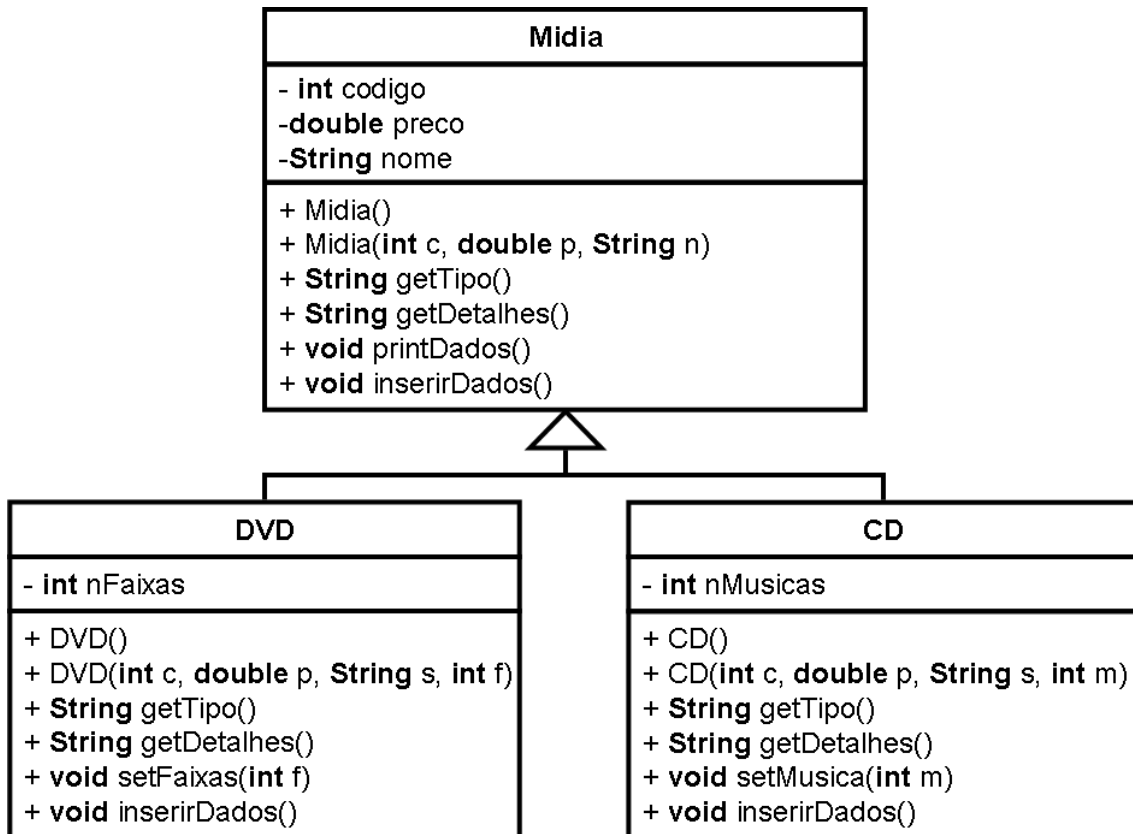


**Exercício 1:** Uma loja que vende CD e DVDS deseja construir um cadastro com dos seus produtos. Para tanto, foi elaborado o diagrama de classes dado na Figura 1.1.



**Figura 1.1:** Hierarquia de classes para construir um cadastro de mídias.

A Tabela 1.1 fornece uma descrição dos métodos que deverão ser elaborados para cada uma das classes.

Método	Descrição
getTipo()	Retorna uma String com o nome da classe.
getDetalhes() ( )	Retorna uma String com as informações contidas nos campos.
printDados() ( )	Imprime as informações contidas nos campos da classe. Para tanto, usa dois métodos para recuperar estas informações: getTipo() e getDetalhes(). Estas funções por sua vez são polimórficas, ou seja, seu tipo retorno varia de acordo com a classe escolhida, tal que este método é sobreposto nas subclasses.
inserirDados() ( )	Insere os dados necessários para se preencher os campos de um objeto de uma dada classe. Seu comportamento é polimórfico.

