# Modelagem de Classes - Java

Informática

Centro Universitário Anhanguera de São Paulo

20 pag.

# Projetando Classes e Relacionamento (Herança) em Modelo de Classes

A UML (Unifed Modeling Language) consiste de um número de elementos gráficos combinados para formar diagramas. Pelo fato de ser uma linguagem, a UML tem regras para a combinação desses elementos.

O propósito dos diagramas é apresentar múltiplas visões de um sistema, o conjunto dessas múltiplas visões é chamado de modelo. É importante deixar claro que um modelo UML diz o que um sistema tem que fazer, mas não como implementa-lo.

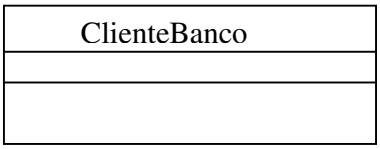
## Diagrama de Classes

Um diagrama de classes descreve os tipos de objetos no sistema e o relacionamento entre eles, mostrando os métodos e atributos de cada classe.

## Classe

Uma classe pode ser representada por uma caixa contendo apenas o nome da classe.

Veja abaixo:



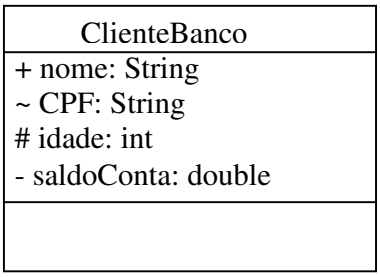

Através do diagrama acima podemos gerar o seguinte código Java.

```
class ClienteBanco{

}
```

## Atributo

A representação de atributos de uma classe em UML é feita diferente da que foi definida em Java: Declara-se primeiramente o nome do atributo e depois o tipo, separando-os pelo símbolo : (dois pontos).

Veja abaixo:




Através do diagrama acima podemos gerar o seguinte código Java.

```
class ClienteBanco{
      public String nome;
      String CPF;
      protected int idade;
      private double saldoConta;
}
```

A tabela abaixo ilustra a representação UML dos modificadores de acesso de visibilidade.

| **Representação UML** | **Código Java** |
|---|---|
| + | public |
| – | private |
| # | protected |
| ~ | default |

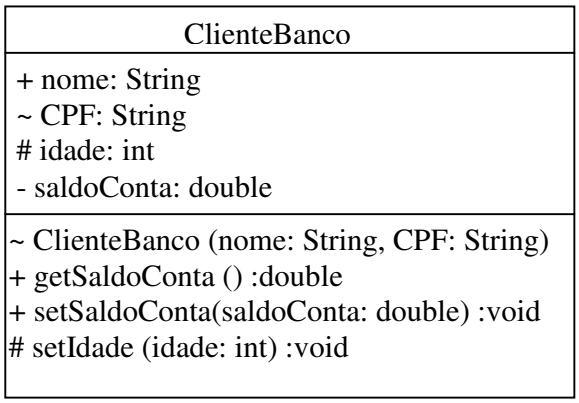
OBS: Modificador default (friendly ou package): Quando não colocamos nenhum modificador de acesso sinalizamos implicitamente o modificador default.
Os elementos demarcados por este modificador podem ser acessados por métodos ou construtores de classes definidas no mesmo pacote.

**Método**

A representação de métodos de uma classe em UML é feita de forma diferente da que foi definida em Java: Declara-se primeiramente o nome do método, e depois o tipo do valor de retorno do método.

A declaração dos parâmetros do método deve ser feita semelhantemente àquela definida para os atributos: Primeiramente declara-se o nome, em seguida o tipo e ambos separados pelo símbolo : (dois pontos).

Veja abaixo:



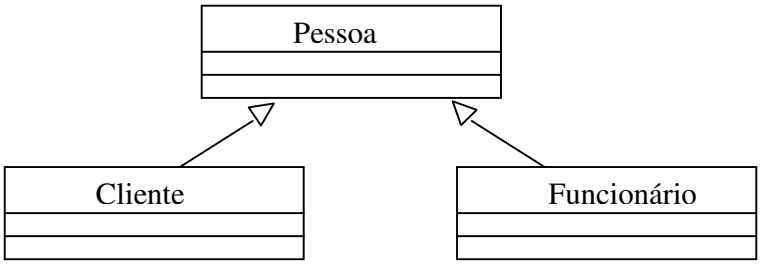

OBS: A representação UML dos modificadores de acesso de visibilidade vale tanto para atributos como para métodos.

