

POO – JAVA

Objeto

- É uma abstração de entidade ou conceito real sobre o objetivo do programa

* Entidade real: árvore, rua, carro

* Conceito real: inscrição, subscrição, entrevista

- Junta dados e operações em uma única estrutura computacional

IMPORTANTE: Não é uma estrutura de dados, é uma estrutura computacional
porque possui operações/comportamento mais os dados

- Pode ter dados, pode ter operações, e pode ter dados e operações

Abstração

- É o ato de concentrar nos aspectos essenciais de um contexto, ignorando características que não ajudam a resolver o problema deste contexto

- Abstração em programação de computadores é uma forma de reduzir a complexidade e tornar o projeto e a implementação mais eficientes em sistemas complexos de software

Atributos

- Estrutura responsável por armazenar dados como características, capacidades e dados de controle do objeto

- O tipo do atributo deve ser especificado

Métodos

- Estrutura responsável pelo comportamento dos objetos

- Contém o código executado para dar esse comportamento ao objeto

- Para fins de comparação com o paradigma estruturado, é similar à função do C

- Pode retornar valor e pode receber parâmetros – depende da sua assinatura

Classe

- Estrutura computacional com a estrutura dos dados (atributos) e o comportamento (métodos) para a criação de objetos
 - Serve de molde/fôrma para a criação de ao menos um objeto (normalmente vários)
 - A criação de objetos é chamada de instanciação
 - Instanciar um objeto é a criação de um novo objeto na memória com base em uma classe
- * Essa operação exige a existência de uma classe para ser usada como “molde”

Método Construtor

- Método especial para instanciar objetos a partir da classe;
 - Serve para inicializar o estado da instância
- * Necessário quando o valor inicial dos atributos deve ser diferente do seu valor padrão de inicialização

```
visibilidade NomeDaClasse(parâmetros) {}
```

Método Destrutor

- Método especial que é executado logo antes do objeto ser destruído, ou seja, ser retirado da memória
- É utilizado principalmente para liberar recursos utilizados pela instância do objeto
- **Atenção: não existe em Java.**

Sobrecarga (Overloading) de Métodos

- Não é possível ter mais de um método com a mesma assinatura
- * Assinatura: visibilidade-nome-parâmetros-retorno
- Porém, podemos ter vários métodos na classe com o mesmo nome (sobrecarga)

Importante: Desde que a assinatura deles seja diferente


- A sobrecarga é utilizada para permitir a utilização de uma lógica com variações nos tipos de dados de entrada (quantidade e/ou tipo parâmetros) e/ou saída (tipo de retorno) do método

void	println() Terminates the current line by writing the line separator string.
void	println(boolean x) Prints a boolean and then terminate the line.
void	println(char x) Prints a character and then terminate the line.
void	println(char[] x) Prints an array of characters and then terminate the line.
void	println(double x) Prints a double and then terminate the line.
void	println(float x) Prints a float and then terminate the line.
void	println(int x) Prints an integer and then terminate the line.
void	println(long x) Prints a long and then terminate the line.
void	println(Object x) Prints an Object and then terminate the line.
void	println(String x)

Visibilidade (Controle de Acesso)


- Mecanismo para determinar quem tem acesso (pode ver) a classe/atributo/membro;
- O controle de acesso pode ser definido para:
 - * **classes** – uma classe inteira
 - * **membros das classes** (atributos e métodos, incluindo os métodos construtores)

Visibilidade para classes



<i>Tipo</i>	<i>Modificador</i>	<i>Visibilidade</i>
privado	private	Somente para classes internas (classes especiais).
pacote	package	Classes dentro do próprio pacote.
protegido	protected	Somente para classes internas (classes especiais).
público	public	Qualquer classe.

Visibilidade para membros de classes



<i>Tipo</i>	<i>Modificador</i>	<i>Visibilidade</i>
privado	<code>private</code>	Dentro da própria classe.
pacote	<code>package</code>	Dentro da própria classe, classes do seu pacote.
protegido	<code>protected</code>	Dentro da própria classe, classes do seu pacote e em subclasses.
público	<code>public</code>	Qualquer classe.

