IFRN– Instituto Federal de Educação, Ciência e Tecnologia do RN
Curso de Tecnologia em Análise e Desenvolvimento de Sistemas
Disciplina: Inglês Técnico
Professor: Sandro Luis de Sousa
Aluno(a)_____Turma:_____Data: ___/___/____.

"Hello World!" for Microsoft Windows

Adapted from: http://java.sun.com/docs/books/tutorial/getStarted/intro/definition.html

Let's write the "Hello World!" application! The following instructions are for users of Windows XP Professional, Windows XP Home, Windows Server 2003, Windows 2000 Professional, and Windows Vista.

A Checklist 

To write this program, you'll need:

1. **The Java SE Development Kit 6 (JDK 6)**
2. **A text editor**

   In this example, we'll use Notepad, a simple editor included with the Windows platforms. You can easily adapt these instructions if you use a different text editor.

These two items are all you'll need to write your first application reading instructions in English.

Creating Your First Application

Your first application, `HelloWorldApp`, will simply display the greeting "Hello world!" To create this program, you will:

- **Create a source file**

  A source file contains code, written in the Java programming language, that you and other programmers can understand. You can use any text editor to create and edit source files.

- **Compile the source file into a .class file**

  The Java programming language *compiler* (`javac`) takes your source file and translates its text into instructions that the Java virtual machine can understand. The instructions contained within this file are known as *bytecodes*.

- **Run the program**

  The Java application *launcher tool* (`java`) uses the Java virtual machine to run your application.

Create a Source File

First, start your editor. You can launch the Notepad editor from the **Start** menu by selecting **Programs > Accessories > Notepad**. In a new document, type in the following code:

```
/**
 * The HelloWorldApp class implements an application that
 * simply prints "Hello World!" to standard output.
 */
Public class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); // Display the string.
    }
}
```

Be Careful When You Type 

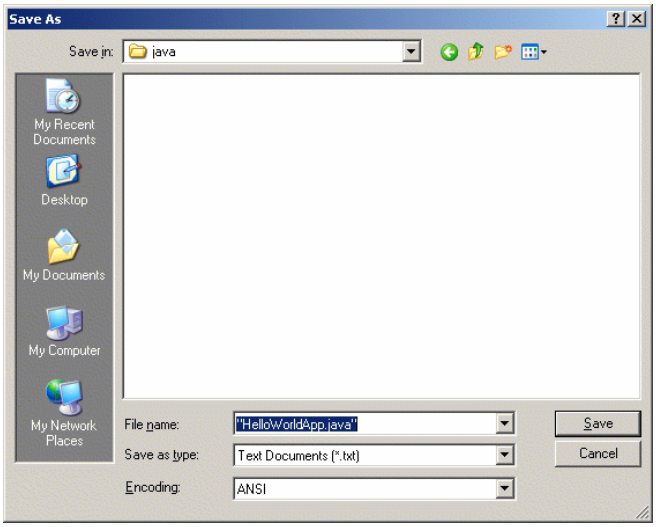
Type all code, commands, and file names exactly as shown. Both the compiler (`javac`) and launcher tool (`java`) are *case-sensitive*, so you must capitalize consistently.

`HelloWorldApp` ≠ `helloworldapp`

Save the code in a file with the name `HelloWorldApp.java`. To do this in Notepad, first choose the **File > Save As** menu item. Then, in the **Save As** dialog box:

1. Using the **Save in** combo box, specify the folder (directory) where you'll save your file. In this example, the **directory** is **`java`** on the *`C`* drive.
2. In the **File name** text field, type `"HelloWorldApp.java"`, including the quotation marks.
3. From the **Save as type** combo box, choose **Text Documents (*.txt)**.
4. In the **Encoding** combo box, leave the encoding as ANSI.

When you're finished, the dialog box should look like this.

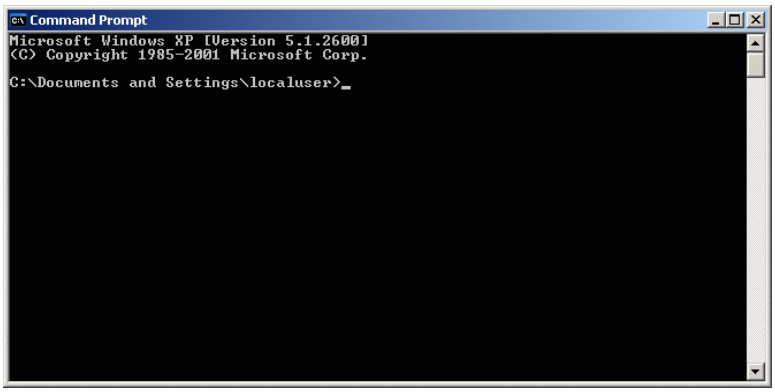


The *Save As* dialog just before you click **Save**.