Através do diagrama acima podemos gerar o seguinte código Java.

```
class ClienteBanco{
	public String nome;
	String CPF;
	protected int idade;
	private double saldoConta;

	ClienteBanco(String nome, String CPF){
		this.nome = nome;
		this.CPF = CPF;
	}

	public double getSaldoConta(){
		return saldoConta;
	}

	public void setSaldoConta(double saldoConta){
		this.saldoConta = saldoConta;
	}

	protected void setIdade (int idade){
		this.idade = idade;
	}
}
```

**Representação de Herança em UML**

A herança é representada na UML com uma seta contínua indicando que uma classe estende outra. No exemplo a seguir é representado que as classes Cliente e Funcionário estendem a classe Pessoa, herdando todos os seus métodos e atributos.




Através do diagrama acima podemos gerar os seguintes códigos Java.

Pessoa.java

```
class Pessoa{
}
```

Cliente.java

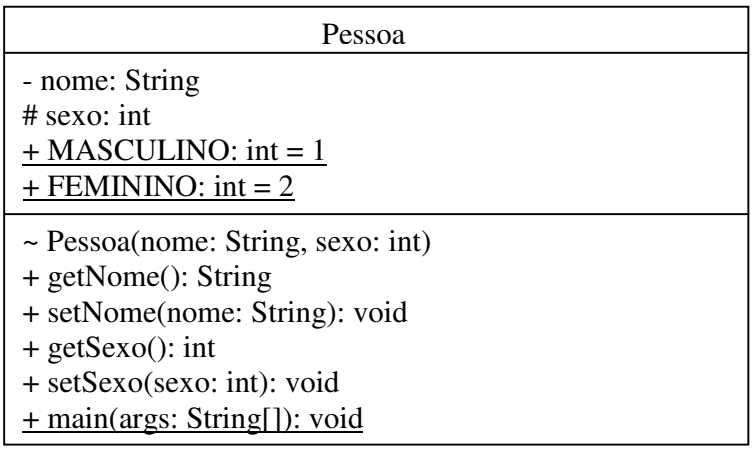
```
class Cliente extends Pessoa{
}
```

Funcionário.java

```
class Funcionario extends Pessoa{
}
```

**Representação do modificador static em UML**

Atributos e métodos estáticos são representados, em UML, utilizando palavras sublinhadas. Veja o exemplo abaixo:

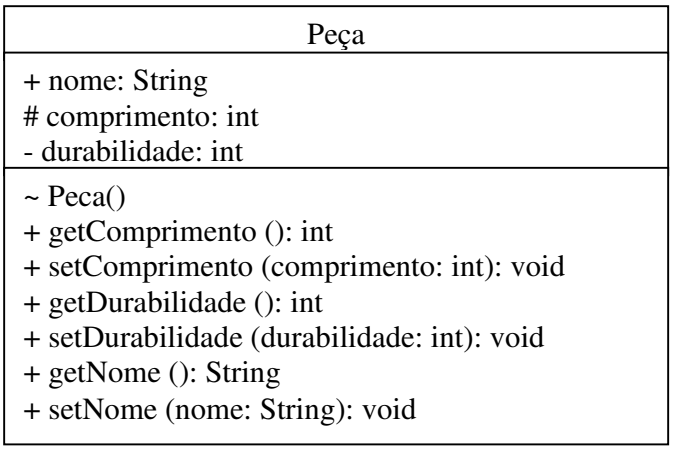


Através do diagrama UML da página anterior podemos gerar o seguinte código Java.

```
class Pessoa{
      private String nome;
      protected int sexo;
      public static int MASCULINO = 1;
      public static int FEMININO = 2;

      Pessoa (String nome, int sexo){
            this.nome = nome;
            this.sexo = sexo;
      }

      public String getNome(){
            return nome;
      }

      public void setNome(String nome){
            this.nome = nome;
      }

      public int getSexo(){
            return sexo;
      }

      public void setSexo(int sexo){
            this.sexo = sexo;
      }

      public static void main (String args[]){
      }
}
```

## Exercícios Sobre Modelagem de Classes

1) A partir do diagrama de classe UML abaixo, implemente a classe Java correspondente.
Para os métodos setDurabilidade e setComprimento, siga as regras abaixo:
SetDurabilidade: Faça a atribuição apenas se o valor informado for maior que 0 (zero) e menor que 180.
SetComprimento: Faça a atribuição apenas se o valor informado for maior que 0(zero) e menor que 25.




