Linguagem de Programação II

Ferramental

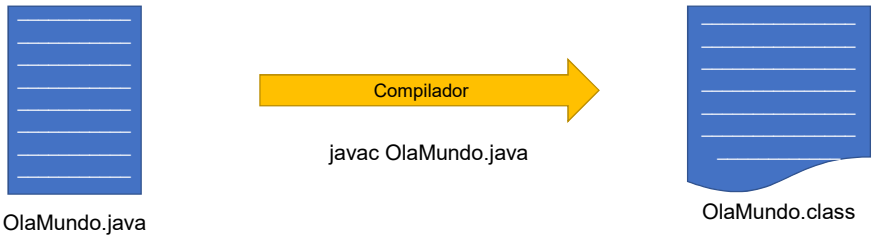
Universidade do Estado do Rio de Janeiro-UERJ
Instituto de Matemática e Estatística-IME
Ciência da Computação
Professor: Alexandre Sztajnberg

Utilitário em linha de comando (JDK)

- Para compilar os arquivos *.java* pela linha de comando, o comando *javac* é utilizado para invocar o compilador java.
- Para executar programas pela linha de comando, o comando *java* é utilizado para incovar a JVM.
- Um comando interessante é o *javadoc*, que gera documentação, a partir de códigos-fonte em java, em formato HTML. É constituído, basicamente, por declarações e comentários inseridos no programa.

O comando *javac*

- É o comando utilizado para executar o compilador.
- Recomendação
  - O `<nome-da-classe>` programada em Java deve começar com letra maiúscula
  - O nome do arquivo fonte deve ser `<nome-da-calsse>.java`
  - O nome do arquivo onde o ByteCode é persistido será `<nome-da-classe>.class`



O comando *javac*

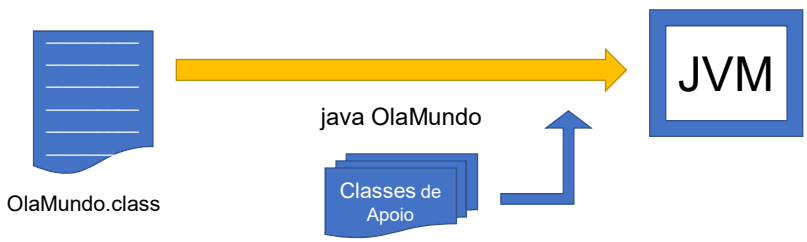
- O comando pode ser chamado da seguinte forma:
  - `javac [options][sourcefiles]`
  - *[options]*
    - campo de parâmetros, não-obrigatório, que permite modificar a forma como o código-fonte será compilado. Lista completa de comandos.
  - *[sourcefiles]*
    - campo obrigatório com o nome do arquivo .java a ser compilado
      - Pode ser uma lista de arquivos .java a ser compilado na mesma passada
      - Obrigatório quando existem chamadas de “um para outro”
      - Pode ser usado wildcards como `*.java`
        - Neste caso, todos os arquivos .java do diretório corrente serão compilados

O comando *javac*

- Alguns parâmetros que podem ser usados no campo *[options]* são:
  - *-verbose*
    - Mostra tudo que o compilador está fazendo. Inclui informações sobre cada classe carregada e cada código-fonte compilado
  - *-nowarn*
    - Desabilita mensagen do tipo *warning*. Faz o mesmo que *-Xlint:none*
  - *--help* **ou** *-help*
    - Mostra uma sinopse sobre os comandos disponíveis em *[options]*
  - *-g*
    - Gera toda informação de debugg
  - *-d {diretório}*
    - Especifica em qual diretório os arquivos .class serão colocados
  - *--class-path {caminho}* **ou** *-classpath {caminho}* **ou** *-cp {caminho}*
    - Especifica onde encontrar os arquivos .java
  - *@filename*
    - Especifica um arquivo que contém todas *[options]* e *[sourcefiles]* que serão usados

O comando *java*

- É o comando utilizado executar um aplicativo *Java.*
- Inicia a JVM
  - carrega as classes de suporte e a classe especifica na chamada ao comando
  - Em seguida chama o método `main()` desta classe.
    - Deve ser declarado como `public` e `static`
    - não retorna nenhum valor (void)
    - deve aceitar uma *array* de *Strings* como argumento.
    - `public static void main(String[] args).`



O comando *java*

- O comando pode ser chamado da seguinte forma:
  - `java [options] <main-class> [args...]`
  - *[options]* tem a mesma função que no comando *javac.* Lista completa de parâmetros.
  - *<main-class>* é o nome da **classe** que contém o método `main()`
    - Que por acaso é o nome do arquivo .classe que contém o seu ByteCode
    - `java HelloWorld.class`, está errado!
  - *[args...]* são os argumentos de linha de comando a serem recebidos pelo array de String do método main. A lista de argumentos é dependente do programa sendo executado
  - Obs: o comando *`javaw`* é similar ao *`java`,* com a exceção que *javaw* não abre uma janela de prompt de comando. Porém, se houver algum falha ao executar o programa, ela exibe uma caixa de diálogo com informações sobre o erro.

