Java Basics

History of Java

The original name of Java was Oak, and it was developed as a part of the Green project at Sun Microsystems. Java was conceived by James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridon at Sun Microsystems in 1991.

The design objectives of the team were to come up with a language that:

- ensured security
- ensured reliability
- would be platform independent – one that would function seamlessly, regardless of the CPU

The World Wide Web (WWW), by nature, had requirements such as reliability, security, and architecture independence, which were in line with the design goals of Java. Sun formally announced the Java language in SunWorld Conference in 1995. On January 27, 2010, Sun was acquired by Oracle Corporation.

Oracle has three major Java products:

1. Java Platform Standard Edition (**Java SE**) for desktop application development.
2. Java Platform Enterprise Edition (**Java EE**) for enterprise application development.
3. Java Platform Micro Edition (**Java ME**) for mobile application development.

Java Major Versions

- JDK Alpha and Beta – 1995
- JDK 1.0 – January 23, 1996
- JDK 1.1 – February 19, 1997
- J2SE 1.2 – December 8, 1998
- J2SE 1.3 – May 8, 2000
- J2SE 1.4 – February 6, 2002
- J2SE 5.0 – September 30, 2004
- Java SE 6 – December 11, 2006
- Java SE 7 – July 28, 2011
- Java SE 8 – March 18, 2014

Java Technology

The word Java refers to many things.

- Java is a programming language.
- Java is a development environment that provides tools such as compiler, interpreter, documentation generator, and so on.
- Java is an application environment to run standalone programs that run on any machine where the Java Runtime Environment (JRE) is installed.
- Java is a deployment environment that supplies Java SE Development Kit with complete set of Application Programming Interface (APIs) as packages.
- Java provides an easy-to-use language by avoiding pitfalls of other languages, such as pointer arithmetic and memory management, which affect the robustness of the code.

Features of Java

1. Simple
    - Java is Easy to write and more readable and eye catching.
    - Java has a concise, cohesive set of features that makes it easy to learn and use.
    - Most of the concepts are drew from C / C++ thus making Java learning simpler.
2. Object Oriented
    - Object-oriented means organizing software as a combination of different types of objects that incorporates both data and behaviour.
    - Object-oriented programming(OOPs) is a methodology that simplify software development and maintenance by providing some rules.
    - Basic concepts of OOPs are:
        - Object
        - Class
        - Inheritance
        - Polymorphism
        - Abstraction
        - Encapsulation

3. Portable

   - The portability actually comes from architecture-neutrality. In C/C++, source code may run slightly differently on different hardware platforms because of how these platforms implement arithmetic operations. In Java, it has been simplified. Unlike C/C++, in Java the size of the primitive data types are machine independent. For example, an int in Java is always a 32-bit integer, and float is always a 32-bit IEEE 754 floating point number. These consistencies make Java programs portable among different platforms such as Windows, Linux and Mac OS.

4. Platform Independent

   - Java source code (.java file) is compiled by the Java compiler (javac) and converted into bytecode (.class file).
   - Java bytecode can be executed on multiple platforms e.g. Windows, Linux and Mac OS, that has Java Virtual Machine (JVM).
   - JVM converts the bytecode to machine specific code. Thus JVM makes Java as platform independent.

5. Robust

   - Reliable - that the code works as desired in normal circumstances for specified periods of time.
   - Robustness - the code responding appropriately in abnormal circumstances.
   - Strong Error handling mechanism: Lot of checks are done at the compilation itself. More checks are performed at runtime system to make sure that code does not malfunction.
   - Automatic garbage collection

6. Multithreaded

   - At the OS level, smallest unit of work that can be scheduled, is a thread.
   - Thread is a sequence of execution of code.
   - A process consists of one or more threads.
   - Multiple threads in the same program share same resources.

- Unlike a multiple process, multiple threads in a process share same memory location. That is why threads are sometimes called Light weight process.
- Java Standard API has rich set of classes that allows us to work with multiple threads simultaneously

7. Dynamic and Extensible

   - Java has Dynamic and Extensible means with the help of OOPS Java provides inheritance and with the help of inheritance we reuse the code that is pre-defined and also uses all the built in functions of Java and Classes

8. Security

   - With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.

9. High performance

   - With the use of Just-In-Time (JIT) compilers, Java enables high performance.

First Java Program

Now let us write a program which displays "Hello World" on console.
Follow the steps below:

Step 1 Open a terminal and type below command to create a file.

```
$ vim Welcome.java
```

Step 2 Type the below code in the file.

```
1 class Welcome {
2     public static void main(String[] args) {
3         System.out.println(''Hello World'');
4     }
5 }
```

Step 3 Save and quit from the file Press **Esc :wq**

Step 4 Type the below command in terminal to compile the source code


```
$ javac Welcome.java
```

Note: If the source code is compiled without any errors, it creates a `.class` *file (*`Welcome.class`*)*

Step 5 Type the below command to execute the file.

```
$ java Welcome
```

Once the execution is done, you can see the output as shown below:

Analysis of the program

As shown in above code, following is the most basic form of a class definition

```
class Welcome {
    . . .
}
```

The keyword class begins the class definition for a class named "Welcome", and the code for each class appears between the opening and closing braces. Every Java application begins with a class definition.

```
public static void main(String[] args) {
    . . .
}
```

Every Java application must contain a main method.

