



INSTITUTO SUPERIOR POLITECNICO DE TECNOLOGIAS E CIENCIAS

1º Semestre

2025/26



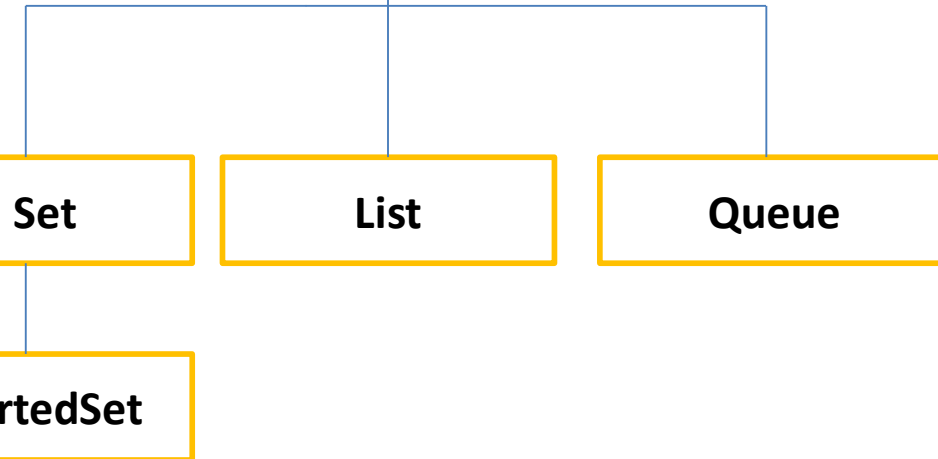
Coleções de Objectos

- Um conjunto de objectos
 - Sistemas, em geral, precisam manipular vários **objetos** de uma mesma **classe**
 - Java possui uma **API** (Application Programming Interface) que fornece interface para colecções
-
- ❖ **Oferecem formas diferentes de coleccionar dados com base em factores como:**
 - ☐ Eficiência no acesso, procura ou na inserção.
 - ☐ Forma de organização dos dados.
 - ☐ Forma de acesso, procura e inserção.

❖ Java fornece vários tipos de colecções

Coleções de elementos individuais

Java.util.**Collection**



Coleções de pares de elementos

Java.util.**Map**

SortedMap

- ❑ Pares de key/value
 - ❑ Vetor associativo
 - ❑ As chaves não podem se repetir

❖ **Conjunto (Set e SortedSet)**

- Sequencia arbitrária (não mantém uma ordem nem uma contagem de elementos)
- Elementos não repetem (cada elemento ou esta no conjunto ou não)

❖ **Lista (List)**

- Sequência definida
- Elementos indexados

❖ **Fila(Queue)**

- Fila de elementos
- Modelo **FIFO** (First in, First out)

❖ **Mapa (Map e SortedMap)**

- uma associação entre chaves e valores
- As chaves não se podem repetir

❖ A **API** organiza as suas coleções do seguinte modo:

- Uma hierquia de interfaces
 - Especificações dos vários tipos
- Uma hiequia de implementação de classes

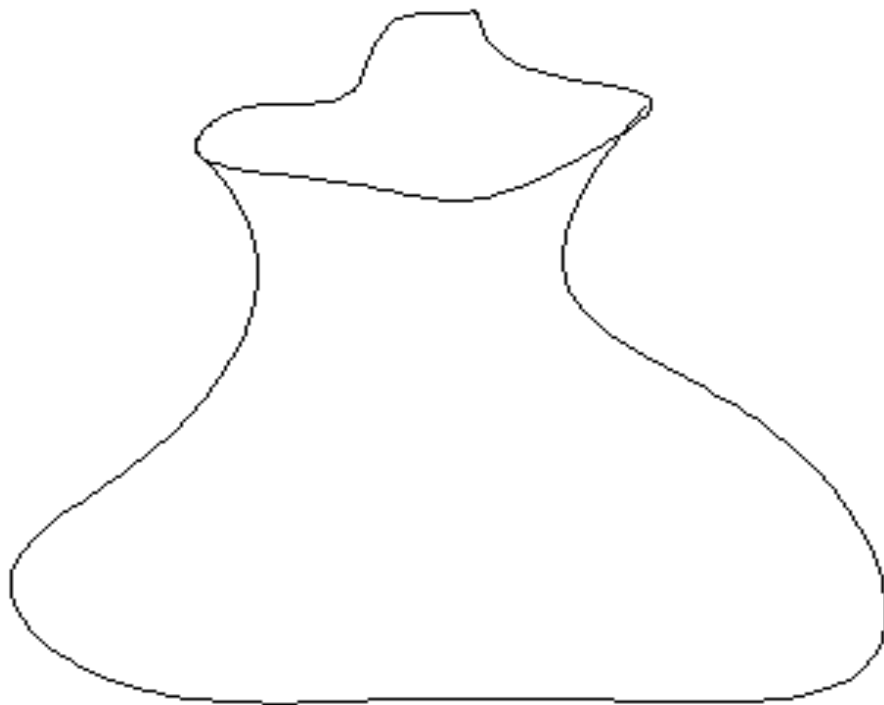
Implementations						
		Hash Table	Resizable Array	Balanced Tree	Linked List	Hash Table + Linked List
Interfaces	Set	HashSet		TreeSet		LinkedHashSet
	List		ArrayList		LinkedList	
	Map	HashMap		TreeMap		LinkedHashMap

