

Module Title: Java Programming

Part I: Introduces the fundamental concepts programming from an object-oriented perspective-Using Java.

Pre-requisite or co-requisite modules

Programming Using C, Data Structures Using C++ and Analysis of Algorithms

Chapter I : Introduction to Java Language**Lecturer Info:**

Mr. Silas MAJYAMBERE

Msc in Computer Science

University of Rwanda

College of Science and Technology

School of ICT

Computer Science

Email: smajyambere@ur.ac.rw

Tel : +250783437123

Pillar of Success

- Work as a team
- Interaction Lecturer-student and vice versa
- To have rhythm

Description and Module objectives

Module consists of two parts of Object-Oriented Programming using Java

#1. Introduces the fundamental concepts programming from an object-oriented perspective. Through the study of object design, this course also introduces the basics of human computer interfaces, graphics, and the social implications of computing, along with significant coverage of software engineering

#2. Introduces the advanced programming concepts for networking, databases, servers and web transactions.

Part I (Java Core) Target or objectives:

1. Basic Concept: JSDK (Java IDE) installation, Software development cycle and First application using Java programming language;
2. Control Flow statements(looping, decision making, exception handling, branching);
3. Methods or functions, Arrays and Available classes in JDK and its packages or libraries;
4. Introduction to OOP(Object Oriented Programming) : What is a Class, what is an Object, How to Create an Object and How to Use an Object;
5. OOP (Object Oriented Programming) and Concepts: Inheritance, Interface, Polymorphism,

HISTORY OF JAVA PROGRAMMING:

Java is one of today's most popular software development languages.

Java was developed by Sun Microsystems.Originally Sun proposed an internal corporate research project in 1991 named Green, with the idea of dealing with

Microprocessor profound impact in intelligent consumer-electronic devices.

The project resulted in a C++ based language that called Oak, then Oak became Java when a group of Sun people visited a local coffee shop.

Sun formally announced Java at an industry conference in May 1995.

Java, Language of Choice for Networked Applications:

Java has become the language of choice for implementing Internet-based applications and software for devices that communicate over a network.

Stereos and other devices in homes are now being networked together by java Technology.

Java is also used in a world wide web technologies, used to develop large-scale enterprise applications, in web servers, to provide applications for consumer devices(e.g.,cellphones,pagers and personal digital assistants),...

Note: Java has grown so large that it has two other editions:

The Java Enterprise Edition (Java EE)

Is geared toward (is suitable) developing large-scale, distributed networking applications and web based applications.

The Java Micro Edition (Java ME)

Is geared toward developing applications for small, memory-constrained devices, such as cell phones, pagers and PDAs.

With java we are able to develop:

- ❖ Applications: software in form of windows (graphical) or console;
- ❖ Applets : java applications used in web based environment;
- ❖ Applications for mobile devices, using J2ME;

Java Official Website:

<http://java.sun.com/>

Introduction to Programming Languages (JAVA)

Terminologies and concepts:

Programming art is an art among arts.

What is programming art?

Programming is nothing than solve problems

(Issues in our everyday life) by using Computers.

Programming language: Is a language used to communicate to the Computer.

A Software /Program: a set of instructions that a programmer (like you) writes to command computers to perform actions and make decisions.

Or, Is a series of instructions to tell the computer to accomplish a given task (those instructions are written in a programming language).

What is a Computer?

A computer is a device capable of performing computations and making logical decisions at speeds millions (even billions) of times faster than human being can. For example, many of today's personal computers can perform several billion calculations in one second.

Note that ;A computer is a non intelligent electronic device; therefore it solves problems through algorithms (step by step method).

Machine Languages, Assembly Languages and High-level Languages:

Programmers write instructions in various programming languages, some directly understandable by computers and others requiring intermediate translation steps.

Hundreds of computer languages are in use today.

These may be divided into three general types:

1. Machine languages
2. Assembly languages
3. High-level languages

A) Machine language

Is the “natural language” of a computer and a such is defined by its hardware design. Any computer can directly understand only its own **machine language**.

Machine languages generally consist of strings of numbers (ultimately reduced to 1s and 0s) that instruct computers to perform their most elementary operations one at a time. (Not understandable for a human being)

e.g.: + 1300042774

+ 1400593419

+ 1200274027

B) Assembly Language:

Machine language programming was simply too slow and tedious for most programmers. Instead of using the string of numbers that computers could directly understand, programmers began using *English-like abbreviations* to represent elementary operations.

We call this type of language: *Assembly language*. Translator programs called *assemblers*, used to convert assembly-language programs to machine language at computer speeds.

e.g.:

```
load basePay  
add overpay  
store grossPay
```

(This is the same code like the one we've seen above in Machine language).

The assembly language is clearer to humans, but incomprehensible to computers until translated to machine language.

C) High-level Language:

Computer usage increased rapidly with the advent of assembly languages, but programmers still had to use many instructions to accomplish even the simplest tasks.

To speed the programming process, high-level languages were developed in which single statements could be written to accomplish substantial tasks.

Translator programs (for High-level languages) called “Compilers”

to convert high-level language programs into machine language.

Example of instruction in high-level language:

GrossPay=basePay+overTimePay

Note: High-level languages are preferable to machine and assembly languages from the programmer's standpoint.

Example of some High-level languages:

Java, C++, Microsoft's .NET languages (e.g. Visual Basic.NET, Visual C++.NET and C#)

Note: Java is the most widely used.

About the Java Technology

Java, Language of Choice for Networked Applications:

Java has become the language of choice for implementing Internet-based applications and software for devices that communicate over a network.

Stereos and other devices in homes are now being networked together by java Technology.

Java is also used in a world wide web technologies, used to develop large-scale enterprise applications, in web servers, to provide applications for consumer devices(e.g., cell phones, pagers and personal digital assistants),...

- ✓ Java technology is both a programming language and a platform.

Installing Java Environment

Java standard Edition Development Kit (JDK)

What _____ is _____ a _____ JDK?

JDK stands for Java Development Kit, and is what you need to develop and run Java applications (it contains a compiler, the JVM - which runs Java applications and several tools such as javadocs, jar etc).

- ✓ Before you can run or build java applications, you must install the java Standard Edition Development Kit (JDK).Or Java development tool that Supports java SE.

You can download the JDK and its documentation from: java.sun.com

Click the download button for JDK (version) for example, JDK6.

You must accept the licence agreement before downloading.

After downloading the JDK installer, double click the installer program to begin installing the JDK.

We recommend that you accept all the default installation options.

On windows, the JDK is placed in the following directory

By default: C:\Program Files\Java\jdk (version)

SETTING the PATH Environment Variable:

The path environment variable on your computer designates which directories the computer searches when looking for applications, such as the applications that enable you to compile and run your java applications (javac.exe and java.exe).

- ✓ See how to set PATH environment variable on one's computer to indicate where the JDK's tools are installed (Live Demo).

SETTING the CLASSPATH Environment Variable:

If you attempt to run a java program and receive a message like:

Exception in thread "main" java.lang.NoClassDefFoundError: className

Then your system has a CLASSPATH environment variable that must be modified. To fix the preceding error, follow the steps seen in setting the PATH environment variable, to locate the CLASSPATH variable, then edit the variable's value to include `.;` at the beginning of its value (with no spaces before or after these characters).

Strength of Java

- ✓ Simple to use
- ✓ Object Oriented
- ✓ Distributed(Networked applications)
- ✓ Dynamic
- ✓ Well structured
- ✓ Portable(JVM)
- ✓ High performance
- ✓ Robust
- ✓ Secure

Weakness of Java:

- ✓ Java technology consumes a huge memory (computer main memory)

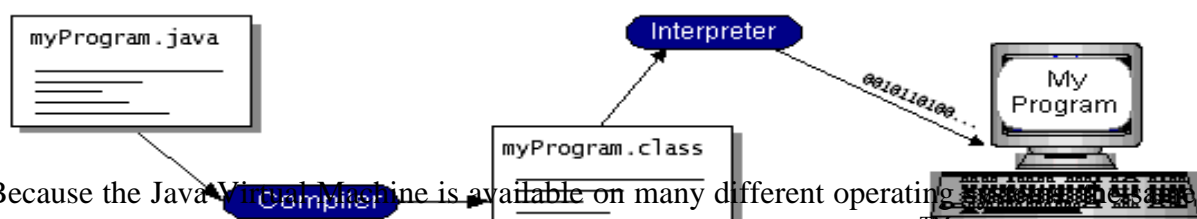
Other Advantages of Java programming language:

- **Get started quickly:** Although the Java programming language is a powerful object-oriented language, it's easy to learn, especially for programmers already familiar with C or C++.
- **Write less code:** Comparisons of program metrics (class counts, method counts, and so on) suggest that a program written in the Java programming language can be four times smaller than the same program in C++.
- **Write better code:** The Java programming language encourages good coding practices, and its garbage collection helps you avoid memory leaks. Its object orientation, its JavaBeans component architecture, and its wide-ranging, easily extendible API let you reuse other people's tested code and introduce fewer bugs.
- **Develop programs more quickly:** Your development time may be as much as twice as fast versus writing the same program in C++. Why? You write fewer lines of code and it is a simpler programming language than C++.

- **Avoid platform dependencies:** You can keep your program portable by avoiding the use of libraries written in other languages.
- **Write once, run anywhere:** Because Java applications are compiled into machine-independent byte codes, they run consistently on any Java platform.
- **Distribute software more easily:** With Java Web Start technology, users will be able to launch your applications with a single click of the mouse. An automatic version check at startup ensures that users are always up to date with the latest version of your software. If an update is available, Java Web Start will automatically upgrade their installation.

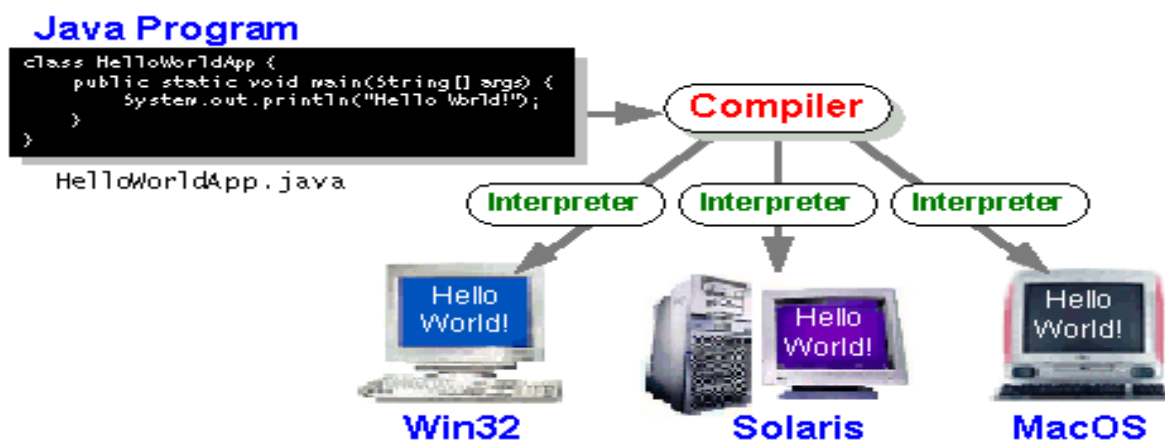
In the Java programming language, all source code is first written in plain text files ending with the **.java** extension.

Those source files are then compiled into **.class** files by the Java compiler (javac). A **.class** file does not contain code that is native to your processor; it instead contains *bytecodes*-- the machine language of the Java Virtual Machine. The Java launcher tool (java) then runs your application with an instance of the Java Virtual Machine.



Because the Java Virtual Machine is available on many different operating systems, **.class** files are capable of running on Microsoft Windows, the Solaris™ Operating System (Solaris OS), Linux, or MacOS. Some virtual machines, such as the [Java Hotspot Virtual Machine](#), perform additional steps at runtime to give your application a performance boost.

This includes various tasks such as finding performance bottlenecks and recompiling (to native code) frequently-used sections of your code.



A *platform* is the hardware or software environment in which a program runs. Here is a list of some of the most popular platforms : Microsoft Windows, Linux, Solaris OS, and MacOS. Most platforms can be described as a combination of the operating system and underlying hardware.

The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.

