# Universidade Federal de Ouro Preto

## CSI032 - Programação de Computadores II

# Collections: Map

Professor: Dr. Rafael Frederico Alexandre
Contato: rfalexandre@decea.ufop.br
Colaboradores: Eduardo Matias Rodrigues
Contato: eduardo.matias@aluno.ufop.edu.br


# Collections framework

# Collections

# Map
## Introdução

1. Trataremos agora da interface **Map**;
    1. Um mapa não extende da Interface **Collection**, logo, um Map tem seus próprios métodos para inserir, remover e pesquisar um dado;

# Collections

# Map
## mapas

1. Um **map** associação *chaves* a *valores*;
   1. Chaves devem ser únicas, mas o valores não precisam ser;
   2. ou seja, muitas chaves podem mapear para um mesmo valor;

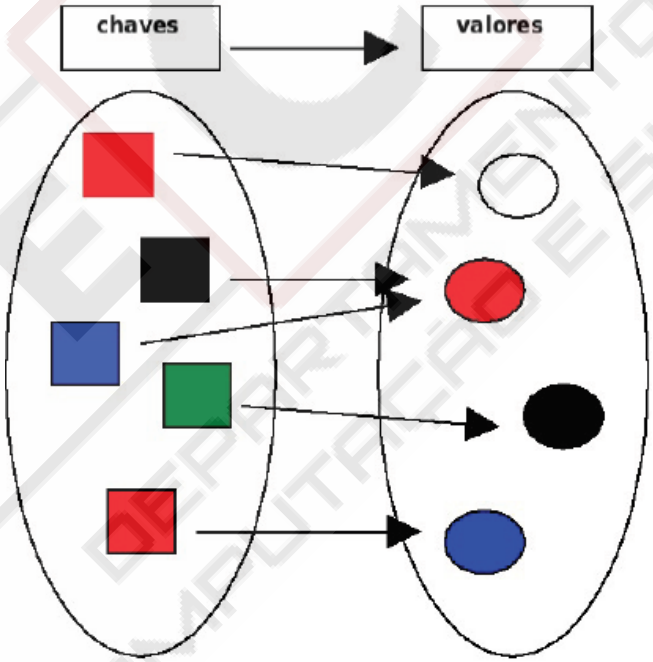



Figura: Fonte: [Caelum, 2017]

# Map
## Diagrama de classes

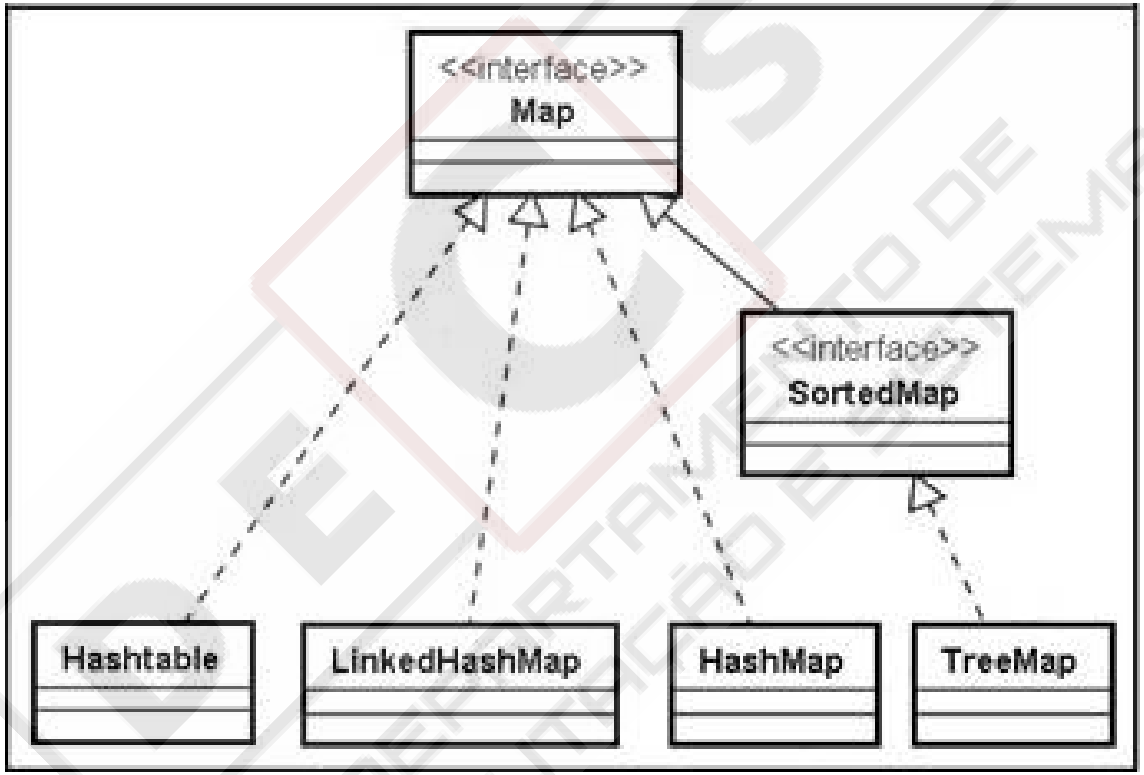



Figura: Fonte: [DEVMEDIA, 2010]

