Fundamentos de Java
7-4: Herança
Atividades Práticas

Objetivos da Lição:

- Demonstrar e explicar os diagramas de classe UML (Unified Modeling Language)
- Usar a palavra-chave prolongada para herdar uma classe
- Comparar e contrastar superclasses e subclasses
- Descrever como a herança afeta o acesso de um membro
- Usar o super para chamar um construtor de superclasse
- Usar o super para acessar membros da superclasse
- Criar uma hierarquia de classes de vários níveis
- Reconhecer quando os construtores são chamados em uma classe hierárquica
- Demonstrar compreensão da herança com o uso de applets
- Reconhecer alterações corretas do parâmetro em um applet existente

Vocabulário:

Identifique a palavra do vocabulário para cada definição a seguir.

	Quando não há modificador de acesso. Mesmo acesso que o público, porém não é visível a outros pacotes.
	As palavras-chave usadas para declarar uma classe, método ou variável como pública, privada ou protegida. Padrão quando não há modificador de acesso.
	Classes que são subconjuntos mais específicos de outras classes e que herdam métodos e campos de classes mais gerais.
	Uma palavra-chave em Java que permite que você declare explicitamente a superclasse da classe atual.
	Uma filosofia de programação que promove a proteção de dados e omissão de implementação a fim de preservar a integridade dos dados e métodos.
	Visível somente à classe onde ele é declarado.
	Uma estrutura que categoriza e organiza as relações entre ideias, conceitos das coisas com o componente mais geral ou abrangente na parte superior e o componente mais específico, ou com o escopo mais restrito, na parte inferior.
	Visível a todas as classes.
	Classes que passam seus métodos para classes mais especializadas.
	O conceito em programação orientada por objetos que permite que as classes obtenham métodos e dados estendendo campos e métodos de outras classes.
	Visível ao pacote onde ele está declarado e a subclasses em outros pacotes.
	Uma linguagem padronizada para modelar sistemas e estruturas em programação.
	Uma palavra-chave que permite que subclasses acessem métodos, dados e construtores da classe pai.
	Um termo útil usado para conceituar as relações entre nós ou ramificações em uma hierarquia de herança.

Tente/solucione:

1. Modifique o applet existente para alterar todas as cores para preto, branco e cinza usando o código abaixo:

```
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.red;
        backgroundColor = Color.orange;

        //Colors can be created using Red, Green, Blue values
        blueColor = new Color(0,0,122);

        //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, int height)
        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, int arcAngle)
        graph.fillArc(110,110,50,50,0,360);

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

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

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

2. Desenhe Diagramas UML simples com as seguintes classes:

    - Árvore, Madeira de Lei, Madeira Conífera, Bétula, Freixo, Cedro, Pinho, Pinho Vermelho, onde Madeira de Lei e Madeira Conífera são tipos de árvores, Bétula e Freixo são tipos de madeira de lei, e Cedro e Pinho são tipos de Madeiras Coníferas. Pinho Vermelho é um tipo de Pinho.
    - A , B, C, D, E, F, onde B e C são tipos de A, D é um tipo de B, E é um tipo de C e F é um tipo de E.

3. Crie uma hierarquia de classes que represente Alunos de uma universidade. Você sabe que Alunos são um tipo de Pessoa, mas Alunos têm características mais específicas, como ter uma Média Geral, número de identificação (ID) escolar e uma disciplina de estudo, ou carreira (Matemática, Informática, Literatura, Psicologia, etc.). Crie uma subclasse de Pessoas chamada Aluno usando o código de Pessoa abaixo e as especificações listadas abaixo:

    - Alunos têm dados pessoais que ajudam a identificá-los pelos administradores da universidade. Crie variáveis para um número do ID do Aluno, uma média geral, que dê os pontos da média para o aluno, uma carreira, um diploma que o aluno esteja adquirindo (Bacharel, Mestrado, Doutorado) e o ano previsto da formatura. Analise com cuidado se esses dados devem ser públicos ou privados.
    - Para os métodos que você declarar privados, forneça métodos de acesso a outras classes para recuperar esses dados.
    - Crie um diagrama de UML detalhado listando todos os dados e métodos de cada classe (tanto para Pessoa quanto para Aluno).
    - Crie um método que permita que os administradores alterem o curso de um aluno
    - Crie um método que calcule a média geral de um aluno considerando um array de suas notas. Calcule a média de todas as notas do aluno usando a divisão a seguir.

Nota	Equivalente da Pontuação da Nota
A	4
A-	3.67
B+	3.33
B	3
B-	2.67
C+	2.33
C	2
D	1
F	0

Código da classe Pessoa:

```
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;
    }
}
```

4. Verdadeiro/Falso – Uma subclasse pode acessar este código na superclasse: Por quê?

    a. public String aString;

    b. protected boolean aBoolean;

    c. int anInt;

    d. private double aDouble;

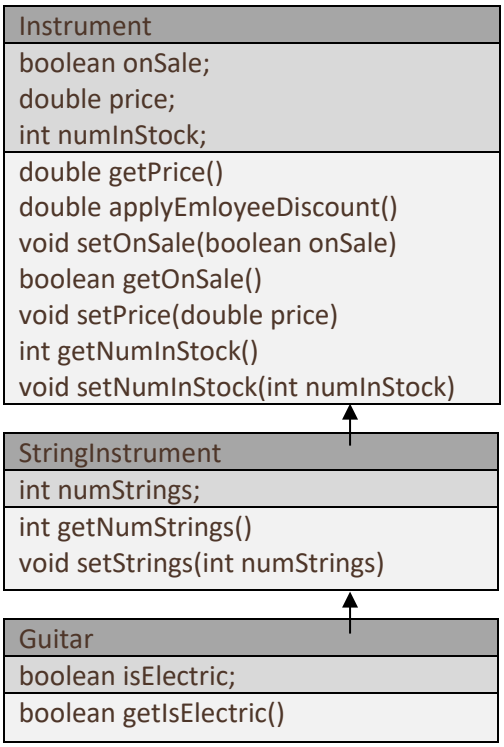
    e. public String aMethod()

    f. private class aNestedClass

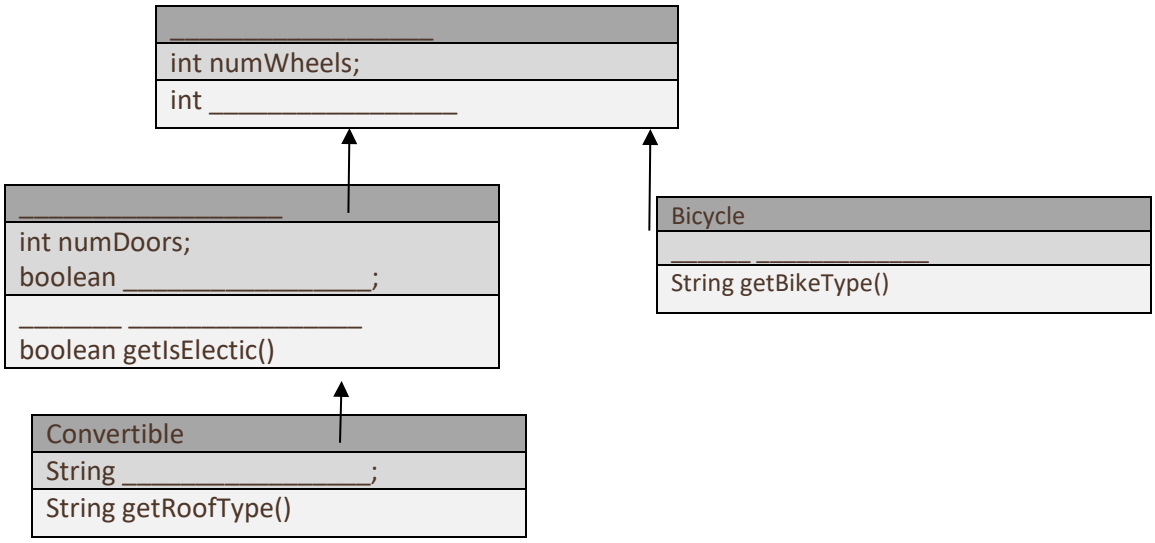
    g. public aClassConstructor()

5. Imagine que você tenha uma loja de música que vende instrumentos musicais. Crie classes que representem uma hierarquia de herança de instrumentos musicais usando o seguinte diagrama UML:

    - O desconto para funcionários é de 25% em instrumentos (Dica: 75% do custo inicial).
    - Se um item estiver onSale, o preço retornado de getPrice() deve ter um desconto de 15%.



6. Usando o código e diagrama UML abaixo, preencha as lacunas com as palavras-chave ou nomes de classes corretos. Se faltar uma palavra-chave na código, preencha com a palavra-chave adequada. Se faltar alguma classe ou algum dado no UML, preencha com as informações corretas.




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

    public int getWheels() {
        return wheels;
    }
}

public class Car _________ Vehicle {
    _________ int numDoors;
    _________ boolean isElectric;

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

    public int getNumDoors() {
        return _________;
    }

    public boolean getIsElectric() {
        return isElectric;
    }
}

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

    public Bicycle(int numWheels, String bikeType) {
```

```
        super(numWheels);
        this.bikeType = _________;
    }

    public _________ getBikeType() {
        return bikeType;
    }
}

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

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

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

7. Dadas as classes Forma, Retângulo, Círculo, Triângulo, Quadrado e Objeto, escreva um diagrama UML usando as seguintes relações:

    - Um quadrado é um retângulo
    - Retângulo, Triângulo e Círculo são todos Formas

    Use a forma simples de diagramar uma classe.

    - **Dica:** Comece com a classe mais ampla, mais genérica.

8. Dadas as classes Caranguejo, Crustáceo, Mamífero, Cachorro, Caranguejo Ermitão, Animal e Objeto, escreva um diagrama UML usando as seguintes relações:

    - Caranguejo é um Crustáceo.
    - Caranguejo Ermitão é um Caranguejo.
    - Cachorro é Mamífero.
    - Mamíferos e Crustáceos são tipos de Animais.

    Use a forma simples de diagramar uma classe.

    - **Dica:** Comece com a classe mais ampla, mais genérica.