Programação Avançada

Introdução à linguagem Java

Linguagem Java

- 1ª versão disponibilizada em 1995 pela *Sun Microsystems*
  - Estudo iniciado em 1991

- Linguagem totalmente orientada aos objetos
- *Cross-platform*
- A sintaxe segue o estilo C/C++

- Objetivos (https://www.oracle.com/java/technologies/introduction-to-java.html)
  - Simples, orientada ao objeto e familiar
  - Robusta e segura
  - Não dependente da arquitetura e portável
  - Alto desempenho
  - Interpretada, multitarefa e dinâmica

Criação de um programa Java

- Tal como noutras linguagens a criação de uma aplicação envolve a escrita do código
  - Editor de texto "tradicional"
    - *notepad, notepad++, ...*
    - *vi, emacs, nano, ...*
    - *TextEdit, TextWrangler, BBEdit, ...*
    - *Visual Studio Code*
    - ...
  - Ambiente integrado de desenvolvimento
    - *NetBeans*
    - *Eclipse*
    - ***IntelliJ*** (existe licença gratuita para estudantes)
    - ***Visual Studio Code*** **(com *Microsoft Extension Pack for Java*)**
    - ...

Compilação de um programa Java

- A compilação, que inclui a verificação sintática e geração de código compilado, é realizada usando o ***Java Development Kit*** **(JDK)**
  - Alguns ambientes de desenvolvimento possuem versões integradas
  - Inclui ferramentas e bibliotecas de classes
  - Inclui uma ferramenta para compilação: `javac`
  - É gerado código intermédio, designado *Java bytecode*
    - Código independente da arquitetura

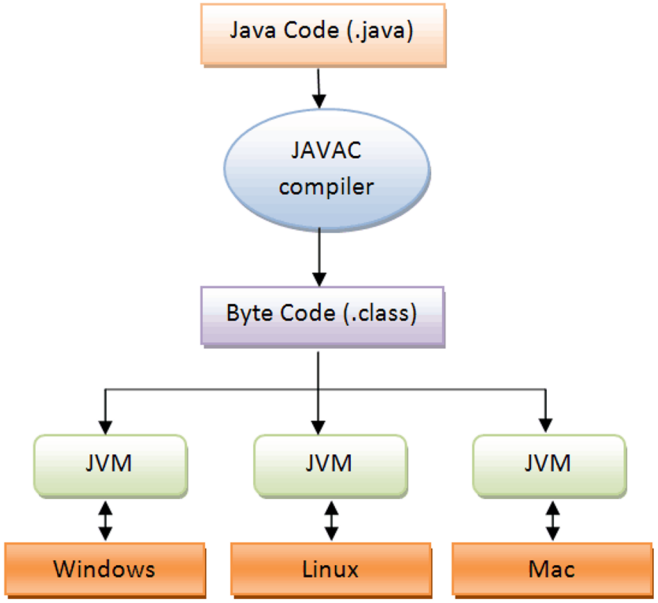
Instalação do JDK

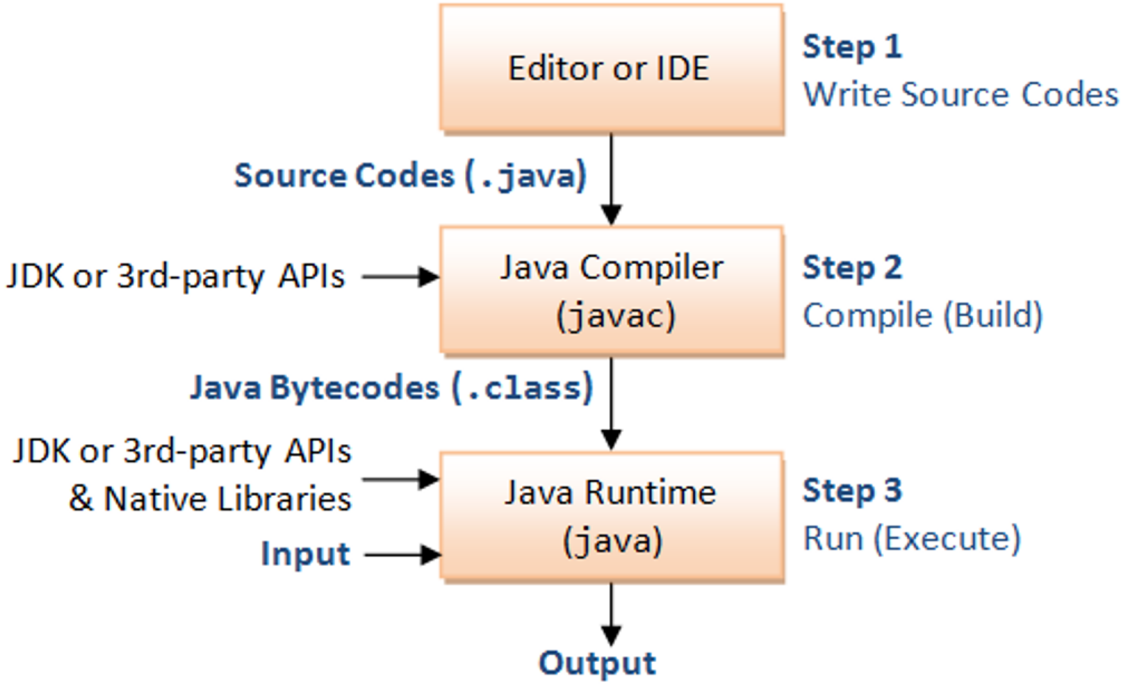
- *OpenJDK* vs *OracleJDK*
  - *OracleJDK*
    - Versão comercial, exigindo licenciamento
    - Suporte dado pela *Oracle*
    - Supostamente, mais estável e com melhor desempenho
  - *OpenJDK*
    - *Open source*, podemos contribuir no seu desenvolvimento
    - Embora o suporte seja essencialmente feito pelas comunidades ativas no seu desenvolvimento, existem distribuições específicas com suporte próprio
  - Versão do JDK a instalar: *Java SE*
    - A última versão disponível é a 17
      - https://jdk.java.net/17/
      - https://www.oracle.com/java/technologies/downloads/

