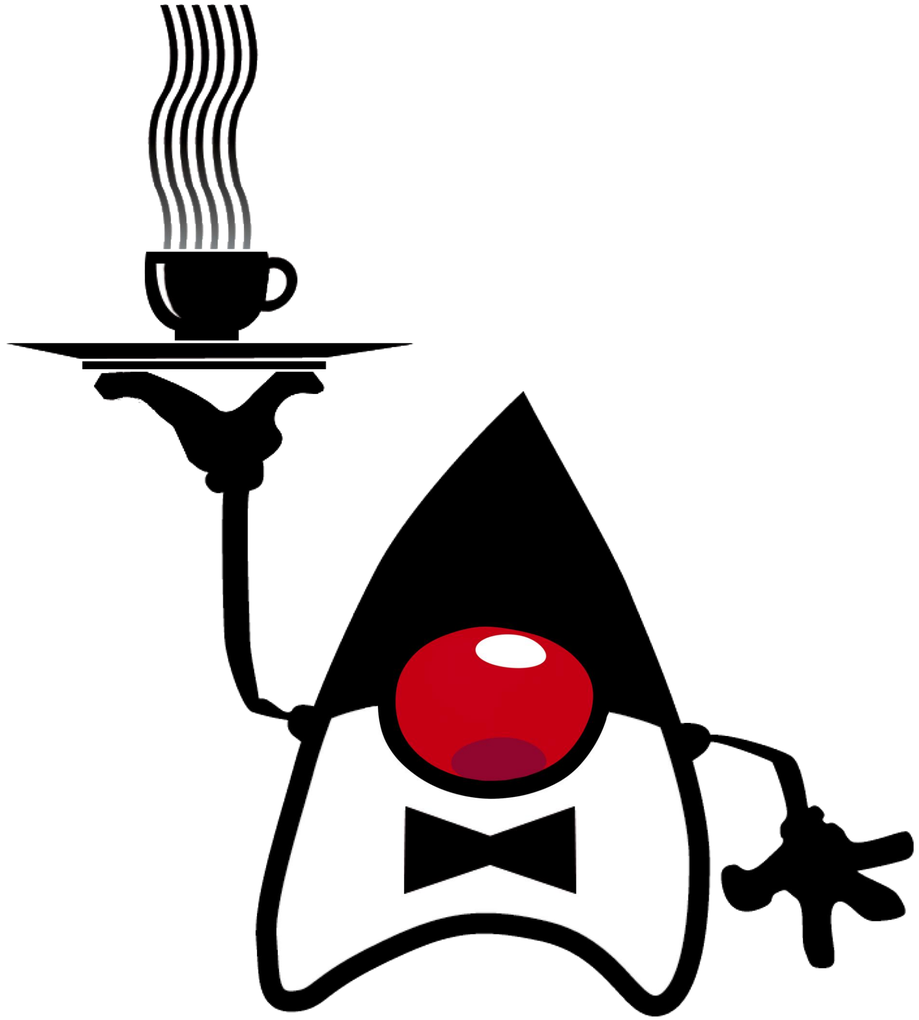
Trilha Java

Encontro 23 – Tópicos Modernos



vnt/school
powered by venturus

Recapitulação

1. Polimorfismo de Sobreposição
2. Assinatura de Classe
3. Polimorfismo de Sobrecarga

vnt/school
powered by venturus

Agenda

1. List
2. Generics
3. hashCode e equals
4. Set
5. Map
5. Exemplos

vnt/school
powered by venturus

Lista

Java

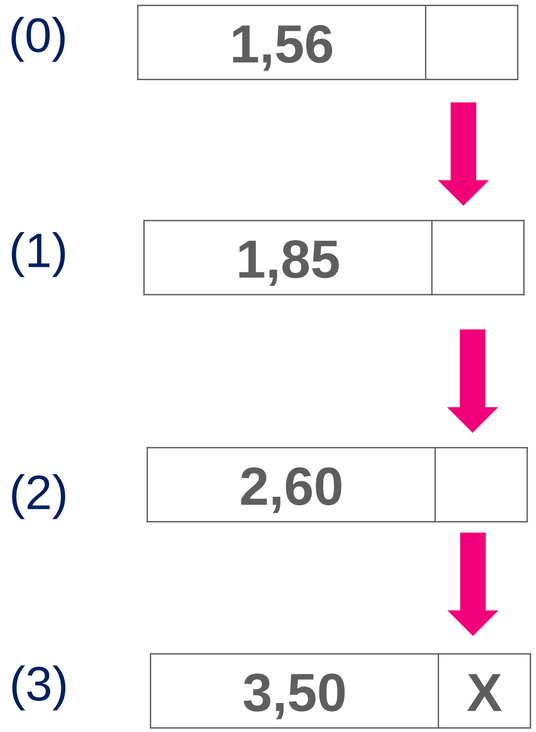
vnt/school

powered by venturus

Lista

Lista é uma estrutura de dados:

Homogênea (dados do mesmo tipo).
Ordenada (elementos acessados por meio de posições).
Inicia vazia, e seus elementos são alocados sob demanda.
Cada elemento ocupa um "nó" (ou nodo) da lista



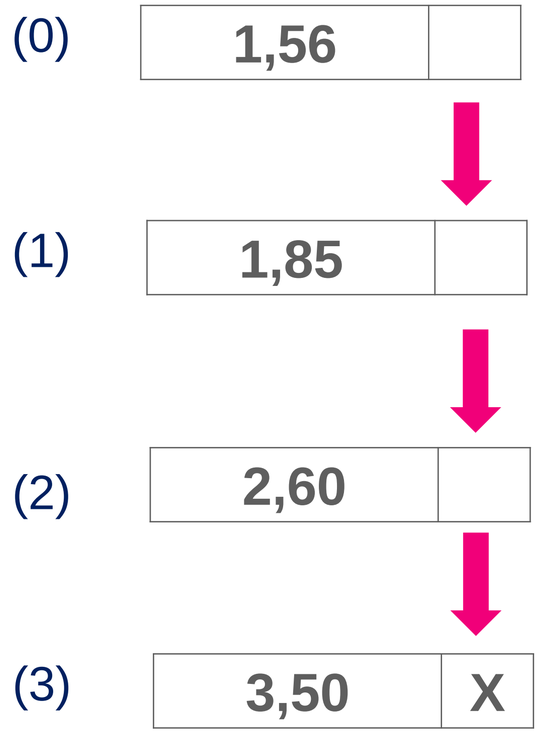
vnt/school
powered by venturus

Lista

Tipo (interface): **List**

Classes que implementam:
ArrayList, LinkedList, etc.