Encapsulamento

- Trata-se de proteger dados e comportamentos dentro de “cápsulas” para serem utilizados de forma controlada
- O mecanismo para implementar essa cápsula é:
 - a. Esconder/ocultar os membros da classe usando os modificadores de visibilidade
 - b. Criar métodos com visibilidade maior para acessar o membro “escondido”

Igualdade de objetos

- Por padrão, a igualdade é dada pela localização na memória (referência), ou seja, se as duas variáveis apontam para o mesmo objeto na memória
- Esse comportamento deve ser alterado para atender a necessidade do programa, se for necessário
- Para alterar, é necessário implementar o método `compareTo`

“`a.compareTo(b)` retornando -1 (ou algum outro número negativo) significa que a antecede b, se retorna +1 (ou algum outro número positivo) então a sucede b e se retorna 0 então a e b são similares. Entretanto a definição exata de ‘antecede’, ‘sucede’ e ‘similar’ fica a critério de cada classe que implementa o `compareTo`. Por exemplo: a classe `String` define por algo parecido com a ordenação alfabética.”

```
Carro cerato = new Carro();  
Carro sedan = cerato;  
cerato == sedan      → verdadeiro
```

```

public class Pessoa {
    public String nome;
    public int compareTo(Pessoa p) {
        return nome.compareTo(p.nome);
    }
}

p1.nome = "Zoe";
p2.nome = "ZOE";
//ambas são Pessoa

// sem implementar o compareTo
p1 == p2    → falso

// depois de implementar o
compareTo (código ao lado)
p1 == p2    → verdadeiro

```

Estado x Representação

- Estado

- * Estado do objeto é dado pelo estado de seus atributos
- * Estado do atributo é dado pela informação contida nele
- * Ou seja, o estado do objeto é dado pelo conjunto de informações contidas nele

- Representação

- * A representação de um objeto é a forma como ele é descrito ou exibido em uma determinada mídia/local
- * **toString** é o método que permite a representação de um objeto em formato String

Pacote

- Programas mais complexos ou com mais funcionalidades inevitavelmente possuem mais código-fonte do que programas menores ou mais simples
- Quanto maior for o código-fonte de um programa, mais difícil é para os programadores conseguirem entendê-lo. Observe:
 - Um programa simples no paradigma OO deve possuir dezenas ou centenas de classes e, cada uma dessas classes públicas, deve ser codificada em um arquivo separado de outras classes;
 - Assim: o programa deverá ter dezenas ou centenas de arquivos;
- Sem falar na grande quantidade de atributos, métodos e classes que precisam de nomes que não podem gerar conflito com outros elementos
- Permite o agrupamento de estruturas de código (classes e outros tipos possíveis na linguagem)

- Serve para facilitar a leitura e a busca por classes
- O pacote corresponde a um sub-diretório/pasta dentro do local do diretório/pasta de código do projeto
- Os pacotes podem ser organizados em árvore, ou seja, **lógica hierárquica**
- * O ponto marca a separação entre os níveis
- É recomendado utilizar o domínio de internet da organização de forma **invertida** no início do domínio

Exemplo: br.edu.idp; org.eclipse; br.jus.cnj;

- Deve-se utilizar um pacote para agrupar classes dentro de algum sentido de código ou de lógica negocial
- * Esse significado deve ser dado pelo programador
- O mais comum é agrupar por significado negocial ou por significado técnico
- * Negocial: faturamento, secretaria, manutenção, etc.
- * Técnico: gui, interface, persistência, io, etc.
- Projetos maiores podem demandar a combinação de ambos

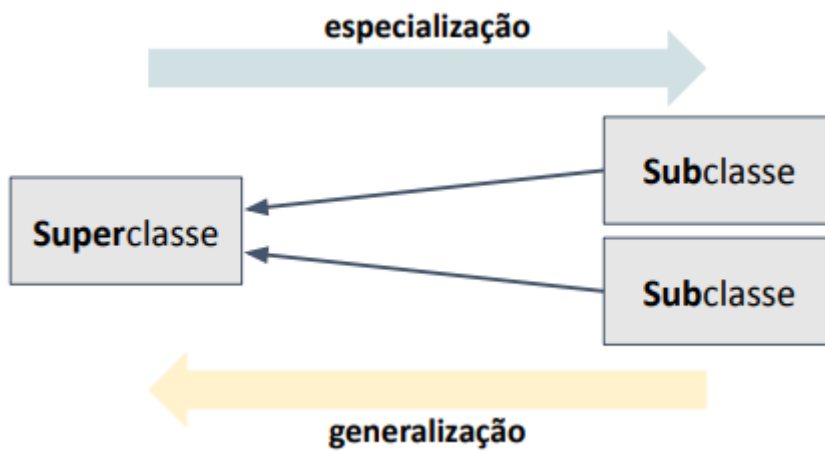
Exemplo: persistência do faturamento (faturamento.persistencia)

