Unit 2 – The Java Environment

Installing Java

Setting Classpath for Java

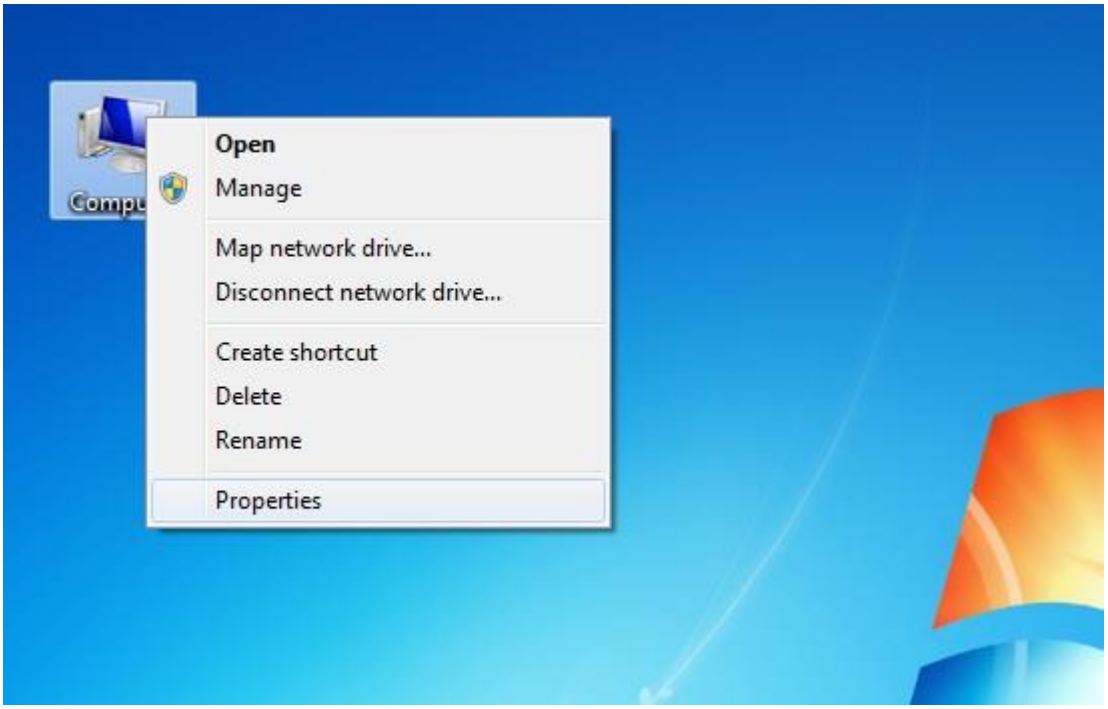
Java is freely available on Oracle's Website. Download the latest version of JDK (Java Development Kit) on your machine. Do check whether your machine is 32 bit or 64 bit and download that particular Java version. Install JDK on your machine. Once you have installed Java on your machine you would need to set environment variable to point to correct installation directory.

An **Environment variable** is a dynamic "object" on a computer that stores a value(like a key-value pair), which can be referenced by one or more software programs in Windows. Like for Java, we will set an environment variable with name "**java**" and its value will be the path of the **/bin** directory present in Java directory. So whenever a program will require Java environment, it will look for the **java** environment variable which will give it the path to the execution directory.

