Java Fundamentals 7-4: Inheritance Practice Activities

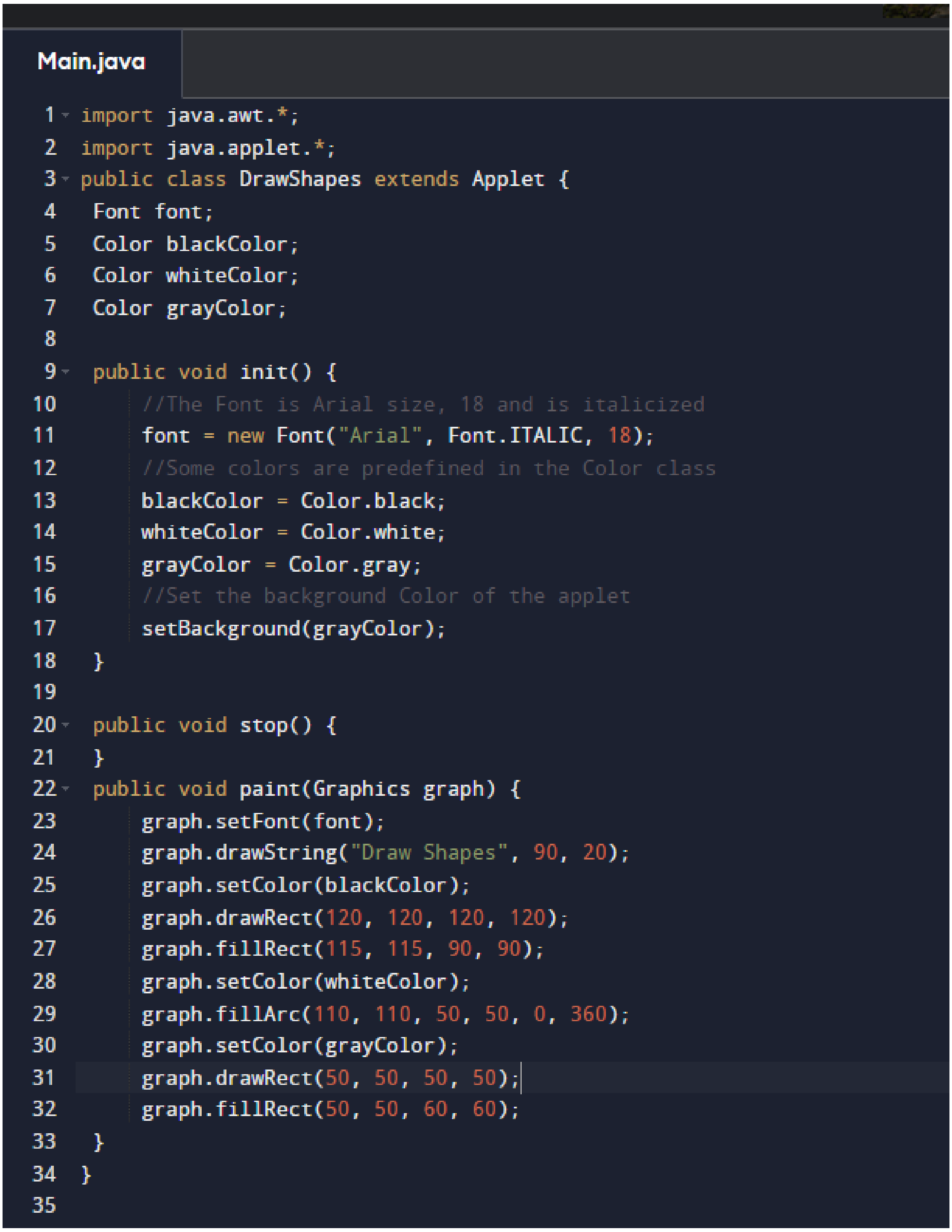
Vocabulary Definitions

1. **Default access** - When there is no access modifier. Same access as public, except not visible to other packages.
2. **Access modifiers** - The keywords used to declare a class, method, or variable as public, private, or protected.
3. **Default** - When there is no access modifier.
4. **Subclasses** - Classes that are more specific subsets of other classes and that inherit methods and fields from more general classes.
5. **extends** - A keyword in Java that allows you to explicitly declare the superclass of the current class.
6. **Encapsulation** - A programming philosophy that promotes protecting data and hiding implementation in order to preserve the integrity of data and methods.
7. **Private** - Visible only to the class where it is declared.
8. **Hierarchy** - A structure that categorizes and organizes relationships among ideas, concepts, or 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.
9. **Public** - Visible to all classes.
10. **Superclasses** - Classes that pass down their methods to more specialized classes.
11. **Inheritance** - The concept in object-oriented programming that allows classes to gain methods and data by extending another class's fields and methods.