Pacote	Funcionalidade
java.awt	AWT, ou Abstract Window Toolkit. Contém todas as classes para a criação de aplicações com interfaces gráficas com o usuário (ou GUI, Graphical User Interface).
java.io	Fornece as classes para realizar operações de entrada (input) e saída (output) através do sistema de fluxos de dados (streams) e serialização de objetos.
java.lang	Fornece classes que são fundamentais para a concepção da linguagem de programação Java.
java.net	Fornece as classes para implementar aplicações de rede.
java.util	Para trabalhar com coleções, entrada de dados (Scanner) e componentes de data e hora.
javax.swing	O pacote Swing é um pacote de extensão que complementa o pacote AWT.

Herança de Classes

- É uma forma de reuso de código
- Uma classe **herda variáveis e métodos** definidos em outra classe
- * Herdar = ter uma “cópia” do que foi definido na outra classe
- A classe mãe é chamada de **superclasse** e a filha é a **subclasse**
- É uma relação de especialização, ou seja, a subclasse pode implementar novas variáveis e métodos além dos que foram herdados

- * Generalização: criação de classe mais genérica, ou seja, classe mãe
- * Especialização: criação de classe mais especialista, ou seja, classe filha



Funcionario.java

```

public class Funcionario {
    private String nome;
    private String cpf;
    private double salario;
    // getters e setters
    public boolean vender() {
        // lógica da venda
        return true;
    }
}
  
```

Gerente.java (com herança)

```

1 public class Gerente extends Funcionario {
2     private int senha;
3     private int numeroDeFuncionariosGerenciados;
4     // getters e setters
5     public void autorizarDesconto() {
6         // lógica da autorização de desconto
7     }
8 }
  
```

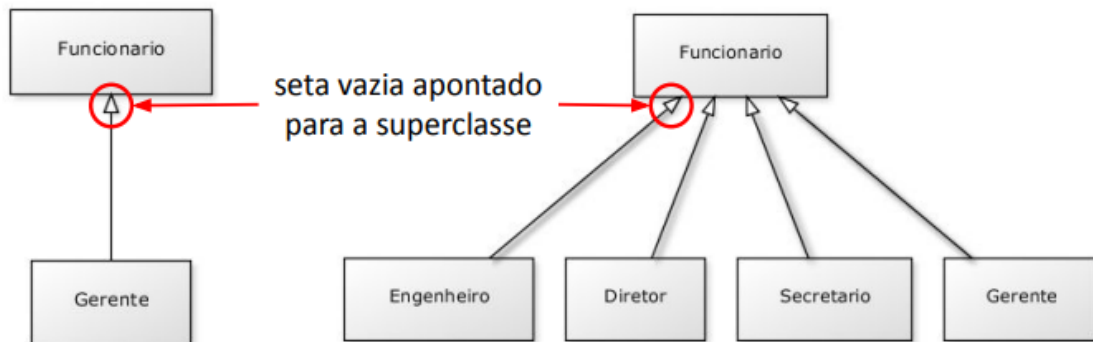
Gerente é um Funcionário
 Gerente é **subclasse** de Funcionário
 Funcionario é **superclasse** de Gerente
 Gerente é classe **filha** da classe Funcionário
 Funcionario é classe **mãe** da classe Gerente

class SUB_CLASSE extends SUPER_CLASSE { }

IMPORTANTE: Quando não há extensão definida explicitamente, a classe herda de Object

- Ou seja, toda classe herda direta ou indiretamente da classe Object
- java.lang.Object

Representação de Herança



Sobrescrita de Métodos

- Method Overriding
- Quando é necessário alterar o método da superclasse
- A anotação “Override” informa ao compilador que o método foi sobrescrito
- * **@Override**
- O método da superclasse pode ser acessado pela palavra reservada “super”
- * Atenção: “this” chama o método da própria subclasse, não o método da superclasse!