Vantagens:
Tamanho variável
Facilidade para se realizar inserções e deleções
Desvantagens:
Acesso sequencial aos elementos



vnt/school
powered by venturus

Lista

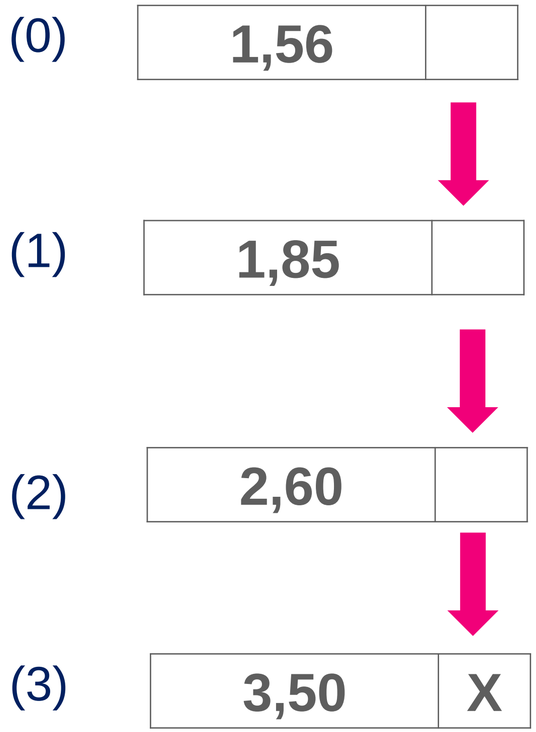
Tamanho da lista: **size()**
Obter o elemento de uma posição: **get(position)**
Inserir elemento na lista: **add(obj), add(int, obj)**

Remover elementos da lista:
remove(obj), remove(int), removeIf(Predicate)

Encontrar posição de elemento:
indexOf(obj), lastIndexOf(obj)

Filtrar lista com base em predicado:

```
List result = list.stream().filter(x -> x >4).collect(Collectors.toList());
```



vnt/school
powered by venturus

Lista

Exemplo 1:

```
public class Exemplo1 {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("Maria");
        list.add("Alex");
        list.add("Bob");
        list.add("Anna");
        //list.add(2, "Marco");
```

Generics

Java

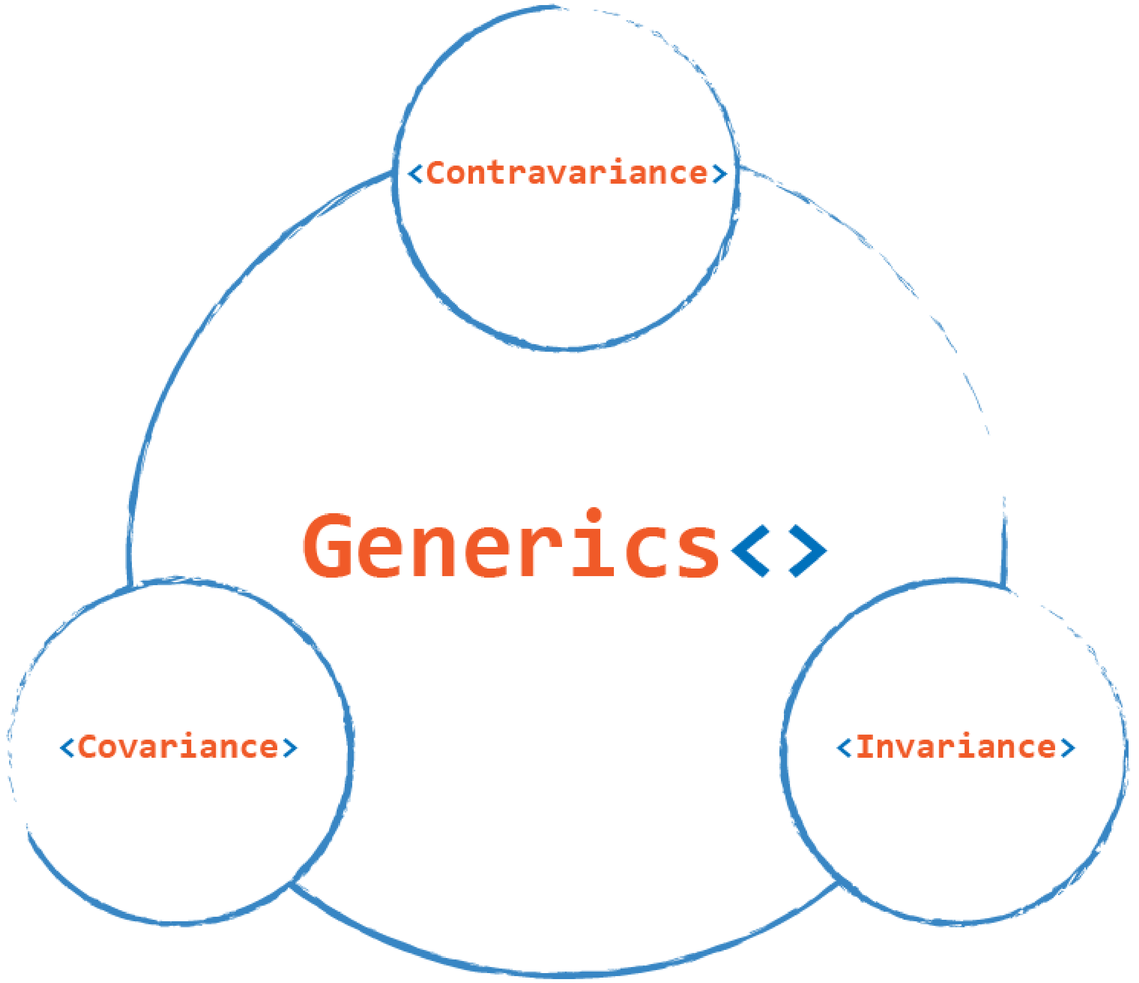
vnt/school

powered by venturus

Generics

Generics é uma maneira de criar parâmetros para classes e definir tipos que podem ser substituídos em vários lugares do programa.

Está associado com a ideia de criar **métodos e classes genéricas** que podem ser declaradas uma vez, mas usadas com vários tipos de dados diferentes.

