Técnicas de Programação 1

Professor Cristhian Riaño

Tópicos

- Decomposição de problemas usando classes e objetos.
- Reuso com herança e composição de classes.
- Polimorfismo por subtipos/parametrizado.
- Tratamento de exceções.
- Técnicas de desenho e programação OO.
- Tópicos especiais (distribuição, concorrência).
- Técnicas avançadas de modularização.

Programação Orientada a objetos - Java

Nesta secção se apresentam os elementos fornecidos pela linguagem de programação Java, associados com os conceitos de programação orientada a objetos.

Os temas abordados serão:

- O conceito de classe e objeto
- Acesso aos atributos e métodos de uma classe
- Memoria em Java (Stack e Heap)
- Encapsulamento
- Contexto Estático
- Herança

.

Memoria em Java (Stack e Heap)

Stack: Tratasse de um fragmento de memória donde se apilham linearmente os valores gerados na execução como:

- Variáveis locais.
- Variáveis de referência.
- Parâmetros e valores de retorno.
- Resultados parciais.
- Também é usado para controlar a invocação e o retorno de métodos.

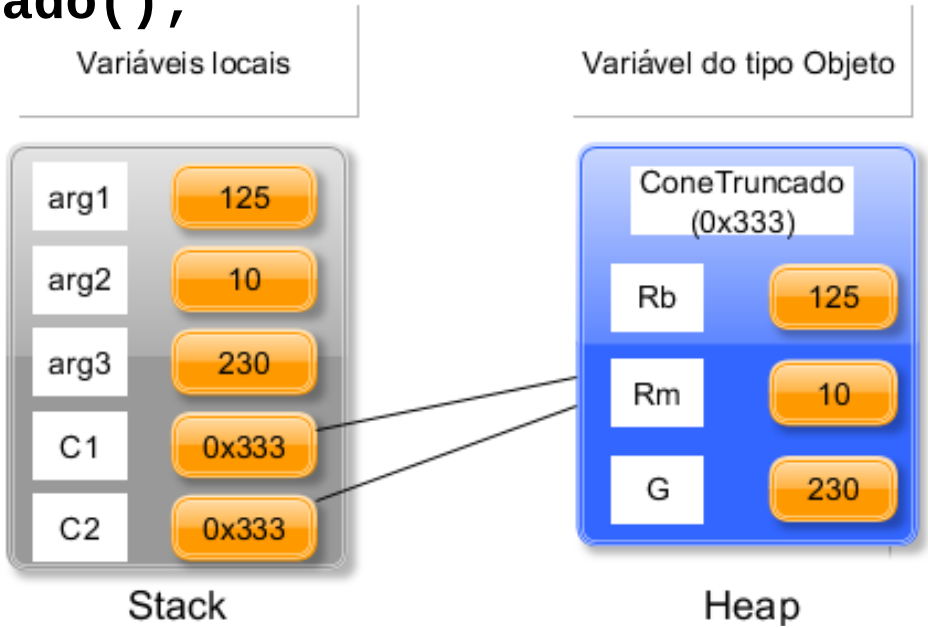
Memoria em Java (Stack e Heap)

Heap:

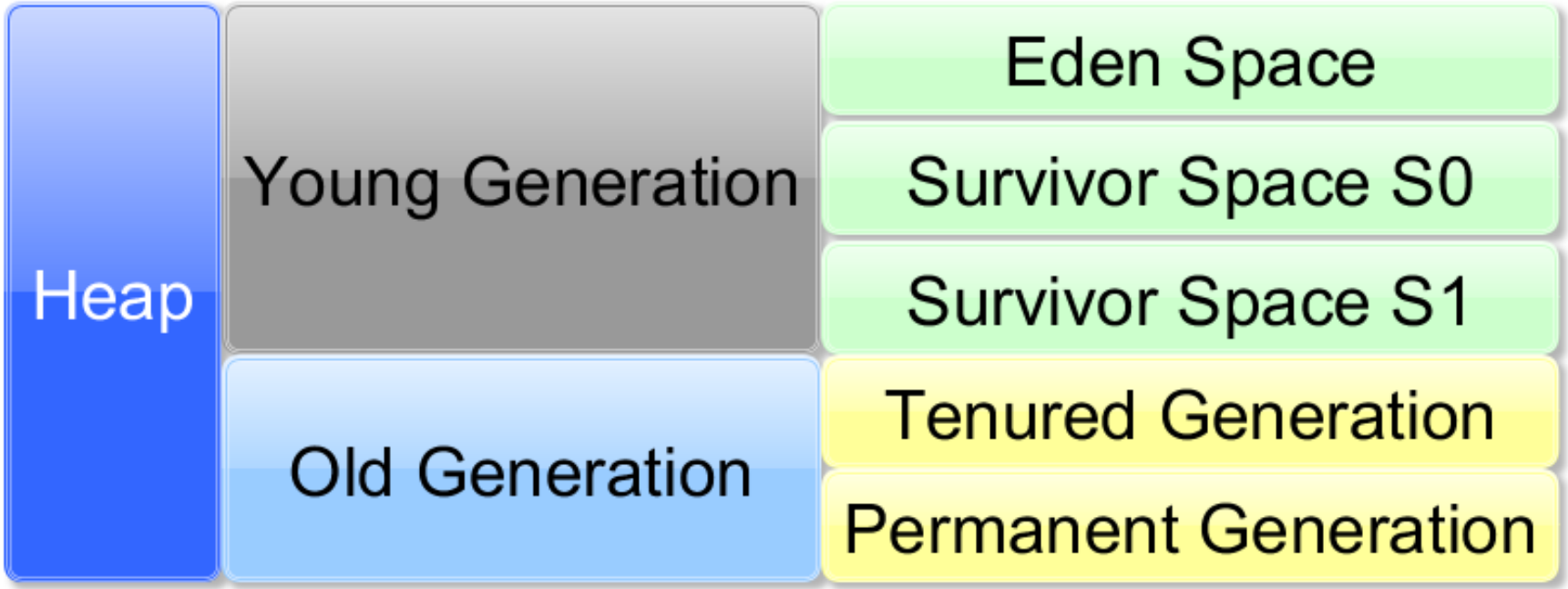
- Armazenar objetos e suas variáveis de instância.
- É um espaço de memória dinâmica criada no início da máquina virtual e exclusivo.
- O administrador deste espaço de memória dinâmica é (Garbage Collector).

Memoria em Java (Stack e Heap)