Funcionario.java

```
public class Funcionario {  
    private String nome;  
    private String cpf;  
    protected double salario;  
    // getters e setters  
    public double getBonificacao() {  
        return this.salario * 0.10;  
    }  
}
```

Gerente.java (com herança)

```
1 public class Gerente extends Funcionario {  
2     private int senha;  
3     private int numeroDeFuncionariosGerenciados;  
4     // getters e setters  
5     @Override  
6     public double getBonificacao() {  
7         return super.getBonificacao() + 1000;  
8     }  
9 }
```

10% de bonificação para funcionário e 1.000 reais para gerente

... mais os 10% de bonificação como funcionário

Funcionario.java

```
public class Funcionario {  
    private String nome;  
    private String cpf;  
    protected double salario;  
    // getters e setters  
    public double getBonificacao() {  
        return this.salario * 0.10;  
    }  
}
```

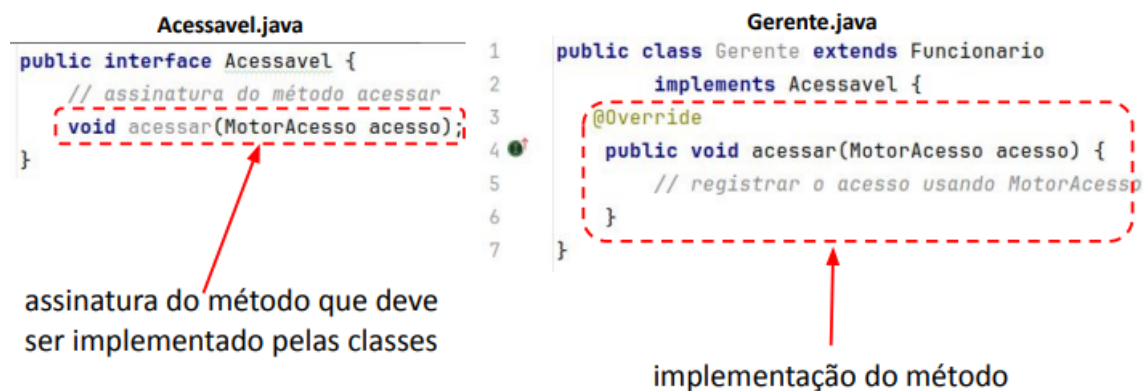
Gerente.java (com herança)

```
1 public class Gerente extends Funcionario {  
2     private int senha;  
3     private int numeroDeFuncionariosGerenciados;  
4     // getters e setters  
5     @Override  
6     public double getBonificacao() {  
7         return this.salario * 0.30;  
8     }  
9 }
```

10% de bonificação para funcionário e 30% para gerente

Interfaces

- É uma forma de reuso de código
- É um tipo especial provido por algumas linguagem para definir apenas contratos, dando a liberdade para que a implementação seja feita conforme as necessidades do seu contexto
- * Esse “contrato” deve ser respeitado pela classe que implementa a interface
- Declara apenas os métodos (e suas assinaturas) que devem existir
- * Ou seja, não é feita a implementação desses métodos
- Torna o código mais flexível e reutilizável do que herança
- Isso ocorre porque foca apenas nas operações essenciais que devem ser executadas