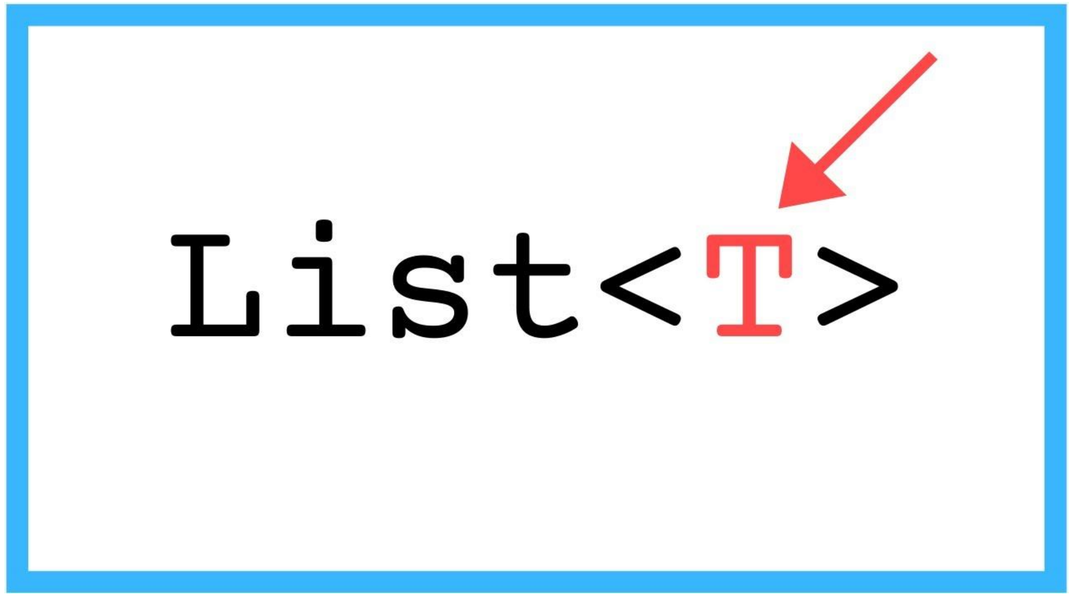


vnt/school
powered by venturus

Generics

Generics permitem que **classes**, **interfaces** e **métodos** possam ser parametrizados por tipo.

Seus benefícios são:
Reuso
Type safety
Performance



vnt/school
powered by venturus

Generics

Uso comum: **coleções**

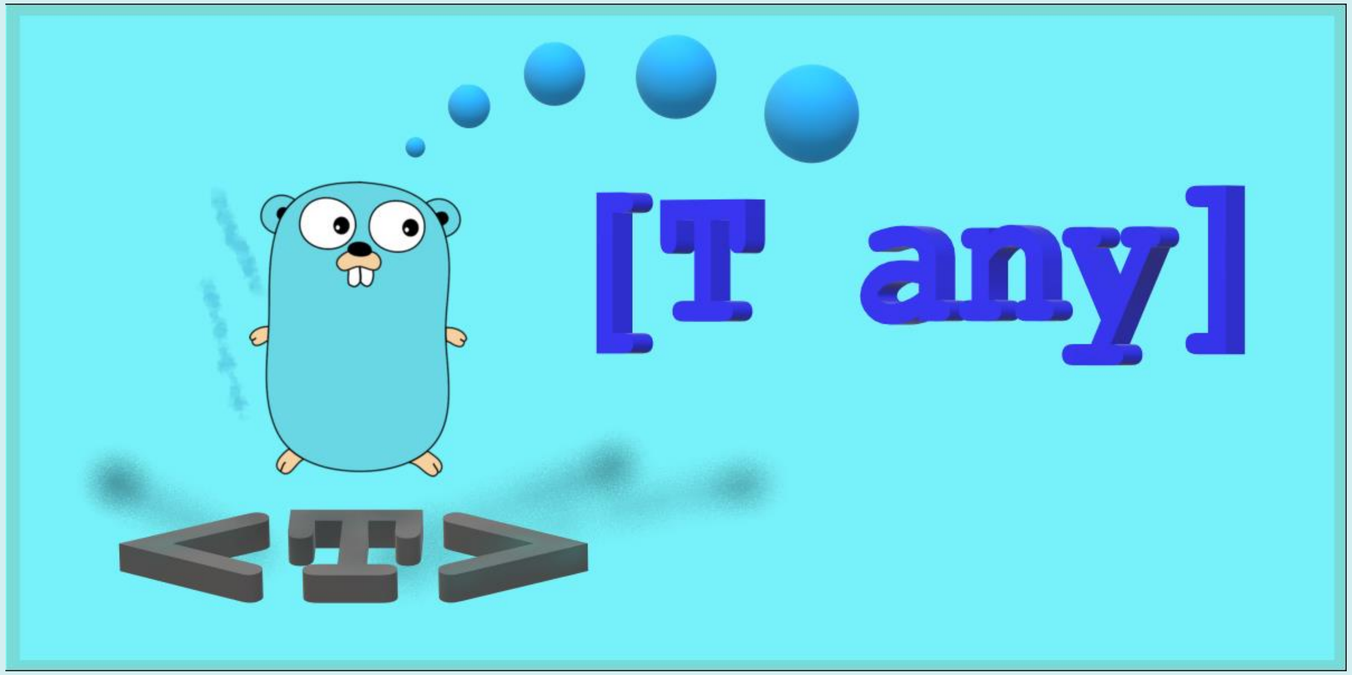
```
List <String> list = new ArrayList<>();
list.add("Maria");
String name = list.get(0);
```

Uma coleção é uma estrutura de dados que permite armazenar vários objetos.
Operações básicas para as coleções são: adicionar, remover, esvaziar, etc.
Os dois grandes tipos de coleções: List e Map.

vnt/school
powered by venturus

Generics

A motivação de estudar **Generics em Java** é de poupar o desenvolvedor de códigos redundantes, como é o caso de casting excessivo.



Generics

Exemplo 2: Faça um programa que leia uma quantidade N, e depois leia os N números inteiros. Ao final, imprima esses números de forma organizada conforme exemplo. Em seguida, informe qual foi o primeiro valor digitado.

Quantidade? 4
10
8
1
6
[10, 8, 1, 6]
Primeiro: 10

Criar um serviço de impressão:

ServicoImpressão
+ adicionaValor(valor: int): void + primeiro(): int + imprime(): void

vnt/school
powered by venturus

Generics

Solução:

```
public class Exemplo2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        ServicoImpressao ps = new ServicoImpressao();
        System.out.print("Quantidade? ");
        int n = sc.nextInt();

        for (int i = 0; i < n; i++) {
            Integer value = sc.nextInt();
            ps.adicionaValor(value);
        }
```

Generics

E se resolvermos tratar com parâmetros do tipo String?