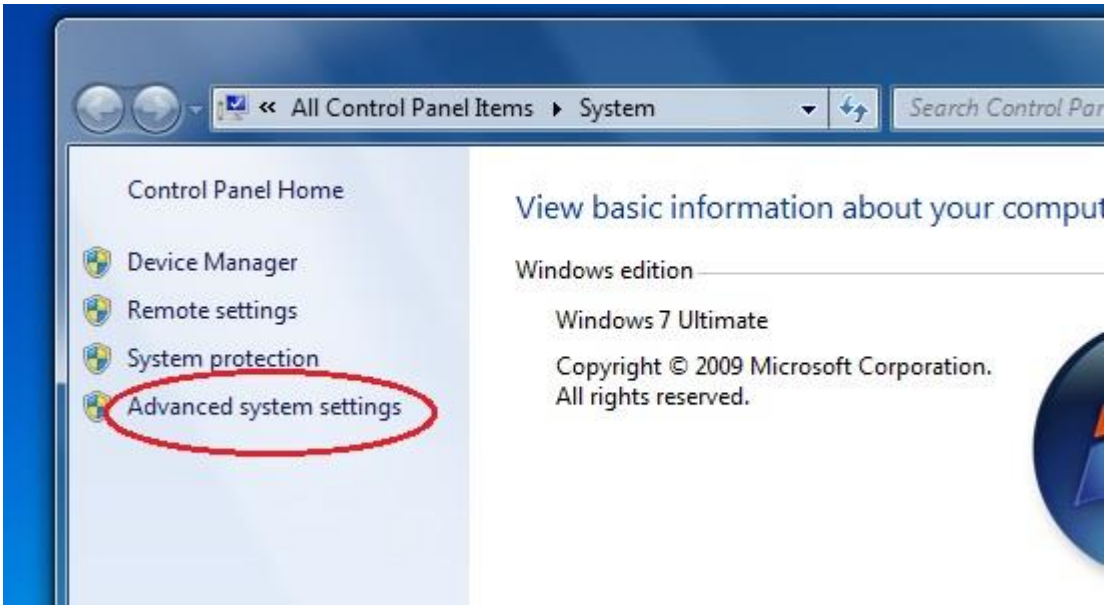
Setting up path for windows (2000/XP/vista/Window 7,8)

Assuming that you have installed Java in **C:\ Program files/ Java / JDK directory**

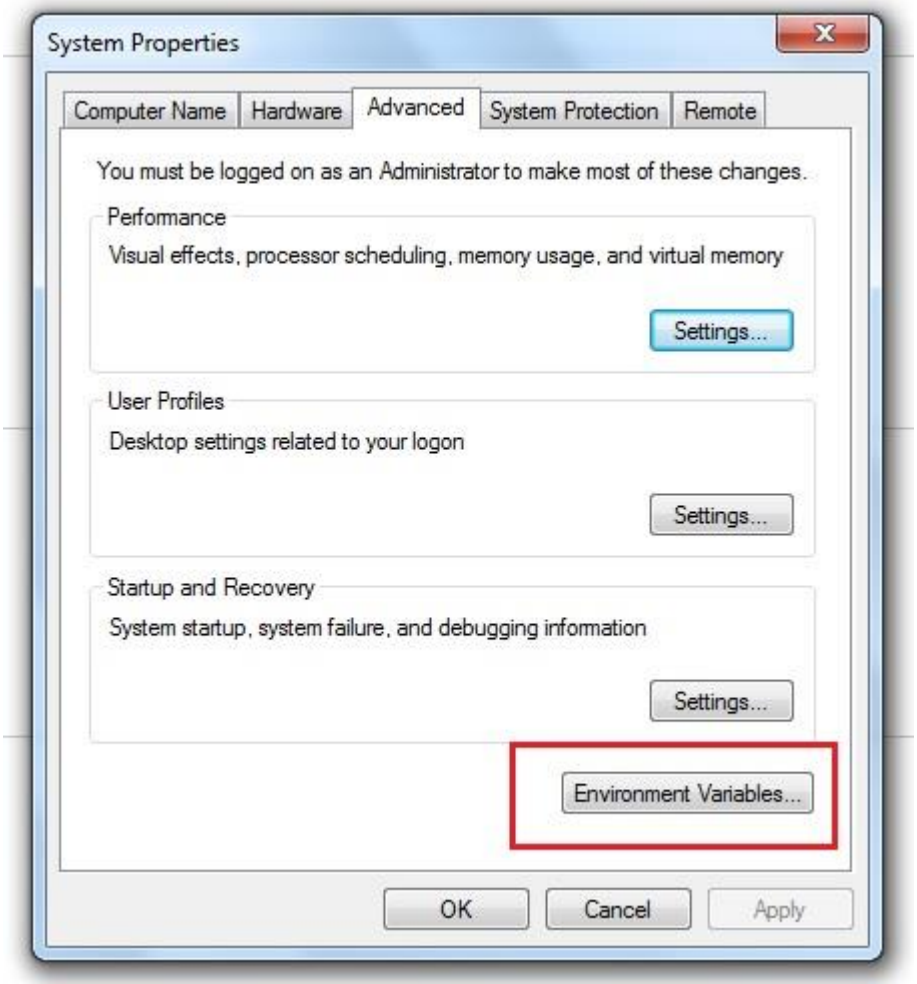
Step 1: Right click on my computer and select properties.