Java Virtual Machine

- Como o resultado da compilação não é código nativo, é necessário possuir um interpretador de *Java bytecode*
- O intérprete de *Java bytecode* designa-se ***Java Virtual Machine* (JVM)**
  - Para que os programas Java possam ser executados numa determinada plataforma é necessário que exista uma JVM para cada
  - A JVM também oferece um ambiente seguro e isolado em que os programas são executados sem afectarem ou serem afetados por outros programas
  - Nota: não é necessário instalar o JDK nas máquinas onde se pretende apenas executar os programas, bastando instalar uma versão simplificada que suporta apenas a execução, designada por *Java Runtime Environment* (JRE)
    - Inclui comando "**`java`**" para executar os programas previamente compiladas em *Java bytecode*

Criação e execução





Exemplo

```
//      Exemplo de um programa em Java
//      Nome do ficheiro: Exemplo1.java

public class Exemplo1 {
        public static void main(String args[]) {
                System.out.println("Java@DEIS-ISEC");
        }
}
```

- Todo o código é encapsulado em classes ou similares
- Uma classe (no exemplo: `class Exemplo1`) é definida num ficheiro com o mesmo nome dessa classe e extensão `.java`
  - Podem existir várias classes no mesmo ficheiro, mas apenas uma pode ser pública
- A primeira função a ser executada num programa em *Java* é `public static void main(String args[])`

Exemplo (compilação e execução)

- Depois de escrito o código deve ser gravado com o nome da classe pública presente no ficheiro de código e com extensão `.java`
  - Neste caso `Exemplo1.java`

- A compilação é realizada fazendo

  `javac Exemplo1.java`
  - Em caso de sucesso é criado o ficheiro `Exemplo1.class`
  - Caso existam erros estes são devidamente indicados através do número de linha no ficheiro em causa

- A execução é realizada fazendo

  `java Exemplo1`

```
PA@deis$ cat > Exemplo1.java
public class Exemplo1 {
        public static void main(String args[]) {
                System.out.println("Java@DEIS-ISEC");
        }
}
PA@deis$ javac Exemplo1.java
PA@deis$ java Exemplo1
Java@DEIS-ISEC
PA@deis$
```

Packages

- As classes são agrupadas em *packages*
  - Organização de código
    - Constituição de *namespaces* para nomear as classes
  - Facilitar reutilização
    - Muitas funcionalidades serão obtidas através da utilização de bibliotecas de classes objetos, definidas através de *packages* adequados, designado por *Java API*
  - Permite evitar conflito de nomes entre diversas classes
    - Quando existe conflito de nomes entre classes de diferentes *packages*, dever-se-á indicar o seu nome completo, ou seja, incluído o nome do package: `<package>.<classe>`

Packages - criação

- Definição do *package*
  - O *package* possui um nome definido através da instrução "`package <nome>;`" indicada, normalmente, como primeira linha efetiva num ficheiro de código
    - O nome base é habitualmente constituído por duas informações distintas, em letra minúscula
      - Nome do domínio da empresa por ordem inversa (`deis.isec.pt => pt.isec.deis`)
      - Nome do projeto concreto
    - Exemplo: `package pt.isec.a200212345.proj_aula1;`
  - Os ficheiros que incluem as classes pertencentes a um *package* devem ser colocados numa hierarquia de diretórios correspondente ao nome do *package*

```
Proj_Aula1
  pt
    isec
      a200212345
        proj_aula1
          Exemplo1.java
```

Packages - utilização

- Para utilizar as classes ou outros elementos definidos noutros *packages* deve-se usar a instrução `import`
  - Podem existir várias instruções de importação (usual!)
  - Podem ser incluídos todos os elementos de um package: `import pt.isec.pa.utils.*;`
  - Pode ser incluído apenas o elemento pretendido: `import pt.isec.pa.utils.FileUtils;`