The Java platform has two components:

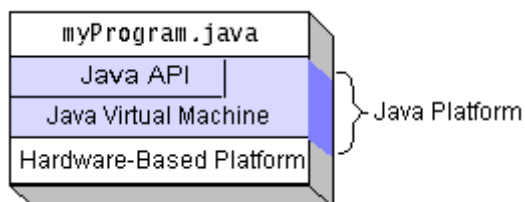
- The *Java Virtual Machine*
- The *Java Application Programming Interface (API)*

Java Virtual Machine. It's the base for the Java platform and is ported onto various hardware-based platforms.

The API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets.

It is grouped into libraries of related classes and interfaces; these libraries are known as *packages*.

The following figure depicts how the API and the Java Virtual Machine insulate the program from the hardware.



To program in Java, we need:

- IDE (Integrated Development Environment): tool that provides everything you'll need for compiling, running, monitoring, debugging, and documenting your applications. As a new developer, the main tools you'll be using are the Java compiler (javac), the Java launcher (java), and the Java documentation tool (javadoc).

✓ JSDK (Java Software Development Kit)and ;

Or any other IDE among the following list :

✓ Eclipse IDE

To download Eclipse IDE, go to: <http://www.eclipse.org/>

SCHOOL OF ICT – COMPUTER SCIENCE DEPARTMENT

MUSANZE Java Programming Part I Level II Computer Science

- JBuilder: Advanced IDE used to develop big projects (enterprises) and integrates more technologies such as XML, JSP/Servlet,...
- Net Beans : developed by SunMicrosystems, and has all features of a good IDE:
 - Editor ;
 - Compiler:
 - Debugger;
 - UML tool;
 - All you need about J2EE applications programming
- JCreator
- IntelliJ
- Sun ONE Studio
- JDeveloper
- JEdit
- Le Bloc-notes de Windows (vi sous Linux), ou encore Notepad++

What we do need to understand:

- ✓ JVM(Java Virtual Machine) is the key in Java programming,
- ✓ JVM ,interprets and read or execute java byte code(java compiled code),
- ✓ Java source code file has a .java extension or format,
- ✓ Java compiled file has a .class extension or format,
- ✓ Byte code is an intermediary code between source code and machine code,
- ✓ Java program edited in windows can be compiled under Mac OS and then be **executed under LINUX(JVM which lead to java portability)**,
- ✓ A computer is not able to understand byte code(Only JVM can understand),
- ✓ Every java program is made of a class (at least one), and must have (except applets) a main method or function.JVM need to use this method to run your program.

Note: A function or method is series of instructions to be executed, and these instructions are grouped together.

Parts of a method or function:

- Header(data type name and parameters)
- Body(what a function or method will do or instructions to be executed)
- Return value(result or output of our method)

Characteristics of Java Programming Language

Features of Java are as follows:

1. Compiled and Interpreted
2. Platform Independent and portable
3. Object- oriented
4. Robust and secure
5. Distributed
6. Familiar, simple and small
7. Multithreaded and Interactive
8. High performance
9. Dynamic and Extensible

1. Compiled and Interpreted

Basically a computer language is either compiled or interpreted. Java comes together both these approach thus making Java a two-stage system. Java compiler translates Java code to Bytecode instructions and Java Interpreter generate machine code that can be directly executed by machine that is running the Java program.

2. Platform Independent and portable

Java supports the feature portability. Java programs can be easily moved from one computer system to another and anywhere. Changes and upgrades in operating systems, processors and system resources will not force any alteration in Java programs. This is reason why Java has become a trendy language for programming on Internet which interconnects different kind of systems worldwide. Java certifies portability in two ways. First way is, Java compiler generates the bytecode and that can be executed on any machine. Second way is, size of primitive data types are machine independent.

3. Object- oriented

Java is truly object-oriented language. In Java, almost everything is an Object. All program code and data exist in objects and classes. Java comes with an extensive set of classes; organize in packages that can be used in program by Inheritance. The object model in Java is trouble-free and easy to enlarge.

4. Robust and secure

Java is a most strong language which provides many securities to make certain reliable code. It is design as garbage – collected language, which helps the programmers virtually from all memory management problems. Java also includes the concept of exception handling, which detain serious errors and reduces all kind of threat of crashing the system. Security is an important feature of Java and this is the strong reason that programmer use this language for programming on Internet. The absence of pointers in Java ensures that programs cannot get right of entry to memory location without proper approval.

5. Distributed

Java is called as Distributed language for construct applications on networks which can contribute both data and programs. Java applications can open and access remote objects on Internet easily. That means multiple programmers at multiple

remote locations to work together on single task.

6. Simple and small

Java is very small and simple language. Java does not use pointer and header files, goto statements, etc. It eliminates operator overloading and multiple inheritance.

7. Multithreaded and Interactive

Multithreaded means managing multiple tasks simultaneously. Java maintains multithreaded programs. That means we need not wait for the application to complete one task before starting next task. This feature is helpful for graphic applications.

8. High performance

Java performance is very extraordinary for an interpreted language, majorly due to the use of intermediate bytecode. Java architecture is also designed to reduce overheads during runtime. The incorporation of multithreading improves the execution speed of program.

9. Dynamic and Extensible

Java is also dynamic language. Java is capable of dynamically linking in new class, libraries, methods and objects. Java can also establish the type of class through the query building it possible to either dynamically link or abort the program, depending on the reply. Java program is support functions written in other language such as C and C++, known as native methods.

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 text, 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 *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 launcher (java) uses the Java Virtual Machine to run your application.



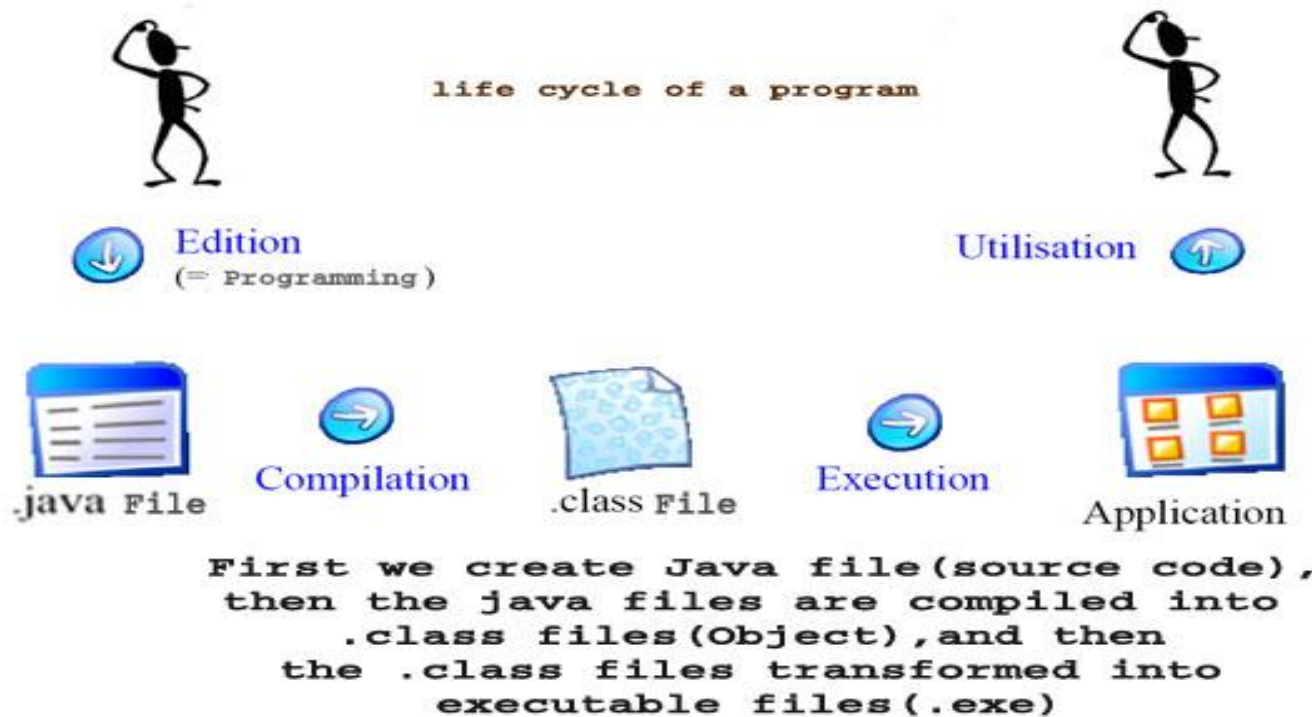
Programming is to communicate to the computer

To program, we use a programming language;
for example, Java programming language



The aim of Programming is to produce applications
Those applications are used by computer users

How to write a program



A First Program in Java: Printing a Line of Text

Helloworld.java

```
//Java Hello world Program
```

```
//Helloworld.java
```

```
public class HelloWorld {
```

```
    //main method begins execution of java application
```

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

```
        System.out.println("Hello World!\nWelcome to Java Programming");
```

```
    } //end of main method
```

```
} //end of class HelloWorld
```

PROGRAM'S EXPLANATIONS

- We created a class named "HelloWorld" containing a simple main function within it. The keyword *class* specifies that we are *defining a class*.
- Every java program begins its execution within the method named *main()*.
- Main method that gets executed has the following signature: *public static void main(String args[])*.
- Declaring this method as public means that it is accessible from outside the class so that the JVM can find it when it looks for the program to start it.
- It is necessary that the method is declared with return type void (i.e. no arguments are returned from the method).
- The main method contains a String argument array that can contain the command line arguments.
- The brackets { and } mark the beginning and ending of the class.
- The program contains a line 'System.out.println("Hello World");' that tells the computer to print out on one line of text namely 'Hello World'. The semi-colon ';' ends the line of code.
- The double slashes '/' are used for comments that can be used to describe what a source code is doing. Everything to the right of the slashes on the same line does not get compiled, as they are simply the comments in a program.

Java	Main	method	Declarations
class	MainExample1	{public static void main(String[] args)	{}}
class	MainExample2	{public static void main(String []args)	{}}
class	MainExample3	{public static void main(String args[])	{}}

Compiling and Running an Application

To compile and run the program you need the JDK distributed by Sun Microsystems. The JDK contains documentation, examples, installation instructions, class libraries and packages, and tools.

Download an editor like Textpad/EditPlus to type your code. You must save your source code with a .java extension.

SCHOOL OF ICT – COMPUTER SCIENCE DEPARTMENT

MUSANZE Java Programming Part I Level II Computer Science

Steps for Saving, compiling and Running a Java

Step 1: Save the program With .java Extension.
Step 2: Compile the file from DOS prompt by typing `javac <filename>.java`
Step 3: Successful Compilation, results in creation of .class containing byte code
Step 4: Execute the file by typing `java <filename without extension>`

Note:

- ✓ Java is case sensitive language(Pay attention while coding; for java capital letter is different from lower letter)
- ✓ Every program in Java consists of at least one class declaration that is defined by you-the programmer. These are known as *programmer-defined classes* or *user-defined classes*.
- ✓ By convention, all class names in java begin with a capital letter and capitalize the first letter of each word they include(for example, HelloWorld)
- ✓ Java programs normally go through three phases: *Create, Compile and Run*

Phase 1: Creating a Program

A Java program is nothing more than a sequence of characters, like a paragraph or a poem, stored in a file with a .java extension. To create one, therefore, you need only define that sequence of characters, in the same way as you do for email or any other computer application. You can use any text editor for this task, or you can use one of the more sophisticated program development environments described on the booksite. Such environments are overkill for the sorts of programs we consider in this book, but they are not difficult to use, have many useful features, and are widely used by professionals. The .java file we create here is called Java source code.

Phase 2: Compiling a Java Program

At first, it might seem that Java is designed to be best understood by the computer. To the contrary, the language is designed to be best understood by the programmer (that's you). The computer's language is far more primitive than Java. A compiler is an application that translates a program from the Java language to a language more suitable for executing on the computer. The compiler takes a file with a .java extension as input (your program) and produces a file with the same name but with a .class extension (the computer-language version). To use your Java compiler, type in a terminal window the `javac` command followed by the file name of the program you want to compile.

Example: `javac Helloworld.java`

If the program compiles, the compiler produces a .class file called Helloworld.class that contains the compiled version(bytecodes) of the program. The .class file is called Java

Byte code and contains the binary information of the java program that can be interpreted by the computer using Java Virtual Machine (JVM).