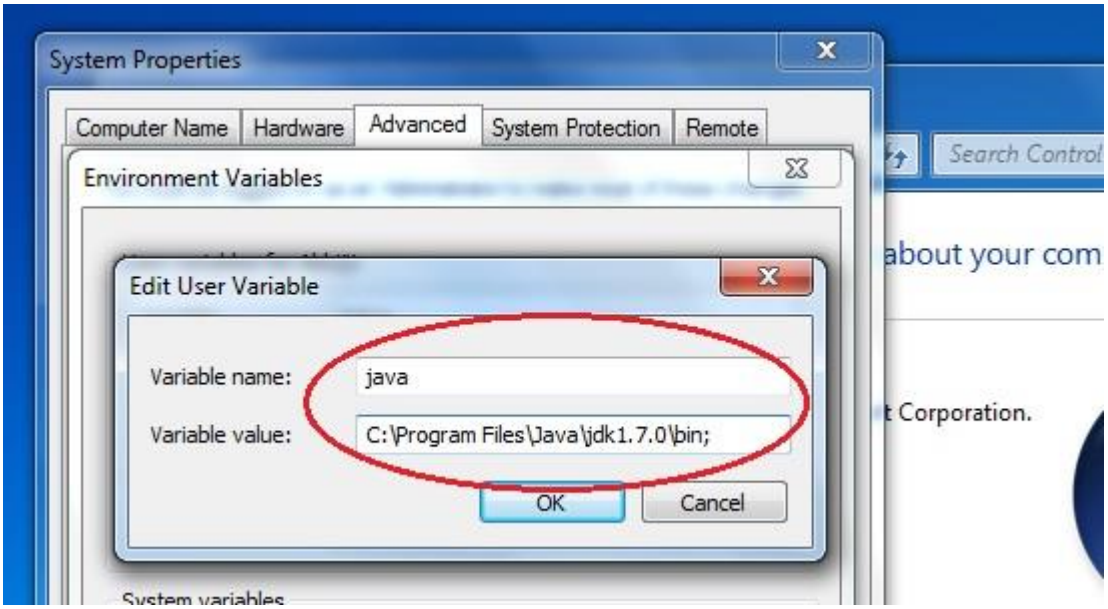
Step 2: Go to the Advance System Settings tab.



Step 3: Click on Environment Variables button.



Step 4: Now alter the path variable so that it also contains the path to JDK installed directory.



For e.g:- Change **C:\windows/ system 32.** to **C:\windows/system 32; C:\program files / Java/ JDK**.

Setting up path for window 95/98/ME

Assuming that you have installed Java in C:\program files\ java\ JDK directory, do the following:

Step 1: Edit the **C:\autoexec.bat** file and add the following line at the end.