R:

```
public class Peca {
	public String nome;
	protected int comprimento;
	private int durabilidade;

	Peca(){
	}

	public int getComprimento() {
		return comprimento;
	}

	public void setComprimento(int comprimento) {
		if (comprimento > 0 && comprimento < 25)
			this.comprimento = comprimento;
	}

	public int getDurabilidade() {
		return durabilidade;
	}

	public void setDurabilidade(int durabilidade) {
		if (durabilidade > 0 && durabilidade < 180)
			this.durabilidade = durabilidade;
	}

	public String getNome() {
```

```
            return nome;
      }

      public void setNome(String nome) {
            this.nome = nome;
      }
}
```

2) A partir da classe Java abaixo, gere o diagrama de classe UML correspondente.

```
public class Pessoa {
      private String nome;
      protected String rg;
      private String cpf;
      byte idade;
      String endereco;
      String foneResidencial;
      String foneComercial;
      String celular;
      protected String nomePai;
      protected String nomeMae;
      private String nomeConjuge;
      private boolean casado;
      private char sexo;

      public static char MASCULINO = 'M';
      public static char FEMININO = 'F';

      public Pessoa (String nome, String cpf,
                        String rg, byte idade){
            this.nome = nome;
            this.cpf = cpf;
            this.rg = rg;
            this.idade = idade;
      }

      public boolean isCasado() {
            return casado;
      }

      public void setCasado(boolean casado) {
            this.casado = casado;
      }

      public String getCelular() {
            return celular;
      }

      public void setCelular(String celular) {
            this.celular = celular;
      }

      public String getCpf() {
            return cpf;
      }
```

```
	public void setCpf(String cpf) {
		this.cpf = cpf;
	}

	public String getEndereco() {
		return endereco;
	}

	public void setEndereco(String endereco) {
		this.endereco = endereco;
	}

	public String getFoneComercial() {
		return foneComercial;
	}

	public void setFoneComercial(String foneComercial) {
		this.foneComercial = foneComercial;
	}

	public String getFoneResidencial() {
		return foneResidencial;
	}

	public void setFoneResidencial(String foneResidencial) {
		this.foneResidencial = foneResidencial;
	}

	public byte getIdade() {
		return idade;
	}

	public void setIdade(byte idade) {
		this.idade = idade;
	}

	public String getNome() {
		return nome;
	}

	public void setNome(String nome) {
		this.nome = nome;
	}

	public String getNomeConjuge() {
		return nomeConjuge;
	}

	public void setNomeConjuge(String nomeConjuge) {
		this.nomeConjuge = nomeConjuge;
	}

	public String getNomeMae() {
		return nomeMae;
	}

	public void setNomeMae(String nomeMae) {
		this.nomeMae = nomeMae;
	}

	public String getNomePai() {
```

```
            return nomePai;
      }

      public void setNomePai(String nomePai) {
            this.nomePai = nomePai;
      }

      public String getRg() {
            return rg;
      }

      public void setRg(String rg) {
            this.rg = rg;
      }

      public char getSexo() {
            return sexo;
      }

      public void setSexo(char sexo) {
            if (sexo == MASCULINO)
                  this.sexo = MASCULINO;
            else if (sexo == FEMININO)
                  this.sexo = FEMININO;
      }
      public boolean isMaiorDeIdade(){
            if (idade >= 18)
                  return true;
            else
                  return false;
      }
}
```

R:

| Pessoa |
| --- |
| - nome: String<br># rg: String<br>- cpf: String<br>~ idade: byte<br>~ endereco: String<br>~ foneResidencial: String<br>~ foneComercial: String<br>~ celular: String<br># nomePai: String<br># nomeMae: String<br>- nomeConjuge: String<br>- casado: boolean<br>- sexo: char<br>+ MASCULINO: char = 'M'<br>+ FEMININO: char = 'F' |