Seria possível fazer reuso do código?

```
for (int i = 0; i < n; i++) {
        String value = sc.next();
        ps.adicionaValor(value);
    }
```

Quantidade? 3
Maria
João
Pedro
[Maria, João, Pedro]
Primeiro: Maria

A Classe principal não vai rodar, pois o Serviço de Impressão não comporta parâmetros do tipo **String**.

vnt/school
powered by venturus

Generics

Exemplo 2:

Solução : refazer o Serviço de Impressão.

Problema: Reuso

```
public class Exemplo2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        ServicoImpressaoString ps = new ServicoImpressaoString();
        System.out.print("Quantidade? ");
        int n = sc.nextInt();

        for (int i = 0; i < n; i++) {
            String value = sc.next();
            ps.adicionaValor(value);
        }
```

Generics

E se resolvermos tratar os parâmetros como "object".

Em java basicamente tudo é objeto. Logo, daria certo a solução.

Mas, teriamos algum problema com com TypeSafety ?

ServicoImpressão
+ adicionaValor(valor: object): void + primeiro(): object + imprime(): void

Generics

Exemplo 3:

Funciona bem para String e Inteiros.

Problema: Se resolver guardar o resultado numa variável Integer.

Type Safety
Performace

```
public class Exemplo3 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        ServicoImpressao ps = new ServicoImpressao();
        System.out.print("Quantidade? ");
        int n = sc.nextInt();

        ps.adicionaValor("Maria");

        for (int i = 0; i < n; i++) {
            //String value = sc.next();
            int value = sc.nextInt();
            ps.adicionaValor(value);
        }
```

vnt/school
powered by venturus

Generics

Solução:

```
ps.imprime();
int x = ps.primeiro();
System.out.println("Primeiro: " + x);
sc.close();
```

A variável **x** do tipo **inteiro** não aceita guardar o resultado do tipo **object**.

```
ps.imprime();
int x = (int) ps.primeiro();
System.out.println("Primeiro: " + x);
sc.close();
```

A solução é forçar a variável aceitar. Basta fazer um "**Casting**"

vnt/school
powered by venturus

Generics

Exemplo 3:

Outro problema é se você resolver adicionar no seu serviço de impressão o objeto "**Maria**"

A melhor solução é criar um parâmetro do tipo mais generico: **Generics**.

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    ServicoImpressao ps = new ServicoImpressao();
    System.out.print("Quantidade? ");
    int n = sc.nextInt();

    ps.adicionaValor("Maria");

    for (int i = 0; i < n; i++) {
        //String value = sc.next();
        int value = sc.nextInt();
        ps.adicionaValor(value);
    }
```

Generics

Exemplo 4:

O tipo **Integer** pode ser alterado pra **String**. O programa **funciona**.

Veja que se estiver inteiro, não aceita "**Maria**".

Não precisa do **Casting**.

```
public class Exemplo4 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        ServicoImpressao<String> ps = new ServicoImpressao<>();
        System.out.print("Quantidade? ");
        int n = sc.nextInt();

        for (int i = 0; i < n; i++) {
            String value = sc.next();
            ps.adicionaValor(value);
        }
```

??

vnt/school
powered by venturus

HashCode e equals

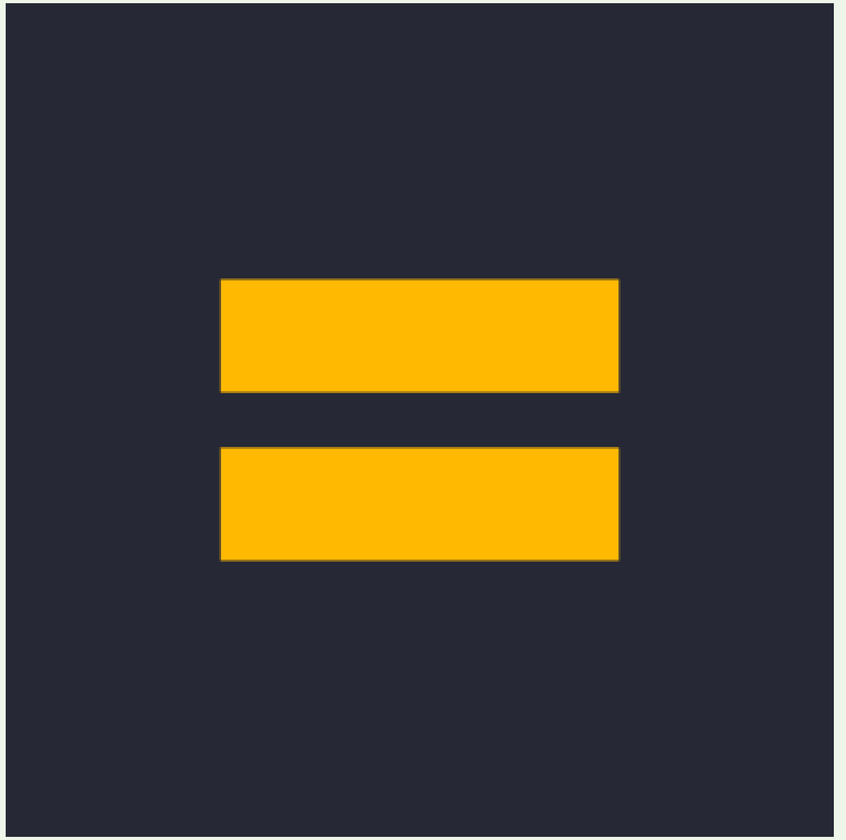
vnt/school
powered by venturus

HashCode e Equals

São operações da classe Object utilizadas para comparar se um objeto é igual a outro.

Equals: lento, resposta 100%

HashCode: rápido, porém resposta positiva não é 100%.



Tipos comuns (String, Date, Integer, Double, etc.) já possuem implementação para essas operações. Classes personalizadas precisam sobrepô-las.

vnt/school
powered by venturus

Equals

Método que compara se o objeto é igual a outro, retornando true ou false.