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.

```
System.out.println(``Hello world'');
```

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

Language Fundamentals

Comments

Comments are used to document and explain your codes and program logic. Comments are not programming statements and are ignored by the compiler, but they VERY IMPORTANT for providing documentation and explanation to understand the program.

There are two kinds of comments in Java:

Single-line Comment begins with // and lasts till the end of the current line.

Multi-line Comment begins with a /* and ends with a */, and can span several lines.

Use comments liberally to explain and document your codes. During program development, instead of deleting a chunk of statements irrevocably, you could comment-out these statements so that you could get them back later, if needed.

Keywords

Keywords are predefined identifiers reserved by Java for a specific purpose. You cannot use keywords as names for your variables, classes and methods.

abstract	default	if	protected
assert***	do	implements	public
boolean	double	import	return
break	else	instanceof	short
byte	enum****	int	static
case	extends	interface	strictfp**
catch	final	long	super
char	finally	native	switch
class	float	new	synchronized
const*	for	package	this
continue	goto*	private	throw

throws	try	volatile
transient	void	while

Note :

*1. * not used, ** added in 1.2, *** added in 1.4, **** added in 5.0*

2. The symbolic constants null, true and false are not keywords. They cannot be used for any other purpose.

Data Types

Data type specifies the size and type of values that can be stored in an identifier. The Java programming language is statically-typed, which means that all variables must first be declared before they can be used. This involves stating the variable's type and name, as you have learnt in C Programming Language.

Data types in Java are classified into two types:

Primitive data types are 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.

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).

int The int data type is a 32-bit signed two's complement integer, which has a minimum value is - 2,147,483,648 and a maximum value - 2,147,483,647.

long The long data type is a 64-bit two's complement integer. The signed long has a minimum value is -9,223,372,036,854,775,808 and a maximum value - 9,223,372,036,854,775,807.

float The float data type is a single-precision 32-bit IEEE 754 floating point. Use a float (instead of double) if you need to save memory in large arrays of floating point numbers.

double The double data type is a double-precision 64-bit IEEE 754 floating point. For decimal values, this data type is generally the default choice.

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.

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).

Non-primitive data types are not defined by the programming language, but are instead created by the programmer. They are sometimes called "reference variables" or "object references", since they reference a memory location, which stores the data.

Literals

A literal, or literal constant, is a specific constant value or raw data, such as 123, -456, 3.14, 'a', "Hello", that is used in the program source. It can be assigned directly to a variable, or used as part of an expression. They are called literals because they literally and explicitly identify their values. We call it literal to distinguish it from a variable.

Integer Literals An integer literal is of type long if it ends with the letter 'L'or 'l', otherwise it is of type int. It is recommended that you use the upper case letter 'L' because the lower case letter 'l' is hard to distinguish from the digit 1.

Values of the integral types byte, short, int, and long can be created from int literals. Values of type long that exceed the range of int can be created from long literals. Integer literals can be expressed by these number systems:

Decimal - Base 10, whose digits consists of the numbers 0 through 9, this is the number system you use every day

Hexadecimal - Base 16, whose digits consist of the numbers 0 through 9 and the letters A through F

Binary - Base 2, whose digits consists of the numbers 0 and 1 (you can create binary literals in Java SE 7 and later)

For general-purpose programming, the decimal system is likely to be the only number system you will ever use. However, if you need to use another number system, the following example shows the correct syntax. The prefix `0x` indicates hexadecimal and `0b` indicates binary:

```
// The number 26, in decimal
int decVal = 26;
```



```
//  The number 26, in hexadecimal
int hexVal = 0x1a;
// The number 26, in binary
int binVal = 0b11010;
```

Floating-Point Literals A number with a decimal point, such as 55.66 and -33.44, is treated as a double, by default. You can also express them in scientific notation, e.g., 1.2e34, -5.5E-6, where e or E denotes the exponent in power of 10. You could precede the fractional part or exponent with a plus (+) or minus (-) sign. Exponent values are restricted to integer. There should be no space or other characters in the number.

You can optionally use suffix 'd' or 'D' to denote double literals.

You MUST use a suffix of 'f' or 'F' for float literals, e.g., -1.2345F. For example,

```
float average = 55.66;       // Error! RHS is a double. Need suffix 'f' fo
float average = 55.66f;
```

Character Literals A printable char literal is written by enclosing the character with a pair of single quotes, e.g., 'z', '$', and '9'.

String Literals A String literal is composed of zero or more characters surrounded by a pair of double quotes, e.g., "Hello, world!", "The sum is ". For example,

```
String directionMsg = ''Turn Right'';
String greetingMsg = ''Hello'';
String statusMsg = '''';     // empty string
```

Type Casting

Type Casting is the mapping type of an object to another. Casting between primitive types enables you to convert the value of one data from one type to another primitive type, commonly occurs between numeric types. Boolean data type can't be converted to other types.

Types of Casting:

1. Implicit Casting
2. Explicit Casting

Implicit Casting means that a value of one type is changed to a value of another type without any special directive from the programmer. A char can be implicitly converted to an int, a long, a float, or a double.

Suppose you want to store a value of int to double.

```
int numInt = 10;
double numDouble = numInt; //implicit cast
```

Explicit Casting is done via casting. The name of the type to which you want a value converted is given, in parentheses, in front of the value. When you convert a data that has a larger type to a smaller type, you must use explicit casting. For example, the following code casts a value of type double to a value of type int.

```
double valDouble = 10.12;
int valInt = (int)valDouble;
```

Operators

An operator is a symbol that operates on one or more operands to produce a result. There are many operators available in Java language.