O comando *java*

- Alguns parametros que podem ser usados no campo *[options]* são:
  - *-verbose:class*
    - Exibe informações sobre cada classe carregada
  - *-verbose:gc*
    - Exibe informações sobre cada evento do *Garbage Collector*
  - *--class-path {caminho}* **ou** *-classpath {caminho}* **ou** *-cp {caminho}*
    - Especifica onde encontrar os arquivos .class
  - *--show-version* **ou** *-showversion*
    - Exibe informações sobre a versão da JVM e continua com a aplicação. A opção *-version* também as exibe, porém instrui a JVM a parar a execução
  - *-Xmn {tamanho}*
    - Define o tamanho inicial (e também o máximo) de memória que a JVM poderá utilizar

Classes X *Structs*

- O que é uma classe?
  - Um elemento de código de linguagens orientadas a objeto
  - Modelo ou "receita", que contém atributos (variáveis) e métodos (funções) comuns a todos os objetos instanciados a partir desta classe
  - "Uma classe está para o tipo, assim como a instância está para uma variável-daquele-tipo"
- Como em uma *Struct* de *C* , as classes de *Java* também definem um "novo tipo" com seus próprios campos (mais comumente chamados de atributos) mas, além disso, permitem:
  - Criar funções (métodos) que operam nos seus atributos
  - Controlar visibilidade ou permissão de acesso aos métodos e atributos
    - Públicos (acesso livre)
    - protegidos ou privados (acesso controlado)
      - Acessíveis através de métodos da própria classe, sendo inacessíveis por código externo.
      - Em *C* isso não seria fácil obter já que não existe o conceito de "métodos de *Struct*".

Classes X *Structs*

Classe em *Java*

```
public class Pessoa {
    private int idade;
    private String nome;
    [...]
    public void getNome(){
        return nome;
    }
}
```

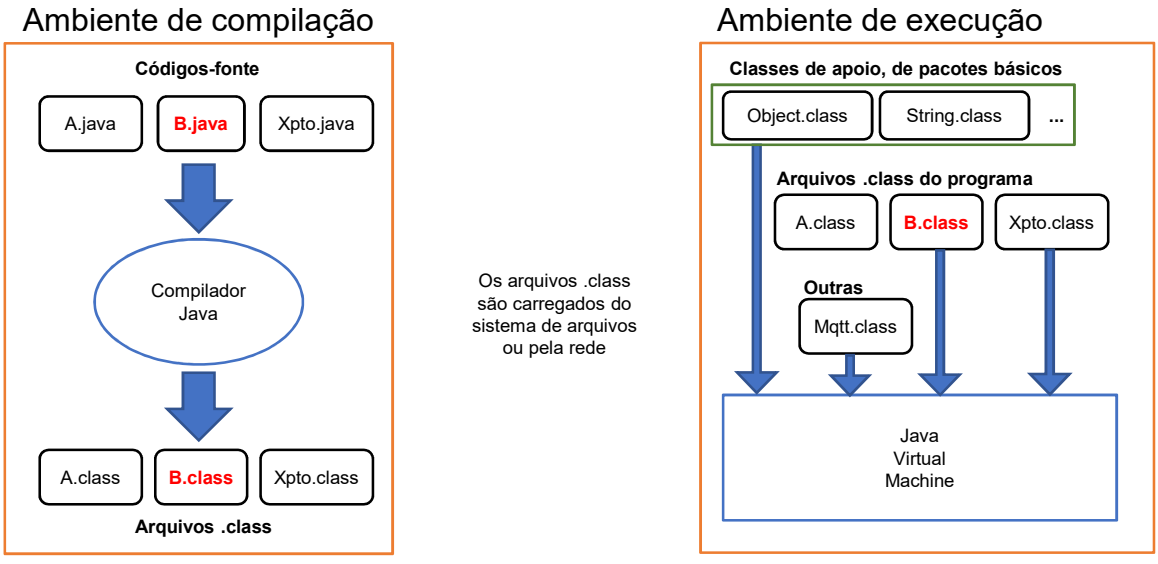
Struct* em *C

```
#include<stdio.h>
struct pessoa{
    char nome[50];
    int idade;
};
```