❖ COLLECTION

❖ Características

- ❖ Métodos comuns para todos os tipos de colecções
- ❖ Possui métodos para adicionar, remover e procurar elementos(e mais alguns outros)
 - **Size()** – retorna a quantidade de objectos na colecção
 - **isEmpty()** – retorna verdadeiro(**true**) se a colecção está vazia e falso(**false**) caso contrário.
 - **add(Object o)** – adiciona um elemento a colecção
 - **remove(Object o)** – remove um objecto da colecção

❖ COLLECTION



Carros

❖ COLLECTION

carro1



carro5



carro2



carro3

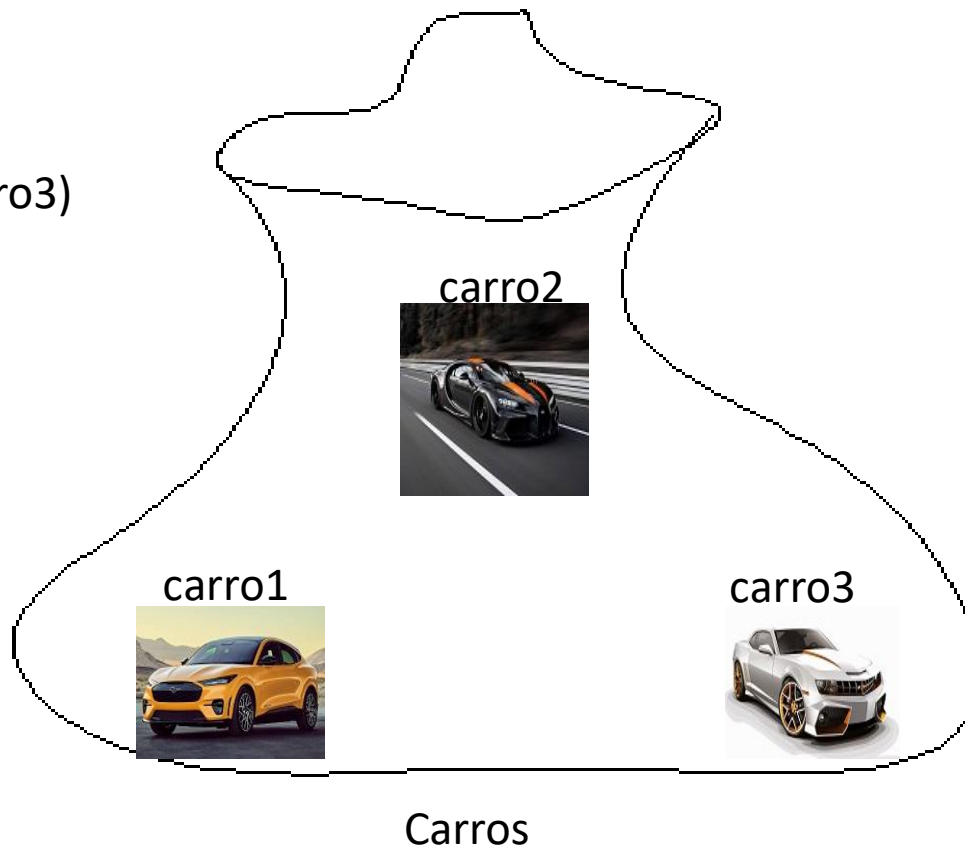


carro4



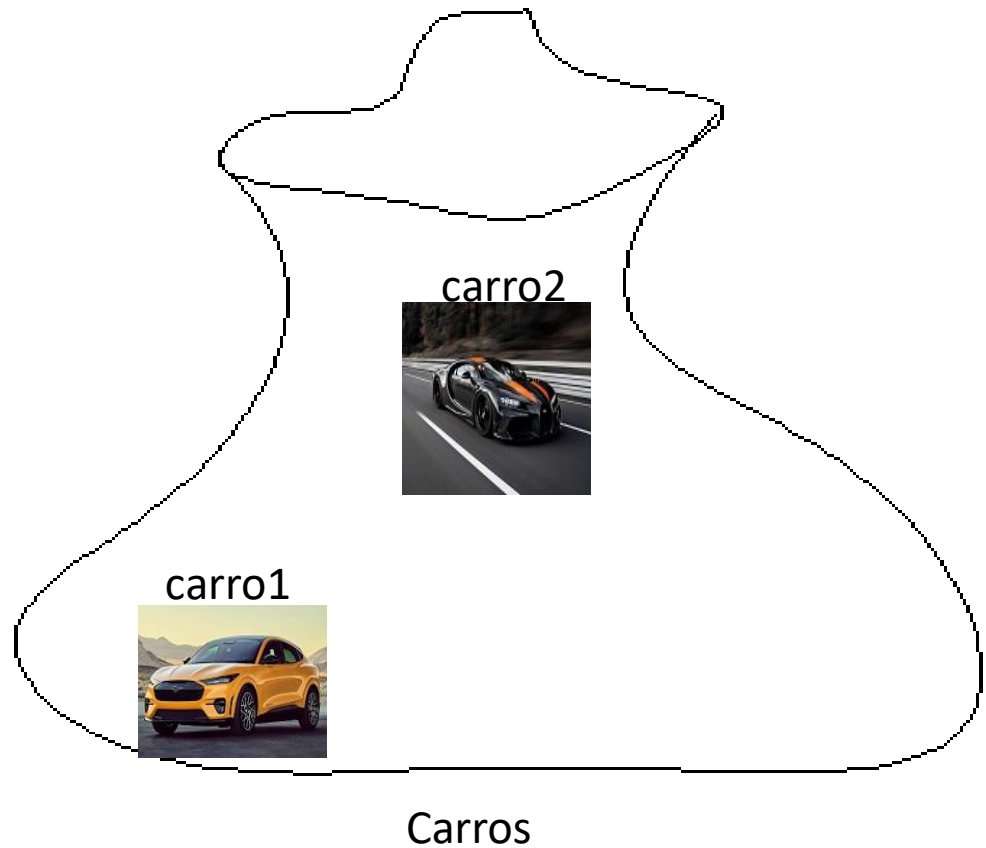
❖ COLLECTION

- `carros.add(carro1)`
- `carros.add(carro2)`
- `carros.add(carro3)`
- `carros.remove(carro3)`



❖ COLLECTION

- `carros.remove(carro3)`
- `carros.isEmpty()`
 - **FALSE**



❖ LIST (ARRAYLIST)

- Uma collection ordenada (também chamada de sequencia)
- Pode conter elementos duplicados
- Herda todos os métodos do Collection(size, isEmpty, add, remove, etc)

carro1



0

carro2



1

carro3



2

carro4



3

carro5



4

❖ LIST (ARRAYLIST)

❖ Inclui métodos de acesso posicional

- **get**(int indice) – retorna o elemento na posição indicada pelo parâmetro índice
- **remove**(int indice) – remove o elemento na posição indicada pelo índice.
- **add**(Object o) – adiciona o objecto depois do último índice

❖ Os elementos começam no índice 0

❖ LIST (ARRAYLIST)

- A lista começa vazia no índice 0

carro1



carro2



carro3



carro4



carro5



❖ LIST (ARRAYLIST)

- `carros.add(carro3)`

carro1



carro2



carro4



carro5



carro3



❖ LIST (ARRAYLIST)

- `carros.add(carro3)`
- `carros.add(carro1)`

carro2



carro4



carro5



carro3



carro1



0

1

❖ LIST (ARRAYLIST)

- carros.**add**(carro3)
- carros.**add**(carro1)
- carros.**add**(carro5)
- carros.**add**(carro4)
- carros.**add**(carro2)

carro3



0

carro1



1

carro5



2

carro4



3

carro2



4

❖ LIST (ARRAYLIST)

