superclass versus Subclass

- Classes can derive from or evolve out of parent classes, which means they contain the same methods and fields as their parents, but can be considered a more specialized form of their parent classes

- The difference between a subclass and a superclass is as follows:

Superclass	Subclass
The more general class from which other classes derive their methods and data	The more specific class that derives or inherits from another class (the superclass)

JavaAPCSA 10-1
Inheritance

Superclass versus Subclass

- Superclasses contain methods and fields that are passed down to all of their subclasses

- Subclasses:
  - Inherit methods and fields of their superclasses
  - May define additional methods or fields that the superclass does not have

General: Superclass

Specific: Subclass

ORACLE
Academy
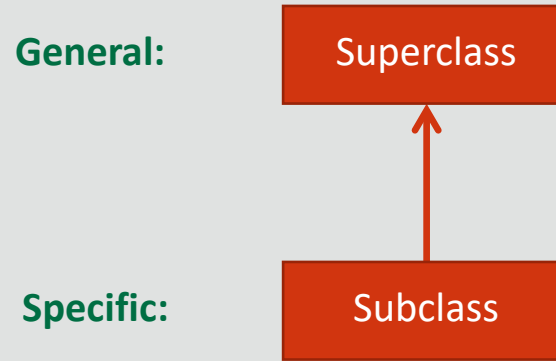
Inheritance Example

- Create a Shape class with a variable, color, and one method which returns the color
  - Create a Rectangle class which inherits the variable and method from Shape, and can have its own methods and variables

```
Shape (superclass)
public String color
public String getColor()
```

```
Rectangle (subclass)
public String color
public String getColor()

//Rectangle-only data
public int length
public int width
public int getLength()
public int getWidth()
```

ORACLE
Academy

Superclass vs. Subclass Example

- Consider the Animal and Crab classes from Greenfoot
- Animal is a more general term than Crab and can apply to more creatures than just Crab
- A Crab is a type of Animal, and that Crab applies to a specific type of Animal
- Therefore, Crab is the subclass and Animal is the superclass

General:

Animal

Specific:

Crab

ORACLE
Academy

Crab Class move Method

- Where does the method move() come from in Crab?
- There is no visible code showing the logic for the move() method in the class Crab

```java
public class Crab extends Animal
{
    public void act()
    {
        move(1);
    }//end method act
}//end class Crab
```

ORACLE
Academy

Inherited move Method

- Even though the code isn't written in the Crab class, we know a Crab object can call the move() method
- Therefore, the code must be inherited from the superclass, Animal, as follows:

```java
public class Animal
{
    public void move(int d)
    {
        //Logic for move()
    }//end method move
}//end class Animal
```

extends Keyword

- In Java, you have the choice of which classes you want to inherit from by using the keyword extends

- The keyword extends allows you to designate the superclass that has methods you want to inherit, or whose methods and data you want to extend

- For example, to inherit methods from the Shape class, use extends when the Rectangle class is created

```java
public class Rectangle extends Shape
{
    //code
}//end class Rectangle
```

extends Keyword Example

- We want the Crab class to extend the methods and data in Animal and inherit methods like move(), turn(), etc
  - Because the Animal class is extended, you can call the move() and turn() methods even though they do not appear within the Crab class code

```java
public class Crab extends Animal
{
    public void act()
    {
        move(1);
        turn(90);
    }//end method act
}//end class Crab
```

ORACLE
Academy
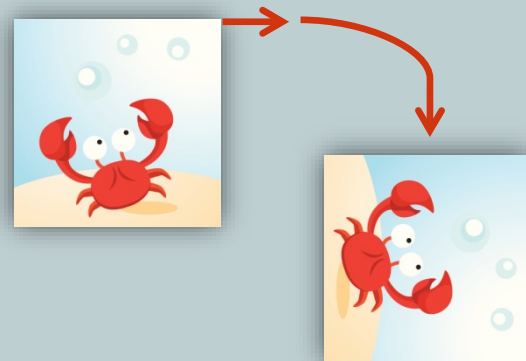
The Rule of Single Inheritance