Phase 3: Executing or Running a Program

Once you compile the program, you can run it. This is the exciting part, where your program takes control of your computer (within the constraints of what the Java system allows). It is perhaps more accurate to say that your computer follows your instructions. It is even more accurate to say that a part of the Java system known as the Java Virtual Machine (the JVM, for short) directs your computer to follow your instructions. To use the JVM to execute your program, type the java command followed by the program name in a terminal window.

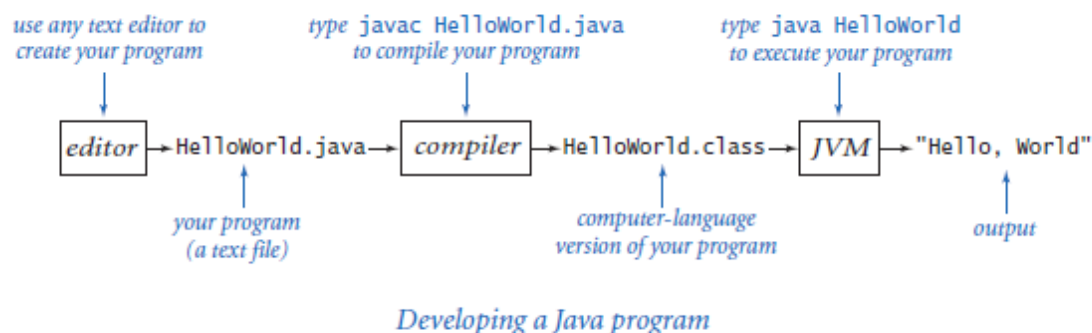


Fig. Process of Creating, Compiling and Executing a Java Program

Problems That May Occur at Execution Time (**Runtime errors**)

Errors like division by zero occur as a program runs, so they are called *runtime errors* or *execution-time errors*.

Basic Language Elements

(Identifiers, keywords, literals, white spaces and comments)

Once you get these basic language concepts you can continue with the other object oriented programming language concepts.

Variable, Identifiers and Data Types

- ✓ **Variables** are cases or boxes inside main memory (RAM) used to store data during program execution.
- ✓ All variables have a **name**, a **data type**, and a **scope**. The programmer assigns the names to variables, known as **identifiers**.
- ✓ Variables have a **data type** that indicates the kind of value they can store.
- ✓ Variables declared inside of a block or methods are said to be **local variables**; they are not automatically initialized. The compiler will generate an error as a result of the attempt to access the local variables before a value has been assigned.

```
public class localVariableEx {
    public static int a;
    public static void main(String[] args) {
        int b;
        System.out.println("a : "+a);
        System.out.println("b : "+b);    //Compilation error
    }
}
```

Note in the above example, a compilation error results in where the variable is tried to be accessed and it was declared without assigned a value.

NB: the (+) in the above program means concatenation.

Variable Declarations

- ✓ There are three ways of variable declarations
 - ❖ data type variableName; // **declaration of a variable**
- int a;
- ❖ data type variableName=value; // **initialization of a variable**
- int a=5;
- ❖ data type variableName=expression; // **dynamic initialization**
- int a=b+c;

Operators

- o An operator performs a function on one, two, or more operands.
- o Operators are divided into the following categories:

_ Arithmetic Operators

- +, -, *, /, %

_ Increment and Decrement Operators

- ++, --

_ Relational Operators

- >, <, >=, <=, ==, !=

_ Logical Operators

- &&, ||, !

_ Assignment Operators

- =, +=, -=, *=, /=, %=

_ Other Operators

- ?:, [], (), (type), new

Data Types

A variable's data type determines the values that the variable can contain and the operations that can be performed on it

There are only three groups of primitive data types:

_ Numeric types

· Integer Types

- byte (1 bytes)
- short (2 bytes)
- int (4 bytes)
- long (8 bytes)

· Floating – point Types

- float (4 bytes)
- double (8 bytes)

_ Character types

- char (1 byte)

_ Boolean types

- boolean (true or false)

NB: Primitive Data Types start with lower letter case but Reference Data Type starts with capital letter they are created using class name. Ex: String s; JOptionPane p;

· Control Flow Statements

1. Looping

_ for

· for(initialization;condition;increment/decrement)

_ while

· initialization;

while(condtion)

{

.....

Increment/decrement;

}

_ do .. while

· initialization;

do

{

Increment/decrement);

} while (condtion);

2. Decision Making

_ Simple if

· if (condtion)

{

true block statements;

}

_ if...else

· if (condtion)

{

true block statements;

}

else

{

false block statements;

}

_ Nested if's

· if (condtion)

{

true block statements;

}

else if

```
{
true block statements;
}
else if
{
true block statements;
}
```

```
else
{
false block statements;
}
```

3. Selection

_ switch case

· switch(choice)

```
{
case caselabel:
statements;
break;
case caselabel:
statements;
break;
```

```
.....
default:
default statements;
}
```

Simple Programs

1. /* Program for simple addition */

```
import java.io.*;
```

```
class prog1
```

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

```
int a,b,c;
```

```
a=5;
```

```
b=10;
```

```
c=a+b;
```

```
System.out.println("a+b equal c,where a=5 and b=10");
```

```
System.out.println("C="+c);
```

```
//end main method
```

```
//end class prog1
```

2. /* Program to find greatest number */

```
import java.io.*;
```

```
class prog2
```

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

```
int a,b;
```

```
a=5;
```

```
    b=10;
    if (a>b)
        System.out.println("A is Greater");
    else
        System.out.println("B is Greater");
} //end main method
```

```
} //end class prog2
```

```
3. /* Program to print 1 to 10 using for loop*/
import java.io.*;
class prog3
{
    public static void main(String args[ ])
    {
        for(int i=1;i<=10;i++)
            System.out.println("i="+i);
    } //end main
} //end class
```

```
4. /* Program to print 1 to 10 using while loop*/
import java.io.*;
class prog4
{
    public static void main(String args[ ])
    {
        int i=1;
        while(i<=10)
        {
            System.out.println("i");
            i++;
        } // end while loop
    } // end main
} // end class prog
```

```
5. /* Program to print 1 to 10 using do..while loop*/
import java.io.*;
class prog5
{
    public static void main(String args[ ])
    {
        int i=1;
        do
        {
            System.out.println("i="+i);
            i++;
        } while(i<=10);
    }
}
```

```
//end main  
} //end class prog5
```

```
6. /* Program to find true or false*/  
import java.io.*;  
class prog6  
{  
public static void main(String args[ ])  
{  
int a=5, b=10;  
System.out.println(a>b);  
}  
} //end of prog6
```

```
7. /* Program for type conversion*/  
import java.io.*;  
class prog7  
{  
public static void main(String args[ ])  
{  
int a=5, b;  
double c=353.769, d;  
d=a;  
System.out.println("Int to Double(Automatic)=" + d);  
b=(int) c;  
System.out.println("Double to Int(External)=" + b);  
} // end main  
} //end class prog7
```

```
8. /* Program for Min, Max value of data types*/  
import java.io.*;  
class prog8  
{  
public static void main(String args[ ])  
{  
System.out.println("Min value of byte =" + Byte.MIN_VALUE);  
System.out.println("Max value of byt =" + Byte.MAX_VALUE);  
System.out.println("Min value of Short =" + Short.MIN_VALUE);  
System.out.println("Max value of Short =" + Short.MAX_VALUE);  
System.out.println("Min value of Long =" + Long.MIN_VALUE);  
System.out.println("Max value of Long =" + Long.MAX_VALUE);  
System.out.println("Min value of Int =" + Integer.MIN_VALUE);  
System.out.println("Max value of Int =" + Integer.MAX_VALUE);  
System.out.println("Min value of Float =" + Float.MIN_VALUE);  
System.out.println("Max value of Float =" + Float.MAX_VALUE);  
System.out.println("Min value of Double =" + Double.MIN_VALUE);  
System.out.println("Max value of Double =" + Double.MAX_VALUE);  
}  
}
```

Another Java Application : Adding Integers

This application reads (or input) two integers (whole numbers, like -22, 7, 0 and 1024) typed by a user at the keyboard, computes the sum of the values and displays the result.

// Addition.java

// Addition program that displays the sum of two numbers.

import java.util.Scanner;//Program uses class Scanner

public class Addition

{

//main method begins execution of Java application

public static void main(String arg[])

{

//Create Scanner to obtain input from command window

Scanner input=new Scanner(System.in);

Int number1;//first number to add

Int number2;//Second number to add

Int sum;//sum of number1 and number2

System.out.print(“Enter first integer :”);//prompt

number1= input.nextInt();//reads first number from user

System.out.print(“Enter Second integer :”);//prompt

number2= input.nextInt();//reads second number from user

sum=number1+number2;//add numbers

System.out.printf(“Sum is %d\n”,sum);//display sum

}//end method main

}//end class addition

Keywords

Keywords are reserved words that are predefined in the language (for example, in java) and have specific meaning in a programming language (for example, in java) such as to tell (inform) the compiler what the program is supposed to do.

Note:

- ✓ These Keywords cannot be used as variable names, class names, or method names.
- ✓ Keywords in java are case sensitive, all characters being lower case.

Some Java Keywords are list in the table below (Taken from Sun Java Site) :

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

Comments

Comments are descriptions that are added to a program to make code easier and clearer to understand for a human being. The compiler ignores comments and comments are used to help a programmer to provide a documentation of the program.

Java supports three comment styles.

- ✓ **Block style comments** begin with /* and terminate with */ that spans multiple lines.
- ✓ **Line style comments** begin with // and terminate at the end of the line.
- ✓ **Documentation style comments** begin with /** and terminate with */ that spans multiple lines.

Good Programming Overview Tips

- ✓ Import declaration helps the compiler to locate a class that is used in this program. (For example, **import java.util.Scanner ;**)
- ✓ A great strength of Java is its rich set of predefined classes that you can reuse rather than “reinventing the wheel”

- ✓ All import declarations must appear before the first class declaration in the file. Placing an import declaration inside a class declaration's body or after a class declaration is a syntax error.
- ✓ Forgetting to include an import declaration for a class used in your program typically results in a compilation error containing a message such as "cannot resolve symbol".
- ✓ By default, package **java.lang** is imported in every java program; thus, classes in java.lang are the only ones in the java API that do not require an import declaration.

An example of classes inside the package java.lang; **Math class**

Math class Methods
final double E final double PI double exp (doublevalue) double log (doublevalue) double pow (doublevalue) double sqrt (doublevalue) double acos (doublevalue) double asin (doublevalue) double atan (doublevalue) double cos (doublevalue) //in radians double sin (doublevalue) //in radians double tan (doublevalue) //in radians ... abs (...) // int,long,float,double ... max (...) // int,long,float,double ... min (...) // int,long,float,double ... round (...) // int from float ... round (...) // long from double

Example of some useful method in the Math class:

Math.round(6.6) --→ 7

$\text{Math.round}(6.3) \rightarrow 6$

$x^4 \rightarrow \text{Math.pow}(x, 4);$

$\sqrt{b^2 - 4ac} \rightarrow \text{Math.sqrt}(\text{Math.pow}(b, 2) - 4 * a * c);$

$E_c = \frac{1}{2}mv^2 \rightarrow 0.5 * m * \text{Math.pow}(v, 2);$

Note: Expression is the term given to formulas in programming language

Example: $\frac{y(x-3)}{x-1} \rightarrow y * (x-3) / (x-1)$

Precedence order

- ✓ Numbers are manipulated by +, -, *, /, % and () arithmetic operators
- ✓ Precedence order is :
 - ()
 - *, /, %
 - +, -

And Expressions are evaluated from left to right : **Example, (4+5)*5/2+7=29**

Variable Tips

- Declare each variable on a separate line. This format allows a descriptive comment to be easily inserted next to each declaration.
- Choosing meaningful variable names helps a program to be self-documenting.
- By convention, variable-name identifiers begin with a lowercase letter, and every word in the name after the first word begins with a capital letter.
- Variable names such as number1, number2... correspond to locations in the computer's memory. Every variable has a name, a type, a size and a value.

Methods and statements

A method is a named sequence of instructions to the computer, written out in a programming language (for example, Java).

A method is used to group together instruction and help to develop software in an easy way by dividing a task into sub tasks

The instructions are properly called **statements**, and fall into the following:

Invocation – Causing a method to be performed;

Assignment – Changing the value of a variable;

Repetition – Performing certain statements over and over again;

Selection – deciding whether to perform certain statements or not;

Exception – detecting and reacting to unusual circumstances.