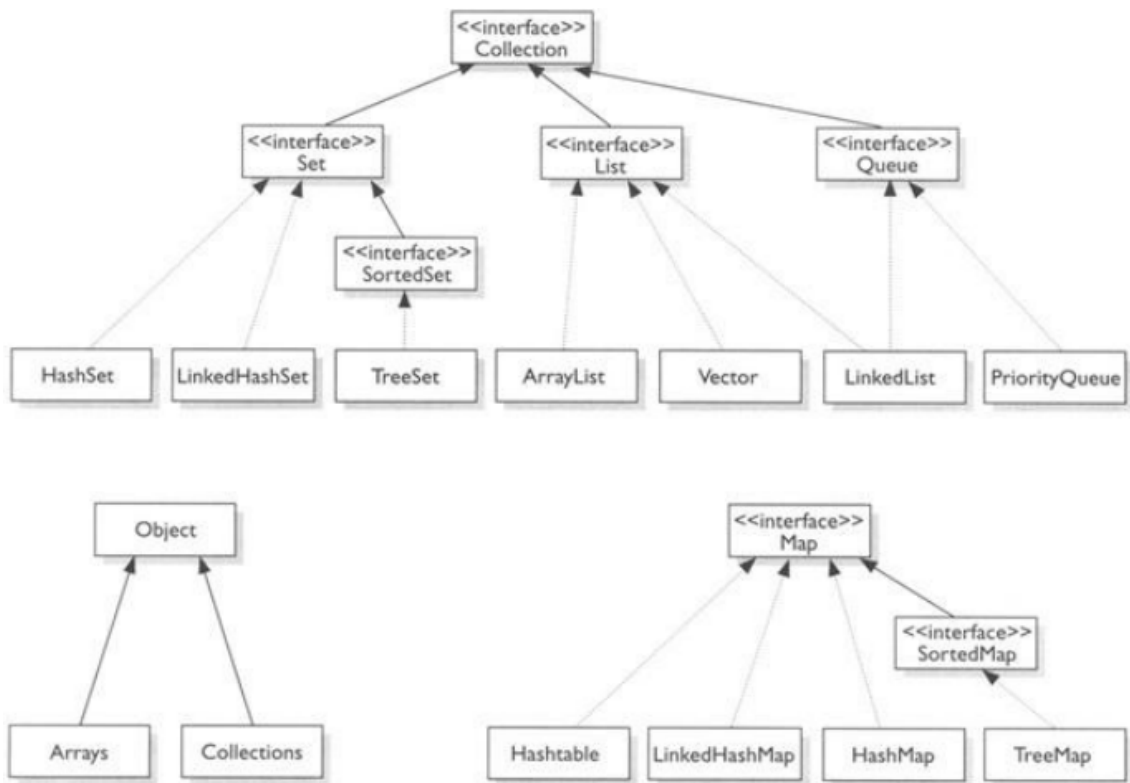
Coleção

- Estrutura de dados que permitem armazenar vários elementos em um só objeto
- * Em OO, uma coleção também é um objeto que contém referências a outros objetos

Java Collections Framework

- É um framework com diversas interfaces e classes para facilitar a utilização de coleções de objetos
- * Essas interfaces e classes já estão prontas para uso (não é necessário fazer qualquer tipo de alteração para utilizá-las)
- Este framework já está embutido no núcleo da JRE desde as primeiras versões da plataforma Java

- Suporta os principais (mais comuns) tipos de estruturas de dados utilizados na programação



Collection

- Interface “coleção”
- A interface Collection define as operações mais elementares que são necessárias para manipular qualquer tipo de coleção em Java que é provido pelo framework: adicionar, remover, limpar, verificar existência, tamanho, dentre outros
- Caso necessário pode-se implementar uma nova classe a partir da Collection para ter uma coleção mais genérica
- Todas as coleções suportadas contém elementos que devem ser objetos, ou seja, não podem ser tipos primitivos
- É super-interface para as duas interfaces mais famosas em Java: Set e List

boolean	add(E e) Ensures that this collection contains the specified element (optional operation).
boolean	addAll(Collection<? extends E> c) Adds all of the elements in the specified collection to this collection (optional operation).
void	clear() Removes all of the elements from this collection (optional operation).
boolean	contains(Object o) Returns true if this collection contains the specified element.
boolean	containsAll(Collection<?> c) Returns true if this collection contains all of the elements in the specified collection.
boolean	equals(Object o) Compares the specified object with this collection for equality.
int	hashCode() Returns the hash code value for this collection.
boolean	isEmpty() Returns true if this collection contains no elements.
Iterator<E>	iterator() Returns an iterator over the elements in this collection.
default Stream<E>	parallelStream() Returns a possibly parallel Stream with this collection as its source.
boolean	remove(Object o) Removes a single instance of the specified element from this collection, if it is present (optional operation).
boolean	removeAll(Collection<?> c) Removes all of this collection's elements that are also contained in the specified collection (optional operation).
default boolean	removeIf(Predicate<? super E> filter) Removes all of the elements of this collection that satisfy the given predicate.
boolean	retainAll(Collection<?> c) Retains only the elements in this collection that are contained in the specified collection (optional operation).
int	size() Returns the number of elements in this collection.
default Spliterator<E>	spliterator() Creates a Spliterator over the elements in this collection.

Iterable e Iterator

- **Iterable** é a interface “iterável”

- É a interface que permite a utilização da iteração for-each

* for-each é um modo de for mais fácil e prático de ser utilizado

* for-each permite percorrer todos os elementos da coleção

- O principal método previsto na iterável é o iterator () que fornece um objeto “iterador”

- Toda coleção que descende de collection é iterável

- **Iterator** é a interface “iterador”

- É a interface que permite a iteração em uma coleção de objetos

default void	forEachRemaining(Consumer<? super E> action) Performs the given action for each remaining element until all elements have been processed or the action throws an exception.
boolean	hasNext() Returns true if the iteration has more elements.
E	next() Returns the next element in the iteration.
default void	remove() Removes from the underlying collection the last element returned by this iterator (optional operation).