- Single inheritance means that you cannot declare or extend more than one superclass per class
- The following code will not compile:

```
public class Crab extends Animal, Crustacean, ...
```

ORACLE
Academy

Extending More than One Class

- Why can't we extend more than one class?
  - Since superclasses pass down their methods and data to all of their subclasses and the subclasses of their subclasses, it isn't really necessary to extend more than one class

ORACLE
Academy

More About Inheritance

- Inheritance is a one-way street
  - Subclasses inherit from superclasses, but superclasses cannot access or inherit methods and data from their subclasses
  - This is just like how parents don't inherit genetic traits like hair color or eye color from their children

ORACLE
Academy

Object: The Highest Superclass

- The superclass of all other classes
- Each class **implicitly** extends the class Object
- Each class **implicitly** calls the constructor in the class Object
- Object:
  - Is considered the highest and most general component of any hierarchy
  - It is the only class that does not have a superclass
  - Contains very general methods that every class inherits

General: | **Superclass**

↑

Specific: | **Subclass**

Object: The Highest Superclass

- Navigate to the Java Docs for the Object class:
  - https://docs.oracle.com/javase/8/docs/api/
  - Select the java.lang package
  - Select the Object class
  - Browse the Object class methods

- Object contains useful methods that can be used by every class, for example:
  - toString()
    - This method returns a String representation of the object
    - This method compares two objects, and then returns `true` if and only if two objects are the same object

JavaAPCSA 10-1
Inheritance

Object Example 1

- After you create a class and construct an instance of it, can you call the method toString() on your object?

```
A_Class class = new A_Class();
class.toString();
```

- Yes
    - Even though you did not write the method toString(), it is still legal for you to call this method because it was inherited from Object

Object Example 2

- Superclass Crustacean is extended by Subclass Crab
- Is c1.toString() legal?

```
Crab c1 = new Crab();
c1.toString();
```

- This is legal since Crustacean and Crab implicitly extend Object

ORACLE
Academy

Why Use Inheritance?

- The central benefit to inheritance is code reuse
- Inheriting methods from a superclass gives your class access to the code and data of the superclass
- You will not have to write the code twice, which saves you time and optimizes your code
- Also, fewer bugs occur

Inheritance Hierarchies

- In many situations, it's common to classify concepts as hierarchies
  - A hierarchy is a way of categorizing the relationship among ideas, concepts or things with the most general or all-encompassing component at the top and the more specific, or the component with the narrowest scope, at the bottom
  - Hierarchies are a useful concept when it comes to inheritance and can be used to model and organize the relationship among superclasses and subclasses
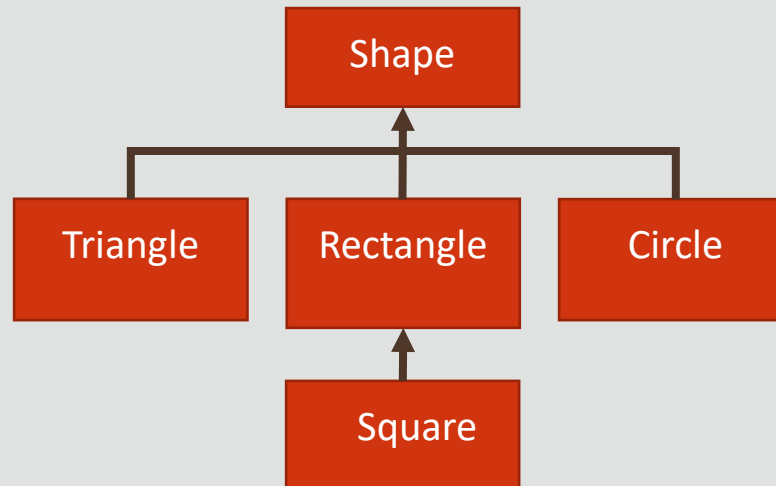
Tree Diagrams