Packages - exemplo

```
PA@deis$ cat Exemplo1.java
package pt.isec.pa.exemplo1;

public class Exemplo1 {
        public static void main(String args[]) {
                System.out.println("Java@DEIS-ISEC");
        }
}
PA@deis$ javac Exemplo1.java
PA@deis$ java Exemplo1
Error: Could not find or load main class Exemplo1
Caused by: java.lang.NoClassDefFoundError: pt/isec/pa/exemplo1/Exemplo1 (wrong n
ame: Exemplo1)
PA@deis$ mkdir -p pt/isec/pa/exemplo1
PA@deis$ mv Exemplo1.class pt/isec/pa/exemplo1
PA@deis$ java pt.isec.pa.exemplo1.Exemplo1
Java@DEIS-ISEC
PA@deis$
```

Packages - exemplo

```
PA@deis$ cat Exemplo1.java
package pt.isec.pa.exemplo1;

public class Exemplo1 {
        public static void main(String args[]) {
                System.out.println("Java@DEIS-ISEC");
        }
}
PA@deis$ javac -d . Exemplo1.java
PA@deis$ java pt.isec.pa.exemplo1.Exemplo1
Java@DEIS-ISEC
PA@deis$
```

Packages - acesso

- Para além das vantagens de organização e reutilização que os packages oferecem, também permitem controlar o acesso a particularidades de implementação dos elementos que definem
- O controlo de acesso é realizado através das palavras-chave `public`, `protected`, `private` ou pela sua não especificação, tendo em conta a seguinte tabela

Etiqueta	classe	*package*	subclasse	outros
`public`	Sim	Sim	Sim	Sim
`protected`	Sim	Sim	Sim	Não
(sem etiq.)	Sim	Sim	Não	Não
`private`	Sim	Não	Não	Não

Ficheiros jar

- Para facilitar as tarefas de *deployment* de programas *Java*, as classes que constituem esse programa deverão ser incluídas num ficheiro `jar`
  - O ficheiro `jar` é na realidade um ficheiro `zip`
  - Permite manter de forma simples a hierarquia de diretórios que representam os vários *packages* que poderão constituir um programa
  - É incluído um ficheiro de manifesto que permite especificar atributos especiais

- Criar ficheiro jar

```
jar cf exemplo1.jar pt/*
```

- Executar jar

```
java -cp exemplo1.jar pt.isec.pa.exemplo1.Exemplo1
```

Ficheiros jar executáveis

- A quando da criação de um ficheiro `jar` pode-se especificar uma classe que inclui uma função `main`, a qual irá ser executada quando o ficheiro `jar` for executado com: `java -jar <ficheiro.jar>`

```
PA@deis$ javac -d . Exemplo1.java
PA@deis$ jar cfe exemplo1.jar pt.isec.pa.exemplo1.Exemplo1 pt/*
PA@deis$ java -jar exemplo1.jar
Java@DEIS-ISEC
PA@deis$ mkdir temp
PA@deis$ cp exemplo1.jar temp/exemplo1.zip
PA@deis$ cd temp
PA@deis$ unzip exemplo1.zip
Archive:  exemplo1.zip
   creating: META-INF/
  inflating: META-INF/MANIFEST.MF
   creating: pt/isec/
   creating: pt/isec/pa/
   creating: pt/isec/pa/exemplo1/
  inflating: pt/isec/pa/exemplo1/Exemplo1.class
PA@deis$ cat META-INF/MANIFEST.MF
Manifest-Version: 1.0
Created-By: 17.0.2 (Oracle Corporation)
Main-Class: pt.isec.pa.exemplo1.Exemplo1

PA@deis$
```

Ficheiro de manifesto

- O ficheiro de manifesto, criado automaticamente no exemplo anterior, pode ser criado e configurado de forma manual

```
PA@deis$ cat > MANIFEST.TXT
Main-Class: pt.isec.pa.exemplo1.Exemplo1
PA@deis$ jar -cfm exemplo1.jar MANIFEST.TXT pt/*
PA@deis$ java -jar exemplo1.jar
Java@DEIS-ISEC
PA@deis$ mkdir temp
PA@deis$ cp exemplo1.jar temp/exemplo1.zip
PA@deis$ cd temp
PA@deis$ unzip exemplo1.zip
Archive:  exemplo1.zip
   creating: META-INF/
  inflating: META-INF/MANIFEST.MF
   creating: pt/isec/
   creating: pt/isec/pa/
   creating: pt/isec/pa/exemplo1/
  inflating: pt/isec/pa/exemplo1/Exemplo1.class
PA@deis$ cat META-INF/MANIFEST.MF
Manifest-Version: 1.0
Main-Class: pt.isec.pa.exemplo1.Exemplo1
Created-By: 17.0.2 (Oracle Corporation)

PA@deis$
```