SET PATH =% PATH% C:\ PROGRAM FILE/JAVA/JDK/bin

Java Program Development

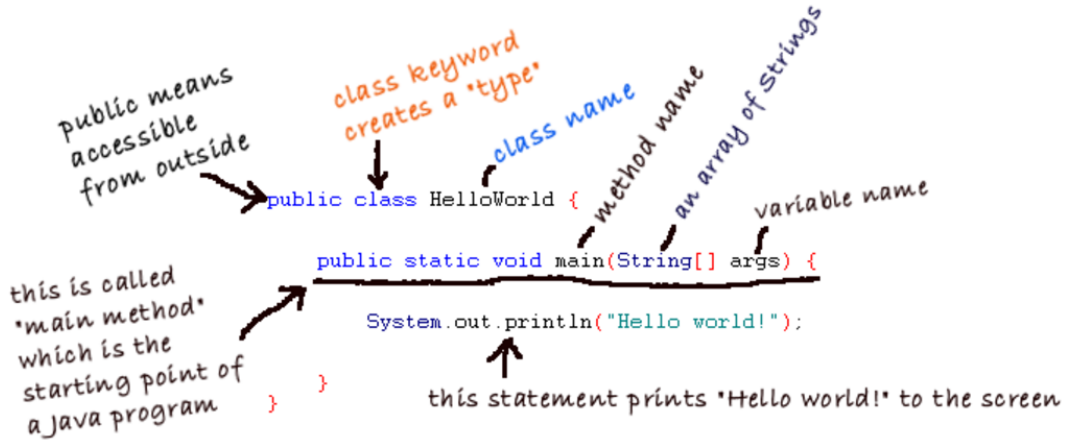
Writing a Java hello world program

Open a simple text editor program such as Notepad and type the following content:

```
1   public class HelloWorld {
2       public static void main(String[] args) {
3           System.out.println("Hello world!");
4       }
5   }
```

Save the file as HelloWorld.java (note that the extension is .java) under a directory, let's say, *C:\Java*.

Don’t worry if you don’t understand everything in this simple Java code. The following picture explains it nicely:



Every Java program starts from the main() method. This program simply prints “Hello world” to screen.

Compilation

Now let’s compile our first program in the HelloWorld.java file using javac tool. Type the following command to change the current directory to the one where the source file is stored:

cd C:\Java

And type the following command:

javac HelloWorld.java

That invokes the Java compiler to compile code in the HelloWorld.java file into bytecode. Note that the file name ends with .java extension. You would see the following output:

```
C:\>cd C:\Java
C:\Java>javac HelloWorld.java
C:\Java>_
```

If everything is fine (e.g. no error), the Java compiler quits silently, no fuss. After compiling, it generates the HelloWorld.class file which is bytecode form of the HelloWorld.java file. Now try to type dir in the command line, we’ll see the .class file:

So remember a Java program will be compiled into bytecode form (.class file).

Execution

It's now ready to run our first Java program. Type the following command:

java HelloWorld

That invokes the Java Virtual Machine to run the program called HelloWorld (note that there is no .java or .class extension). You would see the following output:

It just prints out "Hello world!" to the screen

Java Source File Structure

The structure of a skeletal Java source file is depicted in Figure. A Java source file can have the following elements that, if present, must be specified in the following order:

1. An optional package declaration to specify a package name
2. Zero or more import declarations. Since import declarations introduce class and interface names in the source code, they must be placed before any type declarations.
3. Any number of top-level class and interface declarations. Since these declarations belong to the same package, they are said to be defined at the top level, which is the package level.

The classes and interfaces can be defined in any order. Class and interface declarations are collectively known as type declarations. Technically, a source file need not have any such definitions, but that is hardly useful.

The Java 2 SDK imposes the restriction that at the most one public class definition per source file can be defined. If a public class is defined, the file name must match this public class. If the public class name is NewApp, then the file name must be NewApp.java.

Figure: Java Source File Structure

```
// Filename: NewApp.java

// PART 1: (OPTIONAL) package declaration
package com.company.project.fragilePackage;

// PART 2: (ZERO OR MORE) import declarations
import java.io.*;
import java.util.*;

// PART 3: (ZERO OR MORE) top-level class and interface declarations
public class NewApp { }

class AClass { }

interface IOne { }

class BClass { }

interface ITwo { }
// ...
// end of file
```

Note that except for the package and the import statements, all code is encapsulated in classes and interfaces. No such restriction applies to comments and white space.

Lexical structure of Java

Computer languages, like human languages, have a lexical structure. A source code of a Java program consists of tokens. Tokens are atomic code elements. In Java we have comments, identifiers, literals, operators, separators, and keywords.

Java programs are composed of characters from the Unicode character set.

