



304185

# Fundamentals of JAVA Programming

TE E&TE

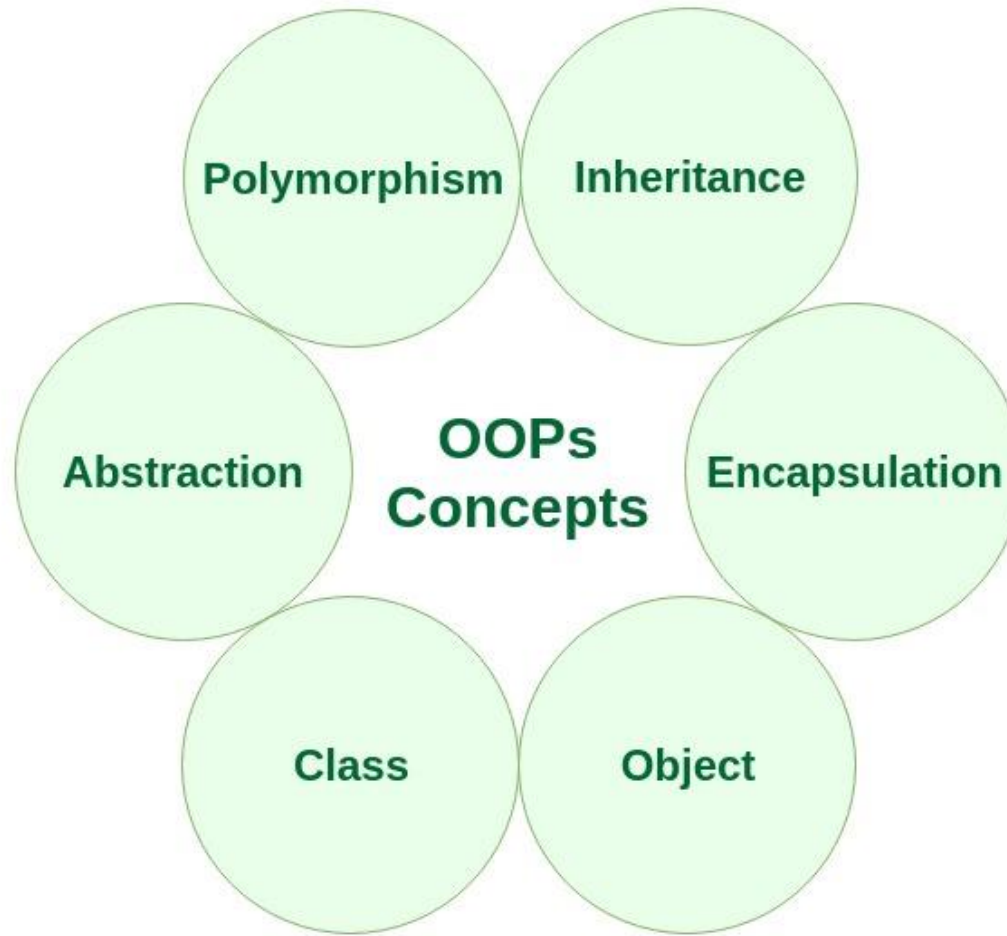
```
mirror_mod = modifier_ob.  
set mirror object to mirror.  
mirror_mod.mirror_object  
operation == "MIRROR_X":  
mirror_mod.use_x = True  
mirror_mod.use_y = False  
mirror_mod.use_z = False  
operation == "MIRROR_Y":  
mirror_mod.use_x = False  
mirror_mod.use_y = True  
mirror_mod.use_z = False  
operation == "MIRROR_Z":  
mirror_mod.use_x = False  
mirror_mod.use_y = False  
mirror_mod.use_z = True  
  
selection at the end -add  
mirror_ob.select= 1  
modifier_ob.select=1  
context.scene.objects.active  
("Selected" + str(modifier_ob.  
mirror_ob.select = 0  
= bpy.context.selected_object  
data.objects[one.name].select  
  
print("please select exactly  
  
-- OPERATOR CLASSES ----  
  
types.Operator):  
X mirror to the selected  
object.mirror_mirror_x"  
mirror X"  
  
context):  
context.active_object is not
```



