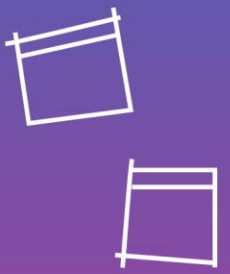
DBC
DIGITAL BUSINESS COMPANY®

VEM SER
DBC



OO III + Collections

Sumário

- Polimorfismo
  - Sobrecarga
  - Sobrescrita
- Collections
  - Listas
  - Filas
  - Pilhas
  - Mapas

Polimorfismo

- "polimorfismo" vem do grego *poli* = muitas, *morphos* = forma

https://www.devmedia.com.br/uso-de-polimorfismo-em-java/26140

Polimorfismo

- "polimorfismo" vem do grego *poli* = muitas, *morphos* = forma
- Usar o mesmo elemento de formas diferentes

https://www.devmedia.com.br/uso-de-polimorfismo-em-java/26140

Polimorfismo

- "polimorfismo" vem do grego *poli* = muitas, *morphos* = forma
- Usar o mesmo elemento de formas diferentes
- Polimorfismo denota uma situação na qual um objeto pode se comportar de maneiras diferentes ao receber uma mensagem.

https://www.devmedia.com.br/uso-de-polimorfismo-em-java/26140

Polimorfismo

- "polimorfismo" vem do grego *poli* = muitas, *morphos* = forma
- Usar o mesmo elemento de formas diferentes
- Polimorfismo denota uma situação na qual um objeto pode se comportar de maneiras diferentes ao receber uma mensagem.
- No Polimorfismo temos dois tipos:
  - Polimorfismo Estático ou Sobrecarga
  - Polimorfismo Dinâmico ou Sobreposição

https://www.devmedia.com.br/uso-de-polimorfismo-em-java/26140

Sobreposição

- O Polimorfismo Dinâmico acontece na herança, quando a subclasse sobrepõe o método original.

Sobreposição

- O Polimorfismo Dinâmico acontece na herança, quando a subclasse sobrepõe o método original.
- O método escolhido se dá em tempo de execução e não mais em tempo de compilação.

Sobreposição

- O Polimorfismo Dinâmico acontece na herança, quando a subclasse sobrepõe o método original.
- O método escolhido se dá em tempo de execução e não mais em tempo de compilação.
- A escolha de qual método será chamado depende do tipo do objeto que recebe a mensagem.

Sobreposição

```
public class Funcionario {
    private double salario;
    public void calcularSalario(double salarioBruto, double descontos){
        salario = salarioBruto - descontos;
    }
}
```

Sobreposição

```
public class Funcionario {
    private double salario;
    public void calcularSalario(double salarioBruto, double descontos){
        salario = salarioBruto - descontos;
    }
}
public class Gerente extends Funcionario {
    @Override
    public void calcularSalario(double salarioBruto, double descontos) {
        super.calcularSalario(salarioBruto * 1.05, descontos); //bonificação
    }
}
```

Sobreposição

```
public class Funcionario {
    private double salario;
    public void calcularSalario(double salarioBruto, double descontos){
        salario = salarioBruto - descontos;
    }
}
public class Gerente extends Funcionario {
    @Override
    public void calcularSalario(double salarioBruto, double descontos) {
        super.calcularSalario(salarioBruto * 1.05, descontos); //bonificação
    }
}
public class Diretor extends Funcionario {
    @Override
    public void calcularSalario(double salarioBruto, double descontos) {
        super.calcularSalario(salarioBruto * 1.10, descontos); //bonificação
    }
}
```

Lets practice;

Sobrecarga

- O tipo de polimorfismo de Sobrecarga permite a existência de vários métodos de mesmo nome

Sobrecarga

- O tipo de polimorfismo de Sobrecarga permite a existência de vários métodos de mesmo nome
- Com assinaturas levemente diferentes, variando no número e tipo de argumentos

Sobrecarga

- O tipo de polimorfismo de Sobrecarga permite a existência de vários métodos de mesmo nome
- Com assinaturas levemente diferentes, variando no número e tipo de argumentos
- Sobrecarga de Métodos são usados nos construtores de uma classe Java