Comments

Comments are used by humans to clarify source code. There are three types of comments in Java.

Comment type	**Meaning**
// comment	Single-line comments
/* comment */	Multi-line comments
/** documentation */	Documentation comments

If we want to add some small comment we can use single-line comments. For more complicated explanations, we can use multi-line comments. The documentation comments are used to prepare automatically generated documentation. This is generated with the javadoc tool.

Comments.java

```
package com.zetcode;


/*
  This is Comments.java
  Author: Jan Bodnar
  ZetCode 2017
*/
public class Comments {
    // Program starts here
    public static void main(String[] args) {
        System.out.println("This is Comments.java");
    }
}
```

The program uses two types of comments.

```
// Program starts here
```

This is an example of a single-line comment.

Comments are ignored by the Java compiler.

```
/*
  This is Comments.java
/*  Author: Jan Bodnar */
  ZetCode 2017
*/
```

Comments cannot be nested. The above code does not compile.

White space

White space in Java is used to separate tokens in the source file. It is also used to improve readability of the source code.

```
int i = 0;
```

White spaces are required in some places. For example between the int keyword and the variable name. In other places, white spaces are forbidden. They cannot be present in variable identifiers or language keywords.

```
int a=1;

int b = 2;

int c  =  3;
```

The amount of space put between tokens is irrelevant for the Java compiler. The white space should be used consistently in Java source code.

Identifiers

Identifiers are names for variables, methods, classes, or parameters. Identifiers can have alphanumerical characters, underscores and dollar signs ($). It is an error to begin a variable name with a number. White space in names is not permitted.

Identifiers are case sensitive. This means that Name, name, or NAME refer to three different variables. Identifiers also cannot match language keywords.

There are also conventions related to naming of identifiers. The names should be descriptive. We should not use cryptic names for our identifiers. If the name consists of multiple words, each subsequent word is capitalized.

```
String name23;

int _col;

short car_age;
```

These are valid Java identifiers.

```
String 23name;

int %col;

short car age;
```

These are invalid Java identifiers.

Literals

A *literal* is a textual representation of a particular value of a type. Literal types include boolean, integer, floating point, string, null, or character. Technically, a literal will be assigned a value at compile time, while a variable will be assigned at runtime.

int age = 29;

String nationality = "Hungarian";

Here we assign two literals to variables. Number 29 and string "Hungarian" are literals.

Operators

An *operator* is a symbol used to perform an action on some value. Operators are used in expressions to describe operations involving one or more operands.

```
+  -  *  /  %  ^  &  |  !  ~

=  +=  -=  *=  /=  %=  ^=  ++  --

==  !=  <  >  &=  >>=  <<=  >=  <=

||  &&  >>  <<  ?:
```

This is a partial list of Java operators. We will talk about operators later in the tutorial.

Separators

A *separator* is a sequence of one or more characters used to specify the boundary between separate, independent regions in plain text or other data stream.

```
[ ]  ( )  { }  ,  ;  .  "
```

String language = "Java";

The double quotes are used to mark the beginning and the end of a string. The semicolon ;character is used to end each Java statement.

System.out.println("Java language");

Parentheses (round brackets) always follow a method name. Between the parentheses we declare the input parameters. The parentheses are present even if the method does not take any parameters. The System.out.println() method takes one parameter, a string value. The dot character separates the class name (System) from the member (out) and the member from the method name (println()).

int[] array = new int[5] { 1, 2, 3, 4, 5 };

The square brackets [] are used to denote an array type. They are also used to access or modify array elements. The curly brackets {} are used to initiate arrays. The curly brackets are also used enclose the body of a method or a class.

int a, b, c;

The comma character separates variables in a single declaration.

Keywords

A keyword is a reserved word in Java language. Keywords are used to perform a specific task in the computer program. For example, to define variables, do repetitive tasks or perform logical operations.

Java is rich in keywords. It has 60 keywords.

abstract continue for new switch

assert default goto package synchronized

boolean do if private this

break double implements protected throw

byte else import public throws

case enum instanceof return transient

catch extends int short try

char final interface static void

class finally long strictfp volatile

const float native super while

In the following small program, we use several Java keywords.