- Para usar, primeiro deve-se criar um iterador para entrar na coleção, em seguida, consumir os método next até que todos os elementos da coleção sejam acessados
- Atenção: o iterador não pode ser utilizado após qualquer tipo de alteração (inclusão, remoção, reordenação, etc.) na coleção porque ele se torna inválido
- * Caso um iterador inválido seja utilizado, ocorre o erro `ConcurrentModificationExceptions`

Lista

- A interface List especifica a implementação de listas em Java
- A lista (às vezes chamada de sequência) é uma coleção que:
 - * permite duplicação de elementos
 - * tem ordenação específica entre os elementos
- Como há garantia da ordem, pode-se percorrer a lista acessando os elementos na ordem pré-definida anteriormente

boolean	add(E e) Appends the specified element to the end of this list (optional operation).
void	add(int index, E element) Inserts the specified element at the specified position in this list (optional operation).
boolean	addAll(Collection<? extends E> c) Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator (optional operation).
boolean	addAll(int index, Collection<? extends E> c) Inserts all of the elements in the specified collection into this list at the specified position (optional operation).
void	clear() Removes all of the elements from this list (optional operation).
boolean	contains(Object o) Returns true if this list contains the specified element.
boolean	containsAll(Collection<?> c) Returns true if this list contains all of the elements of the specified collection.
boolean	equals(Object o) Compares the specified object with this list for equality.
E	get(int index) Returns the element at the specified position in this list.
int	hashCode() Returns the hash code value for this list.
int	indexOf(Object o) Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
boolean	isEmpty() Returns true if this list contains no elements.
Iterator<E>	iterator() Returns an iterator over the elements in this list in proper sequence.
int	lastIndexOf(Object o) Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
ListIterator<E>	listIterator() Returns a list iterator over the elements in this list (in proper sequence).
ListIterator<E>	listIterator(int index)

ArrayList

- É uma classe que implementa a interface List
- Atenção: ArrayList não é um Array
- * ArrayList utiliza um array encapsulado para gerar uma lista
- * ou seja, não tem como acessar o array interno dela e, portanto, não é possível utilizar o operador [] nem atributos como length
- ArrayList é mais rápida na pesquisa do que a LinkedList

LinkedList

- É uma classe que implementa a interface List e a Deque
- É uma implementação de lista duplamente encadeada
- Em relação à ArrayList, a LinkedList é mais rápida na inserção e remoção de itens nas pontas da lista

Vector

- É uma classe que implementa a interface List

- Funciona de forma parecida com o ArrayList, utilizando-se de um array encapsulado
- É thread-safe
- Pode apresentar um desempenho bem superior ao ArrayList

Conjunto

- A interface Set especifica a implementação de conjuntos em Java
- O conjunto é inspirado em conjuntos matemáticos. Portanto, é uma coleção que:
 - * não permite duplicação de elementos
 - * não armazena a ordem dos elementos
- Atenção: não é possível acessar os elementos de um conjunto por índice como o get(int) das listas
- Implementações da Set: HashSet e TreeSet

boolean	add(E e) Adds the specified element to this set if it is not already present (optional operation).
boolean	addAll(Collection<? extends E> c) Adds all of the elements in the specified collection to this set if they're not already present (optional operation).
void	clear() Removes all of the elements from this set (optional operation).
boolean	contains(Object o) Returns true if this set contains the specified element.
boolean	containsAll(Collection<?> c) Returns true if this set contains all of the elements of the specified collection.
boolean	equals(Object o) Compares the specified object with this set for equality.
int	hashCode() Returns the hash code value for this set.
boolean	isEmpty() Returns true if this set contains no elements.
Iterator<E>	iterator() Returns an iterator over the elements in this set.
boolean	remove(Object o) Removes the specified element from this set if it is present (optional operation).
boolean	removeAll(Collection<?> c) Removes from this set all of its elements that are contained in the specified collection (optional operation).
boolean	retainAll(Collection<?> c) Retains only the elements in this set that are contained in the specified collection (optional operation).
int	size() Returns the number of elements in this set (its cardinality).
default Spliterator<E>	spliterator() Creates a Spliterator over the elements in this set.
Object[]	toArray() Returns an array containing all of the elements in this set.
<T> T[]	toArray(T[] a) Returns an array containing all of the elements in this set; the runtime type of the returned array is that of the s