# Syllabus

- Evolution of Java, Comparison of Java with other programming languages, Java features
- Environment, Simple Java Program, Java Tokens, Java Statements, Constants, variables, data types.
- Declaration of variables, Giving values to variables, Scope of variables, arrays, Symbolic constants,
- Typecasting, Getting values of variables, Standard default values, Operators, Expressions, Type
- Conversion in expressions, Operator precedence and associativity, Mathematical functions, Control
- Statements- Decision making & branching, Decision making & looping.

# OOP Concepts



# Evolution of Java

- Java - The programming language developed by Sun Microsystems in 1991.
- Originally created for consumer electronics (TV, VCR, Fridge, Washing Machine, Mobile Phone).
- Java's direct predecessor are C and C++.
- Java developed using syntax from C and OOP concepts from C++.
- Internet and Web was just emerging, so Sun turned it into a language of Internet Programming.
- It allows you to publish a webpage with Java code in it.

## GREEN TEAM





# Why the name JAVA?

- The language was initially called Oak, after an Oak tree stood outside gosling's office.
- It went name green later.
- Again renamed Java, from java coffee, said to be consumed in the large quantities by language creators.

# JAVA Milestones

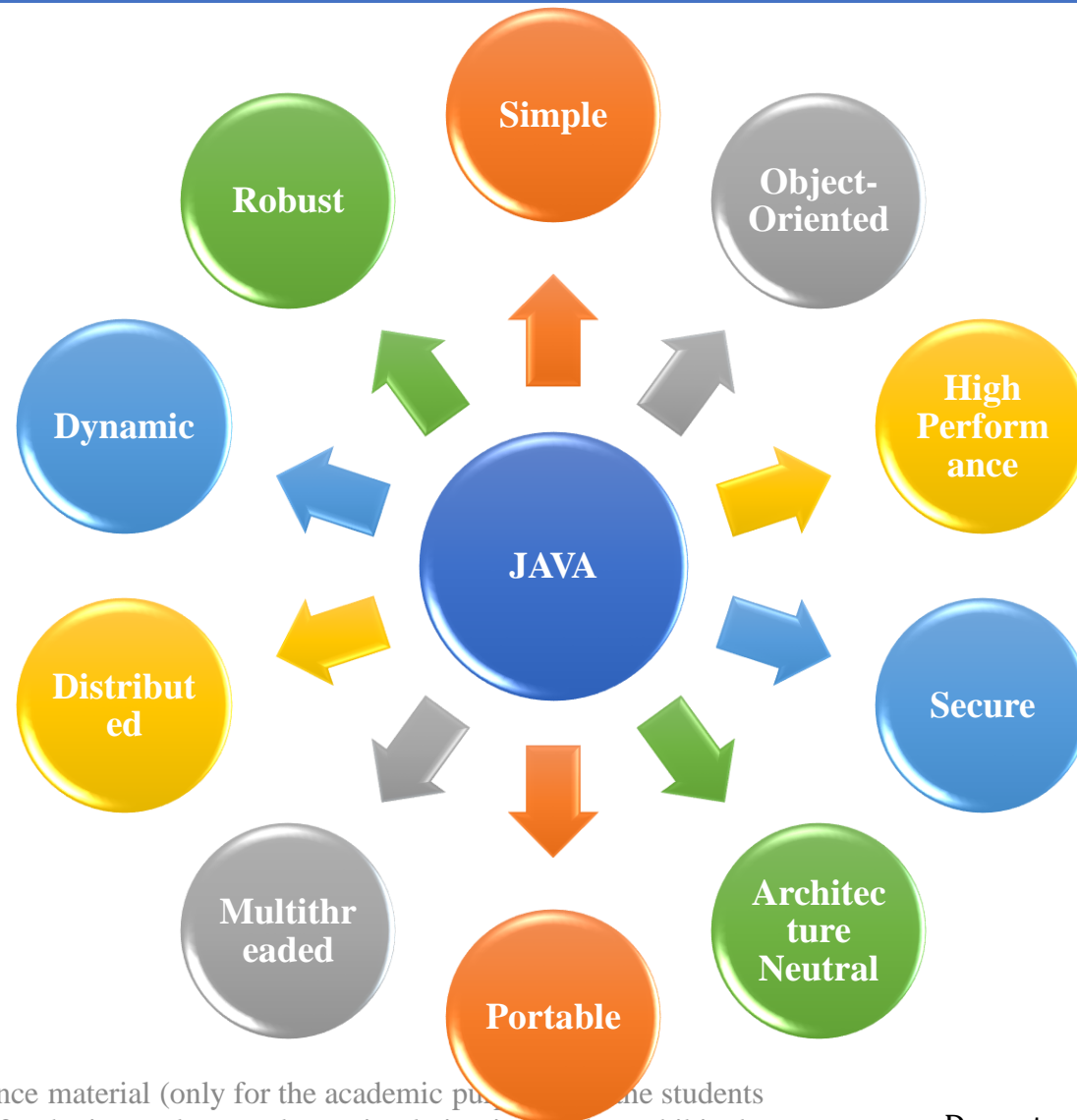
Year	Development
1990	Sun decided to developed special software that could be used for electronic devices. A project called Green Project created and head by James Gosling.
1991	Explored possibility of using C++, with some updates announced a new language named “Oak”.
1992	The team demonstrated the application of their new language to control a list of home appliances using a hand held device with a touch sensitive screen.
1993	The World Wide Web appeared on the Internet and transformed the text-based interface to a graphical rich environment. The team developed Web applets (tiny programs) that could run on all types of computers connected to the Internet.