+ Pessoa (nome: String, cpf: String, rg: String, idade: byte)
+ isCasado (): boolean
+ setCasado (casado: boolean): void
+ getCelular (): String
+ setCelular (celular: String): void
+ getCpf (): String
+ setCpf (cpf: String): void
+ getEndereco (): String
+ setEndereco (endereco: String): void
+ getFoneComercial (): String
+ setFoneComercial (foneComercial: String): void
+ getFoneResidencial (): String
+ setFoneResidencial (foneResidencial: String): void
+ getIdade (): byte
+ setIdade (idade: byte): void
+ getNome (): String
+ setNome (nome: String): void
+ getNomeConjuge (): String
+ setNomeConjuge (nomeConjuge: String): void
+ getNomeMae (): String
+ setNomeMae(nomeMae: String): void
+ getNomePai (): String
+ setNomePai (nomePai: String): void
+ getRg (): String
+ setRg (rg: String): void
+ getSexo (): char
+ setSexo (sexo: char): void
+ isMaiorIdade (): boolean

3) Implemente as classes Java referentes ao diagrama de classes UML abaixo.
Durante a implementação, siga as regras abaixo para cada classe:

**Trapézio**: Todos os métodos setters devem verificar se o valor do parâmetro é positivo e em caso afirmativo deve-se fazer a atribuição.

O método calcularArea deve calcular a área do trapézio e armazenar este valor no atributo área da superClasse.

Fórmula para cálculo da área do trapézio:

$$A = \frac{(baseMaior + baseMenor) * altura}{2}$$

**Triângulo**: Todos os métodos setters devem verificar se o valor do parâmetro é positivo e em caso afirmativo deve-se fazer a atribuição.

O método calcularArea deve calcular a área do triângulo e armazenar este valor no atributo área da superClasse.

Fórmula para calcular a área do triângulo:

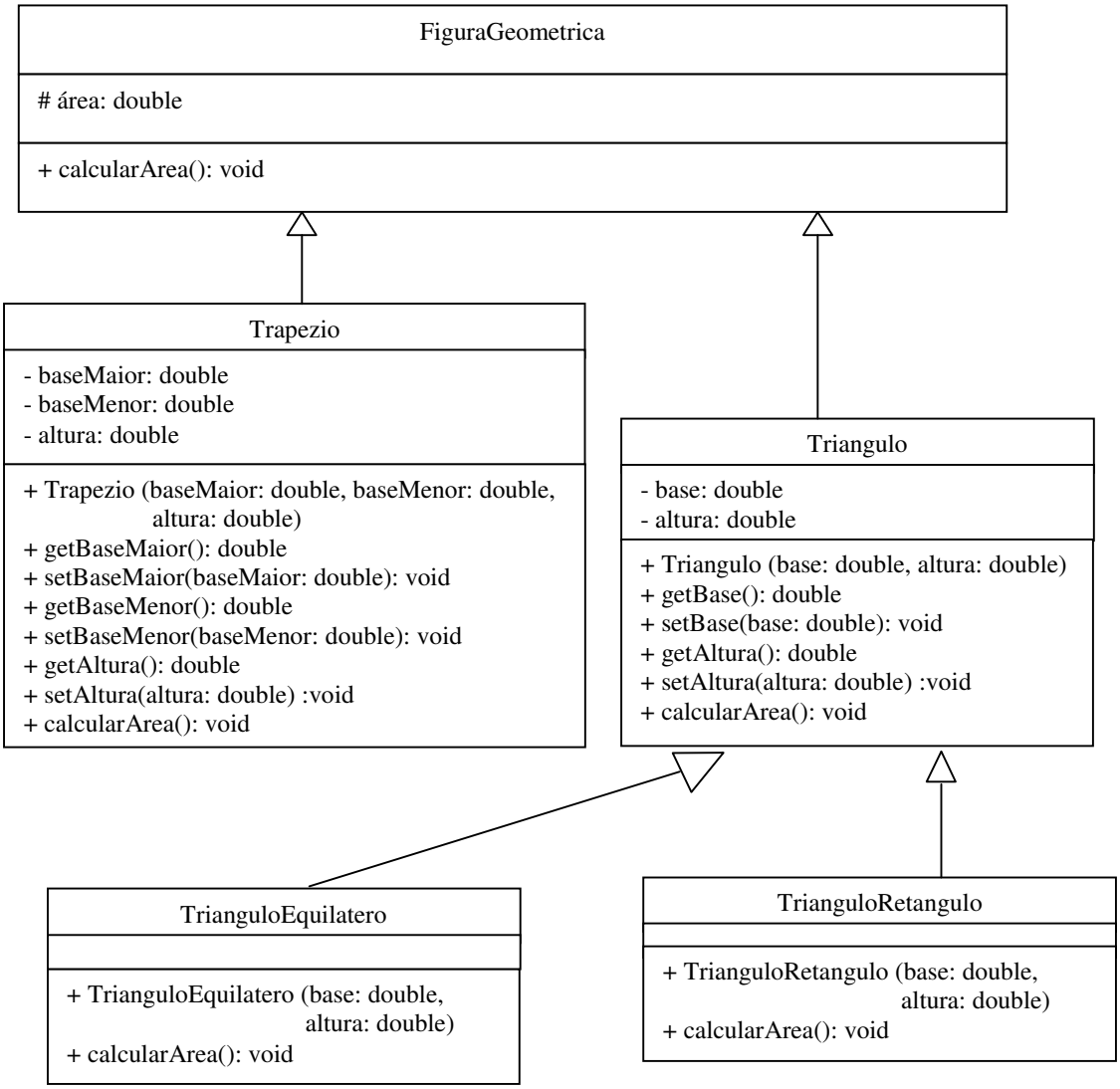
$$A = \frac{base * altura}{2}$$

**Triângulo Eqüilátero**: O método calcularArea deve imprimir a seguinte mensagem: "Calculando a área de um triângulo equilátero" e em seguida chamar o método calcularArea da super classe.

Por último imprima a seguinte mensagem: "A área do triângulo eqüilátero é:" e informe o valor calculado para a área.

**Triângulo Retângulo**: O método calcularArea deve imprimir a seguinte mensagem: "Calculando a área de um triângulo retângulo", e em seguida chame o método calcularArea da super classe.

Por último imprima a seguinte mensagem: "A área do triangulo retângulo é:" e informe o valor calculado para a área.



R:

```
public class FiguraGeometrica {
      protected double area;

      public void calcularArea(){
      }
}
```

```
public class Trapezio extends FiguraGeometrica {
      private double baseMaior;
      private double baseMenor;
      private double altura;

      public Trapezio (double baseMaior, double baseMenor,
                       double altura){
            this.baseMaior = baseMaior;
            this.baseMenor = baseMenor;
            this.altura = altura;
      }

      public double getAltura() {
            return altura;
      }

      public void setAltura(double altura) {
            if (altura > 0)
                  this.altura = altura;
      }

      public double getBaseMaior() {
            return baseMaior;
      }

      public void setBaseMaior(double baseMaior) {
            if (baseMaior > 0)
                  this.baseMaior = baseMaior;
      }

      public double getBaseMenor() {
            return baseMenor;
      }

      public void setBaseMenor(double baseMenor) {
            if (baseMenor > 0)
                  this.baseMenor = baseMenor;
      }

      public void calcularArea(){
            double areaTrapezio;

            areaTrapezio = ((baseMaior + baseMenor)*altura) / 2;

            super.area = areaTrapezio;
      }
}
```

```
public class Triangulo extends FiguraGeometrica {
	private double base;
	private double altura;

	public Triangulo (double altura, double base){
		this.altura = altura;
		this.base = base;
	}

	public double getAltura() {
		return altura;
	}

	public void setAltura(double altura) {
		if (altura > 0)
			this.altura = altura;
	}

	public double getBase() {
		return base;
	}

	public void setBase(double base) {
		if (base > 0)
			this.base = base;
	}

	public void calcularArea(){
		double areaTriangulo;

		areaTriangulo = (base*altura)/2;
		super.area = areaTriangulo;
	}
}
```

```
public class TrianguloEquilatero extends Triangulo {

	public TrianguloEquilatero(double base, double altura){
		super(base, altura);
	}

	public void calcularArea(){
		System.out.println("Calculando a area de um triangulo
				equilatero");
		super.calcularArea();
		System.out.println("A area do triangulo equilatero é: " +
				super.area);
	}
}
```

```
public class TrianguloRetangulo extends Triangulo {

	public TrianguloRetangulo(double base, double altura){
		super(base, altura);
	}

	public void calcularArea(){
		System.out.println("Calculando a area de um triangulo
				retangulo");
		super.calcularArea();
		System.out.println("A area do triangulo retangulo é: " +
				super.area);
	}
}
```