12. **Protected** - Visible to the package where it is declared and to subclasses in other packages.
13. **UML** - A standardized language for modeling systems and structures in programming.
14. **super** - A keyword that allows subclasses to access methods, data, and constructors from their parent class.
15. **Tree** - 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

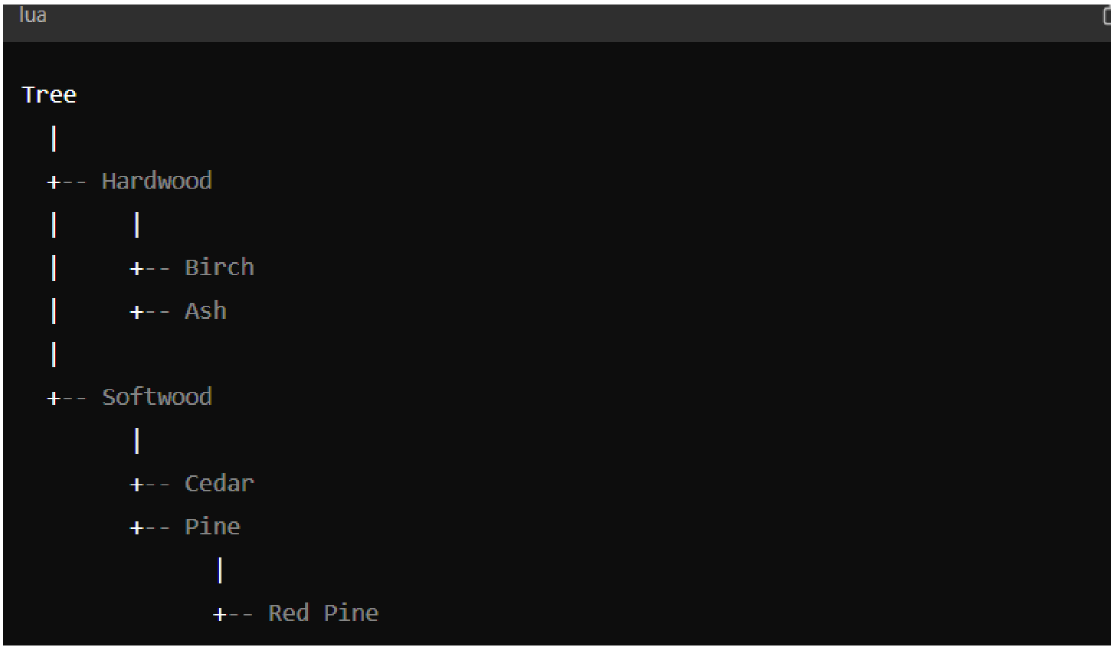
Here is the modified code:



```java
import java.awt.*;
import java.applet.*;
public class DrawShapes extends Applet {
 Font font;
 Color blackColor;
 Color whiteColor;
 Color grayColor;

 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
     blackColor = Color.black;
     whiteColor = Color.white;
     grayColor = Color.gray;
     //Set the background Color of the applet
     setBackground(grayColor);
 }

 public void stop() {
 }
 public void paint(Graphics graph) {
     graph.setFont(font);
     graph.drawString("Draw Shapes", 90, 20);
     graph.setColor(blackColor);
     graph.drawRect(120, 120, 120, 120);
     graph.fillRect(115, 115, 90, 90);
     graph.setColor(whiteColor);
     graph.fillArc(110, 110, 50, 50, 0, 360);
     graph.setColor(grayColor);
     graph.drawRect(50, 50, 50, 50);
     graph.fillRect(50, 50, 60, 60);
 }
}