Now click **Save**, and exit Notepad.

Compile the Source File into a .class File

Bring up a shell, or "command," window. You can do this from the **Start** menu by choosing **Command Prompt** (Windows XP), or by choosing **Run...** and then entering `cmd`. The shell window should look similar to the following figure.



A shell window.

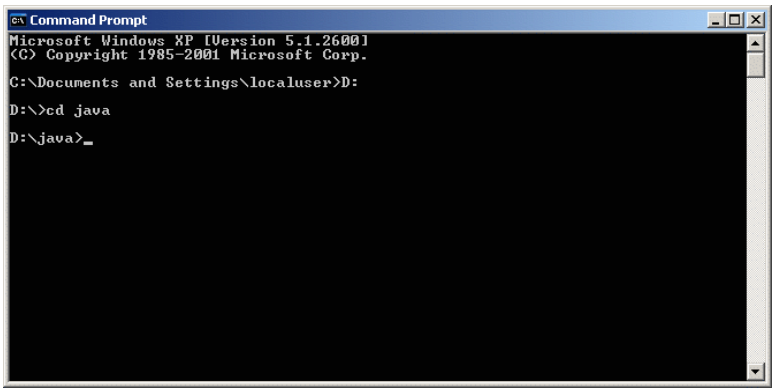
The prompt shows your *current directory*. When you bring up the prompt, your current directory is usually your home directory for Windows XP (as shown in the preceding figure).

To compile your source file, change your current directory to the directory where your file is located. For example, if your source directory is `java` on the `C` drive, type the following command at the prompt and press **Enter**:

```
cd C:\java
```

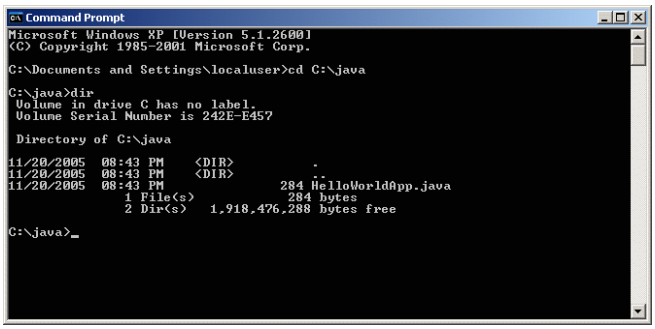
Now the prompt should change to `C:\java>`.

Note: To change to a directory on a different drive, you must type an extra command: the name of the drive. For example, to change to the `java` directory on the `D` drive, you must enter `D:`, as shown in the following figure.



Changing directory on an alternate drive.

If you enter **`dir`** at the prompt, you should see your source file, as the following figure shows.

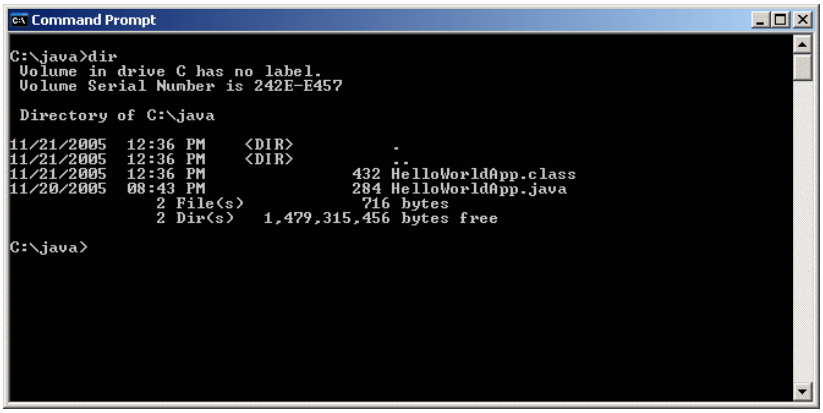


Directory listing showing the `.java` source file.

Now you are ready to compile. At the prompt, **type** the following command and press **Enter**.

```
javac _cp. HelloWorldApp.java
```

The compiler has generated a bytecode file, `HelloWorldApp.class`. At the prompt, **type** `dir` to see the new file that was generated, as shown in the following figure.



Directory listing, showing the generated `.class` file

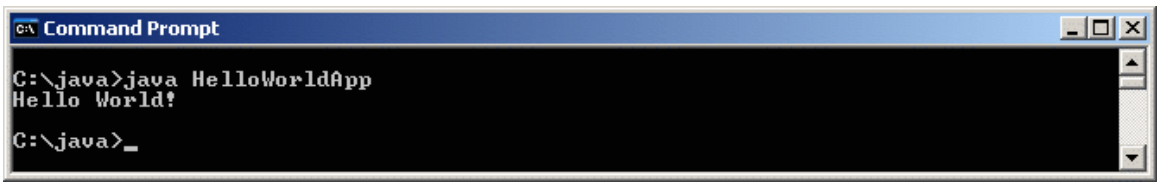
Now that you have a `.class` file, you can **run** your program.