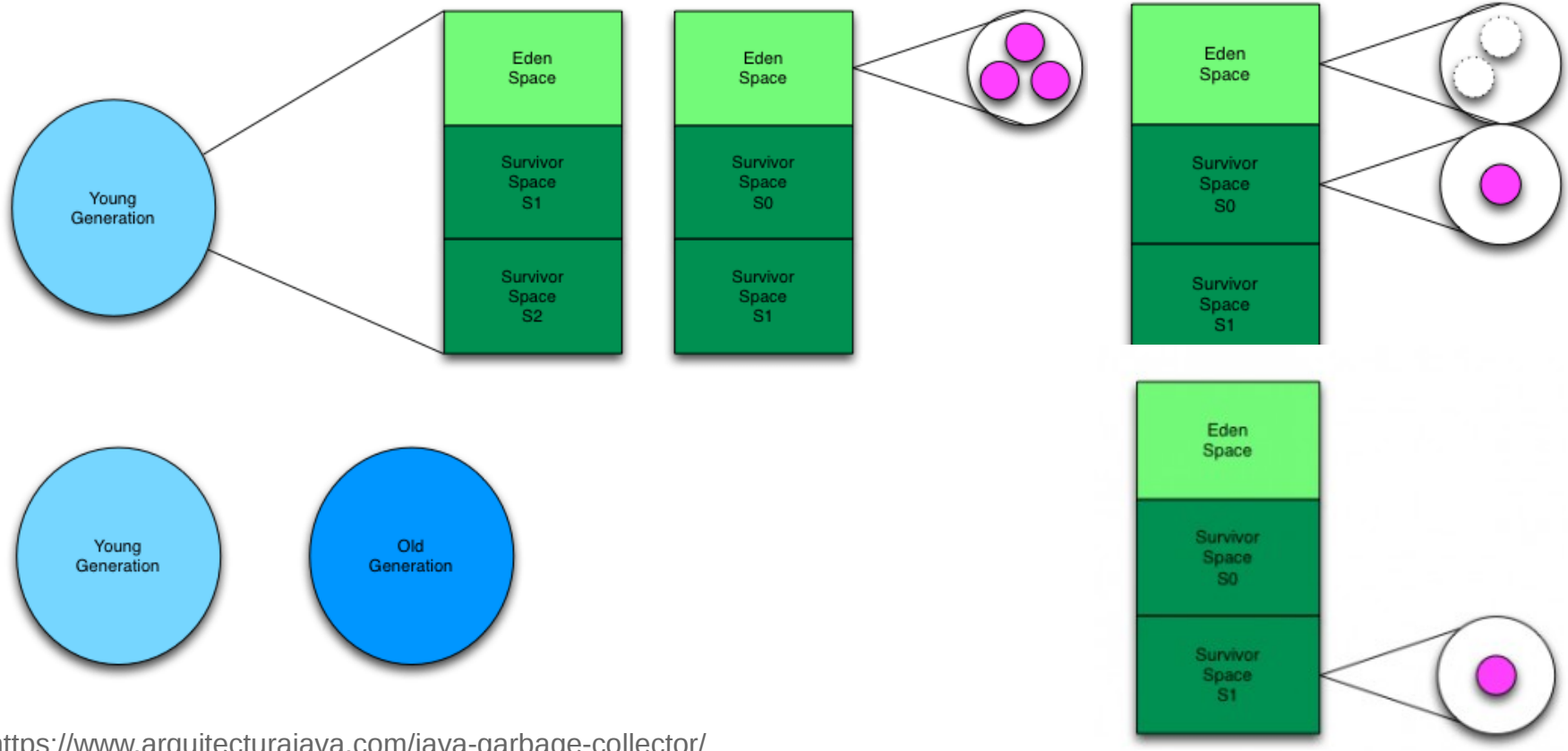
```
int arg1 =125;
int arg2 = 10;
int arg3 = 230;
ConeTrucado C1 = new ConeTrucado();
ConeTrucado C2 =C1;
C2.Rb = arg1;
C2.Rm = arg2;
C2.G = arg3;
```



Coletor de lixo (Garbage Collector)