```

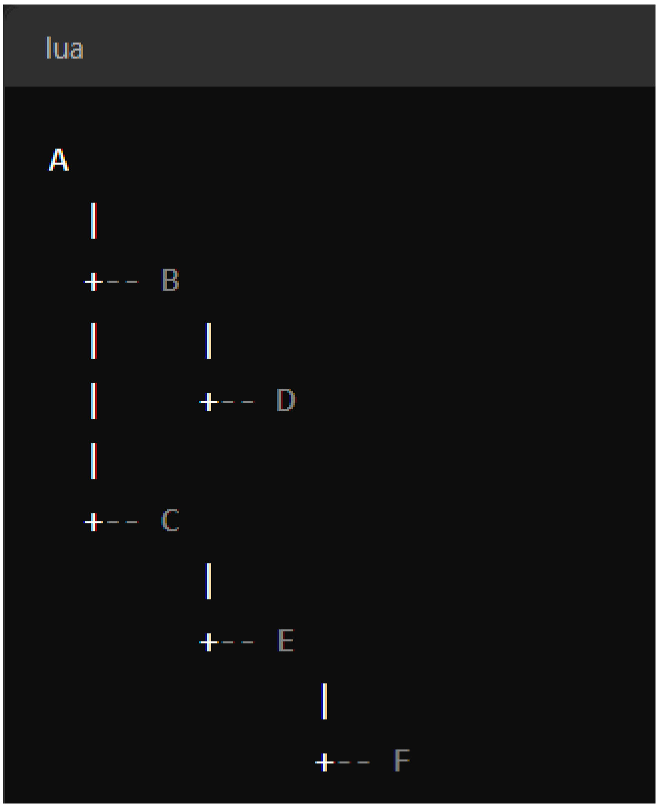
2. Draw simple UML Diagrams with the following classes

Tree Hierarchy UML:



```
Tree
  |
  +-- Hardwood
  |     |
  |     +-- Birch
  |     +-- Ash
  |
  +-- Softwood
        |
        +-- Cedar
        +-- Pine
              |
              +-- Red Pine
```

Class Hierarchy UML:

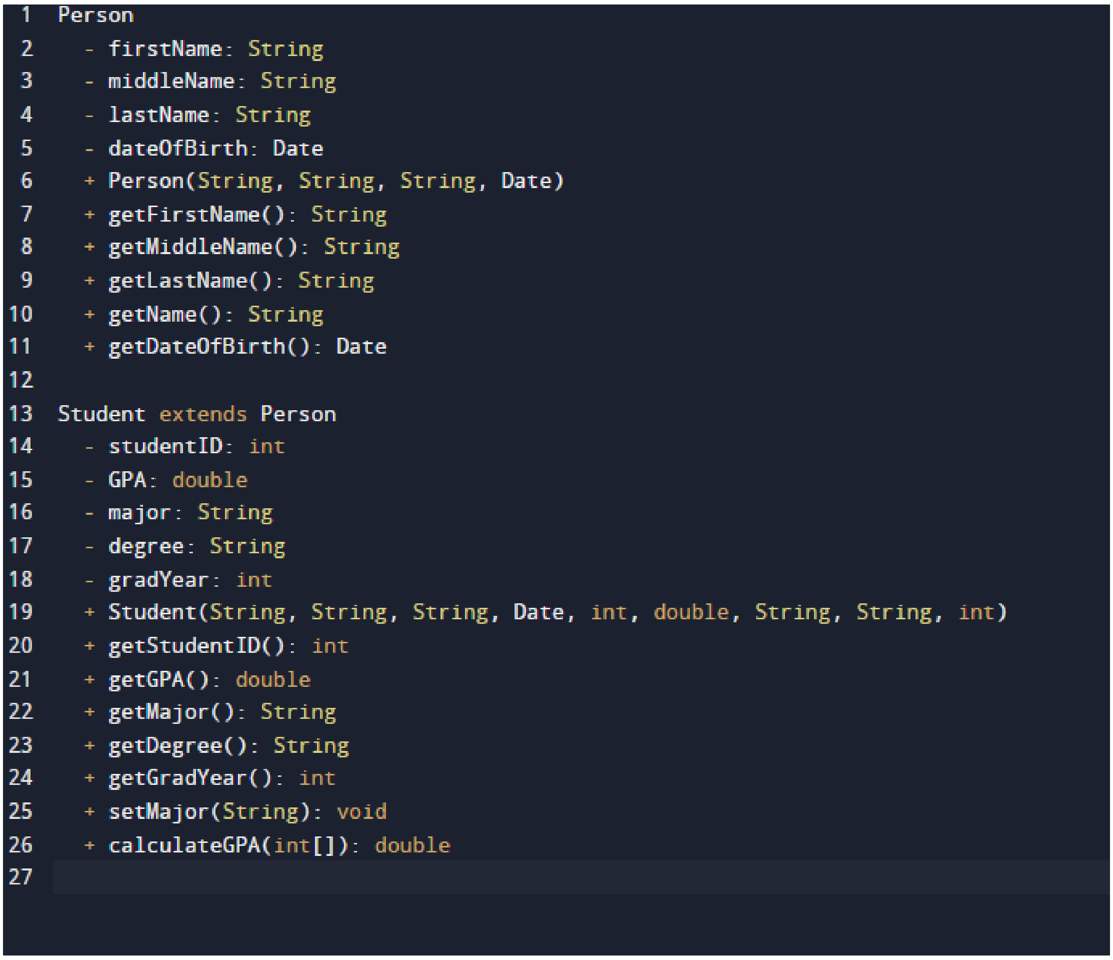


```
A
  |
  +-- B
  |     |
  |     +-- D
  |
  +-- C
        |
        +-- E
              |
              +-- F
```