Example of a program that uses output statements:

Program: Display a Warning (Display a warning message that the computer might have a virus.)

Class DisplayWarning {

/*Displaying a warning program

*-----

*by using output statements

*/

Public static void main (String args[])

{

System.out.println("-----");

System.out.println("|");

System.out.println("| WARNING!! |");

System.out.println("| Possible virus detected |");

System.out.println("| Reboot and run virus |");

System.out.println("| remover software |");

System.out.println("|");

System.out.println("-----");

}//end main

}//end method

Output and Input Using Windows (graphical objects)

Output :

```
import javax.swing.JOptionPane;//importation of JOptionPane class(or package)

public class HelloWorld2 {

    public static void main (String args []) {

        JOptionPane.showMessageDialog(null,"Hello Students!");//output in a window

        System.exit (0);//Allows you to close a program(window)

    }//end main

}//end class
```

Different output tips :

New line by **\n**:

Tab by **\t** :

Double quotes **“”** and back slash ****

```
JOptionPane.showMessageDialog (null,"a\tb\tc\nWelcome to \nMusanze"District\\North
Rwanda\\");
```

Note: **\n** , **\t** , **\”** , **** are called **escape characters**.

Input in a window: (interaction between user and application)

```
import javax.swing.JOptionPane;//importation of JOptionPane class(or package)

public class HelloWorld3 {

    public static void main (String args []) {

        // prompts the user to enter a number

        String temp=JOptionPane.showInputDialog(null,"Enter a Number: ");//Input in //a window

        //Converts the user input into integer number

        int number=Integer.parseInt(temp);

        //output a result

        JOptionPane.showMessageDialog(null,"Dear sir,The Number received from you is :
\t"+number+"\nwe Thank you!");
```

```
System.exit (0); //Allows you to close a program(window)
```

```
} //end main
```

```
} //end class
```

Declaration of Variable and Constants

The form of a variable declaration is one of the following:

```
type name;
```

```
type name1, name2, name3;
```

```
type name=value;
```

For example,

```
int temperature;
```

```
long k, m, n;
```

```
double tax=25;
```

Constant

Constant is a special variable, whose content does not change during execution process cycle.

Constant Declaration:

```
static final type name=value;
```

Examples of constant declarations are:

```
static final int speedLimit = 120;
```

```
static final double kmperMile=1.609;
```

- The declaration modifier **static** indicates a class field. Thus constant declarations can occur only at the class level, and not inside methods, even inside main
- The second modifier, **final**, indicates that the contents of the field cannot be changed during the program's execution; in other words, it is constant.

Declaring a method

A method is a group of fields and statements which is given a name and may be called upon by this name to perform a particular action. The form of a method declaration is

Method declaration
<pre> Modifiers kind name(parameters){ Fields and statements return expression; // typed methods only }</pre>

Examples:

```

static double Fahrenheit (double Celsius) {

Return 9*Celsius/5+32;

}
```

Calling or invoke a method
Object.method(parameters) – declared in another object
Classname.method(parameters) – declared static in another class
Method(parameters) – declared in this class

Example:

```
Fahrenheit (28); //to call a method to perform an action
```

Chapter 2 : INTRODUCTION TO OBJECT ORIENTED PROGRAMMING

We have already understood that every Java application or program is consists of one class(at least).

If you become part of a development team in industry, you might work on applications that contain hundreds or even thousands of classes.

Classes, Objects, Methods and Instance Variables

In java we begin by creating a program unit called a class to house a method (just as a car's engineering drawings house the design of an accelerator pedal).In a class, you provide one or more methods that are designed to perform the class's tasks.

For example, a class that represents a bank account might contain one method to deposit money to an account, an other to withdraw money from an account and a third to inquire what the current balance is.

Just as you can not drive an engineering drawing of a car, you can not “drive” a class.

Just as someone has to build a car from its engineering drawings before you can actually drive a car, you must build an object of a class before you can get a program to perform the tasks the class describes how to do.

A class’s object has attributes. For example, a bank account object has a balance attribute that represents the amount of money in the account.

Attributes are specified by the class’s **instance variables**.

✓ **Briefly**, a class is nothing than a blueprint or a structure of something;

And class object is the object produced by using that structure or blueprint

The remainder of this chapter presents examples that demonstrate the concepts we introduced above.

The first four examples incrementally build a GradeBook class to demonstrate these concepts:

1. The first example presents a GradeBook class with one method that simply displays a welcome message when it is called; we then show how to create an object of that class and call the method so that it displays welcome message.
2. The second example modifies the first by allowing the method to receive a course name as an argument and by displaying the name as part of the welcome message.
3. The third example shows how to store the course name in a GradeBook object. For this version we also show how to use methods to set the course name and obtain the course name.
- 4 . The fourth example demonstrates how the data in a GradeBook object can be initialized when the object is created. The initialization is preformed by the **class’s constructor**.

// Example 1.1 GradeBook.java

//Class declaration with one method

public class GradeBook

{

//Display a welcome message to the GradeBook user

//The following method begins with the keyword public to indicate that the method can be called from outside

```
Public void displayMessage()

{

    System.out.println("Welcome to the Gradebook!");

} //end method displayMessage

} //end class GradeBook
```

Note :

- ✓ Our class above does not have the main method
- ✓ A class that contains method main is a java application
- ✓ Recall that JVM uses the main method to execute an application

//Example 1. 2 GradeBookTest.java

//Create GradeBook Object and call its displayMessage method.

```
Public class GradeBookTest{
```

```
//main method begins program execution
```

```
    Public static void main(String args[]) {
```

```
        //Create a GradeBook objects and assigns it to myGradeBook
```

```
        GradeBook myGradeBook=new GradeBook();// Creating an object
```

```
    } //end main
```

```
} //end GradeBookTest
```

OutPut: **Wecome to the Grade Book!**

Example 2.1

// GradeBook.java improved

// Class declaration with a method that has a parameter

```
public class GradeBook
```

```
{
```



```
//Display a welcome message to the GradeBook user

Public void displayMessage(String courseName)

{

    System.out.printf("Welcome to the Gradebook for %s!\n",courseName);

} //end method displayMessage

} //end class GradeBook
```

//Example 2.2 GradeBookTest.java

//Create GradeBook object and pass a String to

//its displayMessage method.

Import java.util.Scanner;//program uses Scanner

```
public class GradeBookTest{

    //main method begins program execution

    Public static void main(String args[]) {

        //Create a Scanner object to obtain input from command window

        Scanner input=new Scanner(System.in);

        //Create a GradeBook objects and assigns it to myGradeBook

        GradeBook myGradeBook=new GradeBook();

        //prompt for and input course name

        System.out.println("Please enter the course name :");

        String nameOfCourse=input.nextLine();//read a line of text

        System.out.println();//outputs a blank line

        //call myGradeBook's displayMessage method

        //and pass nameOfCourse as an argument

        myGradeBook.displayMessage(nameOfCourse);

    } //end main
```

```
}//end GradeBookTest
```

OutPut: Please enter the Course name:

CS101 Introduction to Java Programming

Welcome to the grade book for

CS101 Introduction to Java Programming!

GradeBook Class with an Instance Variable, a set Method and get Method

In our next application, class GradeBook maintains the course name as an instance variable so that it can be used or modified at any time during an application's execution. The class contains three methods - **setCourseName**, **getCourseName** and **displayMessage**. Method **setCourseName** stores a course name in a GradeBook. Method **getCourseName** obtains a GradeBook's course name. Method **displayMessage**, which now specifies no parameters, still displays a welcome message that includes the course name, as you will see, the method now obtains the course name by calling another method in the same class – **getCourseName**

```
//Fig:GradeBook.java
//GradeBook class that contains a courseName instance variable
//and methods to set and get its value
Public class GradeBook
{
    Private String courseName;//course name for this GradeBook
    //method to set the course name
        public void setCourseName(String name)
        {
            courseName=name;//store the course name
        }// end of setCourseName
    public void getCourseName ()
    {
        return courseName;
    }// end of getCourseName
    public void displayMessage()
    {
        //This statement calls getCourseName to get the
        // name of the course
    }
}
```

```
        System.out.println (“welcome to the grade book    for\n%s!\n”, getCourseName());
    }// end method displayMessage

} //end Class GradeBook
```

```
//GradeBookTest.java
```

```
//Create and manipulate a GradeBook object.
```

```
import java.util.Scanner;//Program uses Scanner
```

```
public class GradeBookTest{
```

```
    //main method begins program execution
```

```
    Public static void main(String args[]) {
```

```
        //Create a Scanner object to obtain input from command window
```

```
        Scanner input=new Scanner(System.in);
```

```
        //Create a GradeBook objects and assigns it to myGradeBook
```

```
        GradeBook myGradeBook=new GradeBook();
```

```
        //Display initial values of courseName
```

```
        System.out.printf(“Initial course name is:%s\n\n”,
```

```
        myGradeBook .getCourseName());
```

```
        //prompt for and read course name
```

```
        System.out.println(“Please enter the course name :”);
```

```
        String theName=input.nextLine();//read a line of text
```

```
        myGradeBook.setCourseName(theName);//set the course name
```

```
        System.out.println();//outputs a blank line
```

```
        //display welcome message after specifying course name
```

```
        myGradeBook .displayMessage();
```

```

    }//end main

} //end GradeBookTest

```

OutPut:

Initial course name is:null

Please enter the Course name:

CS101 Introduction to Java Programming

Welcome to the grade book for

CS101 Introduction to Java Programming!

Member (or instance) variables of a class

Are variables that are available for every object which instantiates a class and are used to store information of the object.

```

public class Personne
{
    public int age;
    public String nom;
} //class Personne

public class PersonneTest
{
    public static void main(String[] args)
    {
        Personne personne1 = new Personne();
        personne1.age = 18;
        personne1.nom = "mamadou";

        Personne denis = new Personne();
        denis.nom = "denis";
        denis.age = -50;
    } // end of main()
} //class

```

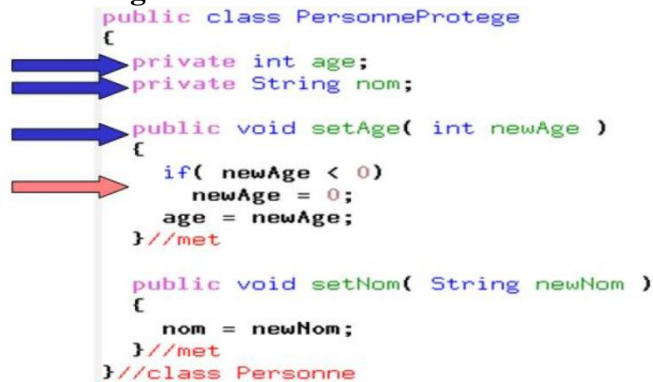
- ✓ If instance variables are declared as public (like in class `Personne`); there are then directly accessible for other classes (as a consequence, they are not protected).