**Tabela 1.1:** Descrição dos métodos a serem implementados. Além dos métodos descritos na Tabela 1.1, deverão ser criados os métodos **get** e **set** correspondentes para retorna e modificar o conteúdo dos campos, respectivamente, bem como os construtores com e sem parâmetros de cada classe. Criar um programa que simule o uso de um cadastro de CD e DVDs.

### Classe Midia

```
import java.util.Scanner;

public class Midia
{
    private int codigo;
    private double preco;
    private String name;

    // Métodos para inserir valores nos campos.
    public void setCodigo(int codigo)
    { this.codigo = codigo; }

    public void setPreco(double preco)
    { this.preco = preco; }

    public void setName(String name)
    { this.name = name; }

    // Métodos para retornar os valores contidos nos campos.
    public int getCodigo()
    { return codigo; }

    public double getPreco()
    { return preco; }

    public String getName()
    { return name; }

    // Construtor sem parâmetros.
    public Midia()
    {
        this(0,0.0,"Nenhum"); // Chamada ao construtor com param.
    }

    // Construtor com parâmetros.
    public Midia(int codigo, double preco, String name)
    {
        setCodigo(codigo);
        setPreco(preco);
        setName(name);
    }
}
```

```

// Função para impressao dos dados do tipo.
public String getTipo()
{
    return "Midia: ";
}

// Função que retorna o conteúdo do campos em forma de String.
public String getDetalhes()
{
    return "Codigo: " + getCodigo() + "\n" +
           "Preco: " + getPreco() + "\n" +
           "Nome: " + getName() + "\n";
}

// Função para impressao dos dados via getDetalhes().
public void printDados()
{
    String s = getTipo() + "\n" + getDetalhes() + "\n";
    System.out.println(s);
}

// Função para leitura dos dados via teclado.
public void inserirDados()
{
    Scanner in = new Scanner(System.in);

    // Leitura dos dados do teclado.
    System.out.printf("\n Entre com o codigo: ");
    int cod = in.nextInt();
    System.out.printf("\n Entre com o preco: ");
    double pre = in.nextDouble();
    in.nextLine(); // Leitura do enter.
    System.out.printf("\n Entre com o nome: ");
    String nam = in.nextLine();

    // Enviando os dados lidos para as funcoes set.
    setCodigo(cod);
    setPreco(pre);
    setName(nam);
}

} // Fim Classe Mídia.

```

## Classe CD

```
import java.util.Scanner;

public class CD extends Midia
{
    private int nMusicas;

    // Construtor sem parâmetros.
    public CD()
    {   this(0,0.0,"Nenhum",0);   // Chamada ao construtor com param.
    }

    // Construtor com parâmetros.
    public CD(int codigo, double preco, String name, int nMusicas)
    {
        // Chamada ao construtor da classe Midia.
        super(codigo, preco, name);
        setMusica(nMusicas);
    }

    //Função para impressao do tipo.
    public String getTipo()
    { return "CD: "; }

    // Função que retorna o conteúdo do campos desta
    // classe e da classe Midia (usando super !).
    public String getDetalhes()
    { return super.getDetalhes() + "\n" +
      "Numero de musicas: " + nMusicas + "\n";
    }

    public void setMusica(int nmus)
    {   nMusicas = (nmus > 0) ? nmus : 0; }

    // Função para leitura dos dados via teclado dos
    // campos desta classe e dos campos da classe
    // Midia (usando super !).
    public void inserirDados()
    {   // Leitura dos dados contidos nos campos pertencentes a classe
        Midia.
        super.inserirDados();

        Scanner in = new Scanner(System.in);
        // Leitura dos dados do teclado.
        System.out.printf("\n Entre com o numero de musicas: ");
        int nmus = in.nextInt();

        // Enviando os dados lidos para as funcoes set.
        setMusica(nmus); }
}
```

```
}
```