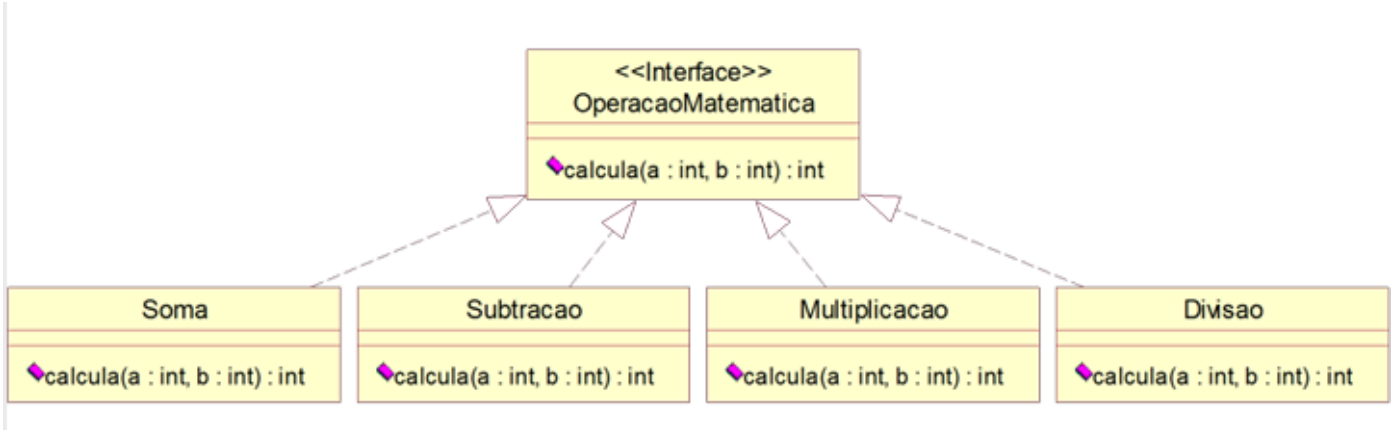
Sobrecarga

```
public class Funcionario {
    private double salario;
    public void calcularSalario(double salarioBruto, double descontos){
        salario = salarioBruto - descontos;
    }
    public void calcularSalario(double salarioBruto, double descontos, double bonificacao){
        double proventos = salarioBruto + bonificacao;
        calcularSalario(proventos, descontos);
    }
}
```

Lets practice;

Exercício #1

- Implemente o diagrama de classes abaixo com os seus respectivos métodos



- Após, sobrecarregue o método calcula em todas as classes com:
  - int calcula(a: double, b: double, c: double) e faça as operações de acordo com cada classe

Crie um método main para testar as operações.

Collections

Collections

- Desde as primeiras versões, Java dispõe das estruturas de arrays e as classes Vector e Hashtable.

https://www.devmedia.com.br/java-collections-como-utilizar-collections/18450

Collections

- Desde as primeiras versões, Java dispõe das estruturas de arrays e as classes Vector e Hashtable.
- A partir de Java 1.2, foi criado um conjunto de interfaces e classes denominado Collections Framework, que faz parte do pacote **java.util**