Data Hiding

```
public class PersonneProtege
{
    private int age;
    private String nom;

    public void setAge( int newAge )
    {
        if( newAge < 0 )
            newAge = 0;
        age = newAge;
    } //met

    public void setNom( String newNom )
    {
        nom = newNom;
    } //met
} //class Personne
```



- ✓ To protect or hide our Object's data; we have to declare them as private.
- ✓ Here we need to declare some public methods which will be used to manipulate our data in controlled way.

```
public class PersonneTest2
{
    public static void main(String[] args)
    {
        PersonneProtege personne1 = new PersonneProtege();
        personne1.setAge( -1 );
    } // end of main()
} //class
```

Scope of a variable and keyword [This](#)

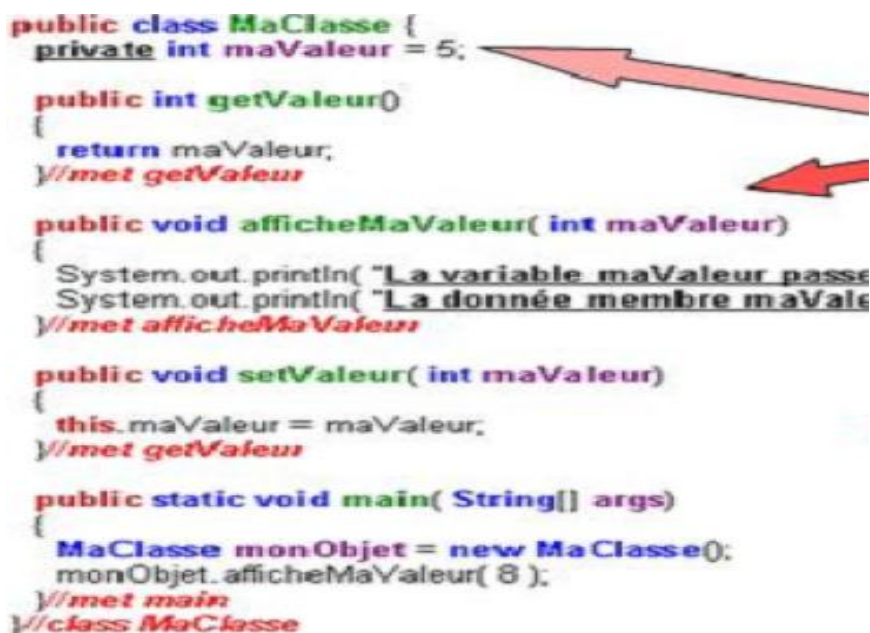
```
public class MaClasse {
    private int maValeur = 5;

    public int getValeur()
    {
        return maValeur;
    } //met getValeur

    public void afficheMaValeur( int maValeur )
    {
        System.out.println( "La variable maValeur passe" );
        System.out.println( "La donnée membre maValeur" );
    } //met afficheMaValeur

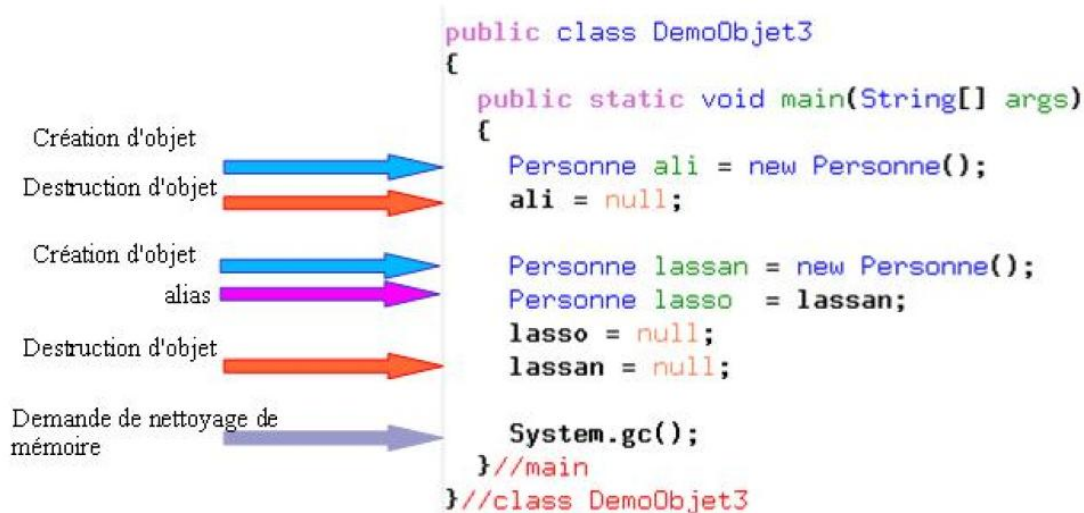
    public void setValeur( int maValeur )
    {
        this.maValeur = maValeur;
    } //met setValeur

    public static void main( String[] args )
    {
        MaClasse monObjet = new MaClasse();
        monObjet.afficheMaValeur( 8 );
    } //met main
} //class MaClasse
```



- ✓ Member variables have a global scope(visibility) within a class.

- ✓ We use keyword “**this**” to refer to a member variable if a local variable have the same name.



- ✓ In java a “**new**” operator is used to create a new object(reference) inside memory

Constructors

```

public class DemoConstructor
{
    int data = 0;

    public DemoConstructor()
    {
    } //met

    public DemoConstructor( int data )
    {
        this.data = data;
    } //met

    public static void main(String[] args)
    {
        DemoConstructor dc0 = new DemoConstructor();
        DemoConstructor dc1 = new DemoConstructor( 2 );
        System.out.println( dc0.data );
        System.out.println( dc1.data );
    } // end of main()
} //class DemoConstructor

```

0

2

- ✓ It is possible in java to define your own constructors to decide how your object is being initialized.
- ✓ Constructors are methods which do not have a return type and must have the same name as the name of the class in which they are defined.
- ✓ By default, the compiler provides a default constructor with no parameters in any class that does not explicitly include a constructor. when a class has only the default constructor, its instance variables are initialized to their default values.

OOP (Object Oriented Programming) leads to :

- ❖ Code or software reusability(**inheritance**);
- ❖ Define generic functionalities (**Polymorphism**)
- ❖ Think in terms of classes and objects(**Abstraction**)

Note: Access Modifiers **public** and **private**

Most instance variable declarations are preceded with the keyword `private` (like `public`, keyword `private` is an access modifier).

Variable or methods declared with access modifier `private` are accessible only to methods of the class in which they are declared. Thus, variable `courseName` can be used only in methods `setCourseName`, `getCoursename` and `displayMessage` of (every object of) class `GradeBook`.

- ✓ Precede every field and method declaration with an access modifier. As a rule, instance variables should be declared `private` and methods should be declared `public`. (It is also appropriate to declare certain methods `private`, if they will be accessed only by other methods of the class).
- ✓ Declaring instance variables with access modifier `private` is known as **data hiding**. When a program creates (instantiates) an object of class `GradeBook`, variable `courseName` is encapsulated (hidden) in the object and can be accessed only by methods of the object's class.
- ✓ A class's `private` fields can be manipulated only by methods of that class. So a client of an object – that is an class that calls the object's methods – calls the class's `public` methods to manipulate the `private` fields of an object of the class.

Primitive Types Vs. Reference Types

Data types in Java are divided into two categories – **Primitive types** and **Reference types** (sometimes called **nonprimitive types**)

- ✓ The primitive types are `boolean`, `byte`, `char`, `short`, `int`, `long`, `float` and `double`. A primitive variable can store exactly one value of its declared type at a time, and when another value is assigned to that variable, its initial value is replaced. Primitive type instance variables are initialized by default – variables of types `byte`, `char`, `short`, `int`, `long`, `float` and `double` are initialized to `0`, and variables of type `boolean` are initialized to `false`.
- ✓ All nonprimitive types are reference types, so classes, which specify the types of of objects, are reference types. Programs use variables of reference types (normally called references) to store the locations of objects in the computer's memory. For example the variable `myGradeBook` of `GradeBook` object contains a reference to that `GradeBook` object. **Reference-type instance variables** are initialized by default to `null` – a reserved word that represents a “reference to nothing.”

Initializing Objects with Constructors

As mentioned above, when an object is created for example, object of class `GradeBook`, its instance variable `courseName` is initialized to `null` by default. what if you want to provide a course name when you create a `GradeBook` object?

```
//Fig:GradeBook.java
//GradeBook class with a constructor to initialize the course name.
public class GradeBook
{
```



```

private String courseName;//course name for this GradeBook
//Constructor initializes courseName with String supplied as argument
public GradeBook(String name)
{
    courseName=name; //initializes courseName
} //end Constructor
//method to set the course name
    public void setCourseName(String name)
    {
        courseName=name;//store the course name
    } // end of setCourseName
// method to retrieve the course name
    public void getCourseName ()
    {
        return courseName;
    } // end of getCourseName
    public void displayMessage()
    {
        //This statement calls getCourseName to get the
        // name of the course
        System.out.println ("welcome to the grade book   for\n%s!\n", getCourseName());
    } // end method displayMessage

} //end Class GradeBook

```

//Fig: GradeBookTest.java

//GradeBook constructor used to specify the course name at the time each //GradeBook object is created

```

public class GradeBookTest
{
    //main method begins program execution
    public static void main(String args[]){
        //create GradeBook object
        GradeBook gradeBook1=new GradeBook("CS101 Introduction to Java
Programming")
        GradeBook gradeBook2=new GradeBook("CS102 Data Structure in Java")
        //display initial value of courseName for each GradeBook
        System.out.printf("gradeBook1 course name is:%s\n",gradeBook1.getCourseName());
        System.out.printf("gradeBook2 course name is:%s\n",gradeBook2.getCourseName());

        } //end main
}

```

How to design user defined classes(Overview)

Class

A class is a collection of objects. A description of a group of objects with same properties. There are two types of elements of a class:

- A data member is an object of any type

- Member functions (methods) to access or operate on those members

Defining Classes

A class definition is always introduced by the reserved word **class**. The name of the class is indicated after the word **class**.

- The data or variables defined within a class are called instance variables.
- The methods and variables defines within a class are called members.

A Simple Class

A class called Box that defines three instance variables: width, height and depth.

```
class Box // defining class
{
double width; //
double height; // // instance variables
double depth; //
}
```

Objects

Objects are the runtime or real time entities. Each object has its own memory.

Each object has a type.

type name= class name();

Declaring Objects

The **new** operator dynamically allocates (that allocates at run time) memory for an object. Java supports the objects which are created by **new**, to destroy automatically and free the memory.

To declare an object of type Box:

```
Box mybox=new Box(); // allocate a Box object
```

The class program

```
/* Program for class Box */
import java.io.*;
class Box
{
int width;
int height;
int depth;
}
class BoxProgram
{
public static void main (String args[ ])
{
Box mybox=new Box();
mybox.width=5;
mybox.height=10;
mybox.depth=15;
System.out.println("Width="+mybox.width);
```

```
System.out.println("Height="+mybox.height);
System.out.println("Depth="+mybox.depth);
}
}
```

Output:

```
Width=5
Height=10
Depth=15
```

Methods

Instead of functions, in Java it's known as methods. A method can be called only for an object and that objects must able to perform that method call. The methods of an object called by following a (.) dot
objectname.method();

Adding method to the Box class

Methods can be defined in the class itself. A method can be called any number of times. The following program adds a method to the Box class to print the output.

```
/* Program for class Box with methods */
import java.io.*;
class Box
{
    int width;
    int height;
    int depth;
    void show()
    {
        System.out.println("Width="+width);
        System.out.println("Height="+height);
        System.out.println("Depth="+depth);
    }
}
class BoxMethods
{
    public static void main (String args[ ])
    {
        Box mybox=new Box();
        mybox.width=5;
        mybox.height=10;
        mybox.depth=15;
        mybox.show();

    }
}
//end main
} //end Class
```

Output:

```
Width=5
Height=10
Depth=15
```

Method with Parameters

Methods can be defined along with the parameters. Parameters allow a method to pass the values from the calling method. The following program adds a method that takes parameters.

```
/* Program for class Box with methods and parameters */
```

```
import java.io.*;
class Box
{
    int width;
    int height;
    int depth;
    void show()
    {
        System.out.println("Width="+width);
        System.out.println("Height="+height);
        System.out.println("Depth="+depth);
    }
}
```

```
void setvalue(int w, int h, int d)
{
    width=w;
    height=h;
    depth=d;
}
}
```

```
class BoxParams
{
    public static void main (String args[ ])
    {
        Box mybox=new Box();
        mybox.setvalue (1, 2, 3);
        mybox.show();
        mybox.setvalue (5, 10, 15);
        mybox.show();
    } //end main
} // end class BoxParams
```

Output:

```
Width=1
Height=2
Depth=3
Width=5
Height=10
Depth=15
```

Overloading Methods

Defining two or more methods within the same class that shares the same name as long as their parameter declarations are different. The following program uses method overloading to overload three types of parameter sets.

```
/* Program for class Box with method overloading */
```

```
import java.io.*;
class Box
{
    int width;
    int height;
    int depth;
    void show()
    {
        System.out.println("Width="+width);
        System.out.println("Height="+height);
        System.out.println("Depth="+depth);
    } // end show
}
```

```
void setvalue(int w)
{
    width=height=depth=w;
}
```

```
void setvalue(int w, int h)
```

```
void setvalue(int w, String h)
{
    width=w;
    height=h;
    depth=30;
}
```

```
void setvalue(int w, int h, int d)
{
    width=w;
    height=h;
    depth=d;
}
}
```

```
class BoxMethodOverload
{
    public static void main (String args[ ])
    {
        Box mybox=new Box();
        mybox.setvalue(10);
        mybox.show();
        mybox.setvalue(10,20);
        mybox.show();
        mybox.setvalue(100,200,300);
        mybox.show();
    }
}
```