- `carros.remove(carro3)`
- `carros.size()`
 - 4

carro1



0

carro5



1

carro4



2

carro2



3

❖ LIST (ARRAYLIST)

- Cada elemento tem o seu sucessor(excepto o último) e o seu antecessor (excepto o primeiro)
- As operações mais importantes em listas são:
 - Adicionar um objecto em qualquer lugar da lista
 - Remover um objecto de qualquer lugar da lista
 - Obter o elemento de qualquer lugar da lista
 - Percorrer os elementos da lista
 - Verificar se um elemento está na lista
 - Descobrir o índice de um elemento na lista
 - Obter o número de elementos da colecção

- Menos eficientes que os vectores
- Não aceitam tipos primitivos (apenas empacotados)
- Não permitir restringir o tipo específico dos objectos guardados (tudo é `java.lang.Object`)
 - Aceitam **qualquer** objeto.
 - Exemplo: uma coleção de Galinhas aceita objectos do tipo Raposa
 - Requer cast na saída para poder usar o objecto

```
List galinheiro = new ArrayList();  
  
galinheiro.add(new Galinha("Chocagilda"));  
galinheiro.add(new Galinha("Cocotalva"));  
galinheiro.add(new Raposa("Fox"));  
  
for(int i = 0; i < galinheiro.size(); i++){  
    Galinha g = (Galinha) galinheiro.get(i);  
    g.ciscar();  
}
```

Vai gerar uma **ClassCastException** quando Object retornado apontar para uma **Raposa** e não para uma galinha.

➤ ArrayList

- Escolha natural quando for necessário utilizar um vetor redimensionável
 - Mais eficiente para leitura
- Implementado internamente com vetores.
- **Ideal para o acesso aleatório**

➤ LinkedList

- Muito mais eficiente que ArrayList para remoção e inserção no meio da lista
- Ideal para implementar pilhas, filas.
- **Ideal para acesso sequencial**

```
List lista = new ArrayList();  
lista.add(new Coisa("um"));  
lista.add(new Coisa("dois"));  
lista.add(new Coisa("tres"));  
(...)
```

```
Coisa c3 = lista.get(2); // == índice de vetor  
ListIterator it = lista.listIterator();  
Coisa c = it.last();  
Coisa d = it.previous();  
Coisa[] coisas = (Coisa[])list.toArray(new Coisa[lista.size()]);
```

- Objectos **Map** são semelhantes a vetores mas, em vez de índices numéricos, usam **chaves**, que são objectos
 - Chaves são **unívocas**(um Set)
 - Valores podem ser duplicados (um Collection)

- Principais subclasses:
 - **HashMap**
 - Escolha natural quando for necessário um vetor associativo
 - Acesso rápido: usa `Object.hashCode()` para organizar e localizar objectos
 - **TreeMap**
 - Mapa ordenado
 - Contém métodos para manipula elementos ordenados.

- **Métodos**
 - void **put(Object key, Object value)**: acrescenta um objeto
 - Object **get (Object key)**: recupera um objeto
 - Set **keySet()**: retorna um Set de chaves
 - Collection **values()**: retorna um Collection de valores
 - Set **entrySet()**: retorna um set de pares chave-valor contendo objetos representados pela classe interna **Map.Entry**

```
Map map = new HashMap();

map.put("um", new Coisa("um"));
map.put("dois", new Coisa("dois"));

Set chaves = map.keySet();
Collection valores = map.values();

Coisa c = (Coisa) map.get("dois");
Set pares = map.entrySet();

Iterator entries = pares.iterator();
Map.Entry one = entries.next();
String chaveOne = (String) one.getKey();
Coisa valueOne = (Coisa) one.getValue();
```



- Pretende-se um programa que leia os dados de um conjunto de estudantes (nome e um conjunto de notas), calcule a sua média e ordene os estudantes por ordem decrescente das médias.
 - ❑ Classe Estudante para representar um aluno.
 - ❑ Classe para representar uma turma de estudantes (Turma).
 - Vai guardar um conjunto de estudantes num ArrayList.
 - Tem como comportamentos:
 - a sua inicialização,
 - a adição de um estudante ao ArrayList,
 - a escrita dos dados de todos os estudantes e
 - a sua ordenação por ordem decrescente de médias.
 - ❑ Classe para fazer a gestão do sistema (GereTurma).