- Hierarchies can be organized into tree diagrams
  - Computer Scientists will often refer to trees having leaves and branches, or will refer to the "leaves" as nodes
  - For example, shapes can be categorized by different properties, such as their number of sides
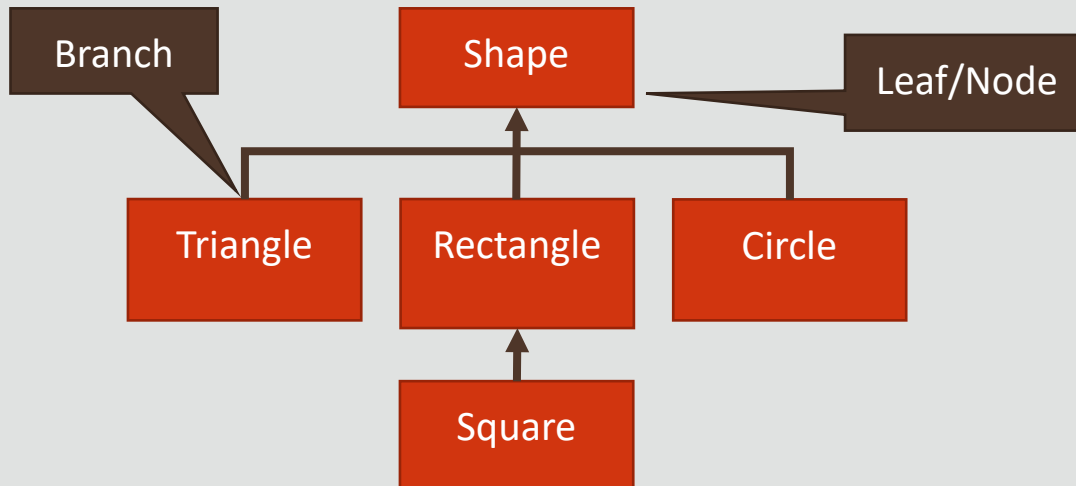
Tree Diagrams

- Notice that Circle, Triangle, and Rectangle all have a different number of sides, so they are different branches in the tree

- Only nodes with the same properties will occupy the same branch

```
                    ┌──────────┐
                    │  Shape   │
                    └──────────┘
                         ▲
        ┌────────────────┼────────────────┐
   ┌──────────┐    ┌──────────┐    ┌──────────┐
   │ Triangle │    │Rectangle │    │  Circle  │
   └──────────┘    └──────────┘    └──────────┘
                         ▲
                    ┌──────────┐
                    │  Square  │
                    └──────────┘
```

Type of Parent Nodes

- Anything below a node on the tree is a type of the parent node

- We know that Square is a type of Rectangle

- Triangle, Rectangle, and Circle are all types of shapes
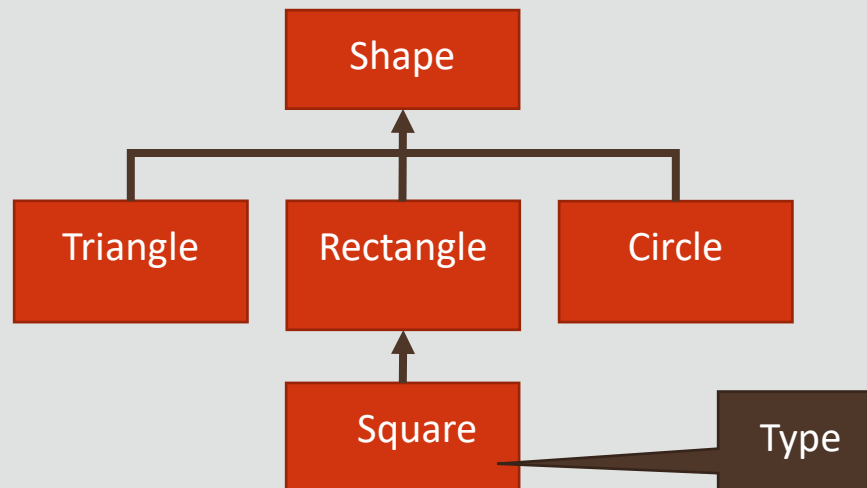
Inheritance Hierarchies: "is-a"