Output:

```
Width=10
Height=10
Depth=10
Width=10
Height=20
Depth=30
Width=100
Height=200
Depth=300
```

Constructors

Constructors are (special methods) used to initialize all the variables in the class, immediately upon the creation of object. It has the same name as class. Constructors have no type, but it can take parameters.

```
classname()
{
// constructing the class
}
```

The following program uses a constructor to initialize the dimensions of class Box

```
/* Program for class Box with constructors */
import java.io.*;
class Box
{
int width;
int height;
int depth;
void show()
{
System.out.println("Width="+width);
System.out.println("Height="+height);
System.out.println("Depth="+depth);
}
Box()
{
width=10;
height=20;
depth=30;
}
}
class BoxCons
{
public static void main (String args[ ])
{
Box mybox=new Box();
mybox.show();
}
}
```

Output:

Width =10

Height=20

Depth=30

Parameterized Constructors

Parameterized constructors are used to set the different values of class members which is specified by those parameters. The following program shows how parameterized parameters are used to set the dimension of class Box

/ Program for class Box with constructors and parameters */*

```
import java.io.*;
class Box
{
    int width;
    int height;
    int depth;
    void show()
    {
        System.out.println("Width="+width);
        System.out.println("Height="+height);
        System.out.println("Depth="+depth);
    }

    Box( int w, int h, int d)
    {
        width=w;
        height=h;
        depth=d;
    }
}
```

```
class BoxCons
{
    public static void main (String args[ ])
    {
        Box mybox=new Box (1, 2, 3);
        Box yourbox=new Box (10, 20, 30);
        mybox.show();
        yourbox.show();
    } //end main
} //end BoxCons
```

Output:

Width =1

Height=2

Depth=3

Width =10

Height=20

Depth=30

Overloading Constructors

The same as method overloading, constructors also overloaded with different set of parameters. The following program shows the constructor overloading.

/* Program for class Box with constructors overloading */

```
import java.io.*;
class Box
{
    int width;
    int height;
    int depth;
    void show()
    {
        System.out.println("Width="+width);
        System.out.println("Height="+height);
        System.out.println("Depth="+depth);
    }
    Box()
    {
        width=10;
        height=10;
        depth=10;
    }
    Box(int w)
    {
        width=height=depth=w;
    }
    Box( int w, int h)
    {
        width=w;
        height=h;
        depth=100;
    }
    Box( int w, int h, int d)
    {
        width=w;
        height=h;
        depth=d;
    }
}

class BoxCons
{
    public static void main (String args[ ])
    {
        Box mybox1=new Box();
        Box mybox2=new Box(50);
        Box mybox3=new Box(100,100);
        Box mybox4=new Box(100,200,300);
    }
}
```

```

        mybox1.show();
        mybox2.show();
        mybox3.show();
        mybox4.show();
    } // end main
} // end class

```

Output:

```

Width =10
Height=10
Depth=10
Width =50
Height=50
Depth=50
Width =100
Height=100
Depth=100
Width =100
Height=200
Depth=300

```

Static Methods and Variables

Understanding Static Methods and Variables

It is possible to create a member that can be used by itself. To create such a member **static** is used. When a member is declared as static, it can be accessed before any objects of its class are created.

The most common example of a static member is **main()**. **main()** is declared as static because it must be called before any objects exists.

Variables declared as static are like global variables. When objects of class are declared, the instances of the class share the same name static variable.

The following program shows how static variables and methods are declared.

```

/
/ demonstrate static variables, methods, and blocks.
import java.io.*;
class UseStatic
{
    static int a = 10;
    static int b;
    static void show()
    {
        System.out.println("a = " + a);
        System.out.println("b = " + b);
    }

    static
    {
        System.out.println("Static block initialized.");
        b = 20;
    }
}

```



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

Output:

a=10

b=20

As soon as the UseStatic class is loaded, all the static statements are run. First a is set to 10, then the static block executes, and b is set to 20. Then main() is called, which calls the method show(), it prints the value for a and b.

Using Static outside of the class

- To call a static method from outside its class, classname.method() can be used.
- To call a static member from outside its class, classname.variable can be used.

The following program shows how the static method and static variables called outside of their class.

```
/  
/demonstrate static methods and variables called outside from the class  
import java.io.*;  
class StaticDemo  
{  
    static int a = 33;  
    static int b = 99;  
    static void callme()  
    {  
        System.out.println("a = " + a);  
    }  
}
```

```
class StaticProg  
{  
    public static void main(String args[])  
    {  
        StaticDemo.callme();  
        System.out.println("b = " + StaticDemo.b);  
    }  
}
```

Output:

a=33

b=99

Characters and Strings

Characters

The data type used to store characters is **char**. In Java the range of char is 0 to 65,536. The standard set of characters known as ASCII ranges 0-127 and extended character set ranges from 0 – 255.

A simple program for char variables:

```
/* program for char demo*/
import java.io.*;
class chardemo
{
    public static void main(String args[])
    {
        char c=0;
        for(int i=0;i<=255;i++)
        {
            System.out.println("ASCII value for "+c+" is "+(int) c);
            c++;
        }
    }
}
```

This program will print the ASCII values from 0 – 255.

Static Methods of Character

Character includes several static methods that categorize characters and alter their case. The following program demonstrates these methods.

```
/
/ Demonstrate several Is... methods.
import java.io.*;
class IsDemo
{
    public static void main(String args[])
    {
        char a[] = {'a', 'b', '5', '?', 'A', ' '};
        for(int i=0; i<a.length; i++) {
            if(Character.isDigit(a[i]))
                System.out.println(a[i] + " is a digit.");
            if(Character.isLetter(a[i]))
                System.out.println(a[i] + " is a letter.");
            if(Character.isWhitespace(a[i]))
                System.out.println(a[i] + " is whitespace.");
            if(Character.isUpperCase(a[i]))
                System.out.println(a[i] + " is uppercase.");
            if(Character.isLowerCase(a[i]))
                System.out.println(a[i] + " is lowercase.");
        } //end main
    }
}
```

```
}

```

Output:

```
a is letter
a is lowercase
b is letter
b is lowercase

```

```
5 is digit
A is a letter
A is a uppercase
is a whitespace

```

Strings

In Java, a string is a sequence of characters. Java implements strings as object of type `String` rather than character arrays. `StringBuffer` is a companion class to `String`, whose objects contain `Strings` that can be modified.

`String` and `StringBuffer` classes are defined in `java.lang`. Thus they are available to all programs automatically.

String Constructors

The `String` class supports several constructors. To create an empty string **new** operator is used.

```
String S=new String();

```

String Length

The length of a string is the number of characters that it contains. To obtain this value the `int length()` method is used.

```
char c[]={ 'a','b','c' };
String s=new String(c);
System.out.println(s.length());

```

The above fragment prints 3.

String Concatenation

Java allows to chain together a series of `+` operations. The following fragment concatenates three strings.

```
String s1=" Welcomes";
String s2="NUR"+s1+" you";
System.out.println(s2);

```

This displays the strings "NUR welcomes you"

Using concat()

To concatenate two strings we can use `concat()`

```
String s1="one";
String s2=s1.concat("two");

```

This will puts the string "onetwo"

String Comparison

· equals()

To compare two strings for equality, we can use `equals()`

```
String s1="hai";

```

```
String s2="hello";
String s3="hai";
System.out.println(s1.equals(s2));
System.out.println(s1.equals(s3));
```

The output for the above fragment will be
false //where s1 and s2 not equals
true //where s1 and s3 equals

· equalsIgnoreCase()

To compare two strings that ignores case differences, we can use equalsIgnoreCase()

```
String s1="hai";
String s2="HAI";
System.out.println(s1.equals(s2));
System.out.println(s1.equalsIgnoreCase(s2));
```

The output for the above fragment will be
false //where s1 and s2 not equals (considering case)
true //where s1 and s2 equals (ignoring case)

Searching Strings

The string class provides two methods that allow you to search the specified character or substring.

· indexOf()

o Searches the first occurrence of the character or substring.

· lastIndexOf()

o Searches the last occurrence of the character or substring.

```
String S="Welcome to the String World in side the Java";
System.out.println(s.indexOf('t'));
System.out.println(s.indexOf("the"));
System.out.println(s.lastIndexOf('t'));
System.out.println(s.lastIndexOf("the"));
```

The above fragment will display

```
8 // 't' starts at 8th position
14 // "the" starts at 14th position
36 // 't' finishes at 36th position
36 //"the" finishes at 36th position
```

Modifying String

· replace()

o The replace() method replaces all occurrences of one character to a specified one.

```
String s="Hello".replace('l','w');
```

Puts the string "Hewwo" into s.

· trim()

o The trim() method returns a copy of string by eliminating leading and trailing whitespaces.

```
String s=" Hello World ".trim();
```

Puts the string "Hello World" into s.

Changing Case

· The method toLowerCase() converts all the characters in a sting to lower case

```
String s="HELLO WELCOME";
System.out.println(s.toLowerCase());
This fragment will display "hello welcome".
· The method toUpperCase() converts all the characters in a sting to upper
case
String s=" hello welcome";
System.out.println(s.toUpperCase());
This fragment will display "HELLO WELCOME".
```

String Buffer

StringBuffer is a peer class of String that provides much of the functionality of strings.

- insert()
 - o The insert() method inserts one string into another

```
StringBuffer sb=new StringBuffer("I Java");
sb.insert(2," like ");
System.out.println(sb);
This fragment will display "I like Java".
```
- delete() and deleteCharAt()
 - o The delete() method deletes a sequence of characters

```
StringBuffer sb=new StringBuffer("Welcome to Strings")
sb.delete(8,10);
System.out.println(sb);
This fragment will display "Welcome Strings".
```

 - o The deleteCharAt() method deletes the character by specifying the index

```
StringBuffer sb=new StringBuffer("Welcome to Strings")
sb.deleteCharAt(6);
System.out.println(sb);
This fragment will display "Welcome Strings".
```
- reverse()
 - o The reverse() method reverse the characters within the string

```
StringBuffer sb=new StringBuffer("Welcome")
sb.reverse();
System.out.println(sb);
This fragment will display "emocleW".
```

JAVA PROGRAMMING Exercises

Question1:

Let Factorial() be a recurring function; such that

Factorial(0)-----→ 1

And

Factorial(n)-----→n*Factorial(n-1),for all $n \in \mathbb{N}^*$ (n belongs to \mathbb{N}^*)

Implement the recurring Factorial() function in Java Programming(e.g., Factorial(5) gives as output 120).

Question2:

Let's define a recurring series(Fibonacci) as follows:

$(u_n)_{n \in \mathbb{N}}$ such that For all $n \geq 2, u_n = u_{n-1} + u_{n-2}$;

Let u be a recurring function such that:

$u(0) \rightarrow 0$

$u(1) \rightarrow 1$

$u(n) \rightarrow u(n-1) + u(n-2)$

Implement the given function u in Java(for example, $u(8)=21$)

More about OOP Tips

Returning values from a method

To get a value out of a method, we use the return process of typed methods.

But **what happens if we want more than one value to be returned?**

Typed methods are not adequate here, instead, we fall back on objects again.

Example:

```
class Worker{
    double x,y;

    void solve (int q){
        //Calculations
        x=q*q;//q square
        y=q*q*q;//q cube
    } // end solve
} //end Worker
```

Then the main program creates an object of this class,

```
Worker work=new Worker ();
```

```
work.solve (10);
```

```
display.println("Answers are"+work.x+"and"+work.y);
```

- ❖ That's it. We have now set up a modern, extensible and object-oriented approach to multiple return values, which does not use of global variables.