## Classe DVD

```
import java.util.Scanner;

public class DVD extends Midia
{
    private int nFaixas;

    // Construtor sem parâmetros.
    public DVD()
    { this(0,0.0,"Nenhum",0); // Chamada ao construtor com param. }

    // Construtor com parâmetros.
    public DVD(int codigo, double preco, String name, int nFaixas)
    { super(codigo, preco, name); // Chamada ao construtor da
    classe Midia.
      setFaixas(nFaixas); }

    //Função para impressao do tipo.
    public String getTipo()
    { return "DVD: "; }

    // Função que retorna o conteúdo do campos desta
    // classe e da classe Midia (usando super !).
    public String getDetalhes()
    { return super.getDetalhes() + "\n" +
      "Numero de faixas: " + nFaixas + "\n"; }

    public void setFaixas(int nfaix)
    { nFaixas = (nfaix > 0) ? nfaix:0; }

    // Função para leitura dos dados via teclado dos
    // campos desta classe e dos campos da classe
    // Midia (usando super !).
    public void inserirDados()
    {
        // Leitura dos dados contidos nos campos
        // pertencentes a classe Midia.
        super.inserirDados();

        Scanner in = new Scanner(System.in);

        // Leitura dos dados do teclado.
        System.out.printf("\n Entre com o numero de faixas: ");
        int nfaix = in.nextInt();

        // Enviando os dados lidos para as funcoes set.
        setFaixas(nfaix);
    }
}
```

```
} // Fim classe DVD.
```

### Classe TestaMidia

```
import java.util.Scanner;

public class TestaMidia
{
    public static void main(String args[])
    {
        // Cria um vetor de elementos que são objetos da classe Midia.
        Midia[] lista = new Midia[10];

        int opcao;

        // Preenchendo o vetor com CDs.
        for (int i=0; i < 2; i++)
        {
            // Usuário escolhe se quer cadastrar CD ou DVD.
            System.out.printf("Digite 1 para CD e 2 para DVD");
            Scanner in = new Scanner(System.in);
            opcao = in.nextInt();
            if (1 == opcao) // Criar CD.
                lista[i] = new CD();
            else // Criar DVD.
                lista[i] = new DVD();
            lista[i].inserirDados(); // Inserir dados no objeto criado seja CD ou
            DVD.
        }

        // Imprimindo o conteudo de cada elemento do vetor de acordo
        com a
        // classe a que ele pertence (isto e, usando polimorfismo).
        for (int i=0; i < 2; i++)
            lista[i].printDados();
        }

} // Fim classe TestaMidia.
```

**Exercício 2:** Refazer o **Exercício 1** de modo que na classe **TestaMidia** a escolha de geração dos objetos CD ou DVD seja realizada de modo aleatório, bem como a atribuição de valores das variáveis de instância dos objetos CD ou DVD sejam gerados aleatoriamente. Para tanto, será necessário empregar o pacote **import java.util.Random**, bem como os comandos dados na Figura 2.1.

```
// Cria um objeto da classe Random que funciona como gerador
aleatório.
Random randomNumbers = new Random();
// Gera valores aleatorios inteiros: valores {{0},{1}}.
```

```
aleat = randomNumbers.nextInt(2);
// Valores reais contidos no intervalo [20,50].
preco = 30*randomNumbers.nextDouble() + 20;
```