- With class hierarchies, you can use the phrase "is-a" to describe a hierarchical relationship

- A node on a branch can be thought of as the same type as the node on the root

- Example: A Rectangle "is-a" Shape, since it has all of the properties of a shape

- To model relationships among classes, we use UML

Shape

is-a

Rectangle
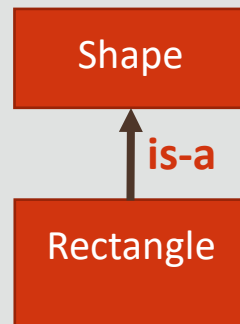
Unified Modeling Language: UML

- Computer scientists model inheritance hierarchies using a modeling language called Unified Modeling Language, or UML

- UML is a way of describing the relationships among classes in a system, or graphical representation of a system

- UML was developed by Grady Booch, James Rumbaugh, and Ivar Jacobson, and is standardized so it can be understood across languages

Basic Components of UML

- Inheritance hierarchies can be modeled simply with UML

- A few simple components are required to start:
  - Class diagram:
    - Shows the name of the class, and any important data or methods within the class
  - Arrows and lines:
    - Show one class's relationship to other classes

Class Diagram in UML

- A class can be drawn as a box that contains the class name, instance variables, and methods

- Classes can also be drawn as simple boxes with just the class name, although including methods is helpful

- It is not necessary to include every attribute or those that represent collections of data (such as arrays) Include only the most helpful attributes

Class Name
Attributes
Methods

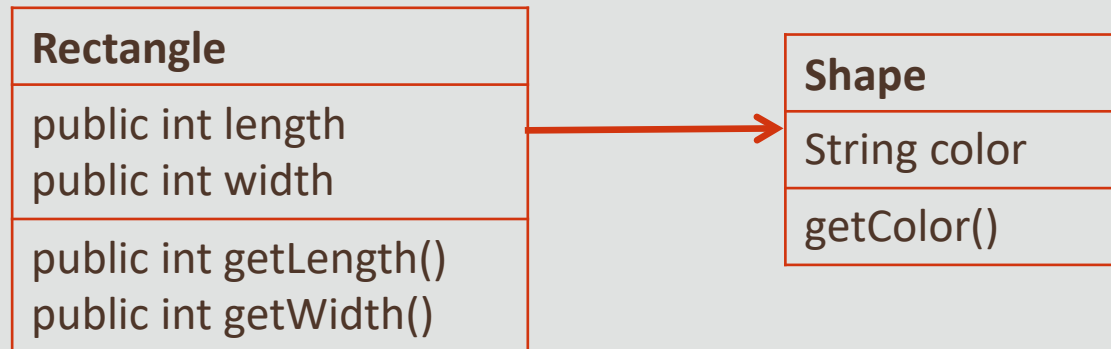| Shape |
|---|
| Color |
| getColor() |
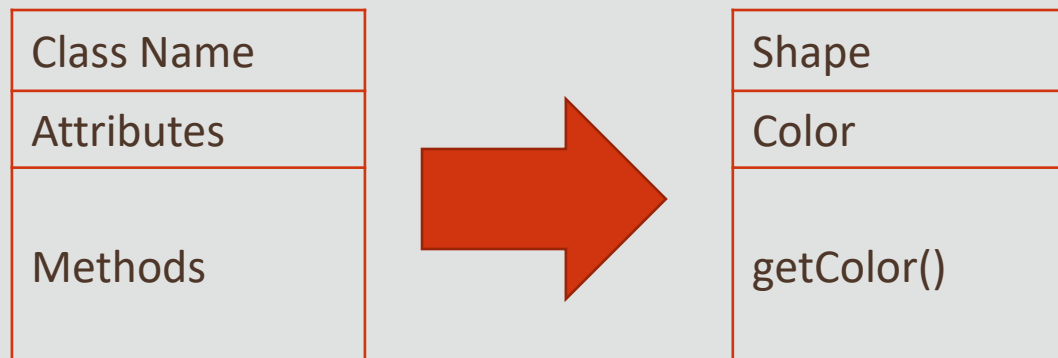
More Details About UML