# JAVA Milestones

Year	Development
1994	The team developed a new Web browser called “Hot Java” to locate and run Applets. HotJava gained instant success.
1995	Oak was renamed to Java, as it did not survive “legal” registration. Many companies such as Netscape and Microsoft announced their support for Java
1996	Java established itself as both, 1. “the language for Internet programming” 2. a general purpose OOP language.
1997	A class libraries, Community effort and standardization, Enterprise Java, Clustering, etc..
2010	Oracle Acquired Sun microsystem

# Features of JAVA





# Features of JAVA

## Simple

- Java is a simplified version of the C++ language.
- It eliminates all redundant and unreliable code.
- There is no support of pointers, preprocessor header files, operator overloading, and multiple inheritances in Java.

## Object-oriented

- Java is an object-oriented language and mainly focuses on objects rather than processes. Java follows the Object-Oriented Programming (OOP) concepts like:
  - Objects & Classes
  - Inheritance
  - Encapsulation / Data hiding
  - Abstraction
  - Polymorphism

Credit: <https://techvidvan.com/tutorials/java-introduction/>



# Features of JAVA

## Platform-independent

- Java is a platform-independent language as the source code of Java can run on multiple operating systems.
- Java programs can run on any machine or the operating system that does not need any special software installed. (JVM)

## Portable

- Java is portable because Java code is executable on all the major platforms.
- Once we compile Java source code to bytecode, we can use it in any Java-supported platform without modification, unlike other languages that require compiling the code for each platform.
- Java is portable because we can carry bytecode over to any other platform it runs on.

Credit: <https://techvidvan.com/tutorials/java-introduction/>



# Features of JAVA

## Robust

- Java provides a strong memory management.
- It supports Automatic garbage collection, so there is no need to delete the unreferenced objects manually.
- Java also provides exception handling and type-checking mechanisms.

## Secure

- Java does not support pointers that make Java robust and secure.
- The Java Runtime Environment (JRE) has a classloader that dynamically loads the classes into the Java Virtual Machine.
- The Bytecode Verifier of Java inspects the parts of code for checking the illegal code that can bypass access.
- The Security Manager of Java decides what resources to allot to a class. Such access includes reading and writing into files.