Arithmetic Operators $\Longrightarrow$ +, -, *, %, /

Relational Operators $\Longrightarrow$ >, <, >=, <=, !=, ==

Assignment Operator $\Longrightarrow$ =

Unary Operators $\Longrightarrow$ ++, - -

LogicalOperators $\Longrightarrow$ &&, ||, !

Control Statements

The control statements are used to control the flow of execution of the program. The execution order depends on the supplied data values and the conditional logic.

Types of control statements:

1. Conditional Statements
   (a) if

(b) if else

(c) else if

(d) nested if

(e) switch

2. Looping Statements

(a) while

(b) for

(c) do while

Conditional Statements

if statement will execute a single statement or a block of statements, when the given boolean expression is true and if it is false then it skips if block and executed the statements after the if block.

```
if (boolean_expression) {
    statement(s);
}
```

if-else statement provides a secondary path of execution when an if statement evaluates to false.

```
if (boolean_expression) {
    A_statement(s);
} else {
    B_statement(s);
}
```

if-else if statement can be followed by an optional else if ... else statement, which is very useful to test various conditions using single if ... else if statement.

```
if (boolean_expression1) {
    A_statement(s);
} else if (boolean_expression2) {
    B_statement(s);
} else if (boolean_expression3) {
```



```
        C-statement(s);
    } else {
        D-statement(s);
    }
```

nested-if statement is an if-else statement with another if statement as the if body or the else body.

```
if (boolean_expression1) {
    if (boolean_expression2) {
        A-statement(s);
    } else {
        B-statement(s);
    }
} else {
    C- statement(s);
}
```

switch statement can have a number of possible execution paths. A switch works with the byte, short, char, and int primitive data types. The expression in switch can be a String (Introduced in Java 1.7) type.

```
switch (expression) {
    case value1:
        statement(s);
        break;
    case value2:
        statement(s);
        break;
    case valueN:
        statement(s);
        break;
    default:
        statement(s);
}
```


Looping Statements

While statement continually executes a block of statement(s) until the given boolean expression is false.

```
while (boolean_expression) {
    statement(s);
}
```

do-while is similar to while, but it evaluates the boolean expression after executing the block of statement(s).

```
do {
    statement(s);
} while (boolean_expression);
```

for loop controls a sequence of repetitions. A basic for structure has three parts: *init*, *test*, and *update*. Each part must be separated by a semicolon (;). The loop continues until the test evaluates to false.

```
for (initialization; condition; increment/decrement) {
    statement(s);
}
```

Arrays

Array is a collection of similar type of data. An array is an indexed collection of fixed number of homogeneous data elements. The length of an array is established when the array is created. After creation, its length is fixed.

Creating Arrays

An array can be declared by specifying the type of the elements and the number of elements you want to store. Every array in a java is an object, Hence we can create an array by using new keyword. When an array is constructed via the new operator, all the elements are initialized to their default values.

An array can be created in any of the two ways:

int [] marks = new int[10];

int marks[] = new int[10];

Types of Arrays

One-Dimensional-Array is used to store the data in a row or column wise. if you want to store the roll numbers of 20 students, we can use one-dimensional array.

int [] rollNos = new int[20];

The above code will create an array named "rollNos" of size 20 and initializes the element values to 0.

Two-Dimensional-Array is used to store data in row and column based index (also known as matrix form). if you want to store roll Numbers and totalMarks secured by 20 students we can use two-dimensional array.

int [][] studMarks = new int[20][2];

The above code will creat an array named "studMarks" of size 20 X 2 and initializes the element values to 0.

Jagged array contains group of arrays within it. It means we can create an array in Java such that other arrays can become its elements. This is a unique feature in Java. A jagged array can store single dimensional array or multi dimensional arrays and also it can store arrays of any size. Jagged arrays are also called as ' irregular multidimensional arrays '. Jagged arrays are useful when dealing with a group of arrays of different sizes.

To create a jagged array that can store two 1D arrays, we can write as: int x [][] = new int [2][];

Here, x is the jagged array with size 2. So, its elements will be x[0] and x[1]. Observe last pair of empty square braces in the expression: new int[2][]. This last pair of braces represents 1D array. So, x[0] and x[1] can store two 1D arrays. Now let us allot memory for x[0] and x[1] so that they can store two 1D arrays of different sizes, as:


```
x[0] = new int[2];
x[1] = new int[3];
```

So, the first array is represented by x[0] and can have 2 elements which can be referenced as

x [0][0], x [0][1];

Similarly, the second array which is represented by x[1] can have 3 elements

x [1][0], x [1][1], x [1][2]

In the same way, we can also cretae a jagged array that contain 2D arrays. For example,

double arr [][][] = new double [3][][];

Here, arr represents a jagged array with size 3. Observe the last two pairs of empty braces. They represents 2D arrays. So, the jagged array can store 3 other 2 dimensional arrays. The elemets arr[0], arr[1] and arr[2] represent them as

```
arr[0] = new double[2][3];
arr[1] = new double[2][2];
arr[2] = new double[3][2];
```

The first 2D array can have 2 rows and 3 columns. Its elements can be referenced starting from arr[0][0][0] to arr[0][1][2]. The second 2D array can have 2 rows and 2 columns and its elemnts can be referenced from arr[1][0][0] to arr[1][1][1]. The third 2D array can have 3 rows and 2 columns and its elements can vary from arr[2][0][0] to arr[2][2][1].