A classe abaixo foi criada apenas para testar, logo não faz parte de exercício. Esta classe é interessante para mostrar aos alunos durante a aula.

```
public class Teste {

	public static void main(String[] args) {
		TrianguloEquilatero te = new TrianguloEquilatero(10,12);
		TrianguloRetangulo tr = new TrianguloRetangulo(25,16);

		te.calcularArea();
		tr.calcularArea();
	}

}
```

4) A partir das classes Java abaixo, gere o diagrama de classe UML correspondente.

```
public class No1 {
	private String nome;
	protected int num1;

	public String getNome() {
		return nome;
	}
	public void setNome(String nome) {
		this.nome = nome;
	}
	public int getNum1() {
		return num1;
	}
	public void setNum1(int num1) {
		this.num1 = num1;
	}
}
```

```
public class No2 extends No1 {
	public String nome;
	protected int num2;

	public String getNome() {
		return nome;
	}
	public void setNome(String nome) {
		this.nome = nome;
	}
	public int getNum2() {
		return num2;
	}
	public void setNum2(int num2) {
		this.num2 = num2;
	}
}
```

```
public class No3 extends No1 {
	String nome;
	private int num3;

	public String getNome() {
		return nome;
	}
	public void setNome(String nome) {
		this.nome = nome;
	}
	public int getNum3() {
		return num3;
	}
	public void setNum3(int num3) {
		this.num3 = num3;
	}
}
```

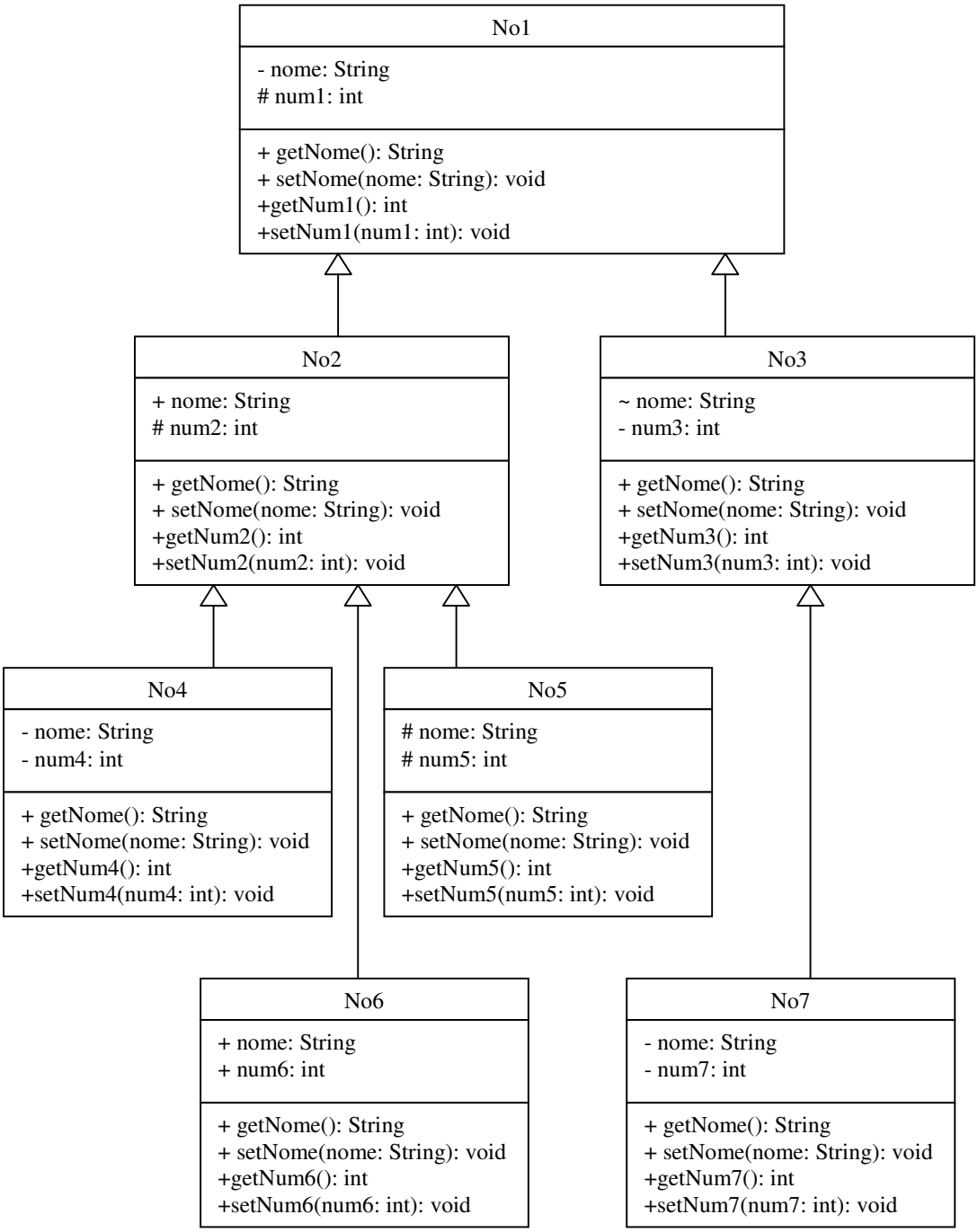
```
public class No4 extends No2 {
	private String nome;
	private int num4;

	public String getNome() {
		return nome;
	}
	public void setNome(String nome) {
		this.nome = nome;
	}
	public int getNum4() {
		return num4;
	}
	public void setNum4(int num4) {
		this.num4 = num4;
	}
}
```

```
public class No5 extends No2 {
	protected String nome;
	protected int num5;
	public String getNome() {
		return nome;
	}
	public void setNome(String nome) {
		this.nome = nome;
	}
	public int getNum5() {
		return num5;
	}
	public void setNum5(int num5) {
		this.num5 = num5;
	}
}
```

```
public class No6 extends No2 {
	public String nome;
	public int num6;

	public String getNome() {
		return nome;
	}
	public void setNome(String nome) {
		this.nome = nome;
	}
	public int getNum6() {
		return num6;
	}
	public void setNum6(int num6) {
		this.num6 = num6;
	}
}
```

```
public class No7 extends No3 {
	private String nome;
	private int num7;
	public String getNome() {
		return nome;
	}
	public void setNome(String nome) {
		this.nome = nome;
	}
	public int getNum7() {
		return num7;
	}
	public void setNum7(int num7) {
		this.num7 = num7;
	}
}
```