```
String a = "Maria";
String b = "Alex";
System.out.println(a.equals(b));
```

HashCode

Método que retorna um número inteiro representando um código gerado a partir das informações do objeto

```
String a = "Maria";
String b = "Alex";

System.out.println(a.hashCode());
System.out.println(b.hashCode());
```

HashCode

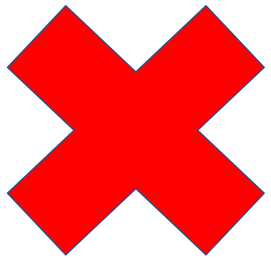
Regra de ouro do HashCode:

Se o hashCode de dois objetos for diferente, então os dois objetos são diferentes

Isso nunca acontece:

Jose Erinaldo → **-242670543**

Jose Erinaldo → **880483901**



Se o código de dois objetos for igual, muito provavelmente os objetos são iguais (pode haver colisão)

vnt/school
powered by venturus

Set<T>

Java

vnt/school

powered by venturus

Set<T>

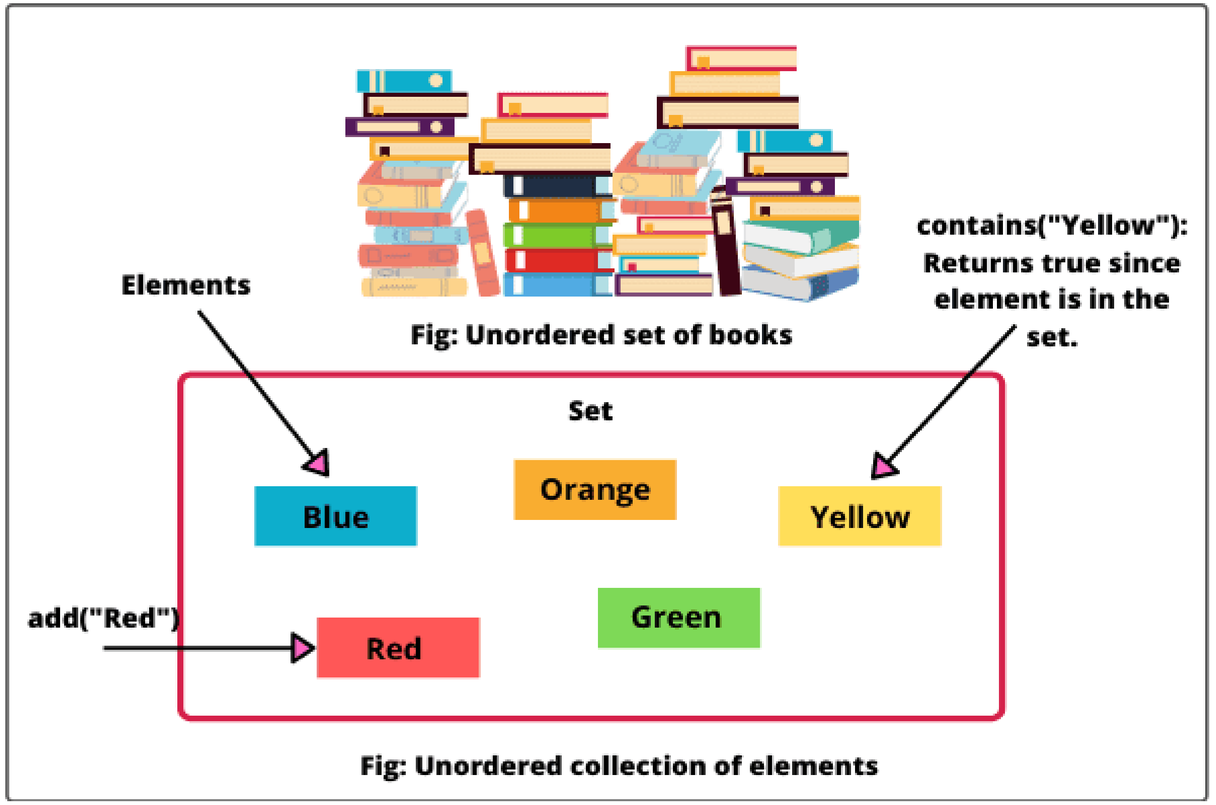
Representa um conjunto de elementos (similar ao da Álgebra):

Não admite repetições

Elementos não possuem posição

Acesso, inserção e remoção de elementos são rápidos

Oferece operações eficientes de conjunto: interseção, união, diferença.



vnt/school
powered by venturus

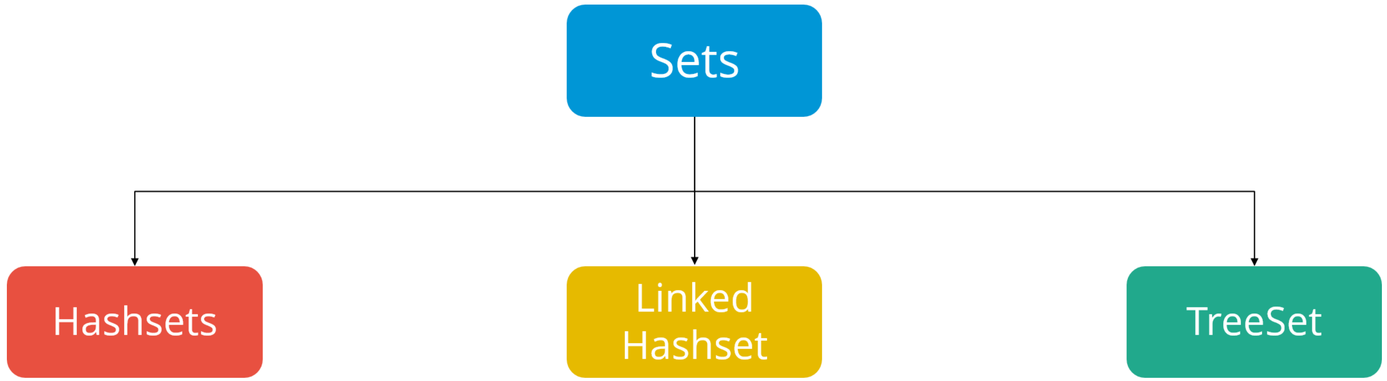
Set

Principais implementações:

HashSet - mais rápido (operações O(1)) e não ordenado.

TreeSet - mais lento (operações O(log(n))) e ordenado pelo compareTo do objeto (ou Comparator).

LinkedHashSet - velocidade intermediária e elementos na ordem em que são adicionados.



vnt/school
powered by venturus

Set

Alguns Métodos Importantes:

add(obj), **remove**(obj), **contains**(obj) : baseado em equals e hashCode.

clear(), **size**(), **removeIf**(predicate) .

addAll(other):
união: adiciona no conjunto elementos do outro conjunto, sem repetição.

retainAll(other):
interseção: remove do conjunto os elementos não contidos em other.

removeAll(other):
diferença: remove do conjunto os elementos contidos em other.

Set

Exemplo 5:

Crie um programa que instancie um objeto do tipo **Set**. Adicione os objetos TV, NOTEBOOK e TABLET. Em seguida faça a confirmação se este Set contém o objeto NOTEBOOK. Imprima o resultado e o conjunto de objetos.

vnt/school
powered by venturus

Set

Não garante a ordem.

A ordem pode ser diferente conforme foi digitado.