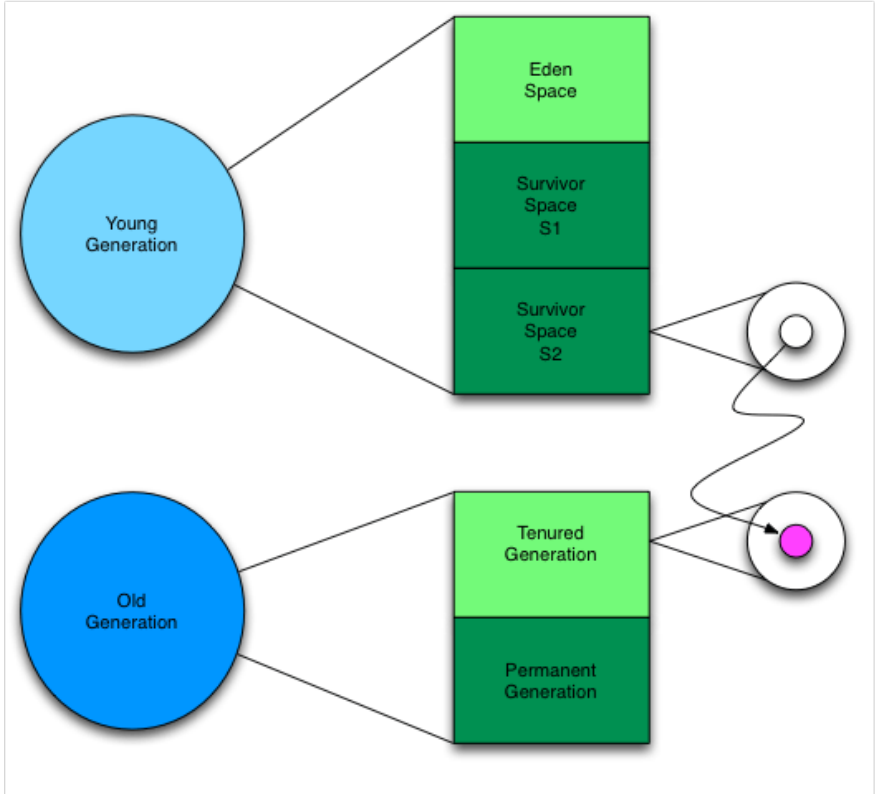


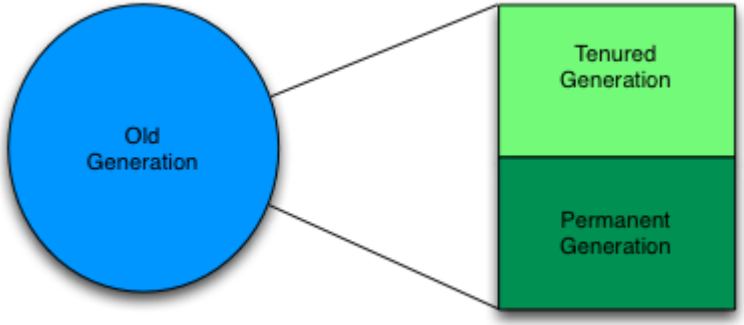
Coletor de lixo (Garbage Collector)



https://www.arquitecturajava.com/java-garbage-collector/

Coletor de lixo (Garbage Collector)





https://www.arquitecturajava.com/java-garbage-collector/

Encapsulamento

- A ocultação de dados é uma parte importante do encapsulamento.
- O objeto não precisa revelar todos os seus atributos e comportamentos.
- Os aspectos internos são ocultos do mundo externo.
- Permite as mudanças com maior controle.