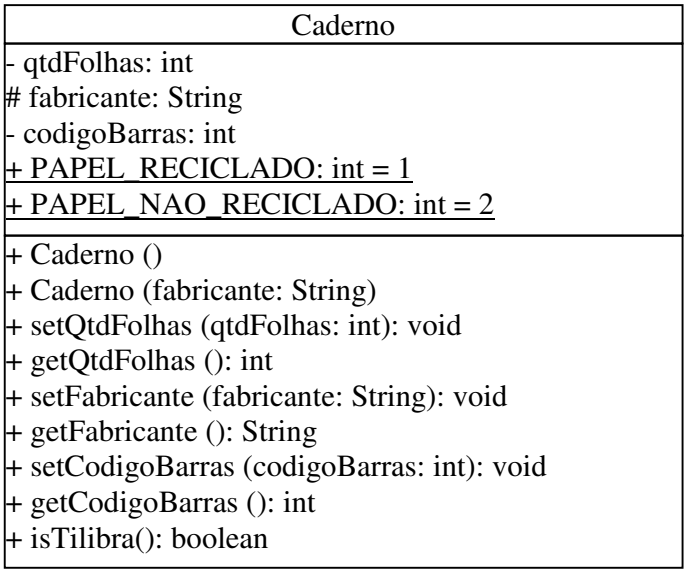
R:



5) Dado o diagrama UML abaixo, implemente a classe Java correspondente.
Durante a implementação da classe Java, siga as regras abaixo:

i. O atributo qtdFolhas deve ser preenchido apenas se o valor informado for maior que zero.

ii. O método "isTilibra" irá retornar "true" se o atributo "fabricante" for igual a "Tilibra" e "false" caso contrário.