- UML is a helpful tool for you to plan out how to organize multilevel class hierarchies
- You are encouraged to use UML for coding projects
- For a more detailed look at UML, visit:
- http://www.oracle.com/technetwork/developer-tools/jdev/gettingstartedwithumlclassmodeling-130316.pdf

Showing Inheritance in UML

- Class diagrams are connected using lines with arrows
- Connecting lines vary depending on the relationship among classes
- For inheritance, a solid line and a triangular arrow are used to represent the "is-a" relationship

Relationship	Symbol
Inheritance	→

Rectangle
public int length public int width
public int getLength() public int getWidth()

Shape
String color
getColor()

ORACLE
Academy

Encapsulation

- Encapsulation is a fundamental concept in object-oriented programming

Encapsulation means to enclose something into a capsule or container, such as putting a letter in an envelope.
In object-oriented programming, encapsulation encloses, or wraps, the internal workings of a Java instance/object

How Encapsulation Works

- In object-oriented programming, encapsulation encloses, or wraps, the internal workings of a Java instance/object

- Data variables, or fields, are hidden from the user of the object

- Methods can provide access to the private data or work with the data, but methods hide the implementation

- Encapsulating your data prevents it from being modified by the user or other classes so that the data is not corrupted

How Encapsulation is Used

- Encapsulation can be used to protect sensitive data, such as personal information, by preventing the data from being changed, except within the scope of the class itself

- Data is protected and implementation is hidden by declaring access modifiers on variables and methods

- Access modifiers (public, private, protected, "default") are keywords that determine whether or not other classes can access the data or methods of the class

Access Modifiers

- Programmers can customize the visibility of their data and methods with several levels of access modifiers

Access Modifier	Accessed by:
public	Any class in any package
private	Only to other methods within the class itself
protected	All subclasses and all classes in the same package
"default"	Any class in the package When no keyword is specified The word default is NOT used

Declaring Access Modifiers

- The general rule for declaring access modifiers is that any data you want to protect from being altered by other classes, or data that is sensitive, should be declared private

- This includes variables

- Methods are generally declared as public so other classes can use them

- However, methods can be declared private when they are meant to be used only by the class itself

Declaring Access Modifiers Example

- If the Shape class contained data for color, the data in this class should be private

```
public class Shape {
    private String color;
}//end class Shape
```

ORACLE
Academy

Member Access

- Use the keyword private to hide data that only the class should be able to change (This is the recommended access modifier)

- If access to the data is needed, then a getter method should be written to achieve this

```java
public class Shape {
    private String color; //the color of the Shape

  //Method which returns the color
  public String getColor() {
    return color;
    }//end method getColor
}//end class Shape
```

With the Shape class, we do not want objects to change the color of this Shape. Writing private in the variable declaration achieves this.

The method getColor() returns the color. Any method that is meant for accessing private data should be public.

ORACLE
Academy

Member Access

- If the private variables need to be (or are allowed to be) changed, then a setter method should be written

```java
public class Shape {
    //the color of the Shape
    private String color;
    //Method which returns the color
    public String getColor() {
        return color;
    }//end method getColor
    //Method to change the color
    public void setColor(String c){
        color = c;
    }//end method setColor
}//end class Shape
```

If the color needs to be changed, a setter method is created.