# Features of JAVA

## Multithreaded and Interactive

- Java is a multithreaded language that means it can handle different tasks simultaneously.
- Java supports multithreaded programs, in which there is no need to wait for one task to finish for another to start.
- This feature of Java significantly improves the interactive performance of graphical applications.
- The code of java is divided into smaller parts and Java executes them in a sequential and timely manner.

## Distributed

- Java is distributed because it encourages users to create distributed applications.
- In Java, we can split a program into many parts and store these parts on different computers.
- A Java programmer sitting on a machine can access another program running on the other machine.

# JAVA vs C

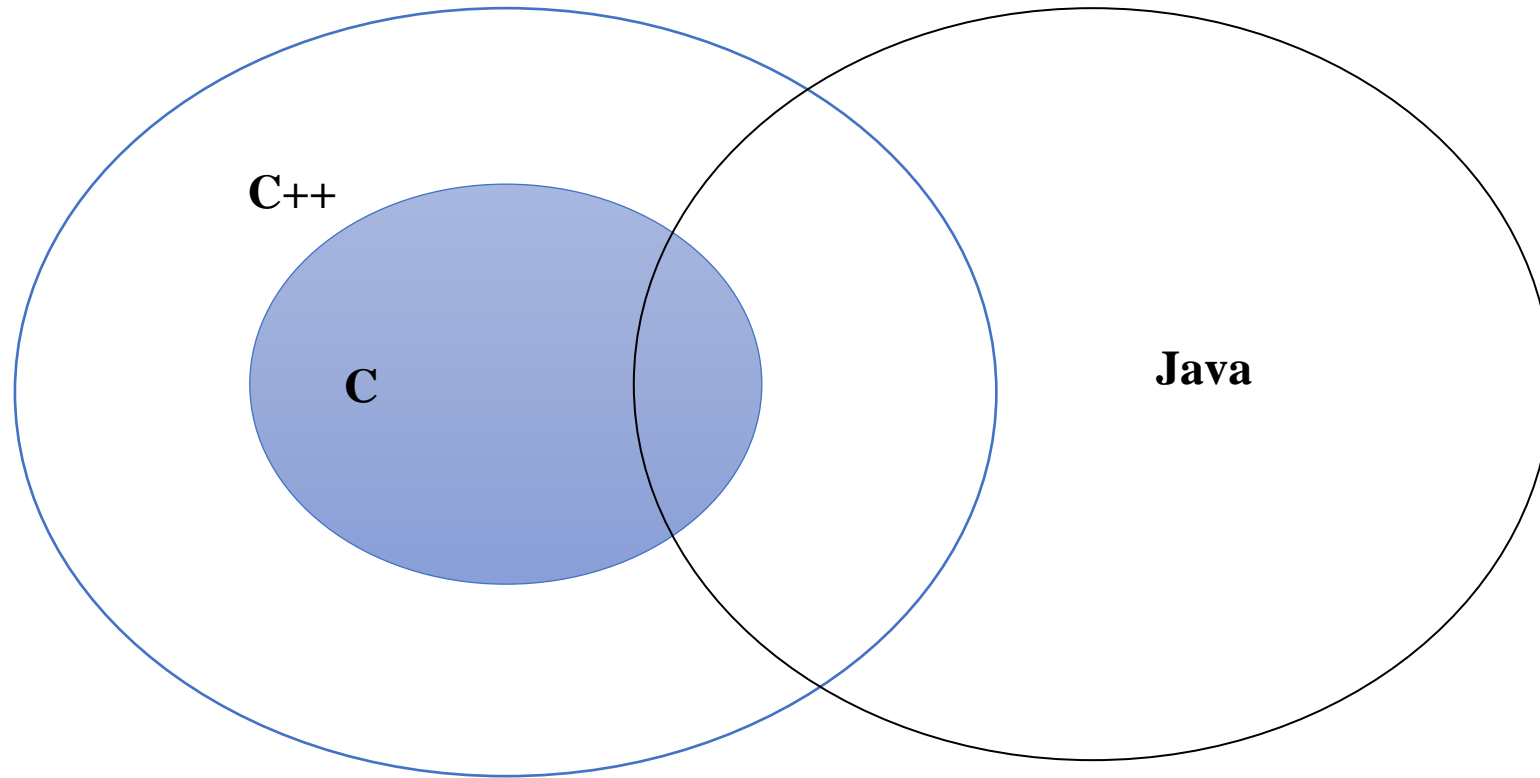
JAVA	C
1) Java does not include sizeof and typedef.	1) C includes keywords sizeof and typedef.
2) Java does not contain the data type struct and union.	2) C contains the data type struct and union.
3) Java does not support pointer type variable.	3) C supports pointer type variable.
4) Java does not have a preprocessor and therefore we cannot use #define, #include and #ifdef statements.	4) C has a preprocessor and therefore we can use #define, #include and #ifdef statements.
5) It does not define type modifiers keywords <b>auto</b> , <b>extern</b> , <b>register</b> , <b>signed</b> , <b>unsigned</b> .	5) It define type modifiers keywords <b>auto</b> , <b>extern</b> , <b>register</b> , <b>signed</b> , <b>unsigned</b> .