Example Program to demonstrate Jagged arrays.

```
1  class JaggedArray {
2      public static void main(String args[]) {
3          //jagged array that can contain two 1D arrays
4
5          int x[][] = new int[2][];
6
7          //create 2 more 1D arrays as part of x
8
9          x[0] = new int[2]; //2 elements in first array
10         x[1] = new int[3]; //3 elements in second array
11
```



```
12          //store 2 elements into first array
13
14          x[0][0] = 10;
15          x[1][1] = 20;
16
17          //store 3 elements into second array
18
19          x[1][0] = 30;
20          x[1][1] = 20;
21          x[1][2] = 40;
22
23          //display first array
24
25          for (int i = 0; i < 2; i++) {
26              System.out.print(x[0][i] + " ");
27          }
28          System.out.println();
29
30          //display second array
31
32          for (int i = 0; i < 3; i++) {
33              System.out.print(x[1][i] + " ");
34          }
35          System.out.println();
36      }
37  }
```

Command Line Arguments

Java application can accept any number of arguments directly from the command line. The user can enter command line arguments when invoking the application. When running the java program from java command, the arguments are provided after the name of the class separated by space. For example, suppose a program named `CommandLineArguments` that accept command line arguments as a string array.

```
1  public class CommandLineArguments {
```



```
2       public static void main(String[] args) {
3           for (int i = 0; i < args.length; i++) {
4               System.out.println(args[i]);
5           }
6       }
7   }
```

Passing Numeric CommandLine Arguments

If an application needs to support a numeric command line argument, it must convert a String argument that represents a number, such as "34", to a numeric value.

Here is a code snippet that converts a command-line argument to an int.

```
1   public class  CommandLineNumArg {
2       public static void main(String[] args) {
3           int firstArg, secondArg;
4           if (args.length==2) {
5               System.out.println(``before conversion:'');
6               System.out.println(args[0] + args[1]);
7               firstArg = Integer.parseInt(args[0]);
8               secondArg = Integer.parseInt(args[1]);
9               System.out.println(``after conversion:'');
10              System.out.println(firstArg + secondArg);
11          } else {
12              System.out.println(``Two values must be required'');
13          }
14      }
15  }
```

`parseInt` throws a `NumberFormatException` if the format of `args[0]` is not valid. All of the `Number` classes `Integer`, `Float`, `Double`, and so on have `parseXXX` methods that convert a `String` representing a number to an object of their type.

Important Points

- Command Line Arguments can be used to specify configuration information while launching your application.

- There is no restriction on the number of command line arguments. You can specify any number of arguments.
- Information is passed as Strings.
- They are captured into the String argument of your main method.

String class

Generally, string is a sequence of characters. But in java, string is an object that represents a sequence of characters. `String` class is used to create string object. The java `String` is immutable i.e. it cannot be changed but a new instance is created. For mutable class, you can use `StringBuffer` and `StringBuilder` class.

We will discuss about immutable string later. Let's first understand what is string in java and how to create the string object

Java String is, however, special. Unlike an ordinary class

- String is associated with string literal in the form of double-quoted text such as "Hello world". You can assign a string literal directly into a String variable, instead of calling the constructor to create a String instance.
- The '+' operator is overloaded to concatenate two String operands. '+' does not work on any other objects.
- String is immutable. That is, its content cannot be modified once it is created.

  String s = "abc";

  String s2 = "abc";

  Memory for s and s2 are allocated as below.

Creating a String object

String objects can be created in two ways.

String Literal is a simple string enclosed in double quotes (" "). A string literal is treated as a String object.

```
String str = ''Talentsprint'';
```

new operator creates an instance of String class.

```
String str = new String(''Java'');
```

String Class Methods

charAt(int index) returns the `character` at the specified index.

- char charAt(int index);

compareTo(String anotherString) compares two strings lexicographically, based on the Unicode value of each character in the Strings.

- int compareTo(String anotherString);

compareToIgnoreCase(String str) compares two strings lexicographically, ignoring case differences.

- int compareToIgnoreCase(String str);

concat(String str) concatenates the specified string to the end of this string.

- String concat(String str);

equals(Object anObject) compares this string to the specified object.

- boolean equals(Object anObject);

equalsIgnoreCase(String anotherString) compares this String to another String, ignoring case considerations.

- String equalsIgnoreCase(String anotherString);

length() returns the number of Unicode characters contained in the String.

- int length();

replace(char oldChar, char newChar) returns a new string resulting from replacing all occurrences of `oldChar` in this string with `newChar`.

- String replace(char oldChar, char newChar);

substring(int beginIndex, int endIndex) returns the substring of this string starting at the specified `beginIndex` up to the `endIndex` index.

- String substring(int beginIndex, int endIndex);

trim() returns a copy of the string, with leading and trailing whitespace omitted.

- String trim();

StringBuffer and StringBuilder

StringBuffer class

The StringBuffer class is used to created mutable (modifiable) string. The StringBuffer class is same as String except it is mutable i.e. it can be changed.