**Figura 2.1:** Gerando números aleatórios inteiros ou reais.

### Classe TestaMidia

```
import java.util.Random;

public class TestaMidia
{
    public static void main(String args[])
    {
        // Cria um vetor de elementos que são objetos da classe Midia.
        Midia[] lista = new Midia[10];

        // Cria um objeto da classe Random que funciona como gerador
        aleatório.
        Random randomNumbers = new Random();
        int aleat, codigo, nData;
        double preco;
        String nome;

        // Laço para inicializar.
        for (int i=0, contc=0, contd=0; i < 2; i++)
        { // Gera valores aleatorios para o construtor: valores {{0},{1}}.
            aleat = randomNumbers.nextInt(2);

            // Construir diferentes midias dependendo do sorteio.
            if (0 == aleat)
            { System.out.println("Criar um CD !");
              // Valores [15,30].
              preco = 15*randomNumbers.nextDouble() + 15;
              // Nome gerado a partir do contador.
              nome = "CD" + contc;
              // Valores inteiros em [5,20].
              nData = randomNumbers.nextInt(15) + 5;
              // Construção aleatoria do objeto CD.
              lista[i] = new CD(contc,preco,nome,nData);
              contc++;
            }
            else
            { System.out.println("Criar um DVD !");
              // Valores [20,50]
              preco = 30*randomNumbers.nextDouble() + 20;
              // Nome gerado a partir do contador.
              nome = "DVD" + contd;
              // Valores inteiros em [20,35].
              nData = randomNumbers.nextInt(15) + 20;
              // Construção aleatoria do objeto DVD.
              lista[i] = new DVD(contd,preco,nome,nData);
            }
        }
    }
}
```

```

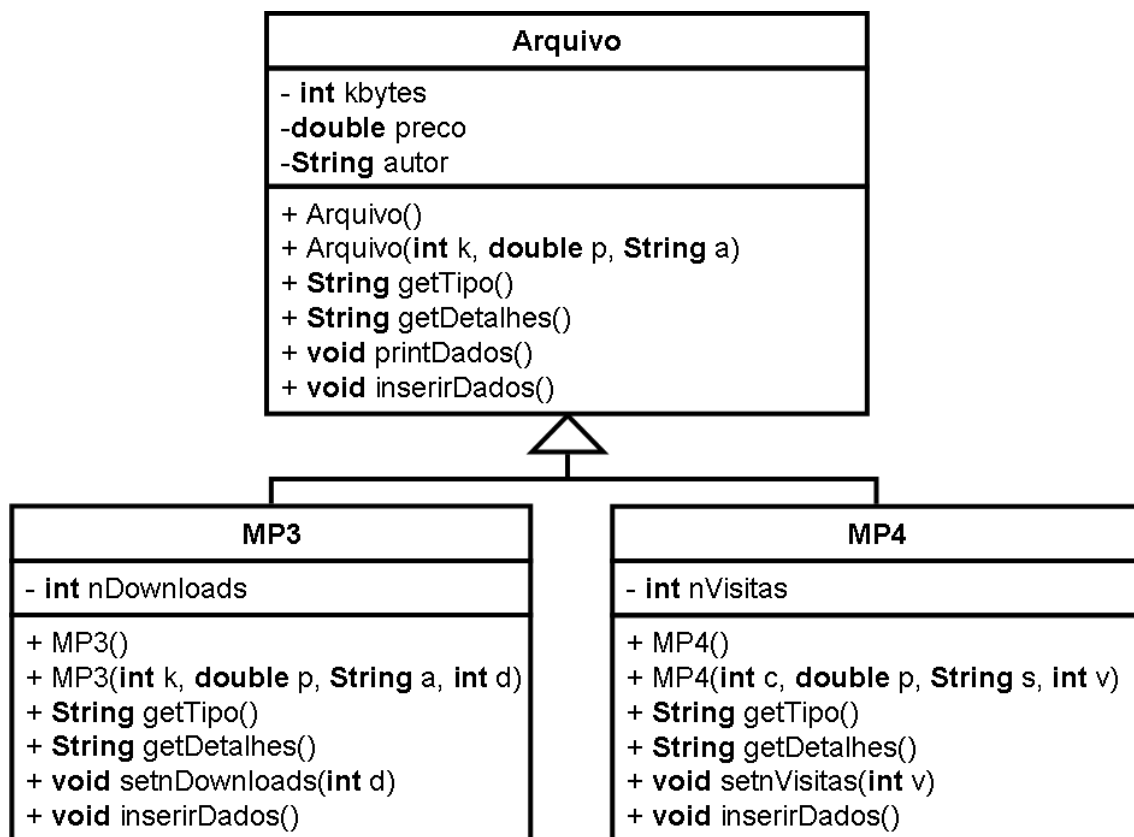
    contd++;
  }

  }
  // Imprimindo o conteúdo de cada elemento do vetor de acordo
com
  // a classe a que ele pertence (isto e, usando polimorfismo).
  for (int i=0; i < 2; i++)
    lista[i].printDados();
  }

} // Fim Classe TestaMidia.