Run the Program

In the same directory, enter the following command at the prompt:

```
java _cp. HelloWorldApp
```

The next figure shows what you should now see:



The program prints "Hello World!" to the screen.

Congratulations! Your program works!

IFRN– Instituto Federal de Educação, Ciência e Tecnologia do RN
Curso de Tecnologia em Análise e Desenvolvimento de Sistemas
Disciplina: Inglês Técnico
Professor: Sandro Luis de Sousa
Aluno(a)_____Turma:_____Data: ___/___/____.

A Closer Look at the "Hello World!" Application

Adapted from: http://java.sun.com/docs/books/tutorial/getStarted/intro/definition.html

Now that you've seen the "Hello World!" application, you might be wondering how it works. Here again is its code:

```
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); // Display the string.
    }
}
```

The "Hello World!" application consists of three primary components: source code comments, the `HelloWorldApp` class definition, and the `main` method. The following explanation will provide you with a basic understanding of the code.

1. Traduza o parágrafo acima:

_____
_____
_____
_____ .

Source Code Comments

The following bold text defines the *comments* of the "Hello World!" application:

```
/**
 * The HelloWorldApp class implements an application that
 * simply prints "Hello World!" to standard output.
 */
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); // Display the string.
    }
}
```

Comments are ignored by the compiler but are useful to other programmers. The Java programming language supports three kinds of comments:

`/*` *text* `*/`
: The compiler ignores everything from `/*` to `*/`.

`/**` *documentation* `*/`
: This indicates a documentation comment (*doc comment*, for short). The compiler ignores this kind of comment, just like it ignores comments that use `/*` and `*/`. The `javadoc` tool uses doc comments when preparing automatically generated documentation.

`//` *text*
: The compiler ignores everything from `//` to the end of the line.

The `HelloWorldApp` Class Definition

The following bold text begins the class definition block for the "Hello World!" application:

```
/**
 * The HelloWorldApp class implements an application that
 * simply displays "Hello World!" to the standard output.
 */
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); // Display the string.
    }
}
```

As shown above, the most basic form of a class definition is:

```
class name {
    . . .
}
```

The keyword class begins the class definition for a class named name, and the code for each class appears between the opening and closing curly braces marked in bold above. For now it is enough to know that every application begins with a class definition.

The `main` Method

The following bold text begins the definition of the `main` method:

```
/**
 * The HelloWorldApp class implements an application that
 * simply displays "Hello World!" to the standard output.
 */
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); //Display the string.
    }
}
```

In the Java programming language, every application must contain a `main` method whose signature is:

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

The modifiers `public` and `static` can be written in either order (`public static` or `static public`), but the convention is to use `public static` as shown above. You can name the argument anything you want, but most programmers choose "args" or "argv".

The `main` method is similar to the `main` function in C and C++; it's the entry point for your application and will subsequently invoke all the other methods required by your program.

The `main` method accepts a single argument: an array of elements of type `String`.

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

This array is the mechanism through which the runtime system passes information to your application. Each string in the array is called a *command-line argument*. Command-line arguments let users affect the operation of the application without recompiling it. For example, a sorting program might allow the user to specify that the data be sorted in descending order with this command-line argument:

```
-descending
```

The "Hello World!" application ignores its command-line arguments, but you should be aware of the fact that such arguments do exist.

Finally, the line:

System.out.println ("Hello World!");

uses the System class from the core library to print the "Hello World!" message to standard output.

Exercises:

1. Complete the definitions below with words taken from the text “the Main Method”.

a) _____ is when a program is running (or being executable).

b) _____ is a number of items arranged in some specified way.

2. Which of the following is *not* a valid comment:

a. `/** comment */`

b. `/* comment */`

c. `/* comment`

d. `// comment`

3. Ao declarar o método *main*, qual modificador deve vir primeiro, *public* ou *static*?

_____
_____ .

4. John tried to compile the *Hello World* program, but he got the following error as response:

HelloWorldApp2.java:7: unclosed string literal

System.out.println("Hello World!); //Display the string.

^

HelloWorldApp2.java:7: ')' expected

System.out.println("Hello World!); //Display the string.

^

O que causou o erro?

_____
_____ .

5. If you want to change the HelloWorldApp.java program below so that it displays Hola Mundo!, what do you have to do?

_____
_____ .

```
/**
 * The HelloWorldApp class implements an application that
 * simply prints "Hello World!" to standard output.
 */
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); // Display the string.
    }
}
```

The `main` method is similar to the `main` function in C and C++; it's the entry point for your application and will subsequently invoke all the other methods required by your program.

Traduza:

_____
_____
_____
_____
_____ .