Use Public or Protected to Access Data?

- If we wanted the ability to alter the Shape's variable color from outside the Shape class code, we could set the String variable color to be public or protected

- However, it is recommended that class variables are declared as private

```
public class Shape {
    protected String color;
}//end class Shape
```

```
public class Shape {
    public String color;
}//end class Shape
```

Changing the Color

- If the variable is declared as public, code that extends or creates a Shape object could change the color without using an accessor method like setColor()

```
/*example for extending the shape class, then changing the
    color*/
super.color = "Blue";          Not recommended



/*example for creating a Shape object and changing the
    color*/
Shape s1 = new Shape();
s1.color = "Blue";          Not recommended
```

Member Access and Inheritance

- How do these access modifiers affect inheritance?
- With encapsulation, even subclasses cannot access private methods and variables
- public and protected modifiers provide access to superclass methods and variables

Access Modifier	Accessed by:
public	All classes
private	Only the class itself
protected	All subclasses and all classes in the same package
"default"	If no keyword is specified, member variables can be accessed by any class in the package

Extending the Shape Class

- Since Shape is not a specific class, we can extend it by creating more specific classes, such as, Rectangle and Square

- We will begin by creating a Rectangle class that extends the Shape class

Inheriting Constructors

- Although a subclass inherits all of the methods and fields from a parent class, it does not inherit constructors

- You can:
  - Write your own constructor or constructors
  - Use the default constructor
    - If you do not declare a constructor, a default no-argument constructor is provided for you
    - If you declare your own constructor, the default constructor is no longer provided

Using the Keyword super in a Constructor

- When creating a Rectangle object, you will need to set the color of the Rectangle

- If the variable color is private in the Shape superclass, how do you set it?

- To construct an instance of a subclass, it is often easiest to call the constructor of the parent class

Using the Keyword super in a Constructor

- The super keyword is used to call a parent's constructor

- It must be the first statement of the constructor

- If it is not provided, a default call to super() is implicitly inserted for you

- The super keyword may also be used to invoke a parent's method or to access a parent's (non-private) field

ORACLE
Academy

Using the Super Keyword Example

```java
public class Rectangle extends Shape
{
    private int length;
    private int width;

    //Constructor
    public Rectangle(String color, int length, int width)
    {
        super(color);
        this.length = length;
        this.width = width;
    }//end constructor
}//end class Rectangle
```

This calls the constructor for Shape, which initializes the variable color.

ORACLE
Academy

Adding methods for the Rectangle Class

- Rectangle methods that would be useful include:

```java
public int getWidth()
public int setWidth(int width)
public int getHeight()
public int setHeight(int height)
public int getArea()
```

ORACLE
Academy

Rectangle Class Methods

- Since Square is a type of Rectangle, or extends the Rectangle class, it will inherit all of the methods from the Rectangle superclass:

```java
public int getWidth()
public int setWidth(int width)
public int getHeight()
public int setHeight(int height)
public int getArea()
```

Set Up the Class

- This code sets up the class
- Use the keyword extends to inherit the methods from Rectangle

```java
public class Square extends Rectangle {

}//end class Square
```

ORACLE
Academy

Write the Constructor

- To write the constructor, consider what values need to be initialized

- If we use the Rectangle super constructor, we need to pass it the values: String color, int length, and int width

- Our Square constructor requires these as parameters if we want to call the super constructor

```java
public class Square extends Rectangle {
    public Square(String color, int length, int width) {

    }//end consructor
}//end class Square
```

Aren't length and width equal for a square?

ORACLE
Academy

Size Parameter

- Although Squares are a type of Rectangle, they have a unique property such that length = width

- Accommodate this by only requiring one size parameter that sets both the width and length values

```java
public class Square extends Rectangle {

    public Square(String color, int size) {
        super(color, size, size);
    }//end constructor
}//end class Square
```

Instead of passing two different parameters of length and width, we can pass a size parameter twice, which will set the length and width values (located in the Rectangle class) to be equal.