```

**Exercício 3:** Baseado nos **Exercícios 1 e 2** simular a operação de uma loja virtual que vende arquivos mp3 e mp4. O diagrama **UML** com as classes necessárias para realizar tal simulação é apresentado na Figura 3.1.



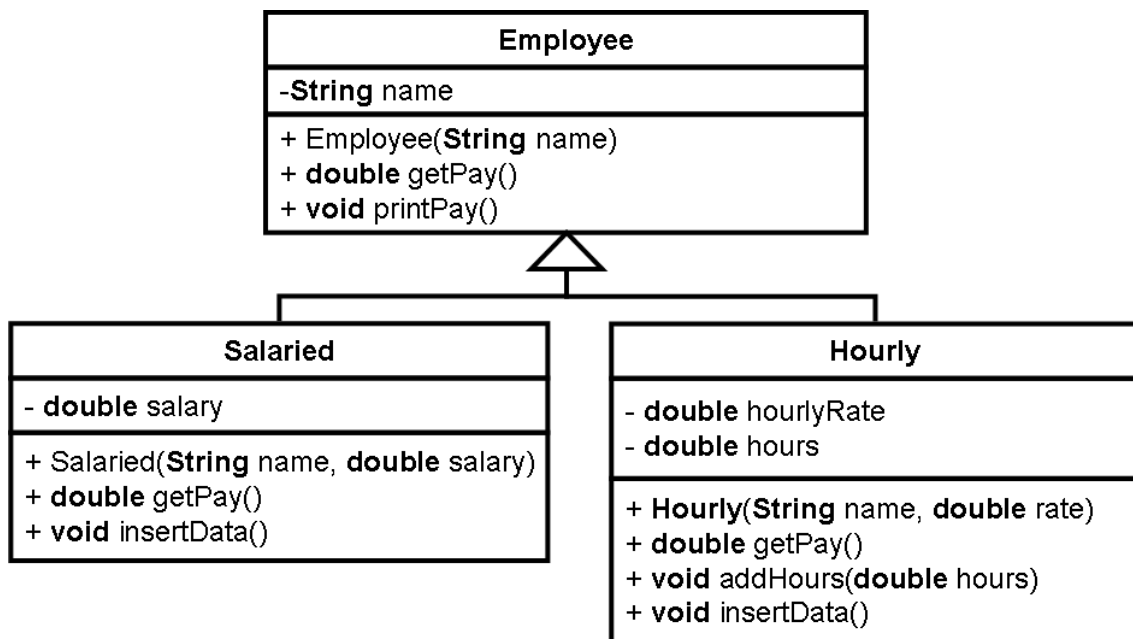
**Figura 3.1:** Diagrama UML com métodos e campos a serem utilizados.

Na simulação, os valores das variáveis de instância dos objetos deverão ser gerados aleatoriamente. Além disso, a lista de arquivos deverá ter 4 elementos mp3 ou mp4 com 50% de chance de ser gerado um ou outro. No final, se um objeto MP3 tiver mais de 5000



downloads, então, seu preço (campo preço) deve ser reajustado em 5%. Já para o objeto MP4 tiver mais de 10000 visitas seu preço deverá ser reajustado em 8%. Exibir os objetos quando da sua criação e após o processo de reajuste de preços descrito anteriormente.

**Exercício 4:** Baseado nos **Exercícios 1 e 2** simular a operação da folha de pagamento de uma empresa. Como existem dois tipos de funcionários (Assalariado - **Salaried** e Horista - **Hourly**). O diagrama **UML** com as classes necessárias para realizar tal simulação é apresentado na Figura 4.1.



**Figura 4.1:** Diagrama UML das classes, com seus campos e métodos, necessárias para simular a folha de pagamento de uma empresa.

Criar um programa que pede para o usuário definir se deseja inserir um empregado assalariado (**Salaried**) ou horista (**Hourly**) e depois preenche os campos de forma adequada. Para determinar, em tempo de execução, se um dado objeto é pertencente a uma dada classe é necessário usar a palavra-chave **instanceof** tal como dado no código da Figura 4.2.