# JAVA vs C++

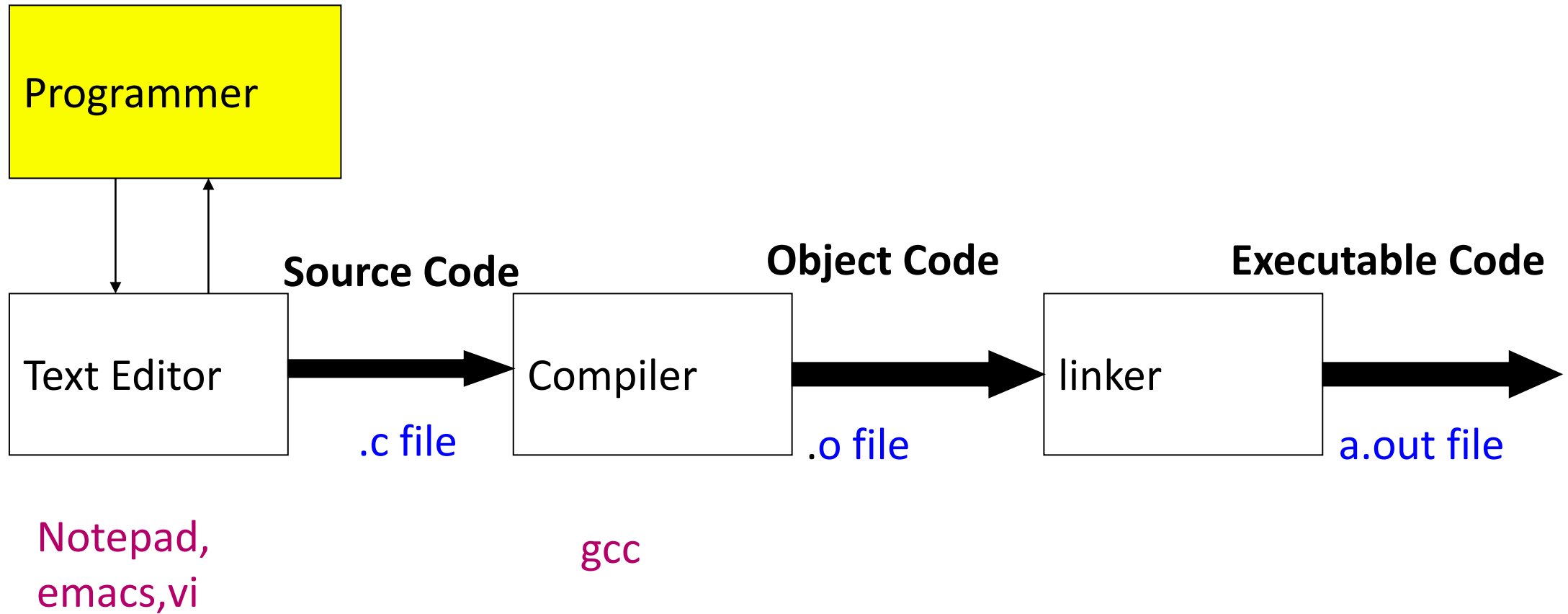
JAVA	C++
1) Java is true Object-oriented language.	1) C++ is basically C with Object-oriented extension.
2) Java does not support operator overloading.	2) C++ supports operator overloading.
3 ) Java does not support global variable. Every variable should be declared in class.	3) C++ support global variable.
4) Java does not use pointer.	4) C++ uses pointer.
5) It does not support multiple inheritance of classes (instead of this java has ' <b>interface</b> ').	5) C++ support Multiple inheritance of classes .
6) There are no header files in Java.	6) We must use header file in C++.