.

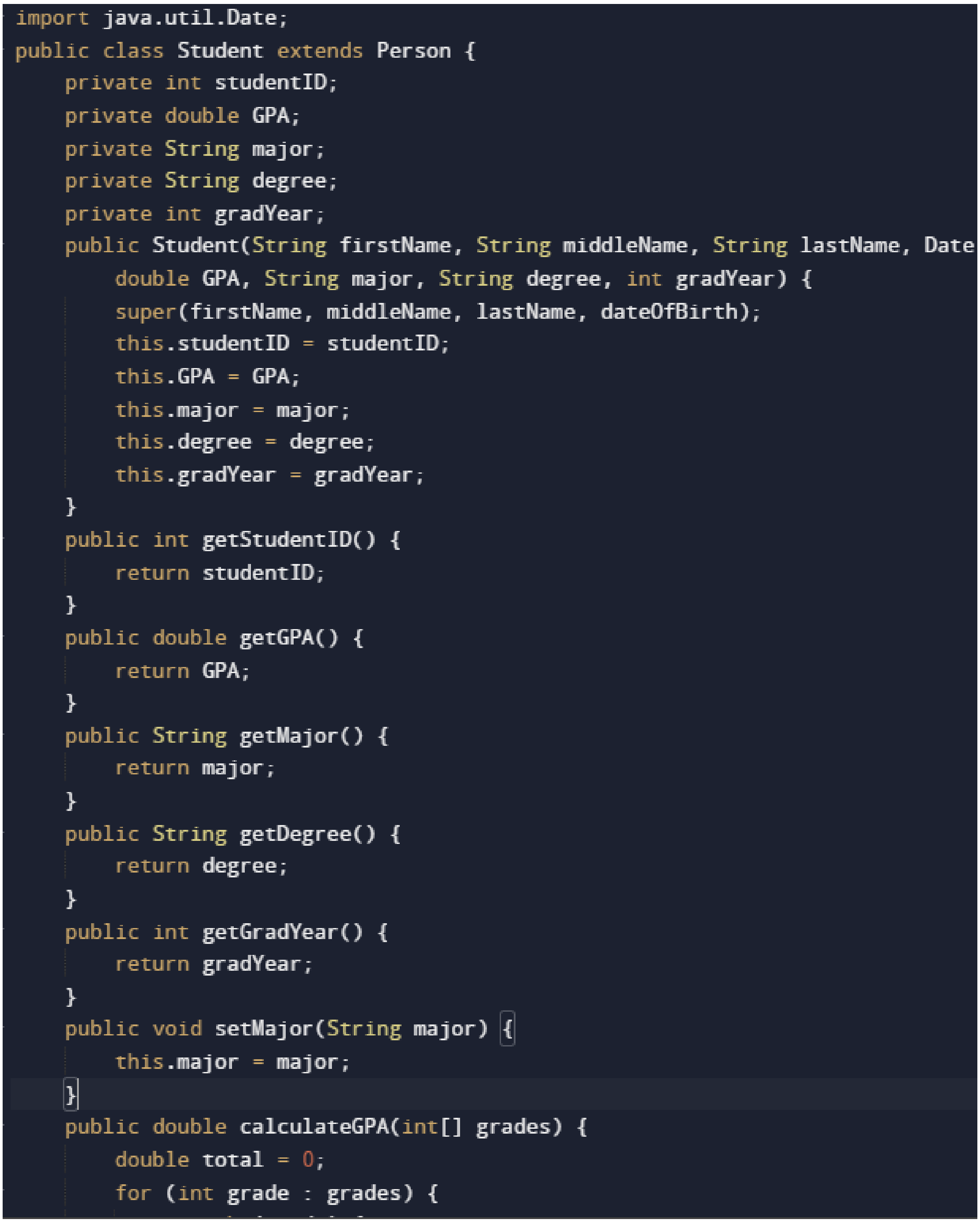
3. Create a class hierarchy representing Students in a university