Strings that need modification are handeled by StringBuffer class After creating a StringBuffer, new string(s) can be inserted or appendedto it. The size of StringBuffer can change whenever needed. StringBuffer objects can be dynamically altered. When a StringBuffer is created, space for 16 more characters is always appened to it. This helps the StringBuffer object to grow by 16 characters without any other process. When the string grows beyound the free 16 character space, StringBuffer is relocated to a new memory space with required size.

StringBuffer Class Methods

append(String s) is used to append the specified string with this string. The `append()` method is overloaded like `append(char)`, `append(boolean)`, `append(int)`, `append(float)`, `append(double)` etc.

- StringBuffer append(String s);

insert(int offset, String s) is used to insert the specified string with this string at the specified position. The `insert()` method is overloaded like `insert(int, char)`, `insert(int, boolean)`, `insert(int, int)`, `insert(int, float)`, `insert(int, double)` etc.

- StringBuffer insert(int offset, String s);

delete(int startIndex, int endIndex) is used to delete the string from specified `startIndex` and `endIndex`.

- StringBuffer delete(int startIndex, int endIndex);

reverse() is used to reverse the string.

- String reverse();

toString() returns a string representing the data in this sequence.

- String toString();

StringBuilder

JDK 1.5 introduced a new StringBuilder class (in package java.lang), which is almost identical to the StringBuffer class, except that it is not synchronized.

StringBuilder API-compatible with the StringBuffer class, i.e., having the same set of constructors and methods, but with no guarantee of synchronization. It can be a drop-in replacement for StringBuffer under a single-thread environment.