Keywords.java

```
package com.zetcode;

public class Keywords {

  public static void main(String[] args) {

      for (int i = 0; i <= 5; i++) {

         System.out.println(i);

      }

   }

}
```

The package, public, class, static, void, int, for tokens are Java keywords.

Conventions

Conventions are best practices followed by programmers when writing source code. Each language can have its own set of conventions. Conventions are not strict rules; they are merely

recommendations for writing good quality code. We mention a few conventions that are recognized by Java programmers. (And often by other programmers too).

- Class names begin with an uppercase letter.
- Method names begin with a lowercase letter.
- The public keyword precedes the static keyword when both are used.
- The parameter name of the main() method is called args.
- Constants are written in uppercase.
- Each subsequent word in an identifier name begins with a capital letter.

Primitive Datatypes

In Java, every variable has a type declared in the source code. There are two kinds of types: reference types and primitive types. Reference types are references to objects. Primitive types directly contain values. There are 8 primitive types:

- byte
- short
- int
- long
- char
- float
- double
- Boolean

A primitive type is predefined by the language and is named by a reserved keyword. Primitive values do not share state with other primitive values. The eight primitive data types supported by the Java programming language are:

byte: The byte data type is an 8-bit signed two's complement integer. It has a minimum value of -128 and a maximum value of 127 (inclusive). The byte data type can be useful for saving memory in large arrays, where the memory savings actually matters. They can also be used in place of int where their limits help to clarify your code; the fact that a variable's range is limited can serve as a form of documentation.

short: The short data type is a 16-bit signed two's complement integer. It has a minimum value of -32,768 and a maximum value of 32,767 (inclusive). As with byte, the same guidelines apply: you can use a short to save memory in large arrays, in situations where the memory savings actually matters.

int: By default, the int data type is a 32-bit signed two's complement integer, which has a minimum value of -2^{31} and a maximum value of $2^{31}-1$. In Java SE 8 and later, you can use the int data type to represent an unsigned 32-bit integer, which has a minimum value of 0 and a maximum value of $2^{32}-1$. Use the Integer class to use int data type as an unsigned integer. See the section The Number Classes for more information. Static methods like compareUnsigned, divideUnsigned etc have been added to the Integer class to support the arithmetic operations for unsigned integers.

long: The long data type is a 64-bit signed two's complement integer. The signed long has a minimum value of -2^{63} and a maximum value of $2^{63}-1$. In Java SE 8 and later, you can use the long data type to represent an unsigned 64-bit long, which has a minimum value of 0 and a maximum value of $2^{64}-1$. The unsigned long has a minimum value of 0 and maximum value of $2^{64}-1$. Use this data type when you need a range of values wider than those provided by int. The Long class also contains methods like compareUnsigned, divideUnsigned etc to support arithmetic operations for unsigned long.

float: The float data type is a single-precision 32-bit IEEE 754 floating point. Its range of values is beyond the scope of this discussion, but is specified in the Floating-Point Types, Formats, and Values section of the Java Language Specification. As with the recommendations for byte and short, use a float (instead of double) if you need to save memory in large arrays of floating point numbers. This data type should never be used for precise values, such as currency. For that, you will need to use the java.math.BigDecimal class instead. Numbers and Strings covers BigDecimal and other useful classes provided by the Java platform.

double: The double data type is a double-precision 64-bit IEEE 754 floating point. Its range of values is beyond the scope of this discussion, but is specified in the Floating-Point Types, Formats, and Values section of the Java Language Specification. For decimal values, this data type is generally the default choice. As mentioned above, this data type should never be used for precise values, such as currency.

boolean: The boolean data type has only two possible values: true and false. Use this data type for simple flags that track true/false conditions. This data type represents one bit of information, but its "size" isn't something that's precisely defined.

char: The char data type is a single 16-bit Unicode character. It has a minimum value of '\u0000' (or 0) and a maximum value of '\uffff' (or 65,535 inclusive).