UML Diagram:

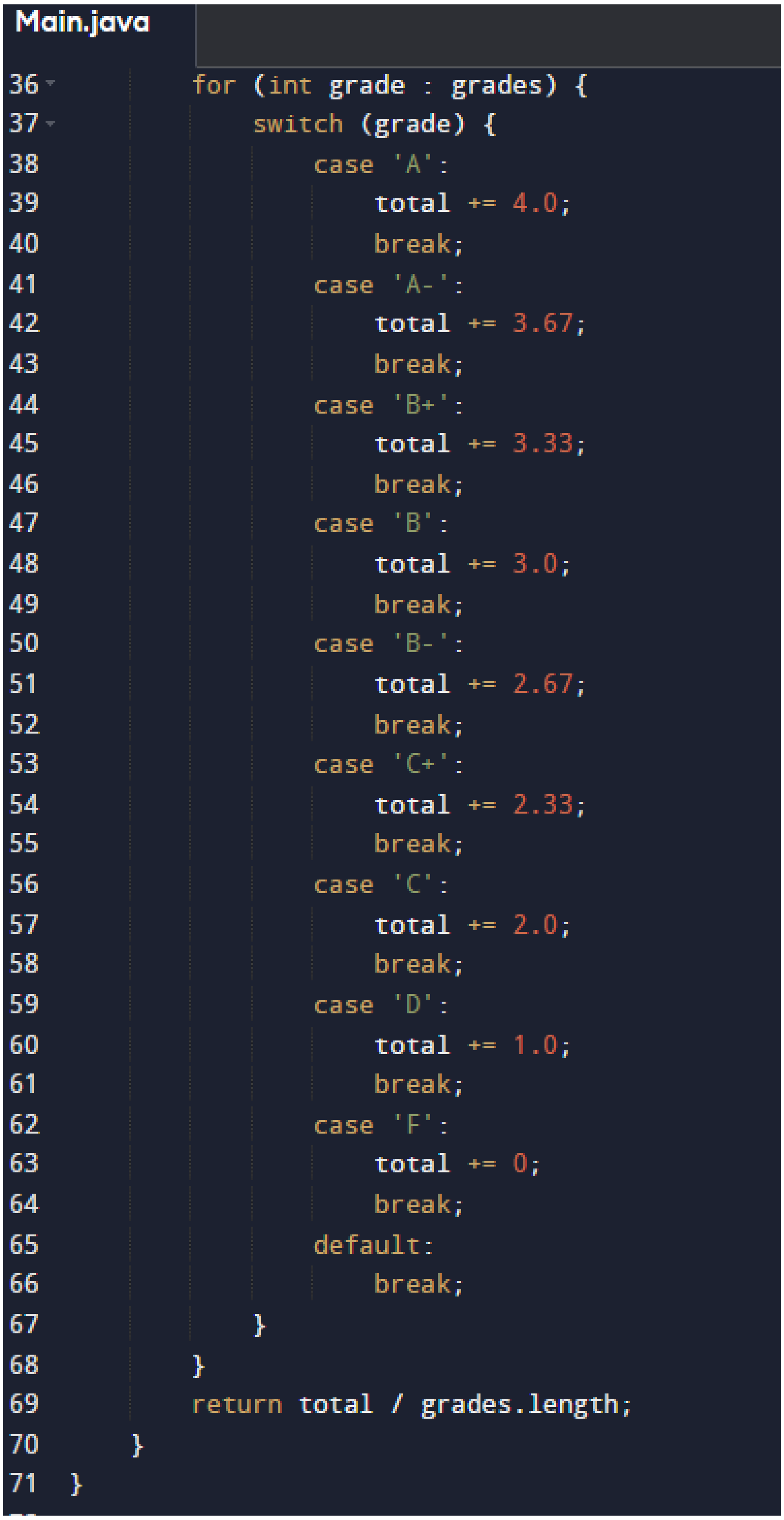


```
Person
  - firstName: String
  - middleName: String
  - lastName: String
  - dateOfBirth: Date
  + Person(String, String, String, Date)
  + getFirstName(): String
  + getMiddleName(): String
  + getLastName(): String
  + getName(): String
  + getDateOfBirth(): Date

Student extends Person
  - studentID: int
  - GPA: double
  - major: String
  - degree: String
  - gradYear: int
  + Student(String, String, String, Date, int, double, String, String, int)
  + getStudentID(): int
  + getGPA(): double
  + getMajor(): String
  + getDegree(): String
  + getGradYear(): int
  + setMajor(String): void
  + calculateGPA(int[]): double
```

