Java Fundamentals
7-4: Inheritance
Practice Activities

Lesson 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
- 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

Vocabulary:

Identify the vocabulary word for each definition below.

default	When there is no access modifier. Same access as public, except not visible to other packages.
access modifiers	The keywords used to declare a class, method, or variable as public, private, or protected. Default is when there is no access modifier.
subclasses	Classes that are more specific subsets of other classes and that inherit methods and fields from more general classes.
extends	A keyword in Java that allows you to explicitly declare the superclass of the current class.
encapsulation	A programming philosophy that promotes protecting data and hiding implementation in order to preserve the integrity of data and methods.
private	Visible only to the class where it is declared.
hierarchy	A structure that categorizes and organizes relationships among ideas, concepts of things with the most general or all-encompassing component at the top and the more specific, or component with the narrowest scope, at the bottom.
public	Visible to all classes.
superclass	Classes that pass down their methods to more specialized classes.
inheritance	The concept in object-oriented programming that allows classes to gain methods and data by extending another classes fields and methods.
protected	Visible to the package where it is declared and to subclasses in other packages.
UML	A standardized language for modeling systems and structures in programming.
super	A keyword that allows subclasses to access methods, data, and constructors from their parent class.
is-a	A helpful term used to conceptualize the relationships among nodes or leaves in an inheritance hierarchy.

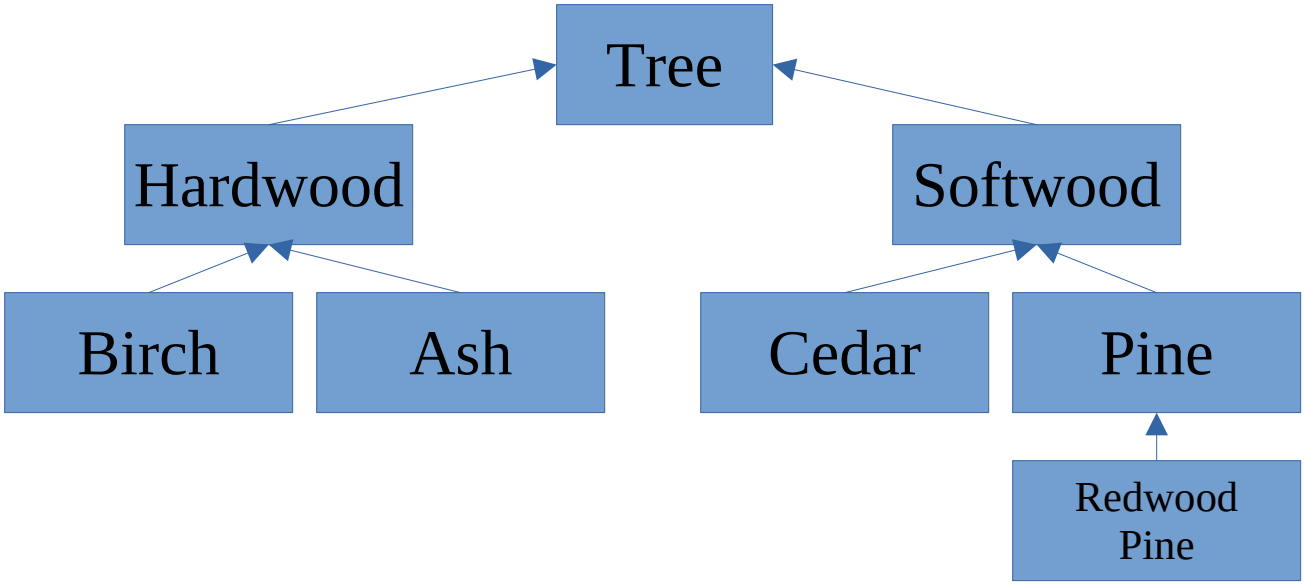
Try It/Solve It:

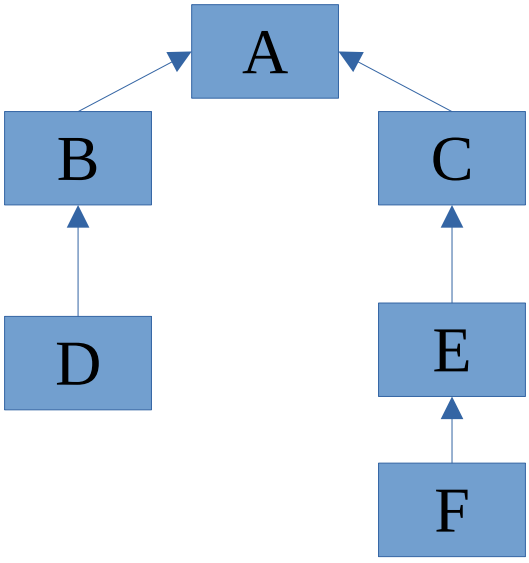
1. Modify the existing applet to change all the colors to black, white, and gray using the code below:

```
import java.awt.*;
import java.applet.*;

public class DrawShapes extends Applet  {
        Font font;
   Color redColor;
   Color blueColor;
   Color backgroundColor;
   public void init() {
      //The Font is Arial size, 18 and is italicized
     font = new Font("Arial",Font.ITALIC,18);

     //Some colors are predefined in the Color class
      redColor = Color.BLACK;
      backgroundColor = Color.WHITE;

      //Colors can be created using Red, Green, Blue values
      blueColor = Color.GREY;

      //Set the background Color of the applet
      setBackground(backgroundColor);
   }
   public void stop() {
   }
   /**
    * This method paints the shapes to the screen
    */
   public void paint(Graphics graph) {
      //Set font
       graph.setFont(font);
       //Create a title
      graph.drawString("Draw Shapes",90,20);

      //Set the color for the first shape
      graph.setColor(blueColor);

      // Draw a rectangle using drawRect(int x, int y, int width, intheight)
      graph.drawRect(120,120,120,120);

      // This will fill a rectangle
      graph.fillRect(115,115,90,90);

      //Set the color for the next shape
      graph.setColor(redColor);

      //Draw a circle using  drawArc(int x, int y, int width, int height, int startAngle, intarcAngle)
      graph.fillArc(110,110,50,50,0,360);

      //Set color for next shape
      graph.setColor(Color.BLACK);

      //Draw another rectangle
      graph.drawRect(50,50,50,50);

      // This will fill a rectangle
      graph.fillRect(50,50,60,60);
   }
}
```

2. Draw simple UML Diagrams with the following classes:

- Tree, Hardwood, Softwood, Birch, Ash, Cedar, Pine, Red Pine, where Hardwood and Softwood are types of trees, Birch and Ash are types of Hardwoods and Cedar and Pine are types of Softwoods. Red Pine is a type of Pine.
- A , B, C, D, E, F, where B and C are a type of A, D is a type of B, E is a type of C, and F is a type of E.





3. Create a class hierarchy representing Students in a university. You know that Students are a type of Person, but Students have more specific characteristics like having a grade point average (GPA), student identification number (ID) and a discipline of study, or major (Mathematics, Computer Science, Literature, Psychology, etc.). Create a subclass of Person called Student using the code for Person below and the specifications listed below:

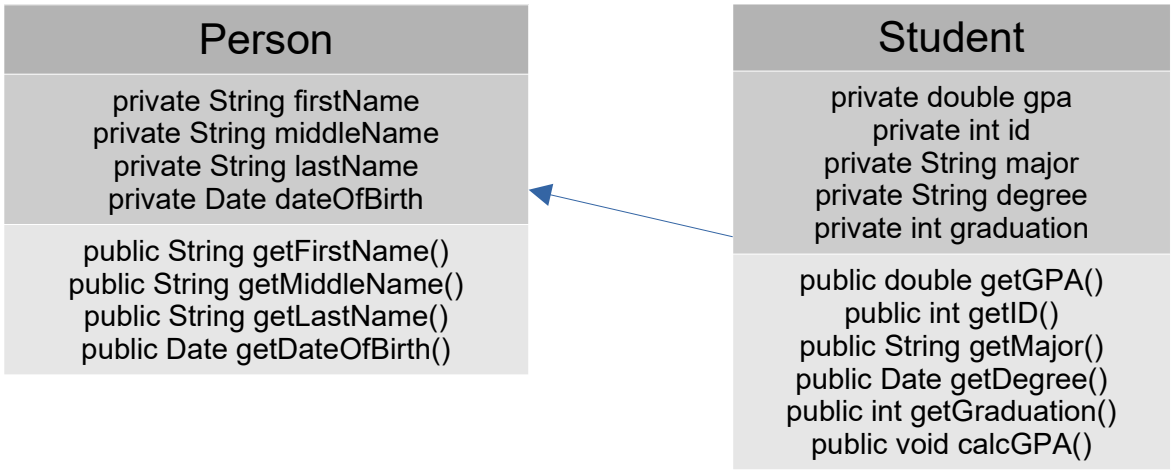
   - Students have personal data that helps identify them to university administrators. Create variables for a Student ID number, a GPA, which gives the average grade points for the student, a major, degree that he or she is earning (Bachelor's, Masters, Ph.D), and his or her expected graduation year. Carefully consider whether these should be public or private data.
   - For methods you declare private, you may want to provide access methods for other classes to retrieve this data.
   - Create a detailed UML diagram listing all of the data and methods of each class (for both Person and Student).
   - Create a method that allows administrators to change a student's major
   - Create a method that calculates a student's GPA by taking in an array of his or her grades. Take the average of all of the student's grades using the following breakdown.

Grade	Grade Point Equivalent
A	4
A-	3.67
B+	3.33
B	3
B-	2.67
C+	2.33
C	2
D	1
F	0

Code for Person class:

```
import java.util.Date;

public class Person {
    private String firstName;
    private String middleName;
    private String lastName;
    private Date dateOfBirth;

    public Person(String firstName, String middleName,
                  String lastName, Date dateOfBirth){
        this.firstName = firstName;
        this.middleName = middleName;
        this.lastName = lastName;
        this.dateOfBirth = dateOfBirth;
    }


    /**
     * Returns a String of the firstName
     * @return firstName
     */
    public String getFirstName(){

        return firstName;
    }

    /**
     * Returns a string of the middleName
     * @return middleName
     */
    public String getMiddleName(){
        return middleName;
    }
```

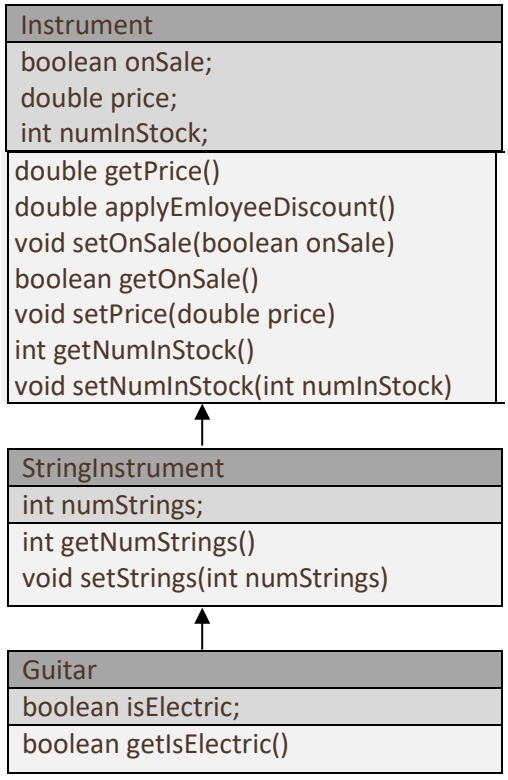
```
    /**
     * Returns a String of the lastName
     * @return lastName
     */
    public String getLastName(){
        return lastName;
    }

    /**
     * Returns a concatenated string of the Person's name
     * @return the Person's first, middle, and last name.
     */
    public String getName(){
        return firstName + " " + middleName + " " + lastName;
    }

    /**
     * Returns the Person's date of birth as a date type
     * @return a Date type of the Person's date of birth.
     */
    public Date getDateOfBirth(){
        return dateOfBirth;
    }
}
```



```
import java.util.Date;

public class Student extends Person{

private int graduation, id;
private double gpa;
private String major, degree;

public Student(String firstName, String middleName, String lastName, Date dateOfBirth,
int id, double gpa, String major, String degree, int graduation){
      super(firstName, middleName, lastName);
      this.id = id;
      this.gpa = gpa;
      this.major = major;
      this.degree = degree;
      this.graduation = graduation;
}

public int getID(){ return this.id; }
public double getGPA() { return this.gpa; }
public String getMajor() { return this.major; }
public String getDegree() { return this.degree; }
public int getGraduation() { return this.graduation; }
public void changeMajor(String major) { this.major = major; }
public void calcGPA(String[] grades) {
      double sum = 0.00;
      for (String g : grades){
            if (g == null) continue;
            switch (g){
                  case “A”: sum += 4.00; break;
                  case “A-”: sum += 3.67; break;
                  case “B+”: sum += 3.33; break;
                  case “B”: sum += 3.00; break;
                  case “C+”: sum += 2.67; break;
                  case “C”: sum += 2.00; break;
                  case “D”: sum += 1.00; break;
            }

      }

}
}
```