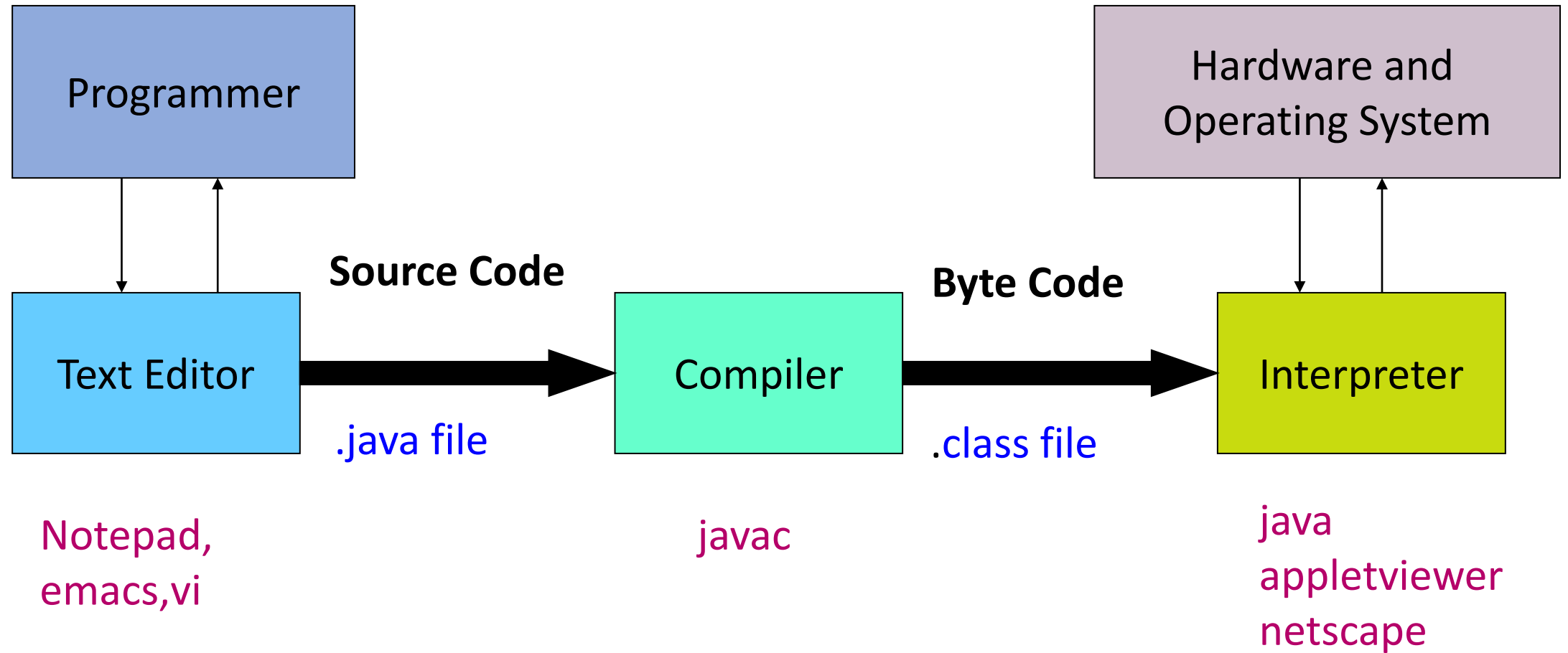
# Overlap of C, C++ and JAVA

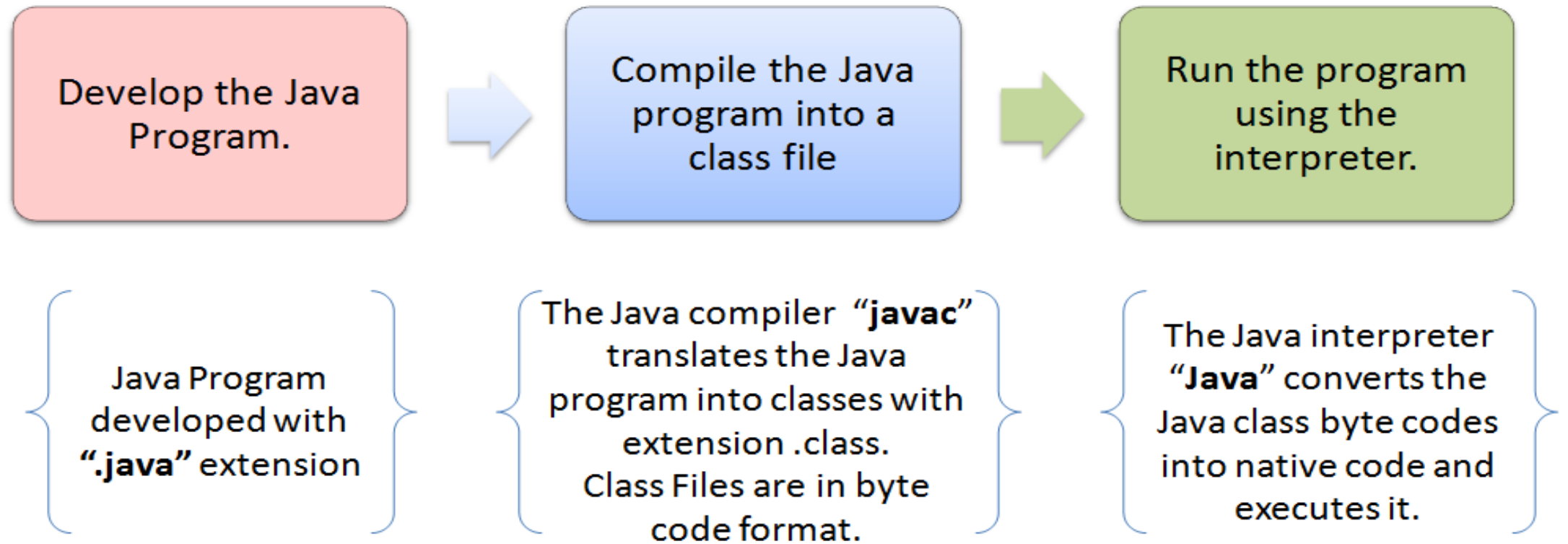


# Compiled Language



# JAVA: Compiled & Interpreted







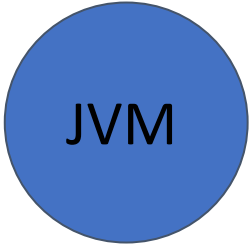
# Java Platforms: JDK, JRE & Java Virtual Machine

Platform is a software environment used to execute programs or applications.

**Java Platform:** Java platform is a software or collection of programs required to execute applications written in Java programming language. A Java platform consists of a Java compiler, a set of libraries, and an execution engine.