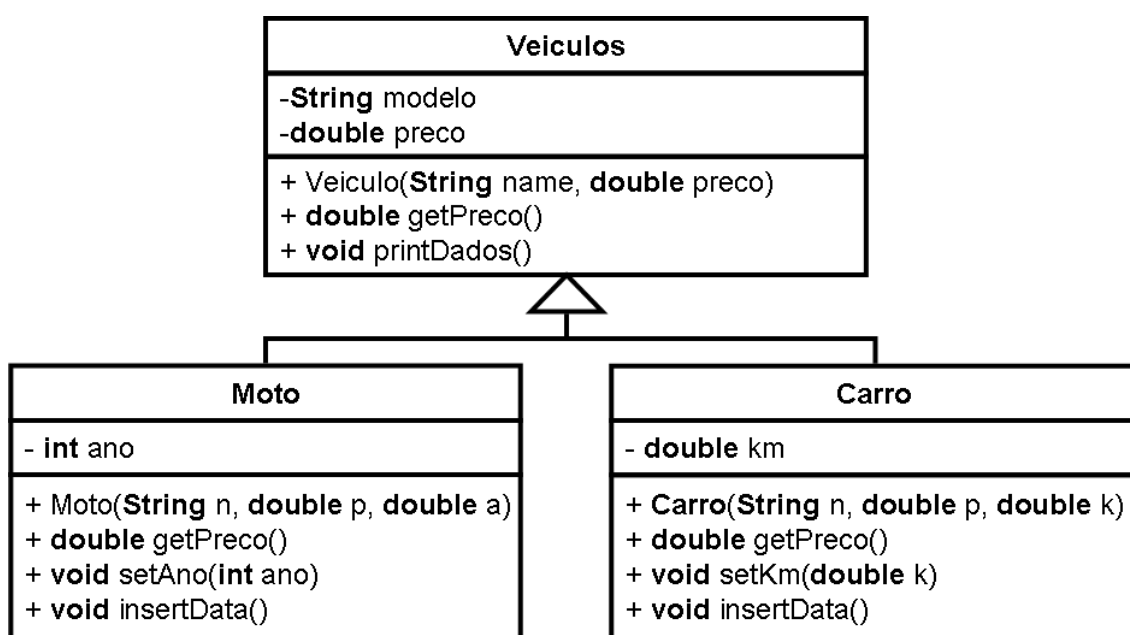
```

if (v[i] == instanceof Hourly)
    v[i].addHours(7.0);
  
```

**Figura 4.2:** Uso da palavra-chave **instanceof** para determinar qual a classe a que pertence um dado objeto contido no elemento v[i].

Após isso, deseja-se conhecer o gasto total da empresa com a folha salarial. Para tanto, será necessário conhecer o quanto cada empregado ganha a partir do método **getPay()**.

**Exercício 5:** Baseado nos **Exercícios 1 e 2** simular a operação de um cadastro de veículos de uma revenda. Existem dois tipos de veículos: **Carro, e Moto**. O diagrama **UML** com as classes necessárias para realizar tal simulação é apresentado na Figura 5.1.



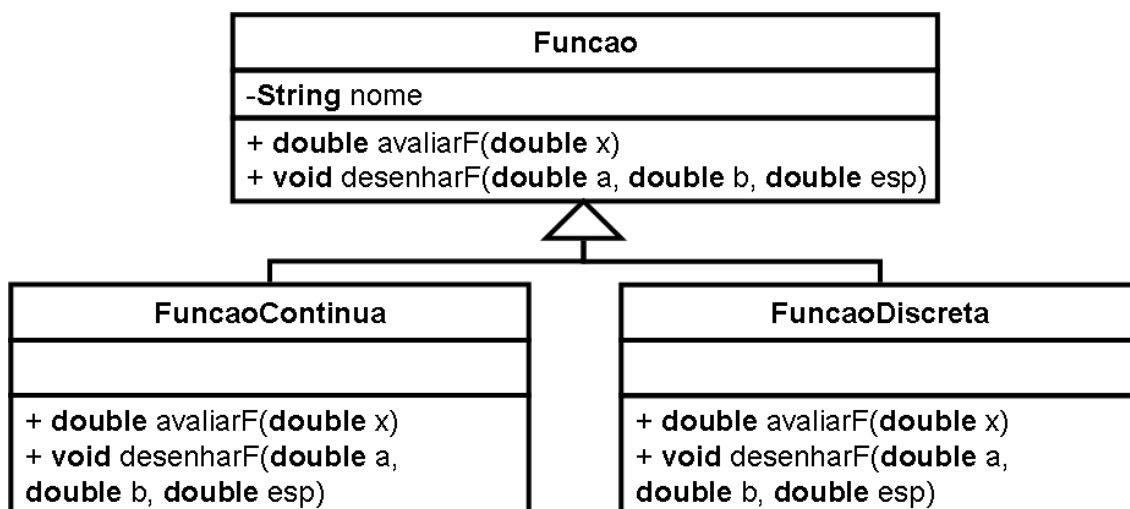
**Figura 5.1:** Diagrama UML das classes para o cadastro de veículos.