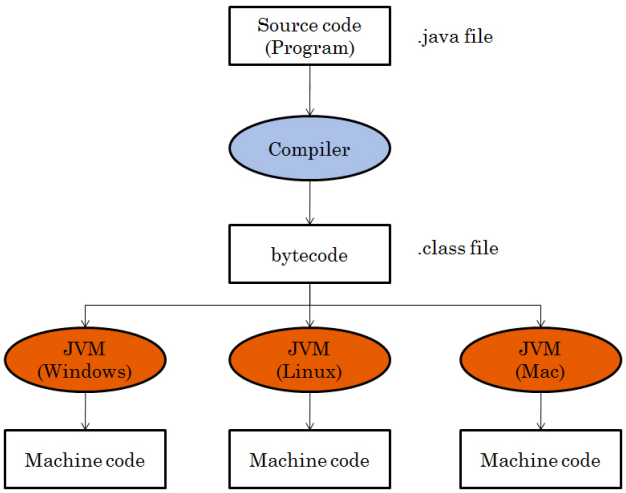
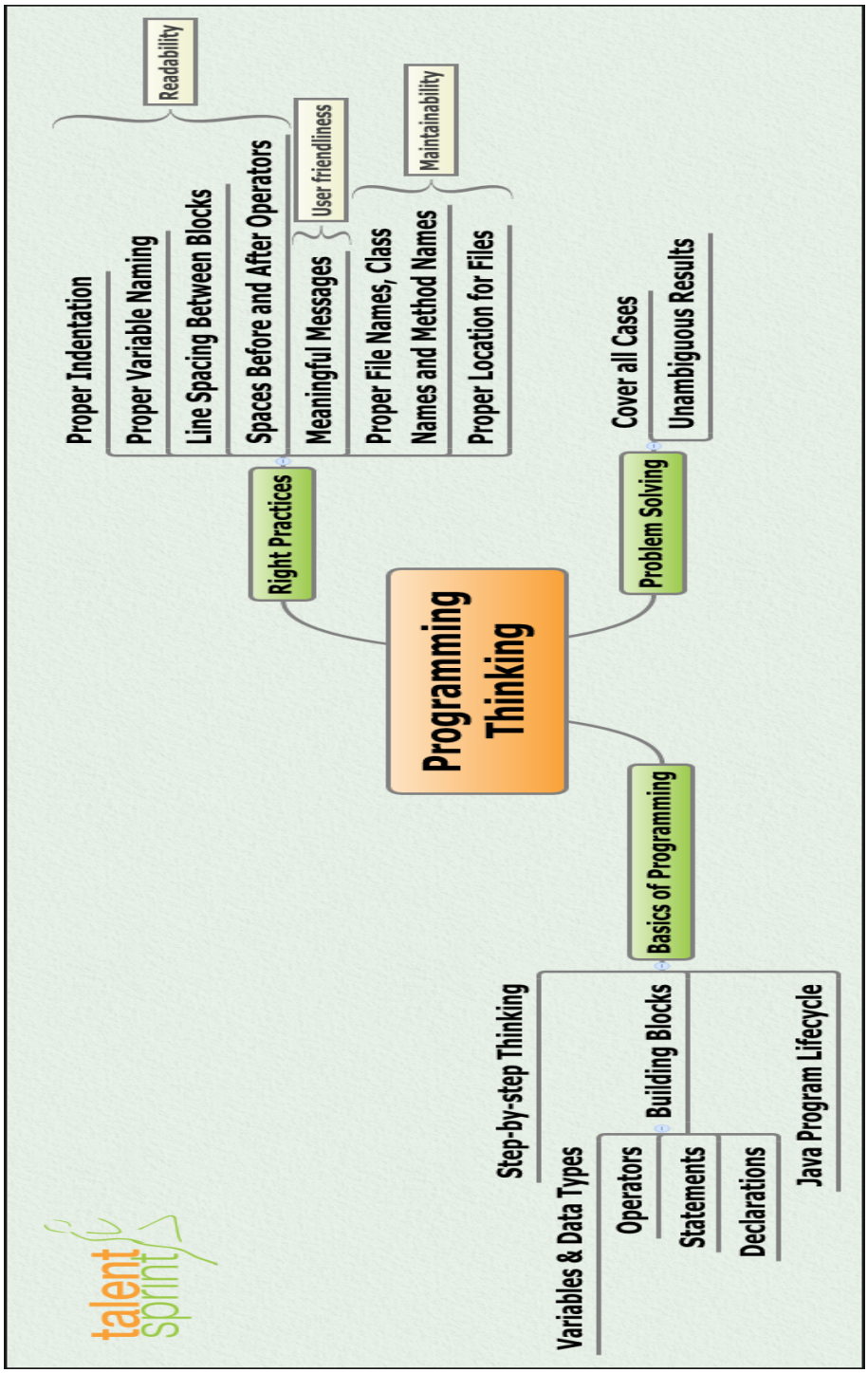
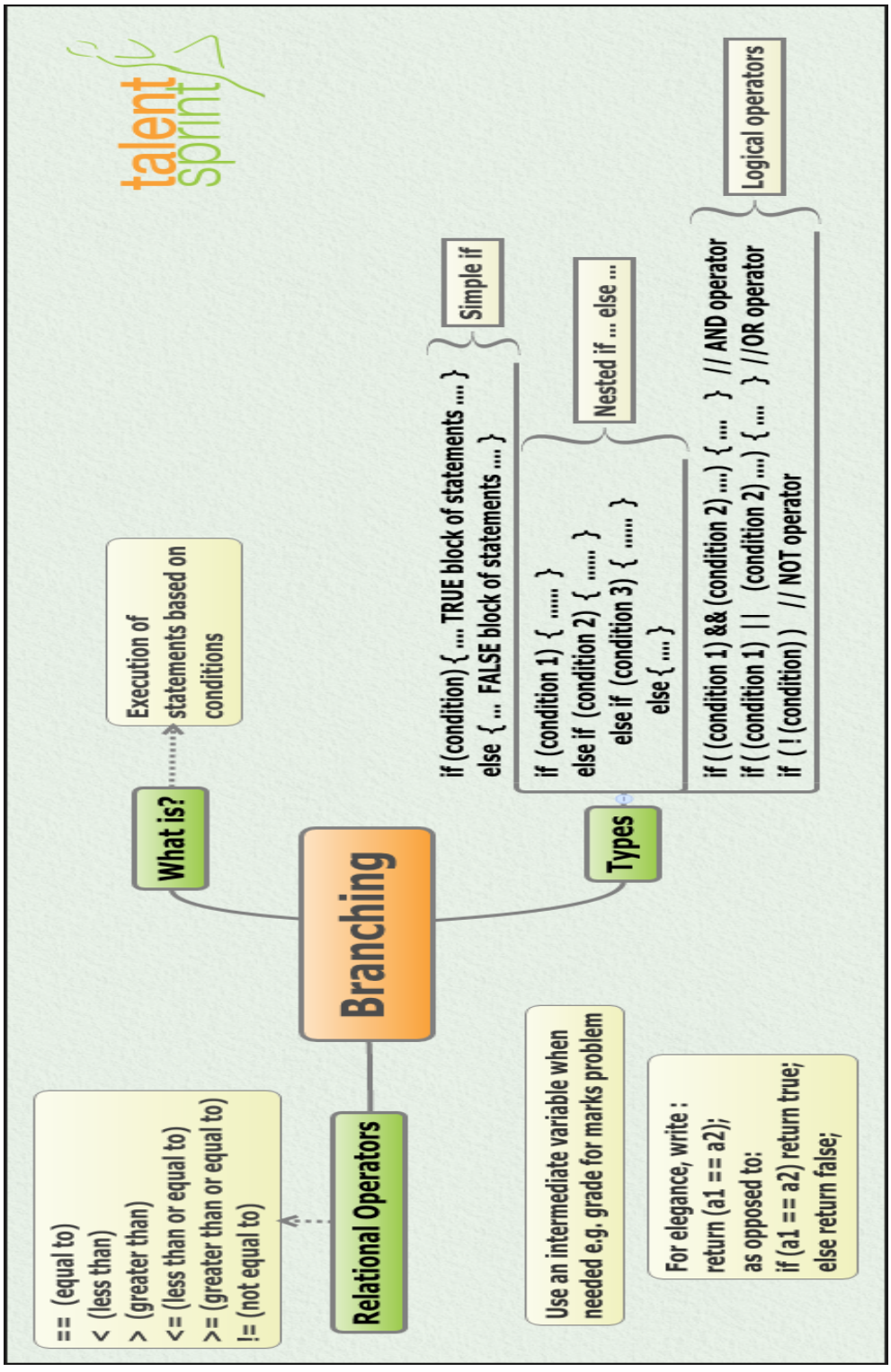
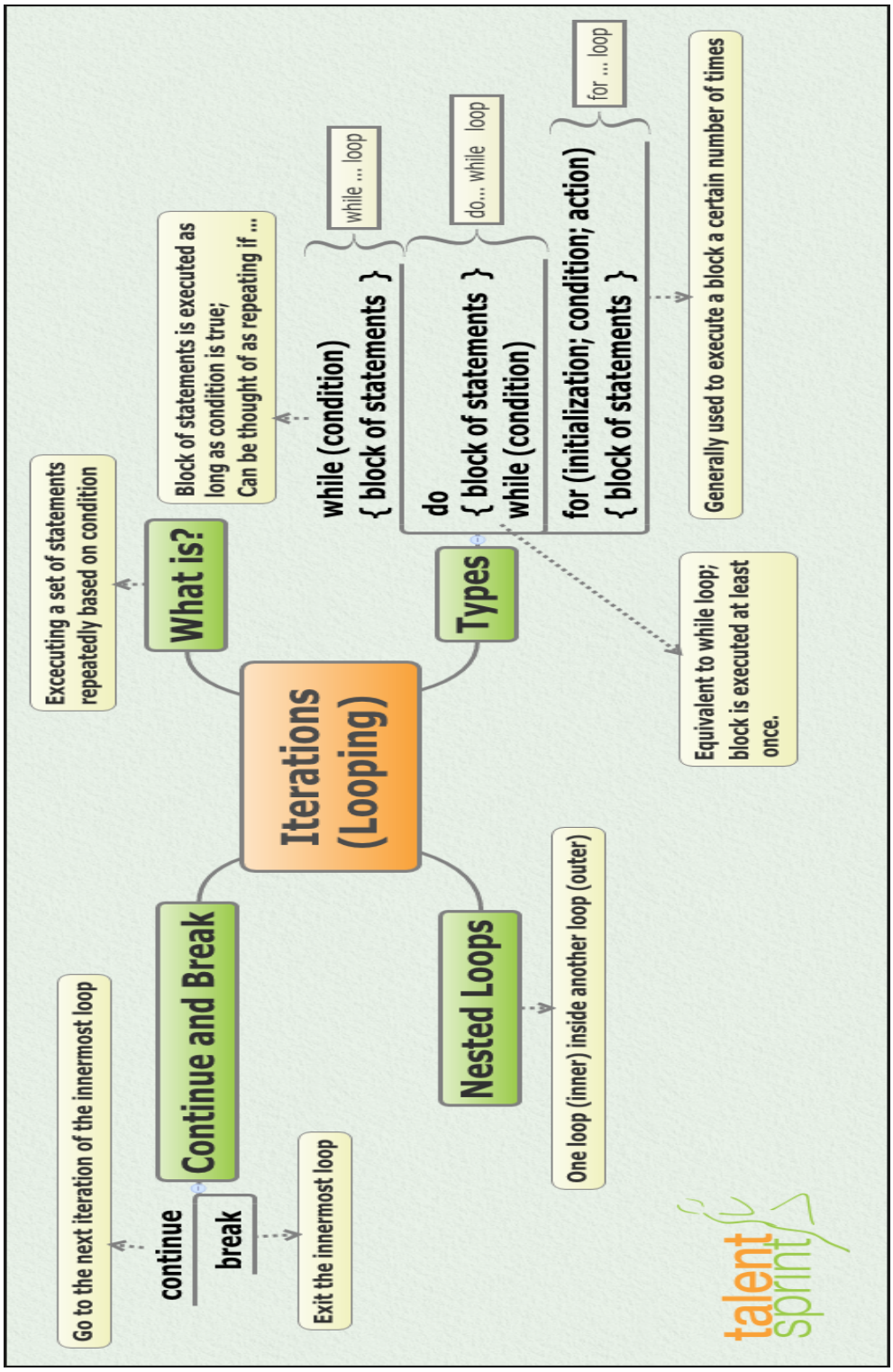