Unique Variables for the Subclass

- What about unique variables that apply only to Squares and not Rectangles?

- For example, a feature that tells us whether or not to fill in a Square

- Add a Boolean value in the parameters list to add this unique variable for the Square class:

```java
public class Square extends Rectangle {
    private boolean isFilled;
    public Square(String color, int size, boolean isFilled){
        super(color, size, size);
        this.isFilled = isFilled;
    }//end constructor
}//end class Square
```

The variable isFilled is unique to the Square class and is an example of how subclasses can contain more methods or fields than their superclasses.

Customize Methods

- Because a Square has the same values for height and width, we want to customize the methods setWidth(int width) and setHeight(int height) so that both are updated when the method is called

- Use the keyword super to call the superclass's methods setLength() and setWidth() and set them both to the parameter value passed to the method

```java
public int setWidth(int width){
    super.setLength(width);
    super.setWidth(width);
}//end method setWidth
```

Square Subclass

- The final product will look like the following:

```java
public class Square extends Rectangle {
  private boolean isFilled;
  public Square(String color, int size, boolean isFilled){
      super(color, size, size);
      this.isFilled = isFilled;
  }//end constructor
  public void setLength(int length) {
      super.setLength(length);
      super.setWidth(length);
  }//end method setLength
  public void setWidth(int width) {
      super.setWidth(width);
      super.setLength(width);
  }//end method setWidth
  public boolean getIsFilled() {
      return isFilled;
  }//end method getIsFilled
}//end class Square
```

Inheritance and Applets

- Java applets are another example of using inheritance

- A Java applet is a web-based Java program that can be embedded into a web browser

- The class Applet can be extended to create special applets using some of the core methods in the Applet class

Java Documentation for Applet Class

- Visit the Java Documentation for the Applet class to learn more

- For all documentation, visit:
  - http://docs.oracle.com/javase/8/docs/api/

- For just the class Applet's documentation, visit:
  - http://docs.oracle.com/javase/8/docs/api/java/applet/Applet.html

ORACLE
Academy

Creating Applets

- To create an applet, you can borrow all of the core methods in the Applet class and customize these methods to suit the particular needs of your applet

Creating Applets

- For example, to make an applet that draws Shapes, start by setting up the inheritance with extends:

```
public class DrawShapes extends Applet {
    ...
}//end class DrawShapes
```

- Now our applet class DrawShapes will inherit methods from Applet that we can customize to build the applet

Applet Example

```java
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Graphis2D;
import java.awt.Rectangle;

public class RectangleApplet extends Applet{

    public void paint(Graphics g){
        Graphics2D g2 = (Graphics2D)g;
        Rectangle testRectangle = new Rectangle(5,10,20,30);
        g2.draw(testRectangle);
    }//end method paint

}//end class RectangleApplet
```

Terminology

- Key terms used in this lesson included:
  - Access modifiers
  - Child class
  - default
  - Encapsulation
  - extends
  - Hierarchy
  - Inheritance
  - "is-a" relationship

Terminology

- Key terms used in this lesson included:
  - Parent class
  - private
  - protected
  - public
  - Subclass
  - super
  - Superclass
  - Unified Modeling Language (UML)

Summary

- In this lesson, you should have learned how to:
  - Demonstrate and explain UML (Unified Modeling Language) class diagrams
  - Use the extends keyword to inherit a class
  - Compare and contrast superclasses and subclasses
  - Describe how inheritance affects member access
  - Use super to call a superclass constructor
  - Use super to access superclass members

Summary

- In this lesson, you should have learned how to:
  - Create a multilevel class hierarchy
  - Recognize when constructors are called in a class hierarchy
  - Demonstrate understanding of inheritance through the use of applets
  - Recognize correct parameter changes in an existing applet