Code for the Student class:



```java
import java.util.Date;
public class Student extends Person {
    private int studentID;
    private double GPA;
    private String major;
    private String degree;
    private int gradYear;
    public Student(String firstName, String middleName, String lastName, Date
        double GPA, String major, String degree, int gradYear) {
        super(firstName, middleName, lastName, dateOfBirth);
        this.studentID = studentID;
        this.GPA = GPA;
        this.major = major;
        this.degree = degree;
        this.gradYear = gradYear;
    }
    public int getStudentID() {
        return studentID;
    }
    public double getGPA() {
        return GPA;
    }
    public String getMajor() {
        return major;
    }
    public String getDegree() {
        return degree;
    }
    public int getGradYear() {
        return gradYear;
    }
    public void setMajor(String major) {
        this.major = major;
    }
    public double calculateGPA(int[] grades) {
        double total = 0;
        for (int grade : grades) {
```



Main.java

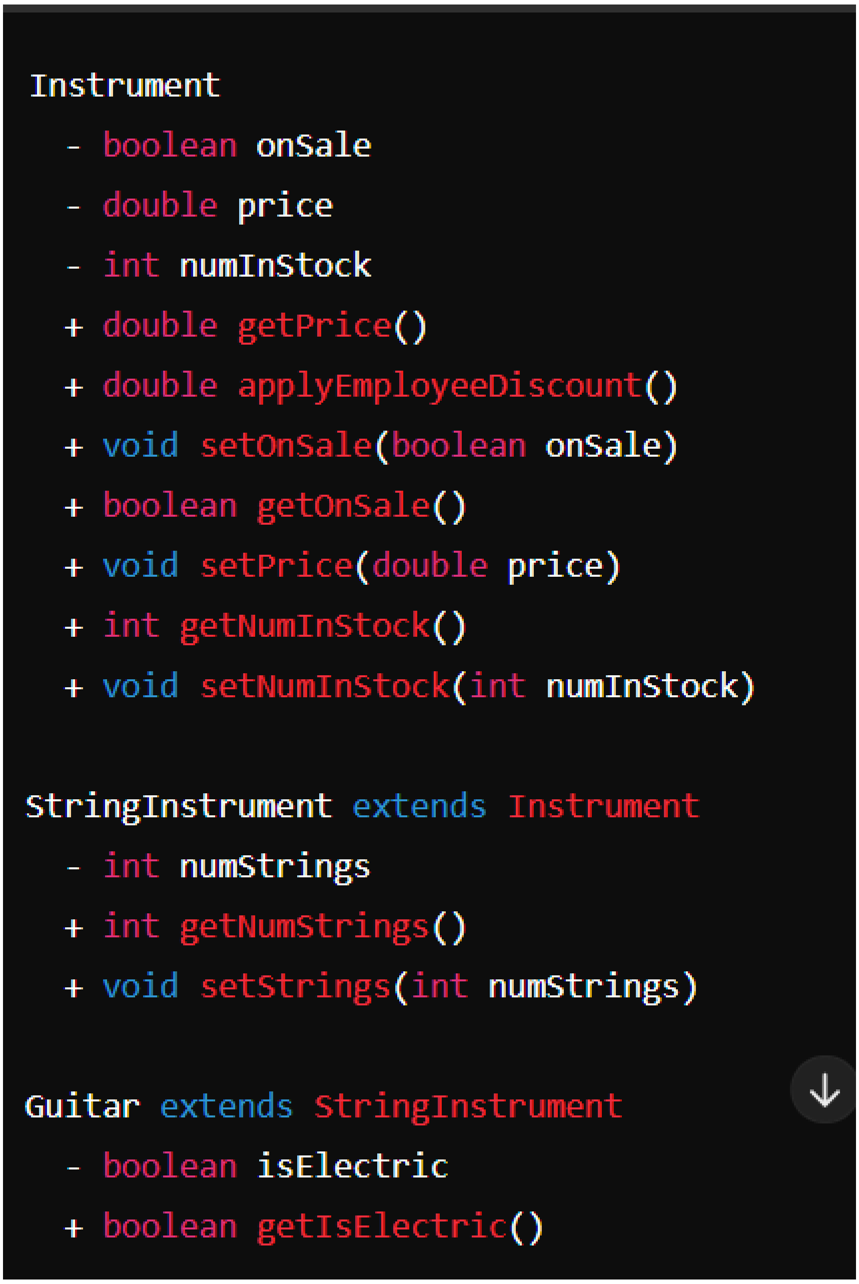
```java
        for (int grade : grades) {
            switch (grade) {
                case 'A':
                    total += 4.0;
                    break;
                case 'A-':
                    total += 3.67;
                    break;
                case 'B+':
                    total += 3.33;
                    break;
                case 'B':
                    total += 3.0;
                    break;
                case 'B-':
                    total += 2.67;
                    break;
                case 'C+':
                    total += 2.33;
                    break;
                case 'C':
                    total += 2.0;
                    break;
                case 'D':
                    total += 1.0;
                    break;
                case 'F':
                    total += 0;
                    break;
                default:
                    break;
            }
        }
        return total / grades.length;
    }
}
```

4. True/False – A subclass is able to access this code in the superclass: Why?

a. `public String aString;` - **True**: Public members are accessible to subclasses. b. `protected boolean aBoolean;` - **True**: Protected members are accessible to subclasses. c. `int anInt;` - **True**: Default (package-private) members are accessible if the subclass is in the same package. d. `private double aDouble;` - **False**: Private members are not accessible to subclasses. e. `public String aMethod()` - **True**: Public methods are accessible to subclasses. f. `private class aNestedClass` - **False**: Private nested classes are not accessible to subclasses. g. `public aClassConstructor()` - **True**: Public constructors are accessible to subclasses.

