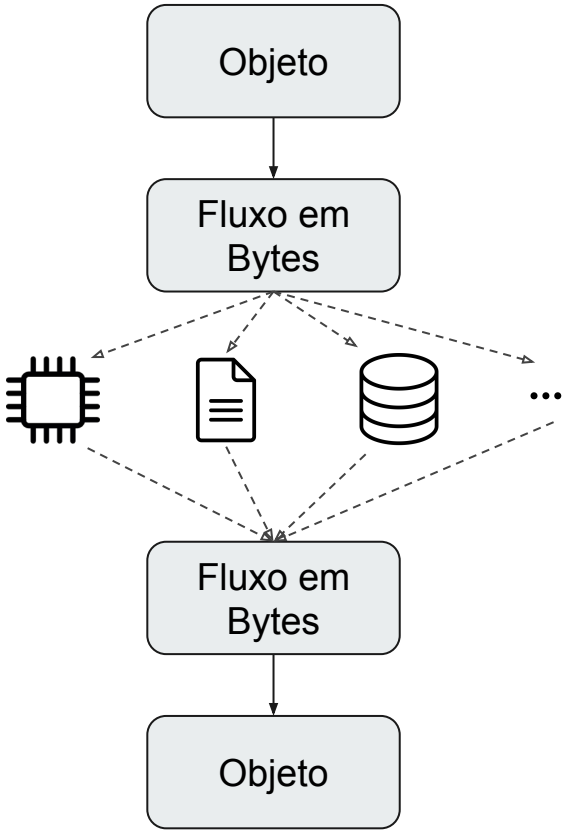
Aula 24
Serialização e Genericidade

Iniciativa Conjunta:
apdc Digital Business Community
Conselho Coordenador dos Institutos Superiores Politécnicos
Instituto do Emprego e Formação Profissional
Com o apoio de:
República Portuguesa
Portugal Digital

Serialização

- Mecanismo que permite transformar objetos, primitivos e dados em bytes que podem ser guardados em ficheiros, enviados pela internet, etc.
- Objetos depois de serializados podem ser desserializados para serem transformados de volta no objeto
- No Java é possível serializar/deserializar mantendo o tipo e dados de forma coerente, mesmo entre vários dispositivos



Interface *Serializable*

```
public class Map implements Serializable {
    private List<ImageTile> tiles = new ArrayList<>();
    private Hero hero;
    private int level;

    public Map(Hero hero, int level) {
        this.hero = hero;
        readMapFile(level);
    }

    private List<ImageTile> readMapFile(int level) {
        //etc..
```

Implementar a interface *Serializable*

Todos os atributos também têm de ser *Serializable*

Primitivos do Java e classes da Java Collections já são *Serializable* por defeito

Serialização - Exemplo

```
Hero hero = new Hero(new Position(3, 5));
Map map = new Map(hero, 0);
try {
   FileOutputStream fileOut = new FileOutputStream("save.dat");
   ObjectOutputStream out = new ObjectOutputStream(fileOut);
   out.writeObject(map);
   out.close();
   fo.close();
} catch (IOException e) {
   System.out.println("Erro a salvar o mapa no ficheiro!");
}
```

Usamos o ObjectOutputStream para transformar o objeto serializável num fluxo de dados

FileOutputStream para salvar esse fluxo de dados num ficheiro

Deserialização - Exemplo

Para ler o fluxo de dados do ficheiro usamos o FileInputStream e ObjectInputStream

```
try {
   FileInputStream fileIn = new FileInputStream("save.dat");
   ObjectInputStream in = new ObjectInputStream(fileIn);
   Map loadedMap = (Map) in.readObject();
   in.close();
   fileIn.close();
   System.out.println("Vida do hero: " + loadedMap.getHero().getHealth());
} catch (IOException e) {
   System.out.println("Erro a ler o ficheiro com o save do mapa!");
} catch (ClassNotFoundException e) {
   System.out.println("Não foi possível converter o objeto salvo no mapa!");
}
```

É necessário fazer *cast* do objeto lido para o Java saber como interpretar os dados

Exercício 1

Criar um menu na consola: pedir ao utilizador para introduzir "**1**" para ***Salvar Dados*** e "**2**" para ***Carregar Dados.***

Na opção ***Salvar Dados*** pedimos ao user para introduzir o seu nome, um campo de texto livre e um título. Por fim pedimos ao user o nome do ficheiro a guardar e salvamos esses dados num ficheiro.

Na opção ***Carregar Dados*** pedimos ao user para introduzir o nome do ficheiro a carregar e mostramos na consola a nota guardada.

Genericidade

- Adicionada no Java 5 como forma de providenciar verificações de tipos de dados a nível do compilador
- Permite trabalhar com tipos de dados diferentes dentro das mesmas classes
- *Java Collections Framework* usa genericidade para criar as ***Lists***, ***Sets***, ***Maps***, etc
- Interface Comparable também usa genericidade para criar o método de interface para comparar um objeto com o outro do mesmo tipo

Genericidade

```
List<Integer> list = new ArrayList<>();
list.add(10);
list.add("10");

...

List list = new ArrayList();
list.add(10);
list.add("10");

Integer num = (Integer) list.get(0);
Integer num2 = (Integer) list.get(1);
```

Instanciar o *list* como uma ArrayList **de** Integers

Compilador queixa-se que é uma lista de inteiros e não podemos adicionar uma String

Instanciação sem genérico

Compilador permite adicionar qualquer objeto

Quando se quer ir buscar o valor à lista é necessário fazer cast, e caso o objeto não seja do tipo certo, lança uma excepção

Genericidade - *Comparable*

```
public interface Comparable<T> {
    public int compareTo(T o);
}
```

Utiliza-se o T (ou qualquer outro caracter) dentro dos símbolos <> para especificar que é do tipo genérico

Interface *Comparable* contém o método compareTo genérico que utiliza o T

```
public class Estudante implements Comparable<Estudante> {
    @Override
    public int compareTo(Estudante outro) {
        return outro.numero - this.numero;
    }
    ...
}
```

Quando implementado, passamos o tipo de objeto que queremos usar como genérico para ficar automaticamente especificado no método *compareTo*

Genericidade - Limitação de Tipos

- Ao criar uma classe ou método genérico, é possível limitar os tipos de objetos que são aceites:

```
public class Test<T extends ImageTile> {
     private T tile;
     ...
}
```

Exercício 2

Criar uma classe genérica ***Pair*** que aceita quaisquer dois tipos de objetos:

```
public class Pair<K, V> { ... }
```

A classe deve conter um par Key-Value como parâmetros e deve ter um construtor, getters e setters apropriados.