Modificador de Acesso

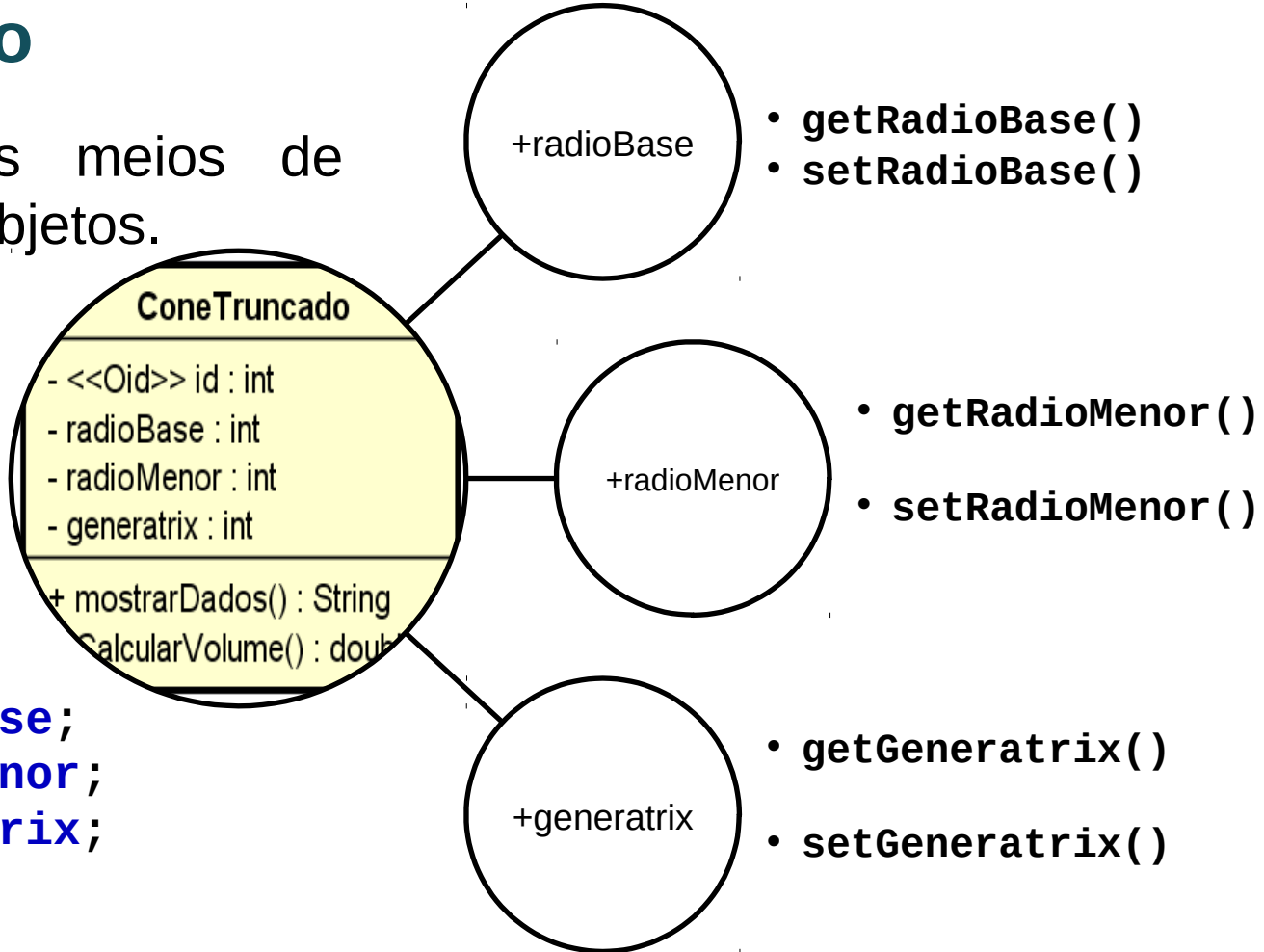
	Classe	Pacote	Subclasse	Outros
public	O	O	O	O
private	O	X	X	X
protected	O	O	O	X
Default	O	O	X	X

O Possível

X Não

Encapsulamento

A interface define os meios de comunicação entre objetos.




```
private int radioBase;
private int radioMenor;
private int generatrix;
```

Exemplo - Encapsulamento

```
package encapsulamento;
public class ConeTruncadoE {
private int radioBase, radioMenor, generatrix;

public ConeTruncadoE() {
}

public ConeTruncadoE(int Rb, int Rm, int G ) {
    this.radioBase =Rb;
    this.radioMenor = Rm;
    this.generatrix =G;
}
```

Exemplo - Encapsulamento

```
public int getRadioBase() {
     return radioBase;
     }
public void setRadioBase(int radioBase) {
     this.radioBase = radioBase;
     }
public int getRadioMenor() {
     return radioMenor;
     }
public void setRadioMenor(int radioMenor) {
     this.radioMenor = radioMenor;
     }
public int getGeneratrix() {
     return generatrix;
     }
public void setGeneratrix(int generatrix) {
     this.generatrix =generatrix;
}
```

Exemplo - Encapsulamento

```
public double CalculoDoVolume() {
    return ((Math.PI)*(Math.sqrt(Math.pow(this.generatrix, 2)-
Math.pow((this.radioBase-this.radioMenor),
2)))*(Math.pow(this.radioBase,2)+Math.pow(this.radioMenor, 2) +
(this.radioBase*this.radioMenor)))/3;
    }
public void ImprimirValores() {
    System.out.println("Valores-> Rb:"+this.radioBase+ "
Rm:"+this.radioMenor+ " G:"+ this.generatrix);
    }
}
```