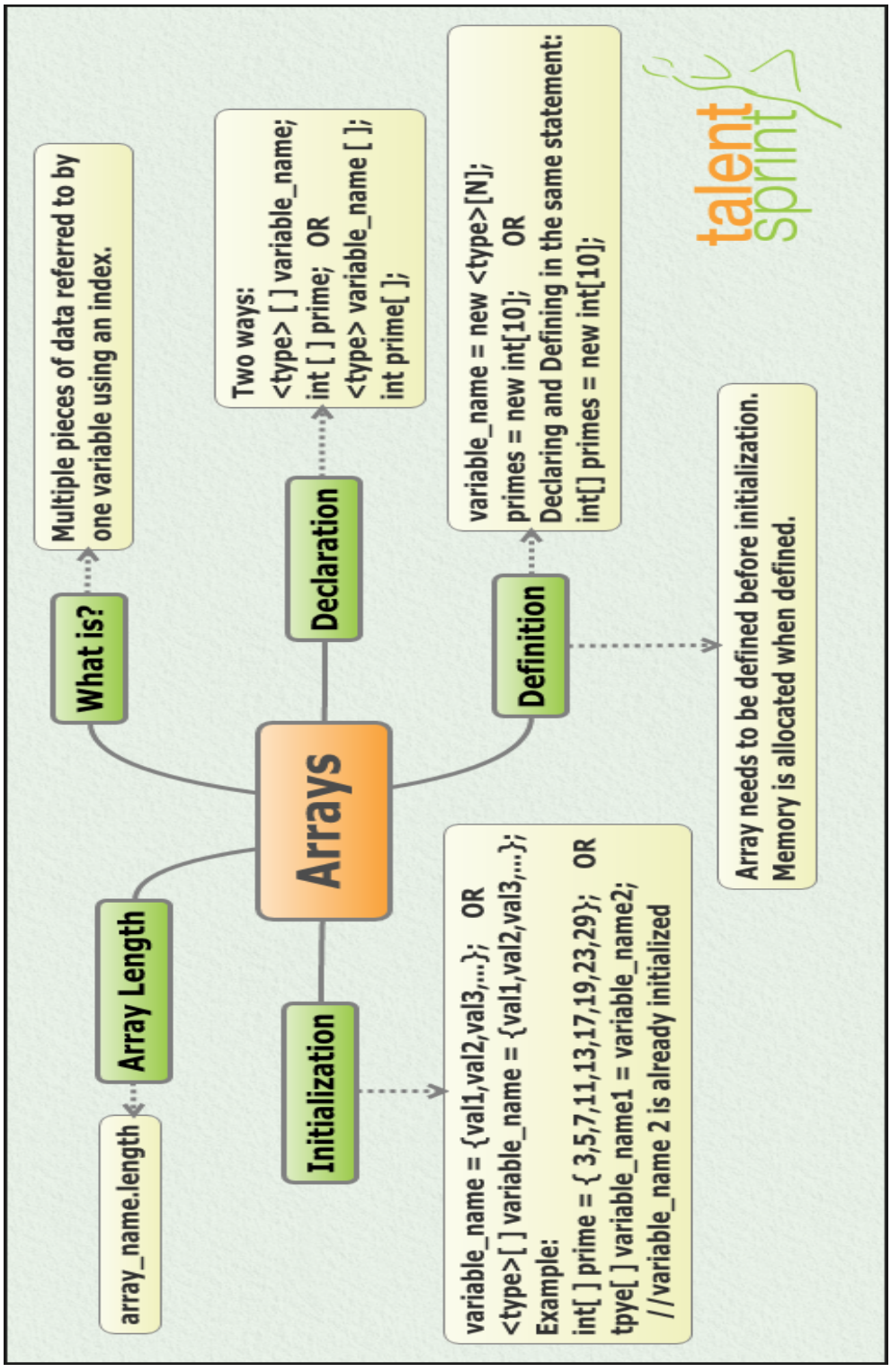
Figure 1: Platform Independency

```
Hello World
```








```
$ javac CommandLineArguments.java
$ java CommandLineArguments one two three
one
two
three
$
```

```
$ javac CommandLineNumArg.java
$ java CommandLineNumArg 12 45
12
$
```