- O conjunto pode parecer desvantajoso porque não permite o acesso por índice, porém ele se torna muito vantajoso porque várias de suas implementações possuem performance muito mais rápida em relação às listas quando, por exemplo, usamos o método contains
- Outro ponto é que pode-se percorrer um conjunto usando iteradores (lembre-se que não há ordem garantida nessa iteração)
- Implementação HashSet:
 - * Armazena os elementos utilizando uma hash-table (tabela de hash)

- Implementação TreeSet

* Armazena os elementos utilizando árvore

Fila

- A interface Queue especifica a implementação de filas em Java

- A fila é uma coleção que:

* permite duplicação de elementos

* mantém a ordem na qual os elementos foram inseridos

- Introduz operações “alternativas” para inserir, remover e acessar elementos em uma fila

* não permite a utilização das operações originais definidas pela Collection (add, remove e elemento)

boolean	add(E e) Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions, returning true upon success and throwing an <code>IllegalStateException</code> if no space is currently available.
E	element() Retrieves, but does not remove, the head of this queue.
boolean	offer(E e) Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions.
E	peek() Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.
E	poll() Retrieves and removes the head of this queue, or returns null if this queue is empty.
E	remove() Retrieves and removes the head of this queue.

- Implementações da Queue: PriorityQueue, LinkedBlockingQueue, ArrayBlockingQueue, PriorityBlockingQueue, DelayQueue, SynchronousQueue e LinkedTransferQueue

Pilha

- A classe Stack implementa a manipulação de pilhas em Java

- A pilha é uma coleção que:

* permite duplicação de elementos

* mantém a ordem na qual os elementos foram inseridos

- Introduz operações “alternativas” para, inserir, remover e acessar elementos em uma pilha
- Estende a classe Vector

boolean	empty() Tests if this stack is empty.
E	peek() Looks at the object at the top of this stack without removing it from the stack.
E	pop() Removes the object at the top of this stack and returns that object as the value of this function.
E	push(E item) Pushes an item onto the top of this stack.
int	search(Object o) Returns the 1-based position where an object is on this stack.

- Atenção: para evitar erros não utilize os métodos herdados de Vector (add, remove, etc.) para manipular pilhas porque eles podem corromper a pilha:
 - * push deve ser usado para empilhar (adicionar no topo da pilha)
 - * pop deve ser usado para desempilhar um elemento (retirar o elemento do topo da pilha e entregá-lo como resultado)
 - * peek deve ser usado para consultar sem consumir o elemento
- Somente push e pop devem ser usados para acrescentar e retirar elementos da pilha

Mapa

- A interface Map especifica a implementação de mapas (dicionários) em Java
- O mapa é uma coleção que:
 - * não permite duplicação de elementos
 - * não armazena a ordem dos elementos
- Um mapa armazena itens formados pelos pares (chave, valor)
- Estrutura utilizada para acessar rapidamente um objeto dado alguma informação sobre ele
- A forma chave-valor permite indexar objetos de acordo com o critério necessário para o caso
- Implementações da Map: AbstractMap, Attributes, ConcurrentHashMap, HashMap, Hashtable, LinkedHashMap, Properties, TreeMap, WeakHashMap

void	<code>clear()</code>	Removes all of the mappings from this map (optional operation).
default V	<code>compute(K key, BiFunction<? super K, ? super V, ? extends V> remappingFunction)</code>	Attempts to compute a mapping for the specified key and its current mapped value (or null if there is no current mapping).
default V	<code>computeIfAbsent(K key, Function<? super K, ? extends V> mappingFunction)</code>	If the specified key is not already associated with a value (or is mapped to null), attempts to compute its value using the given mapping function and enters it into this map unless null.
default V	<code>computeIfPresent(K key, BiFunction<? super K, ? super V, ? extends V> remappingFunction)</code>	If the value for the specified key is present and non-null, attempts to compute a new mapping given the key and its current mapped value.
boolean	<code>containsKey(Object key)</code>	Returns true if this map contains a mapping for the specified key.
boolean	<code>containsValue(Object value)</code>	Returns true if this map maps one or more keys to the specified value.
static <K,V> Map<K,V>	<code>copyOf(Map<? extends K, ? extends V> map)</code>	Returns an unmodifiable Map containing the entries of the given Map.
static <K,V> Map.Entry<K,V>	<code>entry(K k, V v)</code>	Returns an unmodifiable Map.Entry containing the given key and value.
Set<Map.Entry<K, V>>	<code>entrySet()</code>	Returns a Set view of the mappings contained in this map.
boolean	<code>equals(Object o)</code>	Compares the specified object with this map for equality.
default void	<code>forEach(BiConsumer<? super K, ? super V> action)</code>	Performs the given action for each entry in this map until all entries have been processed or the action throws an exception.
V	<code>get(Object key)</code>	Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
default V	<code>getOrDefault(Object key, V defaultValue)</code>	Returns the value to which the specified key is mapped, or defaultValue if this map contains no mapping for the key.