Contexto Estático

- Ao definir um atributo como estático, declaramos que o atributo pertence à classe e não ao objeto.
- O atributo ou método estático é criado uma vez, independentemente de quantos objetos são criados.
- O valor acedido por um objeto de um atributo ou método estático é o mesmo valor que todos os objetos podem aceder.
- Quando o valor do atributo ou método estático é alterado por um objeto os demais objetos acederam ao valor alterado.
- Podemos aceder ao atributo ou método estático sem precisar de instanciar a classe.

Contexto Estático

- Ao definir um atributo como estático, declaramos que o atributo pertence à classe e não ao objeto.
- O atributo ou método estático é criado uma vez, independentemente de quantos objetos são criados.
- O valor acedido por um objeto de um atributo ou método estático é o mesmo valor que todos os objetos podem aceder.
- Quando o valor do atributo ou método estático é alterado por um objeto os demais objetos acederam ao valor alterado.
- Podemos aceder ao atributo ou método estático sem precisar de instanciar a classe.

Contexto Estático

```
package contextoEstatico;
public class ConeEstatico {
    public static void main(String[] args) throws Exception{
        double Vc;
        ConeTruncado C1 = new ConeTruncado(125,10,230);
        C1.ImprimirValores();
        Vc=C1.CalculoDoVolume();
        System.out.println("  Cone : "+ConeTruncado.getContadorObjetos() + "
        Valor do Volume: " + Vc);
                ConeTruncado C2 = new ConeTruncado(150,25,210);
                C2.ImprimirValores();
                Vc=C2.CalculoDoVolume();
        System.out.println("  Cone : "+ConeTruncado.getContadorObjetos()+"
        Valor do Volume: "+ Vc);
            }
        }
```

Contexto Estático

```
class ConeTruncado {
    private int radioBase, radioMenor, generatrix,idObjeto;
    private static int contadorObjetos;
    public ConeTruncado(int Rb, int Rm, int G ) {
        contadorObjetos++;
        this.radioBase =Rb;
        this.radioMenor = Rm;
        this.generatrix =G;
        this.idObjeto=contadorObjetos;
        }
    public static int getContadorObjetos() {
        return contadorObjetos;
        }
```

Contexto Estático

```
    public double CalculoDoVolume() {
        return ((Math.PI)*(Math.sqrt(Math.pow(this.generatrix, 2)-
Math.pow((this.radioBase-this.radioMenor),
2)))*(Math.pow(this.radioBase,2)+Math.pow(this.radioMenor, 2)      +
(this.radioBase*this.radioMenor)))/3;
                }
    public void ImprimirValores() {
                System.out.println("Id Objeto : "+this.idObjeto + "
                Rb:"+this.radioBase+ " Rm:"+this.radioMenor+ " G:"+
                this.generatrix);
                        }
                    }
```