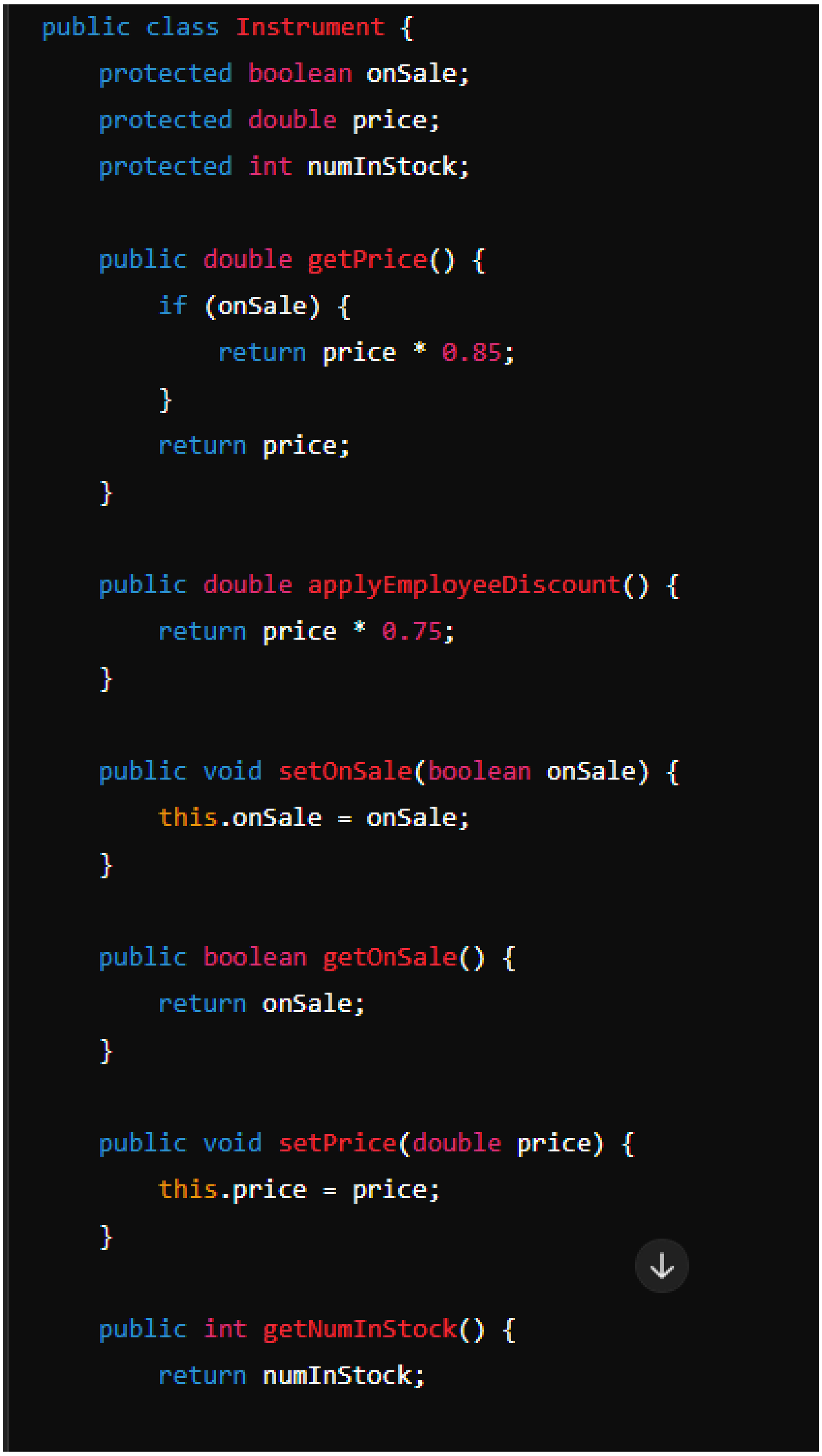
5. Create classes representing an inheritance hierarchy of musical instruments

UML Diagram:



```
Instrument
  - boolean onSale
  - double price
  - int numInStock
  + double getPrice()
  + double applyEmployeeDiscount()
  + void setOnSale(boolean onSale)
  + boolean getOnSale()
  + void setPrice(double price)
  + int getNumInStock()
  + void setNumInStock(int numInStock)

StringInstrument extends Instrument
  - int numStrings
  + int getNumStrings()
  + void setStrings(int numStrings)

Guitar extends StringInstrument
  - boolean isElectric
  + boolean getIsElectric()
```

Code for the classes:



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

    public double getPrice() {
        if (onSale) {
            return price * 0.85;
        }
        return price;
    }

    public double applyEmployeeDiscount() {
        return price * 0.75;
    }

    public void setOnSale(boolean onSale) {
        this.onSale = onSale;
    }

    public boolean getOnSale() {
        return onSale;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    public int getNumInStock() {
        return numInStock;
```