Classe Collections

- A Collections é uma utilitária que fornece diversos métodos para manipular coleções
- A grande vantagem é que os algoritmos implementados nestes métodos são bem otimizados e possuem alto desempenho
- Os métodos são métodos de classe, ou seja, são fornecidos de forma estática (sem a necessidade de instanciar um objeto Collections)
- Os algoritmos são polimórficos, ou seja, eles operam de forma diferente e adequada a cada tipo de coleção fornecida a eles (interface única para várias implementações)
- Vários dos algoritmos da Collections utilizam o método equals do objeto para verificar a igualdade com outro objeto e, por padrão, a igualdade é se os dois objetos forem sobre a mesma referência (como ocorre na comparação feita pelo operador ==)
- * Portanto o método “equals” deve ser sobrescrito se for necessário utilizar outro critério de igualdade
- Atenção: não confunda a classe Collections com a interface Collection (uma no singular e outra no plural)

Comparable x Comparator

- Estão disponíveis duas formas de definir o algoritmo de ordenação: dentro da própria classe alvo ou a criação de uma nova classe específica para fazer a comparação
- Ambas são realizadas por meio da implementação de duas interfaces:

* Comparable

* Comparator

- **Comparable** é a interface para definir como uma classe deve se comparar

- Define o método compareTo que serve para definir o formato padrão de comparação na classe que desejamos comparar

- **Comparator** é a interface para definir uma classe para fazer a comparação

- Define o método compare que serve para definir a lógica de comparação na entre dois objetos do mesmo tipo

Classe Arrays

- A Arrays é uma classe utilitária que fornece diversos métodos para manipular arrays, incluindo a ligação com outros tipos de coleções (como listas, por exemplo)

- É muito importante porque disponibiliza métodos para diversas operações básicas que precisam ser feitas sobre os arrays

- Os métodos são métodos de classe, ou seja, são fornecidos de forma estática (sem a necessidade de instanciar um objeto Arrays)

* Vários métodos são sobrecarregados para atender diferentes tipos de coleções

- Atenção: não confunda a classe Arrays com o tipo de dado array (aquele com a variável multidimensional, sendo que cada valor é acessado pelo índice da sua localização)

static <T extends Comparable<? super T>> parallelSort(T[] a)	Sorts the specified array of objects into ascending order, according to the natural ordering of its elements.
void	
static <T> void parallelSort(T[] a, Comparator<? super T> cmp)	Sorts the specified array of objects according to the order induced by the specified comparator.
static <T extends Comparable<? super T>> parallelSort(T[] a, int fromIndex, int toIndex)	Sorts the specified range of the specified array of objects into ascending order, according to the natural ordering of its elements.
void	
static <T> void parallelSort(T[] a, int fromIndex, int toIndex, Comparator<? super T> cmp)	Sorts the specified range of the specified array of objects according to the order induced by the specified comparator.
static void setAll(double[] array, IntToDoubleFunction generator)	Set all elements of the specified array, using the provided generator function to compute each element.
static void setAll(int[] array, IntUnaryOperator generator)	Set all elements of the specified array, using the provided generator function to compute each element.
static void setAll(long[] array, IntToLongFunction generator)	Set all elements of the specified array, using the provided generator function to compute each element.
static <T> void setAll(T[] array, IntFunction<? extends T> generator)	Set all elements of the specified array, using the provided generator function to compute each element.
static void sort(byte[] a)	Sorts the specified array into ascending numerical order.
static void sort(byte[] a, int fromIndex, int toIndex)	Sorts the specified range of the array into ascending order.
static void sort(char[] a)	Sorts the specified array into ascending numerical order.
static void sort(char[] a, int fromIndex, int toIndex)	Sorts the specified range of the array into ascending order.
static void sort(double[] a)	Sorts the specified array into ascending numerical order.
static void sort(double[] a, int fromIndex, int toIndex)	Sorts the specified range of the array into ascending order.