# Map
## Classes

1. **HashTable** e **HashMap** armazenam elementos em tabelas hash e **TreeMaps** armazenam elementos em árvores;
   1. A classe **TreeMap** implementa a interface **SortedMap** que mantêm as chaves de seus elementos ordenadas (ordem natural);
   2. Ao contrário da **HashMap**, uma **HashTable** é sincronizada (múltiplas threads podem modificar um HashTable de forma concorrente).
2. **LinkedHashMap** mantém uma lista duplamente ligada através de seus itens. A ordem de iteração é a ordem em que as chaves são inseridas no mapa.

# Map
## HashMap

1. Iremos trabalhar apenas com a HashMap;
2. Vide documentação para mais detalhes sobre HashTable, TreeMap e LinkedHashMap;

- HashTable:

`https://docs.oracle.com/javase/8/docs/api/java/util/Hashtable.html`

- TreeMap:

`https://docs.oracle.com/javase/8/docs/api/java/util/TreeMap.html`

- LinkedHashMap:

`https://docs.oracle.com/javase/8/docs/api/java/util/LinkedHashMap.html`

# Collections

# Map

## Operações: [Caelum, 2017]

1. Possíveis operações para se fazer com um Map:
    1. Mapear uma chave para um valor;
    2. O que está mapeado na chave $X$?
    3. Remapeie uma certa chave;
    4. Obter o conjunto de chaves;
    5. Obter o conjunto de valores;
    6. Desmapear a chave $X$.

# Map
## sintaxe

1. Sintaxe para criar um mapa:

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

1. Onde Integer é a classe a qual as chaves pertencem e String é a classe a qual os valores pertencem.

# Map
## Exemplo

1. Faremos um simples exemplo que mapeia um número para seu equivalente por extenso, ou seja:
   1. A chave será um inteiro (devemos informar sua wrapper class, a classe **Integer**);
   2. e o valor será o nome do número (tipo String);
2. O método **put(objeto, objeto)** recebe a chave e o valor de uma nova associação e para saber o que está associado a um determinado objeto-chave, passa-se esse objeto no método **get(objeto)**

# Map
## put e get

```
public static void main(String[] args) {

    Map<Integer, String> mapa = new HashMap<>();

    mapa.put(1, "Um");
    mapa.put(2, "Dois");
    mapa.put(3, "Três");

    // Qual valor está associado a chave 2?
    System.out.println(mapa.get(2));

}
```

# Map
## Saída do programa

```
run:
Dois
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figura: Saída do programa

# Map
## Remapeando chave-valor

```
public static void main(String[] args) {

    Map<Integer, String> mapa = new HashMap<>();

    mapa.put(1, "Um");
    mapa.put(2, "Dois");
    mapa.put(3, "Três");
    mapa.put(2, "D-O-I-S"); // sobrescreve o valor

    // Qual valor está associado a chave 2?
    System.out.println(mapa.get(2));

}

run:
D-O-I-S
BUILD SUCCESSFUL (total time: 0 seconds)
```

# Map
## Percorrendo um mapa

1. Para percorrer um mapa primeiro devemos obter suas chaves com o método *keySet()*;
   1. Retorna um Set com todas as chaves.
2. Podemos percorrer o Set contendo as chaves utilizando um **for aprimorado**.

# Map
## Percorrendo um mapa

```
public static void main(String[] args) {

    Map<Integer, String> mapa = new HashMap<>();

    mapa.put(1, "Um");
    mapa.put(2, "Dois");
    mapa.put(3, "Três");

    Set<Integer> chaves = mapa.keySet();

    for (Integer chave : chaves) {
        System.out.println("Chave: " + chave +
                           ", valor: " + mapa.get(chave));
    }
}
```

# Map

## Percorrendo um mapa: saída do programa

```
run:
Chave: 1, valor: Um
Chave: 2, valor: Dois
Chave: 3, valor: Três
BUILD SUCCESSFUL (total time: 0 seconds)
```

# Map
## Percorrendo um mapa

1. Se precisarmos apenas saber os valores de um mapa, temos o método *values()* que retorna uma **Collection** contendo todos os valores.

# Map

## Percorrendo um mapa: saída do programa