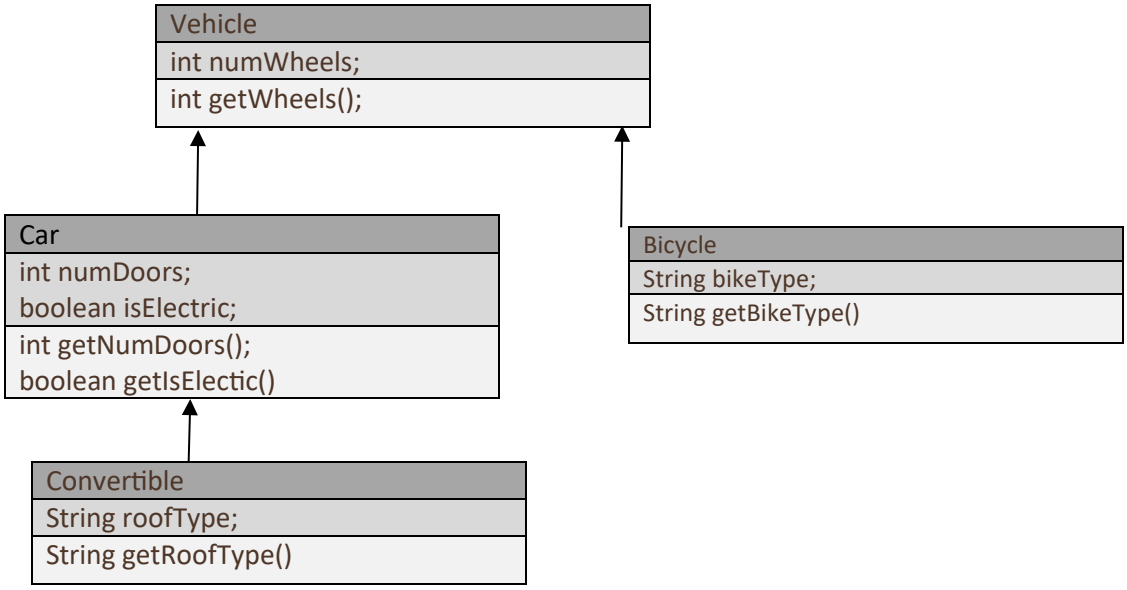
4. True/False – A subclass is able to access this code in the superclass: Why?
   a. public String aString; – public is accessible globally
   b. protected boolean aBoolean; – protected accessible within package
   c. int anInt; – default
   d. private double aDouble; – private not accessible
   e. public String aMethod() – public accessible to children
   f. private class aNestedClass – private inaccessible
   g. public aClassConstructor() – using super()
5. Imagine you own a music shop that sells musical instruments. Create classes representing an inheritance hierarchy of musical instruments using the following UML diagram:

   - Employee discount is 25% off the instrument (Hint: 75% of the initial cost).
   - If an item is onSale, then the price returned for getPrice() should be 15% off.



```
public class Instrument{
	private boolean onSale;
	private double price;
	private int numInStock;

	public Instrument(boolean onSale, double price, int numInStock){
		this.onSale = onSale;
		this.price = price;
		this.numInStock = numInStock;
	}

	public double getPrice() { return this.price; }
	public boolean getOnSale() { return this.onSale; }
	public int getNumInStock() { return this.numInStock; }
	public double applyEmployeeDiscount() { return price * 0.75; }
	public double getPrice() { return (onSale ? : price * 0.85 : price); }
	public void setOnSale(boolean sale) { onSale = sale; }
	public void setNumInStock(int num) { numInStock = num; }
}

public class StringInstrument extends Instrument{
	private int numStrings;
	public StringInstrument(boolean onSale, double price, int numInStock, int strings){
		super(onsale, price, numInStock);
		numStrings = strings;
	}
	public int getNumStrings() { return this.numStrings; }
	public void setNumStrings(int strings) { numStrings = strings; }
}

public class Guitar extends StringInstrument{
	private boolean isElectric;
	public Guitar(boolean sale, double price, int stock, int s, boolean electric){
		super(sale, price, stock, s);
		isElectric = electric;
	}
	public boolean getIsElectric() { return isElectric; }
}
```

6. Using the code and UML diagram below, fill in the blanks with the correct keywords or class names. If a keyword is missing from the code, fill in the keyword that fits. If the class or data is missing from the UML, fill in the correct information.



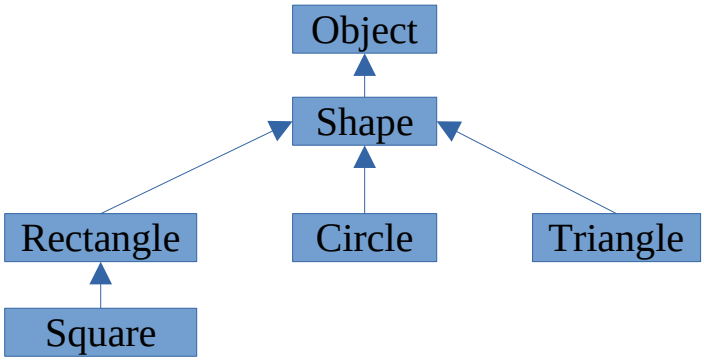
```
public class Vehicle {
    private int numWheels;
    public Vehicle(int numWheels)
     {
        this.numWheels = numWheels;
    }

    public int getWheels() {
        return wheels;
    }
}

public class Car extends Vehicle {
    private int numDoors;
    private boolean isElectric;

    public Car (int numWheels, int numDoors, boolean isElectric) {
        super(numWheels);
        this.numDoors = numDoors;
        this.isElectric = isElectric;
    }

    public int getNumDoors() {
        return numDoors;
    }

    public boolean getIsElectric() {
        return isElectric;
    }
}


public class Bicycle extends Vehicle {
    //Mountain bike, road bike, recumbent bike.. etc
    private String bikeType;

    public Bicycle(int numWheels, String bikeType) {
        super(numWheels);
        this.bikeType = bikeType
    }

    public String getBikeType() {
        return bikeType;
    }
}

public class Convertible extends Car {
    //Soft top or rag top, or hard top
    private String roofType;

    public Convertible(int numWheels, int numDoors, boolean isElectric, String roofType) {
        super(numWheels, numDoors isElectric
        this.roofType = roofType;
    }

    public String getRoofType() {
        return roofType;
    }
}
```

7. Given the classes Shape, Rectangle, Circle, Triangle, Square, and Object, write a UML diagram using the following relationships:

   - A Square is a Rectangle
   - A Rectangle, Triangle, and Circle are all Shapes

   Use the simple way of diagramming a class.

   - **Hint:** Start with the broadest, or most general class.



8. Given the classes Crab, Crustacean, Mammal, Dog, Hermit Crab, Animal, and Object, write a UML diagram using the following relationships:

   - A Crab is a Crustacean.
   - A Hermit Crab is a Crab.
   - A Dog is a Mammal.
   - Mammals and Crustaceans are types of Animals.

   Use the simple way of diagramming a class.

   - **Hint**: Start with the broadest or most general class.