```
public class Exemplo5 {
    public static void main(String[] args) {
        Set<String> set = new HashSet<>();
        set.add("Tv");
        set.add("Tablet");
        set.add("Notebook");
        for (String p : set) {
            System.out.println(p);
        }
```

??

vnt/school
powered by venturus

Set

Como Set testa igualdade?

Como as coleções Hash testam igualdade?

Se hashCode e equals estiverem implementados:

Primeiro: hashCode. Se der igual, usa equals para confirmar.

Lembre-se: String, Integer, Double, etc. já possuem equals e hashCode.

Se hashCode e equals NÃO estiverem implementados:

Compara as referências (ponteiros) dos objeto

Set

Exemplo 6:

Crie um programa com uma classe Produto conforme ao lado. Na classe principal instancie os objetos Set e Produto. Adicione em Set os produtos conforme a tabela. E para o objeto Produto, instancie o "Notebook" com o preço "1200". Faça uma comparação e veja se o objeto Set contém o objeto Produto.

True ou False

Produto
- String name - Double price
Implemente o construtor, Getters e Setters.

Produtos	
TV	900.0
Notebook	1200.0
Tablet	400.0

vnt/school
powered by venturus

Set

O resultado é falso porque a classe Produto não contém a implementação do **HashCode** e **Equals**.

Para implementar, basta clicar com o botão direito, clicar em "Insert Code" e clicar em **HashCode() and Equals()**.

```
public class Exemplo6 {
    public static void main(String[] args) {
        Set<Produto> set = new HashSet<>();
        set.add(new Produto("TV", 900.0));
        set.add(new Produto("Notebook", 1200.0));
        set.add(new Produto("Tablet", 400.0));

        Produto prod = new Produto("Notebook", 1200.0);
        System.out.println(set.contains(prod));
    }
}
```

vnt/school
powered by venturus

Set

O resultado agora será **True**, pois o **hashCode** e o **equals** foram **implementados** na classe **Produto**.

```java
@Override
public int hashCode() {
    int hash = 3;
    hash = 29 * hash + Objects.hashCode(this.name);
    hash = 29 * hash + Objects.hashCode(this.price);
    return hash;
}

@Override
public boolean equals(Object obj) {
    if (this == obj) {
        return true;
    }
    if (obj == null) {
        return false;
```

??

Set

Como o TreeSet compara os elementos?

No geral, podemos dizer que ele implementa o **Comparable** e usa o **CompareTo** para comparar.

Analisando o exemplo anterior dos produtos, você pode fazer a comparação via "**name**" ou "**price**".

Para este exemplo iremos fazer a comparação via "**name**". Portanto, o resultado deve ordenar os produtos de forma alfabética.

Produtos	
TV	900.0
Notebook	1200.0
Tablet	400.0

vnt/school
powered by venturus

Set

A solução basicamente segue o código anterior, devendo ser acrescentado a implementação: **implements Comparable<Produto>**

```
@Override
public String toString() {
    return "Produto [name=" + name + ", price=" + price + "]";
}
@Override
public int compareTo(Produto other) {
    return name.toUpperCase().compareTo(other.getName().toUpperCase());
}
```

Coffee time!

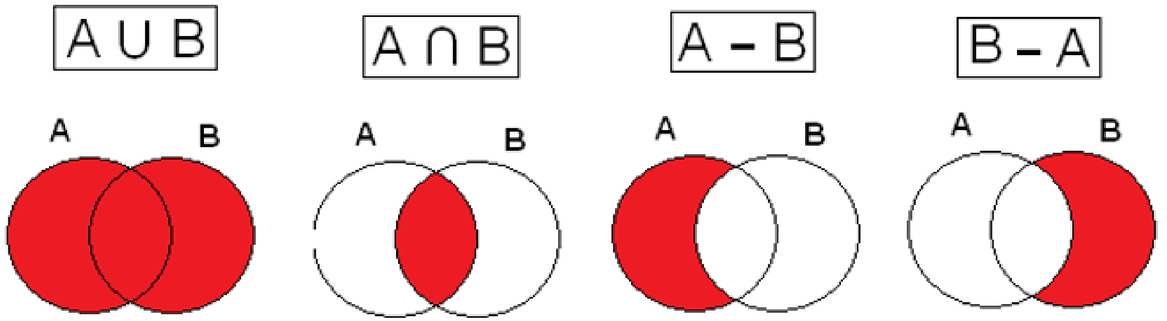
Set

Atividade 1:

Crie um programa que instancie dois objetos do tipo **Set,** veja o exemplo abaixo. Em seguida faça as operações UNIÃO, INTERSEÇÃO e DIFERENÇA entre esses conjuntos. Imprima o resultado.

a -> (0,2,4,5,6,8,10)

b -> (5,6,7,8,9,10)



vnt/school
powered by venturus

Set