Um programa em *java*

- Um programa em *java* consiste de uma coleção de classes
  - Não existe um executável,
  - Uma classe principal com entry-point (nesse caso, o método `main()`).
- Ao começar a execução de um programa usando o comando *java*,
  - Os módulos do JRE é são iniciados
  - Uma instância da JVM é criada
  - O ClassLoader é acionado
    - Classes de suporte para o funcionamento de qualquer programa são carregadas
    - A classe passada como argumento no comando java é carregada
    - O método main desta classe é executado
    - Outras classes podem ser carregadas durante a execução

Um programa em *java*

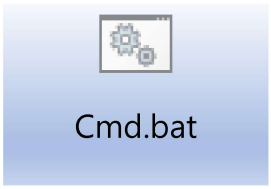


ClassLoader

- É uma classe cuja finalidade é carregar os *bytecodes* de classes na memória. Assim, podendo ser usada pelo programa ou aplicação em tempo de execução de uma forma mais dinâmica.
- *"Se tudo é carregado por um ClassLoader e o próprio é uma classe, quem carrega o primeiro ClassLoader?".* O "*Bootstrap ClassLoader*" , escrito em linguagem nativa, é o primeiro a ser carregado na JVM e fica responsável por carregar todo o resto.
- Graças ao *ClassLoader* não é necessário saber, em tempo de compilação, todas as classes que farão parte da execução do programa em *Java*. Enquanto é executado, o programa determina quais classes extras serão necessárias e o *ClassLoader* as carrega para a memória. Isso se chama "Carregamento dinâmico de classes"

Primeiro programa em *java*: Olá Mundo!

- Agora, vamos executar um programa que exibe na tela do prompt de comando "Olá mundo".



Primeiro programa em *java*: Olá Mundo!

- O que temos no programa?
  - `public class OlaMundo // declaração de uma classe pública`
  - `public static void main(String[] args)// declaração do métdo main`
  - `// Mostra uma frase no console (Comentário de uma linha)`
  - `System.out.println("Ola, Mundo!") // chamada de método`
    - Método *println*
    - Do objeto *out,* da classe PrintStream, e representa a saída padrão (“stdout”)
    - Da classe *System*, uma das classes básicas do JDK, que representa o Sistemas Operacional
  - `/* Comentario`
    `de mais de uma linha */`

Primeiro programa em *java*: Olá Mundo!

- Chamada (ou invocação, ou ativação) de métodos:
  - `objeto.NomeDoMétodo(parâmetros-de-entrada)`
  - `System.out.println(“Ola, Turma!”)`
- Possíveis erros:
  - Erros de sintaxe (são detectados pelo compilador):
    - `System.ouch.println(“...”);`
    - `System.out.println(“Ola);`
  - Erros lógicos (detectados pelo próprio desenvolvedor):
    - `System.out.println(“Oka”);`

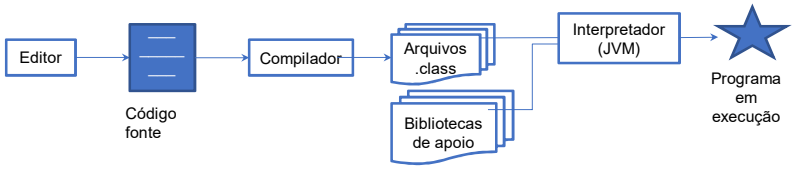
Erros detectados pelo compilador

```
OlaMundo.java:4: error: cannot find symbol
                  System.ouch.println("Ola, Mundo!");
                         ^
  symbol:   variable ouch
  location: class System
1 error
```

```
OlaMundo.java:4: error: unclosed string literal
                    System.out.println("Ola, Mundo!);
                                       ^
OlaMundo.java:4: error: ';' expected
                    System.out.println("Ola, Mundo!);
                                                    ^
OlaMundo.java:8: error: reached end of file while parsing
}
 ^
3 errors
```

Compilando e executando o exemplo

- Escrever o programa em um editor de textos
  - Salvá-lo com o NomeDaClasse, e extensão .java
- Abrir o interpretador de comandos
  - `command` ou `bash/terminal` ou `ssh`
- Tornar o diretório do arquivo .java o diretório corrente
  - `cd <diretório>`
- Compilar em ByteCode
  - `javac OlaMundo.java`
- `Executar o programa`
  - `java OlaMundo`



Exercícios

- Comece a criar uma classe *Pessoa*, com, pelo menos, os atributos *idade* e *nome*.
- Desenvolva um programa principal para criar instâncias desta classe.
- Explore as opções existentes para os comandos *javac* e *java*.
- Continue avançando no tutorial da *Oracle*