ARRAYS and Matrices

Simple arrays

- ✓ An array is a variable which can store multiple values.
- ✓ An array is a bounded collection of elements of the same type, each of which can be selected by indexing with an integer from 0 upwards to a limit specified when the array is created. The relevant form is

Array declaration

```
type arrayname[]=new type[limit];
//The limit gives the number of elements in the array,
//with each element being indexed by a number in the
//range from 0 to limit -1
type arrayname[]={value1,value2,...};
```

Examples of array declarations of the first form shown above are:

```
int frequencies[]=new int[20];
```

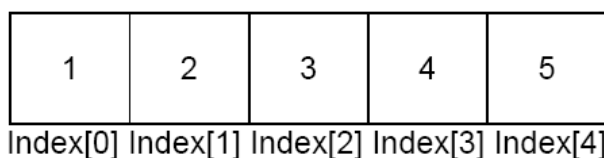
```
String rainbow[]=new String[7];
```

```
double results[]=new double[101];
```

In array, the indexes will be start from 0 and end with (array.length-1). Say for example, to retrieve the value 3, the subscript used is

Conceptual Diagram for 1D array

Conceptual Diagram for 1D array



```
Sytem.out.println(intarray[2]);
```

The above output statement will print 3.

The following program specifies how 1D array are declared with initialization and access those array elements.

```
/
* program for 1D Arrays */
import java.io.*;
class oneDarray
{
public static void main(String args[ ])
{
int intarray[ ]={ 1,2,3,4,5 };
Sytem.out.println("First Element =" +intarray[0]);
Sytem.out.println("Second Element =" +intarray[1]);
Sytem.out.println("Third Element =" +intarray[2]);
Sytem.out.println("Fourth Element =" +intarray[3]);
Sytem.out.println("Fifth Element =" +intarray[4]);
}
}
```

Output:

```
1
2
3
4
5
```

Here is one more example that uses one-dimensional arrays

```
/
* Program to calculate average using arrays*/
import java.io.*;
class Average
{
public static void main(String args[])
{
double nums[] = { 10.1, 11.2, 12.3, 13.4, 14.5 };
double result = 0;
int i;
for(i=0; i<5; i++)
result = result + nums[i];
System.out.println("Average is " + result / 5);
}
}
```

Output:

```
12.3
```

Arrays of dynamic allocation declaration

- Dynamic allocation of arrays will be declared by using the **new** keyword.
- The elements in the array allocated by new will automatically be initialized to zero.

```
int intarray[ ]=new int[5];
```

The size of the array can be dynamically received by user input also.

For example

```
int a;
```


a will be assigned by some value from the user input

now

```
int intarray[]=new int[a];
```

This statement will allocate the array of a size a.

The following program shows how the values been set the array elements

```
/
```

```
/ Demonstrate a one-dimensional array with new.
```

```
import java.io.*;
```

```
class Array
```

```
{
```

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

```
{
```

```
int month_days[];
```

```
month_days = new int[12];
```

```
month_days[0] = 31;
```

```
month_days[1] = 28;
```

```
month_days[2] = 31;
```

```
month_days[3] = 30;
```

```
month_days[4] = 31;
```

```
month_days[5] = 30;
```

```
month_days[6] = 31;
```

```
month_days[7] = 31;
```

```
month_days[8] = 30;
```

```
month_days[9] = 31;
```

```
month_days[10] = 30;
```

```
month_days[11] = 31;
```

```
System.out.println("April has " + month_days[3] + " days.");
```

```
}
```

```
}
```

Output:

April has 30 days.

To get the input from the user

Java is not directly supporting to input the number values by the user. To perform this it has some indirect methods. The following steps can be made to input a number value from the user.

1. We can use `DataInputStream` or `BufferedReader` to get the input.
2. The input values will assign to `String` variables
3. Using `parse` method we can convert the strings to required number values

The following program shows how to read the input from the user

```
/
```

```
/ Demonstrate a one-dimensional array with user input.
```

```
import java.io.*;
```

```
class ArrayInput
```

```
{
```

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

```
{
```

```
int n;
```

```
String s;
```

```

    DataInputStream ds=new DataInputStream(System.in);
    System.out.println("Enter the size of the Array=");
    s=ds.readLine();
    n=Integer.parseInt(s);
    int a[]=new int[n];
    System.out.println("Enter the elements of the Array");
    for(int i=0;i<a.length;i++)
    {
        s=ds.readLine();
        a[i]=Integer.parseInt(s);
    }
    System.out.println("The elements of the Array");
    for(int i=0;i<a.length;i++)
        System.out.println(a[i]);
    }//end main
} //end class

```

The above program will get the input from the user and according to user specified size the array will be declared dynamically. According to the values specified by the user, the array will hold the elements and display the elements.

The same program can be written by using `BufferedReader`

```

int a;
String s;
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter the size of the Array=");
s=br.readLine();
a=Integer.parseInt(s);

```

The above fragment will puts the input value which is entered by the user, into the variable `a`.

Example of an array program: Frequency count

```

Random generator=new Random ();

int frequency [] =new int [20];

int score;

for (int i=0;i<1000;i++) {

    score=Math.abs(generator.nextInt());

    //reduces the number to between 0 and 19

    score%=20;

    frequency[score]++;

}

```

Note:

- ✓ **References:** When an array is declared, a reference is set up for it. This reference will point to the place where the array's elements are stored.
- ✓ When arrays are assigned or passed as parameters, it is their references that move around, not the array itself. This is efficient, and useful for scientific programming.
- ✓ **Array Operator:** The only operator that applies to a whole array is assignment. Assigning one array to another, though, does not create a copy of the whole array. Instead, it copies the references, so that both arrays will refer to the same storage, and changes made to one will affect the other.

Multi - Dimensional Arrays

Arrays of Arrays

More than one dimensional, the arrays are called as multi-dimensional arrays. In

Java multi-dimensional arrays are actually arrays of arrays.

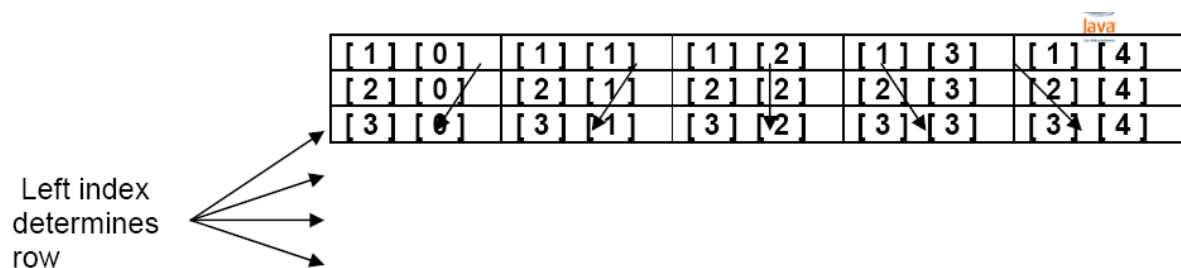
The following declaration states the two dimensional array

```
int twoD[ ] [ ] = new int[ 4] [ 5 ];
```

4 represent the row and 5 represent the column of the array.

Conceptual Diagram for two dimensional arrays

Right index determines column



The following program numbers each element in the array from left to right, top to bottom and displays the values.

```
/
/ Demonstrate a two-dimensional array.
import java.io.*;
class TwoDArray
{
    public static void main(String args[])
    {
        int twoD[][] = new int[4][5];
        int i, j, k = 0;
        for(i=0; i<4; i++){
            for(j=0; j<5; j++) {
                twoD[i][j] = k;
                k++;
            }
            for(i=0; i<4; i++) {
```

```

        for(j=0; j<5; j++)
        System.out.print(twoD[i][j] + " ");
        System.out.println();
    }
} //end main
} //end class

```

Output:

```

0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19

```

Initializing two dimensional arrays

The two dimensional arrays are initialized like one dimensional array with a curly braces. But here, we use curly braces inside curly braces.

```

int aa[][]={
    { 1, 2, 3 },
    { 4, 5, 6 },
    { 7, 8, 9 }
};

```

The above declaration initializes the two dimensional arrays with a size of 3x3.

Two dimensional arrays can be initialized with some arithmetic operation like the following program has.

```

/
/ Initialize a two-dimensional array.

```

```

class Matrix {
    public static void main(String args[]) {
        double m[][] = {
            { 0*0, 1*0, 2*0, 3*0 },
            { 0*1, 1*1, 2*1, 3*1 },
            { 0*2, 1*2, 2*2, 3*2 },
            { 0*3, 1*3, 2*3, 3*3 }
        };
        int i, j;
        for(i=0; i<4; i++) {
            for(j=0; j<4; j++)
                System.out.print(m[i][j] + " ");
            System.out.println();
        }
    } //end main
} //end class

```

Output:

```

0.0 0.0 0.0 0.0
0.0 1.0 2.0 3.0
0.0 2.0 4.0 6.0
0.0 3.0 6.0 9.0

```

Algorithms for Sorting and Searching Arrays

Sorting Arrays

Sorting is the process of putting data in order; either numerically or alphabetically. There are literally hundreds of different ways to sort arrays. The basic goal of each of these methods is the same: to compare each array element to another array element and swap them if they are in the wrong position.

The selection sort

In selection sort during each iteration, the unsorted element with the smallest (or largest) value is moved to its proper position in the array.

The number of times the sort passes through the array is one less than the number of items in the array. In the selection sort, the inner loop finds the next smallest (or largest) value and the outer loop places that value into its proper location.

The following table shows the swap of elements in each iteration. Here each iteration includes the combination of both loops.

Selection Sort (Descending Array)

Array at beginning: 84 69 76 86 94 91

After iteration 1: 84 91 76 86 94 69

After iteration 2: 84 91 94 86 76 69

After iteration 3: 86 91 94 84 76 69

After iteration 4: 94 91 86 84 76 69

After iteration 5: 94 91 86 84 76 69

The algorithm for selection sort:

```
int [ ] selection_sort(int [ ] array)
{
    int i, j, first, temp;
    int array_size = array.length( );
    for (i=0; i<array.lenth; i++)
    {
        for (j=i+1; j<a.length; j++) //Find smallest element between the positions 1 and i.
        {
            if (array[j] < array[i])
            {
                temp = array[i]; // Swap smallest element found with one in position i.
                array[j] = array[i];
                array[i] = temp;
            }
        }
    }
    return array;
}
```

The following program shows how selection sort can be implemented using class.

```
/
/demonstration of selection sort
import java.io.*;
class selection
```

```

{
int i,j,temp=0;
int []sortt(int []a)
{
for(i=0;i<a.length;i++)
for(j=i+1;j<a.length;j++)
if(a[i]<a[j])
{
temp=a[i];

a[i]=a[j];
a[j]=temp;
}
return a;
}
}
class sorting
{
public static void main(String args[])
{
int x[]={ 84, 69,76,86,94,91 };
int y[]=new int[x.length];
selection sl=new selection();
y=sl.sortt(x);
for(int s=0;s<y.length;s++)
System.out.println(y[s]);
}
}

```

Output:

```

94
91
86
84
76
69

```

Searching Arrays

A fast way to search a sorted array is to use a binary search. The idea is to look at the element in the middle. If the key is equal to that, the search is finished. If the key is less than the middle element, do a binary search on the first half. If it's greater, do a binary search of the second half.

The algorithm for binary search:

```

int AL (int[] array, int seek)
{
int bot = -1;
int top = size;
while (top - bot > 1) {

```

```

int mid = (top + bot)/2;
if (array[mid] < seek) bot = mid;
else /* array[mid] >= seek */ top = mid;
}
return top;
}

```

The implementation of the above algorithm is implemented using class.

```

/
/demonstration of binarch algorithm
import java.io.*;
class bsearch
{
int search (int[] array, int seek)
{
int bot = -1;
int top = array.length-1;
while (top - bot > 1)
{
int mid = (top + bot)/2;
if (array[mid] < seek) bot = mid;
else top = mid;
}
return top;
}
}
class binarysearch
{
public static void main(String args[])
{
int x[]={1,2,3,4,5};
int n=3;
bsearch bs=new bsearch();
int y=bs.search(x,n);
System.out.println("The index of array element "+n+" = "+y);
}
}

```

Output:

The index of array element 3 = 2.

Composition of Classes

Defining a class within another class is known as nested classes or inner classes or composition classes.