```
public static void main(String[] args) {

    Map<Integer, String> mapa = new HashMap<>();

    mapa.put(1, "Um");
    mapa.put(2, "Dois");
    mapa.put(3, "Três");

    Collection<String> valores = mapa.values();

    System.out.print("Valores: ");
    for (String valor : valores) {
        System.out.print(valor + "  ");
    }
    System.out.println("");
}
```

```
run:
Valores: Um  Dois  Três
BUILD SUCCESSFUL (total time: 0 seconds)
```

# Map
## Demaspeando uma chave

```
public static void main(String[] args) {

    Map<Integer, String> mapa = new HashMap<>();

    mapa.put(1, "Um");
    mapa.put(2, "Dois");
    mapa.put(3, "Três");

    System.out.println(mapa);
    mapa.remove(2);
    System.out.println(mapa);
}
```

```
run:
{1=Um, 2=Dois, 3=Três}
{1=Um, 3=Três}
BUILD SUCCESSFUL (total time: 0 seconds)
```

# Map
## chave-valor

1. Podemos criar mapas de chaves e valores de qualquer tipo:
    1. Suponha uma concessionária de carros, quando o funcionário da loja deseja visualizar os dados de um carro específico ele deve digitar sua placa e o sistema deve retornar as informações do automóvel que contém aquela placa.

# Map
## Classe Carro

```
public class Carro {

    private int ano;
    private String modelo;
    private String placa;
    private double preco;

    public Carro(int ano, String modelo, String placa, double preco) {
        this.ano = ano;
        this.modelo = modelo;
        this.placa = placa;
        this.preco = preco;
    }

    @Override
    public String toString() {
        return "Modelo: " + this.modelo + "\n" +
               "Ano: " + this.ano + "\n" +
               "Placa: " + this.placa + "\n" +
               "Preço: " + this.preco;
    }
}
```

# Map
## Classe de teste

```
public static void main(String[] args) {

    Map<String, Carro> mapa = new HashMap<>();

    Carro focus = new Carro(2015, "Ford Focus", "JAX-1010", 30000);
    Carro fiatUno = new Carro(2000, "Fiat Uno", "ABC-9999", 8000);
    Carro palio = new Carro(2008, "Palio", "FAC-5412", 10000);
    Carro fusca = new Carro(1990, "Fusca", "HLG-5241", 4000);

    mapa.put("JAX-1010", focus);
    mapa.put("ABC-9999", fiatUno);
    mapa.put("FAC-5412", palio);
    mapa.put("HLG-5241", fusca);

    Carro c = mapa.get("FAC-5412");
    System.out.println(c.toString());
}
```

# Map
## Saída do programa

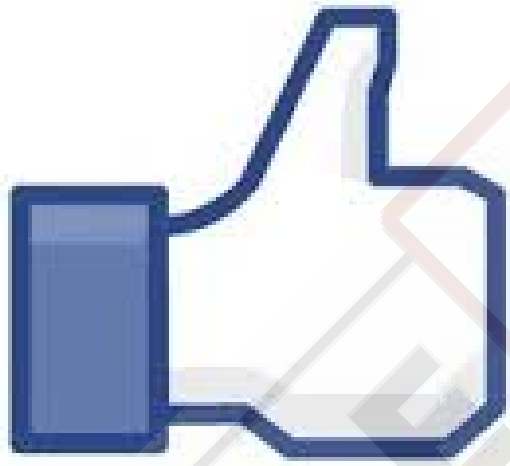
```
run:
Modelo: Palio
Ano: 2008
Placa: FAC-5412
Preço: 10000.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

# Map
## equals e hashCode

1. Toda classe que for utilizada como chave em um mapa deve sobrescrever os métodos *equals* e *hashCode*, já que em um mapa as chaves não podem ser duplicadas.

# Considerações
## Map

1. Porque não utilizar sempre um ArrayList ou array para armazenar nossos dados e quando necessários localizar um elemento, percorremos os elementos de forma linear?
   1. Para coleções grandes essa abordagem é ineficiente;
   2. Opte por vetores ou ArrayList apenas quando a ordem dos elementos for importante;
2. Sempre quando precisar utilizar uma estrutura para armazenar seus dados, analise qual a melhor opção para cada necessidade que aparecer!



**MUITO OBRIGADO!!!**

# Referências Bibliográficas I

Caelum (2017).
Java e orientação a objeto.
https://www.caelum.com.br/download/caelum-java-objetos-fj11.pdf.

DEVMEDIA (2010).
Java collections: Como utilizar collections.
https://www.devmedia.com.br/java-collections-como-utilizar-collections/18450.