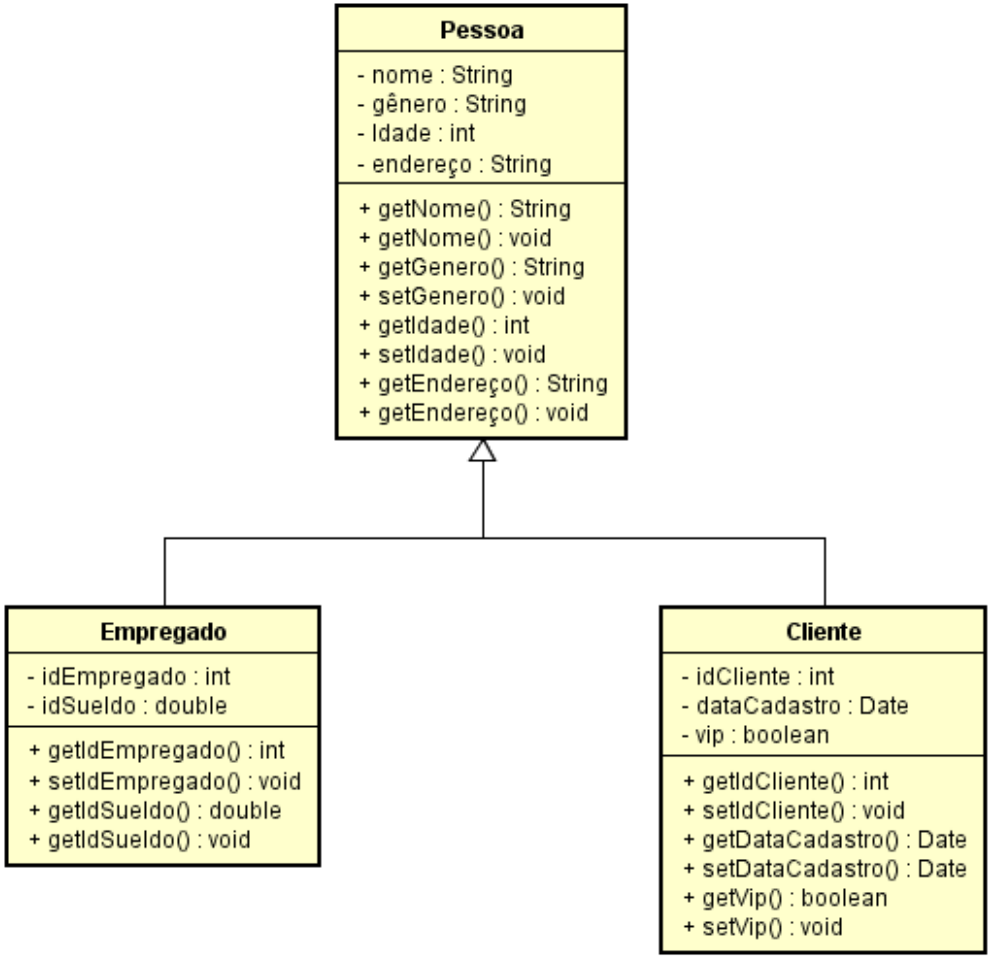
Herança

- As subclasses a serem derivadas a partir de uma superclasse, herdam suas características, ou seja, seu atributos e métodos.

- Em uma herança os atributos e os relacionamentos pertencentes às superclasses serão “herdados” por todas as subclasses. Cada subclasse pode possuir atributos adicionais e participar de outros relacionamentos.

- Um objeto obtido de uma subclasse sempre pode substituir um objeto de tipo superclasse o seja tudo o que podemos deduzir a respeito de uma superclasse também será valido para suas subclasses.

Herança


Pessoa
- nome : String
- gênero : String
- Idade : int
- endereço : String
+ getNome() : String
+ getNome() : void
+ getGenero() : String
+ setGenero() : void
+ getIdade() : int
+ setIdade() : void
+ getEndereço() : String
+ getEndereço() : void
Empregado
- idEmpregado : int
- idSueldo : double
+ getIdEmpregado() : int
+ setIdEmpregado() : void
+ getIdSueldo() : double
+ getIdSueldo() : void
Cliente
- idCliente : int
- dataCadastro : Date
- vip : boolean
+ getIdCliente() : int
+ setIdCliente() : void
+ getDataCadastro() : Date
+ setDataCadastro() : Date
+ getVip() : boolean
+ setVip() : void

Herança - class Pessoa (Parte 1)

```
package herencia;
public class Pessoa {
	private String nome;
	private char genero;
	private int idade;
	private String endereco;
	public Pessoa() {
	}
	public Pessoa(String nome) {
		this.nome = nome;
	}
	public Pessoa(String nome, char genero, int idade, String endereco) {
		this.nome =nome;
		this.genero = genero;
		this.idade = idade;
		this.endereco =endereco;
	}
```