https://www.devmedia.com.br/java-collections-como-utilizar-collections/18450

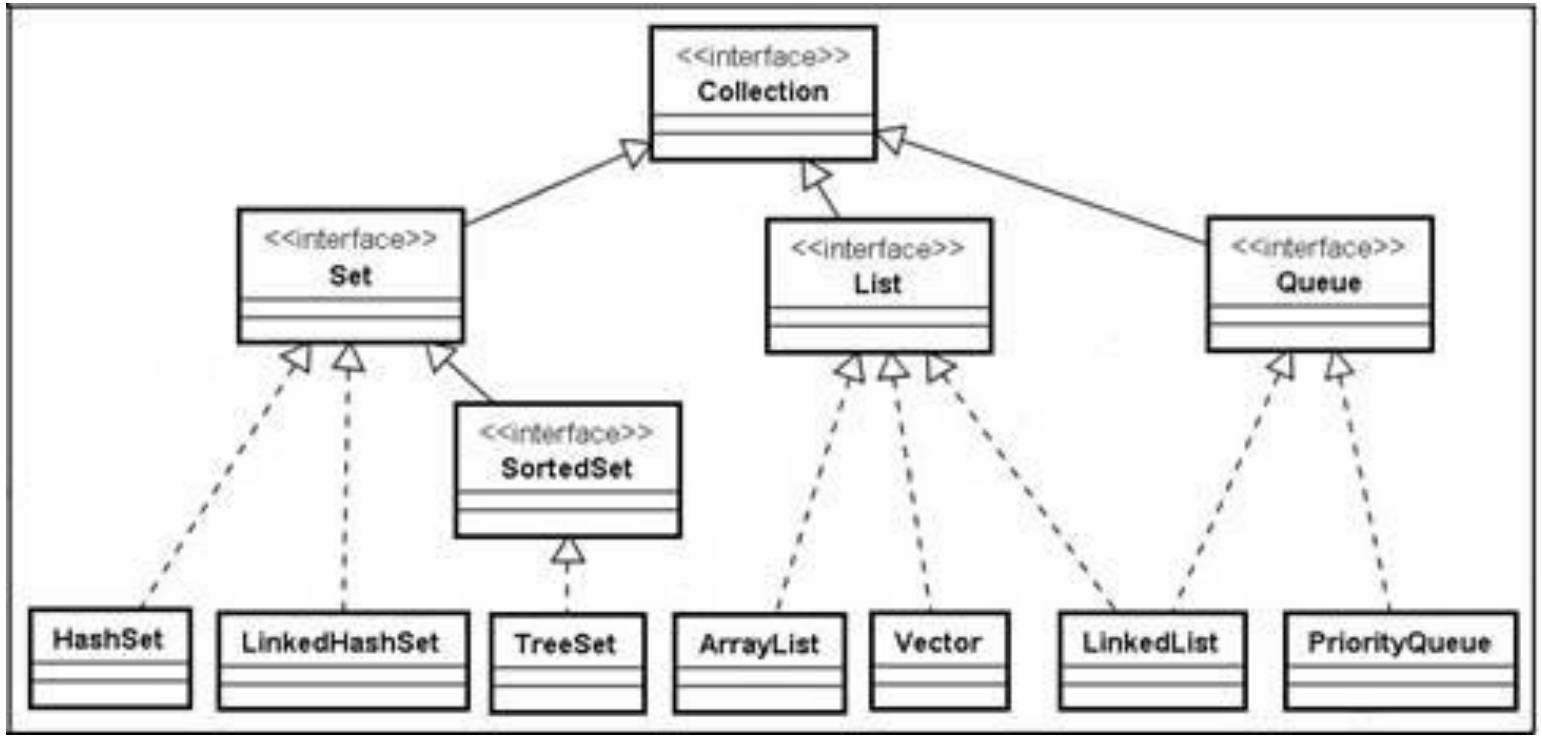
Collections

- Desde as primeiras versões, Java dispõe das estruturas de arrays e as classes Vector e Hashtable.
- A partir de Java 1.2, foi criado um conjunto de interfaces e classes denominado Collections Framework, que faz parte do pacote **java.util**
- Collections Framework é um conjunto bem definido de interfaces e classes para representar e tratar grupos de dados como uma única unidade

https://www.devmedia.com.br/java-collections-como-utilizar-collections/18450

Collections

- Listas => **java.util.**ArrayList
- Filas => **java.util.**LinkedList
- Pilhas => **java.util.**Stack
- Mapas => **java.util.**HashMap



Collections - Listas

```
ArrayList<String> lista = new ArrayList<>();
```

- add(elemento)
- remove(index)
- size()
- get(index)

Lets practice;

Exercício #2

- Crie uma lista de nomes e peça ao usuário para ir digitando nomes até que ele digite “SAIR”,
  - Imprima o penúltimo nome da lista
  - Imprima o primeiro nome da lista
  - Remova o último nome da lista
  - Ao final imprima todos os nomes e também a quantidade de nomes que sobraram na lista.

Collections – Filas

```
Queue<String> fila = new LinkedList<>();
```

- add(elemento)
- poll()
- size()

Exercício #3

- Crie um sistema de senhas onde cada pessoa que chegar pegue uma senha numérica sequencial...
- Faça 5 pessoas entrarem na fila
- Faça 2 pessoas serem atendidas
- Faça 1 pessoa ser atendida
- Faça mais 3 pessoas entrarem na fila
- Faça 3 pessoas serem atendias
- Imprimir ao final a fila com todos os valores da mesma

Lets practice;

Collections - Pilhas

```
Stack<String> pilha = new Stack<>();
```

- add(elemento)
- pop()
- size()

Lets practice;

Exercício #4

- Crie um programa que leia 15 números e proceda, para cada um deles, como segue:
  - se o número for par, insira-o na pilha;
  - se o número lido for ímpar, retire um número da pilha;
  - ao final, se ainda conter elementos, esvazie a pilha imprimindo os elementos.

Collections - Mapas

```
Map<String, String> tabela = new HashMap<>();
```

- put(chave, valor)
- remove(chave)
- get(chave)

Lets practice;

Exercício #5

- Crie um programa que leia o cpfs, o nomes de pessoas e adicione a um Mapa.
  - Em seguida, peça ao usuário para consultar um cpf e exiba o nome daquela pessoa com o cpf digitado e remove o cpf caso esteja cadastrado. Mostrar a mensagem que o cpf não existe caso não exista.
  - Ao final imprima o conteúdo do Mapa.

Comparação de Listas

- Utiliza a interface **Comparable** para tal...
- Ordena os elementos de acordo com algum critério pré-determinado pelo dev.

```
carros.sort(new Comparator<Carro>() {
    @Override
    public int compare(Carro o1, Carro o2) {
        return o1.getQuilometragem() - o2.getQuilometragem();
    }
});
```

Exercício #6

- Crie uma lista de nomes e idades (objeto Pessoa) (pode ser fixa) e ordene de forma crescente por nome e imprima os valores.
- Na sequência, ordene por idade (do mais velho para o mais novo) e imprima os valores.
- Por fim, ordene por idade e por nome respeitado o critério:
  - Nome crescente
    - Se nome for igual, ordena por idade

Homework

- Troque os atributos “arrays” da classe cliente para ArrayLists.
- Crie uma nova classe **ContaPagamento** que deverá estender de **Conta** e implementar a interface **Imprimir**, porém com um atributo estático TAXA_SAQUE com o valor de R$4,25
  - Sobrescreva o método sacar(valor) e faça descontar o valor da taxa do saldo da conta pagamento (lembrando que a conta não pode ficar negativa)
- Crie um método main, crie 2 clientes sendo:
  - 1 cliente com conta pagamento e conta corrente
  - 1 cliente com conta poupança
  - Realize ao menos 3 movimentações entre as contas (saque, depósito e transferência)
  - Ao final, imprima os dados da conta