The following program illustrates how to define an inner class.

```

/
/ Demonstrate an inner class.
class Outer
{
int outer_x = 100;

```

```

void test() {
    Inner inner = new Inner();
    inner.display();
}
// this is an innner class
class Inner
{
    void display() {
        System.out.println("display: outer_x = " + outer_x);
    }
}
}
class InnerClassDemo
{
    public static void main(String args[])
    {
        Outer outer = new Outer();
        outer.test();
    }
}

```

Output:

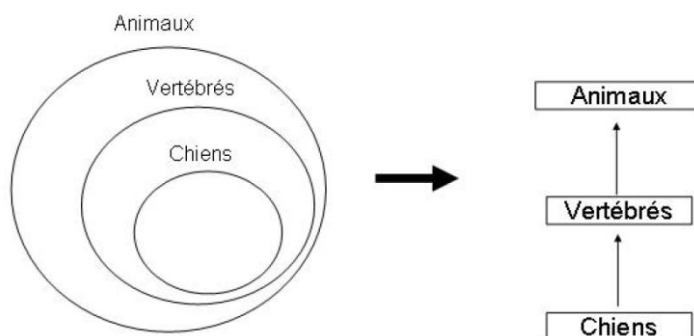
display: outer_x=100

Inheritance

Inheritance

Inheritance is used to inherit the properties from one class to another. The class that is inherited is called a **super class**. The class that does inheriting is called a **sub class**. The following program creates a super class called A and a subclass B. The keyword **extends** is used to create a sub class of A.

Inheritance Diagram





Here we say :

MaClass extends MaSuperClass
 MaClass inherits MaSuperClass
 MaClass derives from MaSuperClass
 MaClass is a subclass of MaSuperClass
 MaClass is a daughter class of MaSuperClass
 MaSuperClass is the Super class of MaClass
 MaSuperClass is the mother class of MaClass
 MaSuperClass is the father class of MaClass

Note:

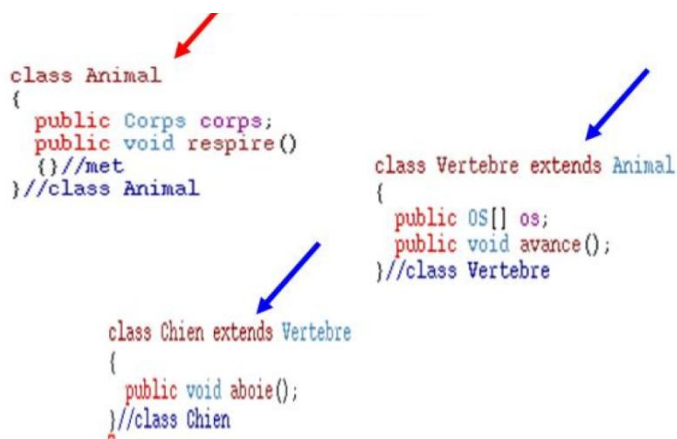
- ✓ Derivation(inheritance) is a specification
- ✓ No multiple inheritance in java(a class can't derives from more than one class in java)
- ✓ A class represent a set and a subclass represent a subset of its super class

Constructor in subclass

As explained ,instantiating a subclass object begins a chain of constructor calls in which the subclass constructor,before performing it own tasks, invokes its direct supclass's constructor either explicitly(via the **super** reference) or implicitly(by calling the super class's default constructor).

`Super(firstname,lastname,phone);`// a call to the superclass constructor in a subclass

Example of Inheritance;



```
//le chien de sébastien
Chien belle = new Chien();

//est un chien, il peut donc aboyer
belle.aboie();

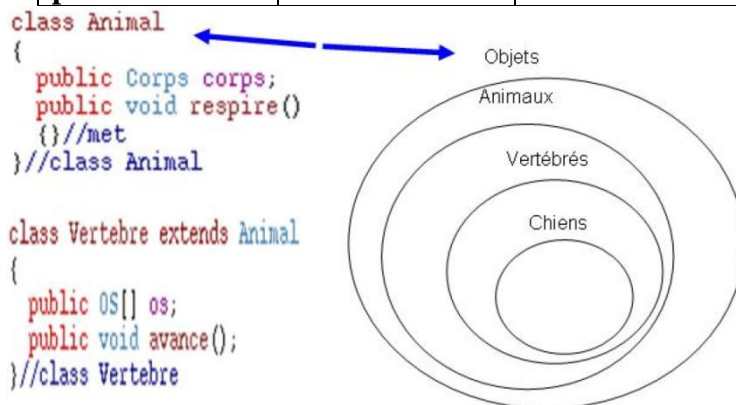
//Mais c'est aussi un animal, donc il a un corps
belle.corps = null;

//Il hérite également de la faculté d'avancer car
//c'est un vertébré
belle.avance();
//et de respirer car c'est un animal
belle.respire();
```

- ✓ A sub class receives functionalities of its super class such as its data members and its methods.

Data availability

Access Modifier	Visible in a Class	Visible in a class of same package	Visible in a subclass	Visible for all classes
public	Yes	Yes	Yes	Yes
protected	Yes	Yes	Yes	No
package	Yes	Yes	No	No
private	Yes	No	No	No



Hierarchical Diagram

- ✓ In java all classes are subclass of java.lang.Object
- ✓ java.lang.Object is the overall super class

Example,

```

import javax.swing.*;


public class DemoFrame extends JFrame
{
    private JButton button = new JButton();

    public DemoFrame()
    {
        button.setText( "Click !" );
        this.getContentPane().add( button );
        this.show();
    } //cons

    public static void main(String[] args)
    {
        new DemoFrame();
    } // end of main()

} //class DemoFrame

```



- ✓ To deal with graphical interface objects ,often we do need to import javax.swing.JFrame

OVERRIDING or Redefinition

```

class Rectangle
{
    protected int largeur;
    protected int longueur;

    public void setLargeur(int largeur)
    {
        this.largeur = largeur;
    } //met

    public void setLongueur(int longueur)
    {
        this.longueur = longueur;
    } //met

    public int getAire()
    {
        return largeur*longueur;
    } //met
} //class

class Carre extends Rectangle
{
    public void setLargeur(int largeur)
    {
        this.largeur = largeur;
        this.longueur = largeur;
    } //met

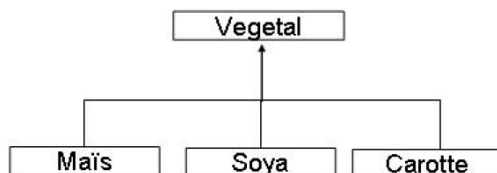
    public void setLongueur(int longueur)
    {
        this.largeur = longueur;
        this.longueur = longueur;
    } //met

    public int getAire()
    {
        return largeur*largeur;
    } //met
} //class

```

- ✓ A subclass can redefine or overrides methods of its super class.
/ An other example of inheritance.

POLYMORPHISM



Each of these 4 classes has a method called `public void pousse()`;

Let `Vegetal[]` be an array of vegetable objects;

`Vegetal[] vegetal = new Vegetal{ new carotte(),...}`

To grow up all vegetables let do:

```

for( int i=0;i<30;i++)
    vegetal[i].pousse();

```

- Java knows dynamically the type of object and then executes the specific method for that object

- In particular, polymorphism enables us to write programs that process objects that share the same superclass in a class hierarchy as they are all objects of the super class; this can simplify programming.

ABSTRACT CLASSES

```

public abstract class Animal
{
}

public abstract class Animal
{
    public abstract void respire();

    public void getNomLatin()
    {
        ...
    } //met
}

Animal animal = new Animal();

public class Animal
{
    public void respire();

    public void getNomLatin()
    {
        ...
    } //met
}

```

- ✓ To be an abstract class, you may need only one method declared as abstract;
- ✓ Abstract class can not be instantiated (we can not create an object of this Class)
- ✓ Abstract class are used for generalization (abstraction); they are used to give an idea on how something will look like in general view.

ABSTRACT CLASSES and Inheritance

```

public abstract class Animal
{
    public abstract void respire();
} //class Animal

class Chien extends Animal
{
}

} //class Chien

```

Note:

- ✓ Chien class does not get compiled because it inherits an abstract method respire () from Animal class.
- ✓ To solve this problem ,we have two options;
 - 1.To make abstract our Chien class;
 2. Is to override(redefine) respire() method

```

public abstract class Animal
{
    public abstract void respire();
} //class Animal

class Chien extends Animal
{
    public void respire()
    {
        //instructions de respiration
    } //met
} //class Chien

```

More examples of inheritance :

```

/
/ Create a superclass.
class A
{
    int i, j;
    void showij() {
        System.out.println("i and j: " + i + " " + j);
    }
}

```

```

// Create a subclass by extending class A.
class B extends A
{
    int k;
    void showk() {
        System.out.println("k: " + k);
    }
    void sum() {
        System.out.println("i+j+k: " + (i+j+k));
    }
}
class SimpleInheritance
{
    public static void main(String args[])
    {
        A superOb = new A();
        B subOb = new B();
        // The superclass may be used by itself.
        superOb.i = 10;
        superOb.j = 20;
        System.out.println("Contents of superOb: ");
        superOb.showij();
        System.out.println();
        /* The subclass has access to all public members of
        its superclass. */
        subOb.i = 7;
    }
}

```

```
subOb.j = 8;
subOb.k = 9;
System.out.println("Contents of subOb: ");
subOb.showij();
subOb.showk();
System.out.println();
System.out.println("Sum of i, j and k in subOb:");
subOb.sum();
}
```

Output:

Contents of SuperOb:

i and j: 10 20

Contents of SubOb:

i and j: 7 8

k: 9

Sum of I, j and k in subOb:

i+j+k: 24

Recursion

Recursion is the process of defining something in terms of itself. A method that calls itself is said to be recursive.

The classical example of recursion is the computation of the factorial number.

The factorial of a number N is the product of all the whole numbers between 1 and N.

for example, 3 factorial is 1x2x3, or 6.

The following program illustrates the recursion.

```
/
/ A simple example of recursion.
class Factorial
{
// this is a recursive function
int fact(int n) {
int result;
if(n==1) return 1;
result = fact(n-1) * n;
return result;
}
}
class Recursion
{
public static void main(String args[])
{
Factorial f = new Factorial();
System.out.println("Factorial of 3 is " + f.fact(3));
System.out.println("Factorial of 4 is " + f.fact(4));
System.out.println("Factorial of 5 is " + f.fact(5));
}
}
```

Output:

Factorial of 3 is 6
Factorial of 4 is 24
Factorial of 5 is 120

4. Interface

Format:

```
Access Specifier InterfaceName  
  
{  
Access Specifier Data Type Function1();  
Access Specifier Data Type Function2();  
Access Specifier Data Type FunctionN();  
}
```

Example:

```
interface MusicPlayer{  
// Cannot have method implementations:  
void on();  
void off();  
void play();  
void stop();  
}
```

Points to note above:

- A semicolon after the method definition
- No implementation logic in the method above
- interface keyword instead of class

Accessing an Interface

In the above example, we've not defined whether the interface is public, private or protected. A private interface makes no sense. If not defined the above interface is visible in the package where the interface belongs. You can define an interface public – which means the interface is visible outside the package as well. Methods inside the interface are public by default. So in the above example, the methods are public and visible outside of the package as well. The class which inherits the methods must explicitly define the methods to be public.

NB: To access the interface's methods we use the keyword implements.

Example:

```
class MP3Player implements MusicPlayer{  
public void on(){  
  
System.out.println("the MP3 Player is ON");
```

```
}  
public void off(){  
System.out.println("the MP3 Player is OFF");  
}  
public void play(){  
System.out.println("the MP3 Player is playing");  
}  
public void stop(){  
System.out.println("the MP3 Player is off");  
}  
}
```

MULTIPLE INHERITANCES

In Java, there is nothing which prevents from inheriting from multiple interfaces. Since there are no implementations in the methods (code in the methods), there is no danger or overwriting any implementations between multiple interfaces.

Example:

```
interface MusicPlayer {  
void on();  
void off();  
void play();  
  
void stop();  
  
}  
}  
interface VideoPlayer{  
void on();  
void off();  
void play();  
void stop();  
void changeContrast(int x);  
void changeBrightness(int x);  
  
}  
}  
class iPod implements MusicPlayer, VideoPlayer{  
public void on(){  
System.out.println("the MP3 Player is ON");  
}  
public void off(){  
System.out.println("the MP3 Player is OFF");  
}  
public void play(){  
  
System.out.println("the MP3 Player is playing");  
}  
}
```



```
public void stop(){
System.out.println("the MP3 Player is off");

}
public void changeContrast(int x){
System.out.println("Constrast Changed by" + x);
}
public void changeBrightness(int x){
System.out.println("Brightnesss Changed by" + x);
}
}
```

NB: Multiple Inheritances is implemented using interfaces