Deseja-se construir um programa que:

**Item (A):** O usuário decide se deseja inserir uma moto ou um carro. Depois insere as informações necessárias e finalmente imprime o relatório com as características de cada veículo contido no cadastro.

**Item (B):** Após o usuário inserir as informações deverá ser realizado um levantamento acerca das motos e carros. Primeiro será obtido o total preços dos veículos contidos no cadastro. Depois, é verificado e uma moto tiver no campo ano um valor maior ou igual que 2008 seu valor será reajustado em 10%. Se um carro tiver um valor no campo km um valor maior que 100000, então, seu valor será reduzido em 8%. Por fim, um novo cálculo acerca do total de preços é realizada e impressa.

**Exercício 6:** Construir as classes dadas no Diagrama UML da Figura 6.1.



**Figura 6.1:** Diagrama de classes para construir funções matemáticas.

A partir deste diagrama deseja-se construir uma lista de funções a serem exibidas. Para tanto, o método desenharF deverá se comportar polimorficamente tal como dado nas Figuras 6.2 e 6.3.

### Classe FuncaoContinua

```

void desenharF(double a, double b, double esp)
{
    double ya, yb, y;
    StdDraw.clear();
    StdDraw.setXscale(a, b);
    ya = avaliarFuncao(a);
    yb = avaliarFuncao(b);
    StdDraw.setYscale(ya, yb);
    for (double x=a;x<=b;x=x+esp)
    {
        y = avaliarFuncao(x);
        StdDraw.setPenColor(StdDraw.RED);
        StdDraw.filledCircle(x,y,0.008);
        StdDraw.show(20);
    }
}
  
```

**Figura 6.2:** Detalhamento do método desenharF para a classe FuncaoContinua.

**Classe FuncaoDiscreta**

```
void desenharFuncao(double a, double b, double esp)
{
    double ya, yb, y, maior, menor;
    StdDraw.clear();
    StdDraw.setXscale(a, b);
    ya = avaliarFuncao(a);
    maior = ya;
    menor = ya;
    yb = avaliarFuncao(b);
    StdDraw.setYscale(ya, yb);
    for (double x=a;x<=b;x=x+esp)
    {
        y = avaliarFuncao(x);
        if (maior < y)
            maior = y;
        if (menor > y)
            menor = y;
    }

    StdDraw.setXscale(a-1, b);
    StdDraw.setYscale(0, maior);
    StdDraw.setPenRadius(((b-a-2)/(esp))/10.0);
    for (double x=a;x<=b;x=x+esp)
    {
        y = avaliarFuncao(x);
        StdDraw.line(x, 0, x, y);
    }
}
```

**Figura 6.3:** Detalhamento do método desenharF para a classe FuncaoDiscreta.

Observe ainda que o método avaliarF() contém a informação necessária para descrever a avaliação de uma função  $f(x)$  em um dado ponto  $x_0$ . Para que o método retorne o valor da função  $y = ax + b$  pode-se usar o código da Figura 6.4.

```
double avaliarFuncao(double x)
{
    double y;
    y = a*x + b;
    return y;
}
```

**Figura 6.4:** Detalhamento do método avaliarFuncao() da classe FuncaoContinua.