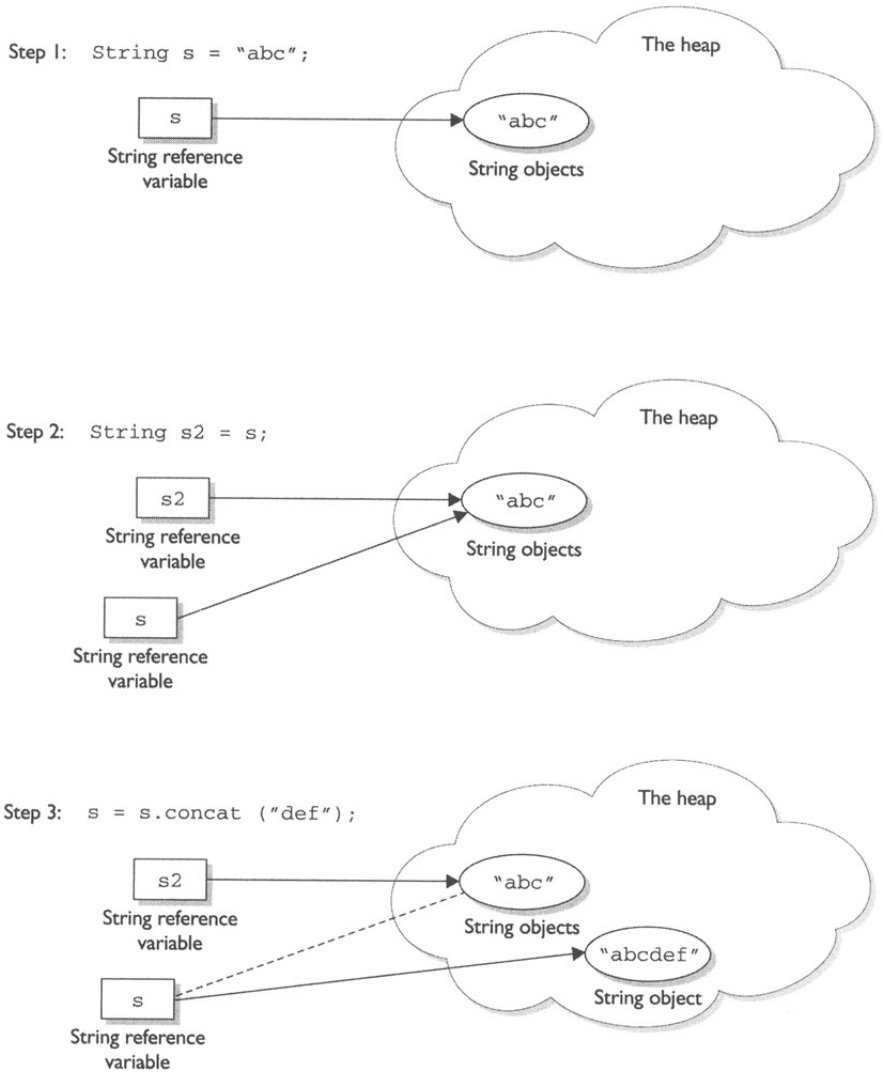



Figure 2: String objects