```java
public class Atividade1 {
    public static void main(String[] args) {
        Set<Integer> a = new TreeSet<>(Arrays.asList(0, 2, 4, 5, 6, 8, 10));
        Set<Integer> b = new TreeSet<>(Arrays.asList(5, 6, 7, 8, 9, 10));
//union
        Set<Integer> c = new TreeSet<>(a);
        c.addAll(b);
        System.out.println(c);
//intersection
        Set<Integer> d = new TreeSet<>(a);
        d.retainAll(b);
        System.out.println(d);
//difference
        Set<Integer> e = new TreeSet<>(a);
        e.removeAll(b);
        System.out.println(e);
    }
```

Map<K,V>

Coleção

vnt/school

powered by venturus

Map<K,V>

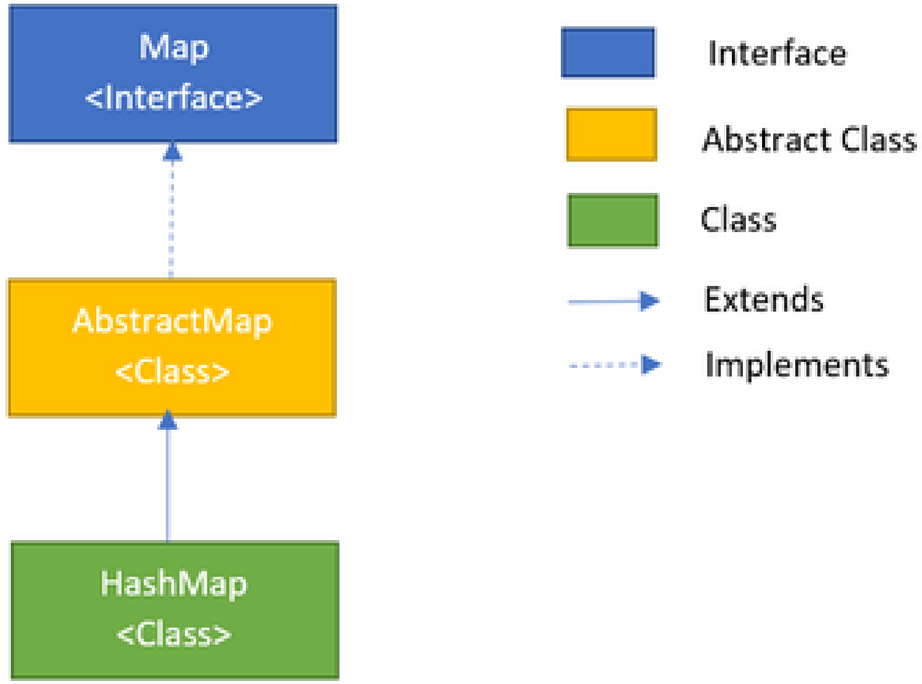
É uma coleção de pares chave / valor:

Não admite repetições do objeto chave.

Os elementos são indexados pelo objeto chave (não possuem posição).

Acesso, inserção e remoção de elementos são rápidos.

Uso comum: cookies, local storage, qualquer modelo chave-valor.



vnt/school
powered by venturus

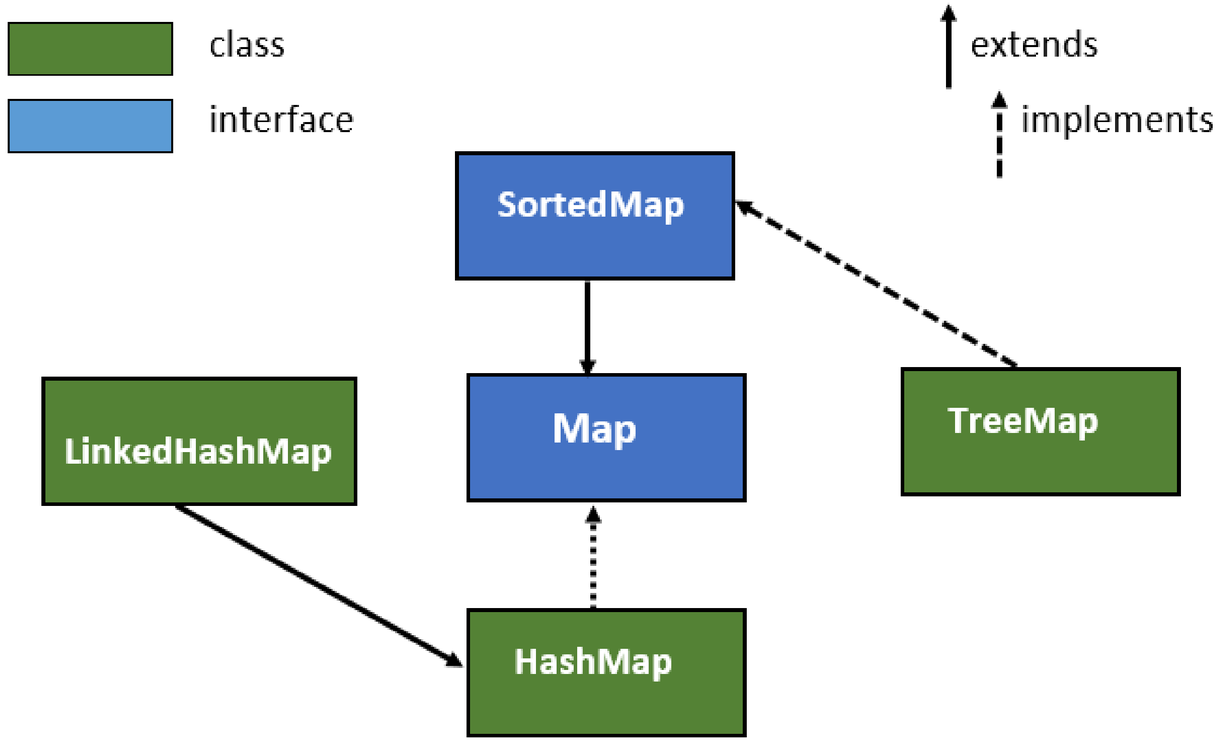
Map<K,V>

Principais implementações:

HashMap - mais rápido (operações O(1)) e não ordenado.

TreeMap - mais lento (operações O(log(n)) e ordenado pelo compareTo do objeto (ou Comparator).

LinkedHashMap - velocidade intermediária e elementos na ordem em que são adicionados.



vnt/school
powered by venturus

Map<K,V>

Alguns métodos importantes:

put(key, value), remove(key), containsKey(key), get(key)
Baseado em equals e hashCode
Se equals e hashCode não existir, é usada comparação de ponteiros.

clear()
size()

keySet() - retorna um Set

Map<K,V>

Exemplo 8:

Crie um programa que instancie um objeto Map. Insira os valores/chaves conforme mostrado na tabela. Em seguida faça testes como remover() e put() para algumas chaves e veja o resultado. Imprima o resultado final para cada caso.

Chave	Valor
username	Maria
email	maria@gmail.com
phone	99128097

Map<K,V>