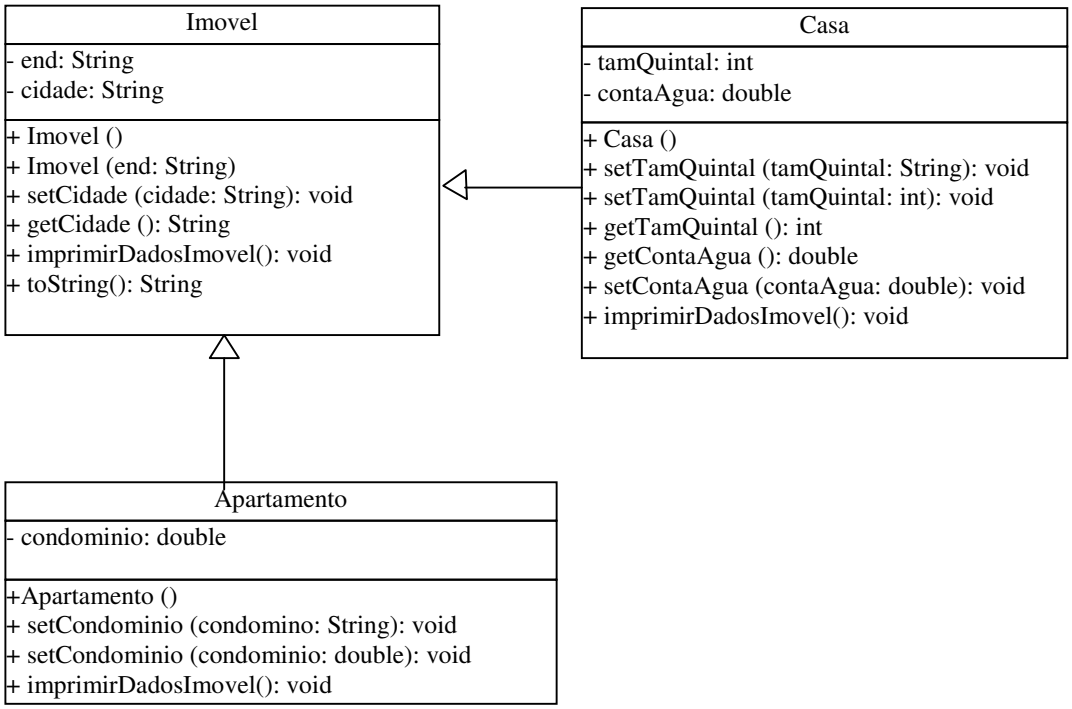



Resposta:

```
public class Caderno {
	private int qtdFolhas;
	protected String fabricante;
	private int codigoBarras;
	public static int PAPEL_RECICLADO = 1;
	public static int PAPEL_NAO_RECICLADO = 2;

	public Caderno(){
	}

	public Caderno(String fabricante){
		this.fabricante = fabricante;
	}

	public int getQtdFolhas() {
		return qtdFolhas;
	}

	public void setQtdFolhas(int qtdFolhas) {
		if (qtdFolhas > 0)
			this.qtdFolhas = qtdFolhas;
	}
```

```
	public String getFabricante() {
		return fabricante;
	}

	public void setFabricante(String fabricante) {
		this.fabricante = fabricante;
	}

	public int getCodigoBarras() {
		return codigoBarras;
	}

	public void setCodigoBarras(int codigoBarras) {
		this.codigoBarras = codigoBarras;
	}

	public boolean isTilibra(){
		if (fabricante.equals("Tilibra"))
			return true;
		else
			return false;
	}
}
```

6) Dado o diagrama de classes UML abaixo, liste todos os métodos que são exemplos de sobrescrita e sobrecarga. Justifique sua resposta.
Implemente todas as classes Java presentes neste diagrama.



Resposta

Na classe Imóvel o método construtor é um exemplo de sobrecarga, pois temos duas versões do mesmo.

Ainda na classe Imóvel o método toString() é um exemplo de sobrescrita porque o mesmo está implementado na classe Object. Devemos lembrar que a classe Imóvel é filha da classe Ojbect.

Na classe Apartamento o método setCondomínio é um exemplo de sobrecarga, pois temos duas versões do mesmo.

Ainda na classe Apartamento o método imprimirDadosImovel é um exemplo de sobrescrita, pois o mesmo está sobrescrevendo o método da super classe (Imóvel).

Na classe Casa o método setTamQuintal é um exemplo de sobrecarga, pois temos duas versões do mesmo.

Ainda na classe Casa o método imprimirDadosImovel é um exemplo de sobrescrita, pois o mesmo está sobrescrevendo o método da super classe (Imóvel).