Herança - class Pessoa (Parte 2)

```
public String getNome() {
        return nome;
}
public void setNome(String nome) {
        this.nome=nome;
}
public char getGenero() {
        return genero;
}
public void setGenerp(char genero) {
        this.genero=genero;
}
public int getIdade() {
        return idade;
}
public void setIdade(int idade) {
        this.idade =idade;
}
```

Herança - class Pessoa (Parte 3)

```
    public String getEndereco() {
        return endereco;
    }

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

    @Override
    public String toString() {
        return "pessoa{" + "nome= "  + nome + ", genero= " + genero + ", idade= " +
idade+ ", endereco= "+endereco+ '}';
        }

}
```

Herança - class Empregado (Parte 1)

```
package herencia;
public class Empregado extends Pessoa{
        private int idEmpregado;
        private double salario;
        private static int contadorEmpregados;
        public Empregado(String nome, double salario) {
                super(nome);
                this.idEmpregado = ++ contadorEmpregados;
                this.salario =salario;
        }
        public int getIdEmpleado() {
                 return idEmpregado;
        }
```

Herança - class Empregado (Parte 2)

```
	public double getSalario() {
		return salario;
	}
	public void setSalario(double salario) {
		this.salario = salario;
	}

	@Override
	public String toString() {
		return super.toString() + " Empregado{" + "idEmpregado= " + idEmpregado + ", salario= " +
salario + '}';
	}


}
```

Herança - class Cliente (Parte 1)

```
package herencia;
import java.util.Date;
public class Cliente extends Pessoa{
	private int idCliente;
	private java.util.Date dataCadastro;
	private boolean vip;
	private static int contadorCliente;

	public Cliente(Date dataCadastro, boolean vip){
		this.idCliente = ++contadorCliente;
		this.dataCadastro = dataCadastro;
		this.vip = vip;
	}

	public int getIdCliente() {
		return idCliente;
	}
```

Herança - class Cliente (Parte 2)

```
        public Date getDataCadastro() {
                return dataCadastro;
        }
        public boolean isVip() {
                return vip;
        }
        public void setVip(boolean vip) {
                this.vip =vip;
        }
        @Override
        public String toString() {
         return super.toString() + " Cliente{" + ", idCliente= " + idCliente + ", dataCadastro= " +
dataCadastro + ", vip= " + vip + '}';
}
}
```

Herança - class TestHerenca (Parte 1)

```
package herencia;
public class TestHerenca {
    public static void main(String[] args) {
        Empregado e1 = new Empregado ("Jose", 2300);
        System.out.println("Imprimimos objeto e1: ");
        System.out.println(e1);

        Empregado e2 = new Empregado ("Carlos", 5600);
        e2.setIdade(35);
        e2.setGenerp('M');
        e2.setEndereco("Quadra 303");
        System.out.println("\n Imprimimos objeto e2: ");
        System.out.println(e2);

        Cliente c1=new Cliente(new java.util.Date() ,false);
        System.out.println("\n Imprimimos Cliente c1: ");
        System.out.println(c1);
```

Herança - class TestHerenca (Parte 2)

```
            c1.setNome("Maria");
            c1.setIdade(25);
            c1.setGenerp('F');
            c1.setEndereco("Rua A");

            System.out.println("\n Imprimimos Cliente c1: ");
            System.out.println(c1);

        }

    }
```

Exercício 1 – Sistema Registro Artistas Musicais

Deseja-se construir um sistema para manter um registro de artistas musicais e seus álbuns. Cada álbum possui várias músicas, as quais poderão ser consultadas pelo sistema. O sistema também deve permitir a busca de artistas por nome ou nacionalidade. O sistema também deve ser capaz de exibir um relatório dos álbuns de um artista, o qual pode ser ordenado por nome, ano, ou duração total do álbum. Um álbum pode ter a participação de vários artistas, sem distinção. Já a música pode possuir um ou mais autores e intérpretes (todos são considerados artistas).