```java
public class Exemplo8 {
    public static void main(String[] args) {
    Map<String, String> cookies = new TreeMap<>();
    cookies.put("username", "maria");
    cookies.put("email", "maria@gmail.com");
    cookies.put("phone", "99771122");
    //cookies.remove("email");
    cookies.put("phone", "99771133");
    System.out.println("Contem chave 'phone': " + cookies.containsKey("phone"));
    System.out.println("Numero Phone: " + cookies.get("phone"));
    System.out.println("Email: " + cookies.get("email"));
    System.out.println("Tamanho: " + cookies.size());
    System.out.println("TODOS COOKIES:");
    for (String key : cookies.keySet()) {
        System.out.println(key + ": " + cookies.get(key));
    }
```

??

vnt/school
powered by venturus

Map<K,V>

Exemplo 9:

Crie um programa com uma classe Produto conforme ao lado. Na classe principal instancie os objetos Map e Produto. Adicione em Map os produtos conforme a tabela. E para o objeto Produto, instancie a "TV" com o preço "900". Faça uma comparação e veja se o objeto Map contém o objeto Produto.

True ou False

Produto
- String name - Double price
Implemente o construtor, Getters e Setters.

Produtos	
TV	900.0
Notebook	1200.0
Tablet	400.0

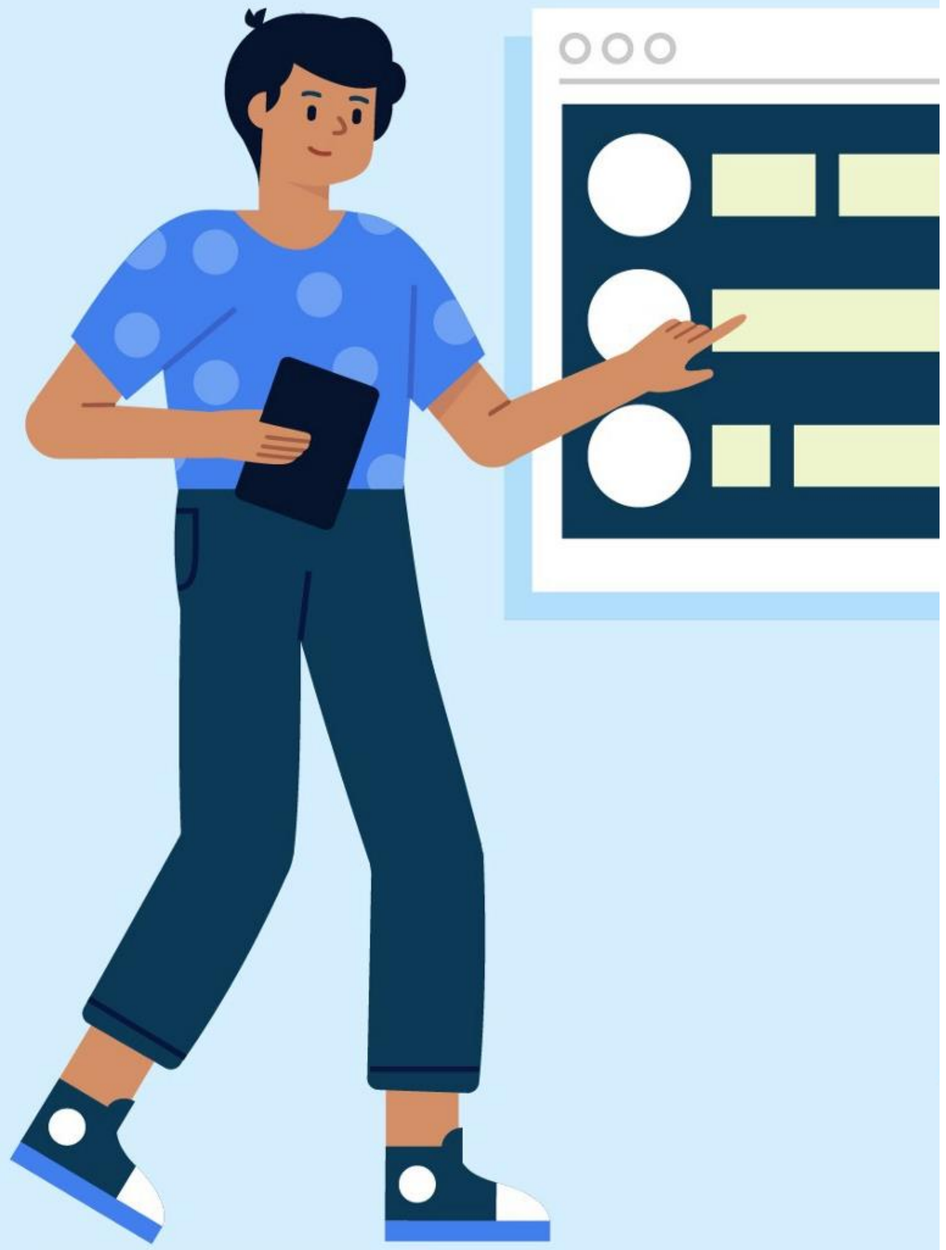
vnt/school
powered by venturus

Map<K,V>

O resultado só será **true** se a **classe Produto** possuir a implementação de **hashCode e equals**.

```
public class Exemplo9 {
    public static void main(String[] args) {
        Map<Produto, Double> estoque = new HashMap<>();
        Produto p1 = new Produto("Tv", 900.0);
        Produto p2 = new Produto("Notebook", 1200.0);
        Produto p3 = new Produto("Tablet", 400.0);
        estoque.put(p1, 10000.0);
        estoque.put(p2, 20000.0);
        estoque.put(p3, 15000.0);
        Produto ps = new Produto("Tv", 900.0);
        System.out.println("Contem chave 'ps': " + estoque.containsKey(ps));
    }
```


Review
e
Preview



Comunidade VNT

Os dois links abaixo apresentam informações na página oficial da Oracle sobre Set e Map. Aproveite para conferir um pouco mais sobre o assunto.

Set (Java SE 10 & JDK 10) (oracle.com)

Map (Java SE 10 & JDK 10) (oracle.com)

Boa leitura!!

[1] A. Goldman, F. Kon, Paulo J. S. Silva; Introdução à Ciência da Computação com Java e Orientação a Objetos (USP). 2006. Ed. USP.
[2] Algoritmo e lógica de programação. Acessado julho/2022: https://visualg3.com.br/
[3] G. Silveira; Algoritmos em Java; Ed. Casa do Código.
[4] M. T. Goodrich, R. Tamassia; Estrutura de dados e algoritmos em Java. Ed Bookman. 2007.
[5] Algoritmo e lógica de programação. Acessado julho/2022: https://www.cursoemvideo.com/
[6] P. Silveira, R. Turini; Java 8 Pratico: lambdas, streams e os novos recursos da linguagem. Ed. Casa do Código.
[7] Linguagem Java: Curso acessado em agosto/2022: https://www.udemy.com/
[8] Linguagem Java: Curso acessado em setembro/2022: https://www.cursoemvideo.com/