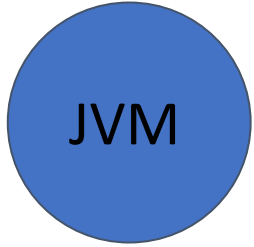
**Java platform consists of the following components.**

- Java language
- The Java Development Kit (JDK)
- The Java Runtime Environment (JRE)
- The Java Compiler
- The Java Virtual Machine (JVM)

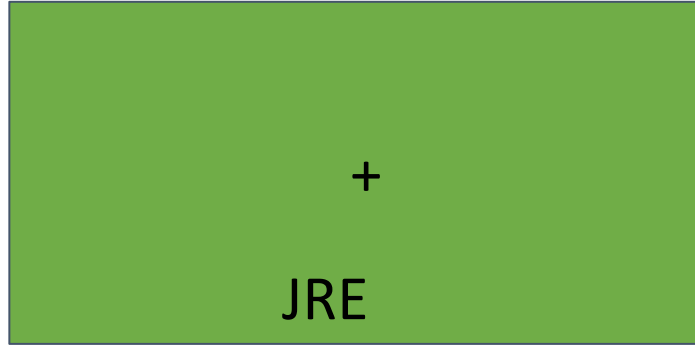


Java Virtual Machine

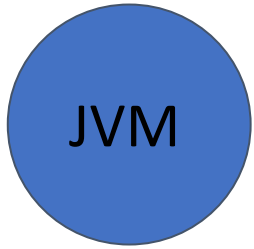




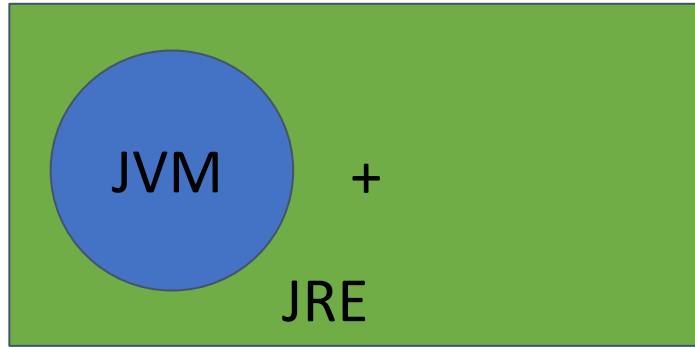
Java Virtual Machine



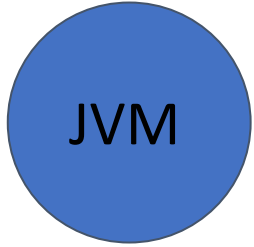
Java Runtime Environment



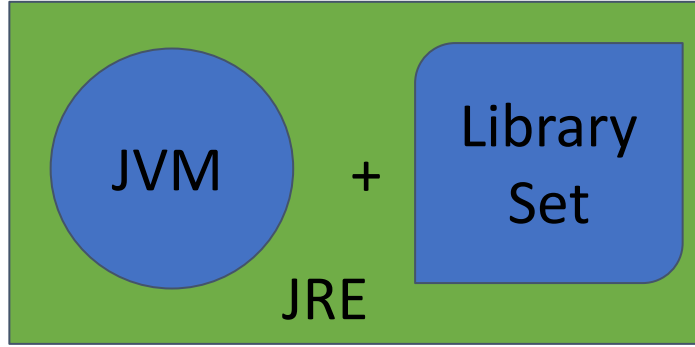
Java Virtual Machine



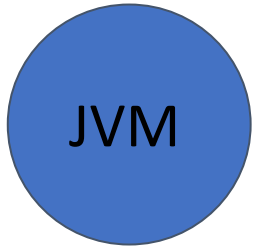
Java Runtime Environment



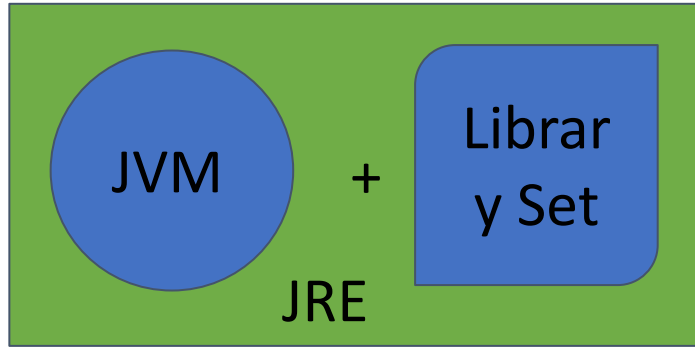
Java Virtual Machine



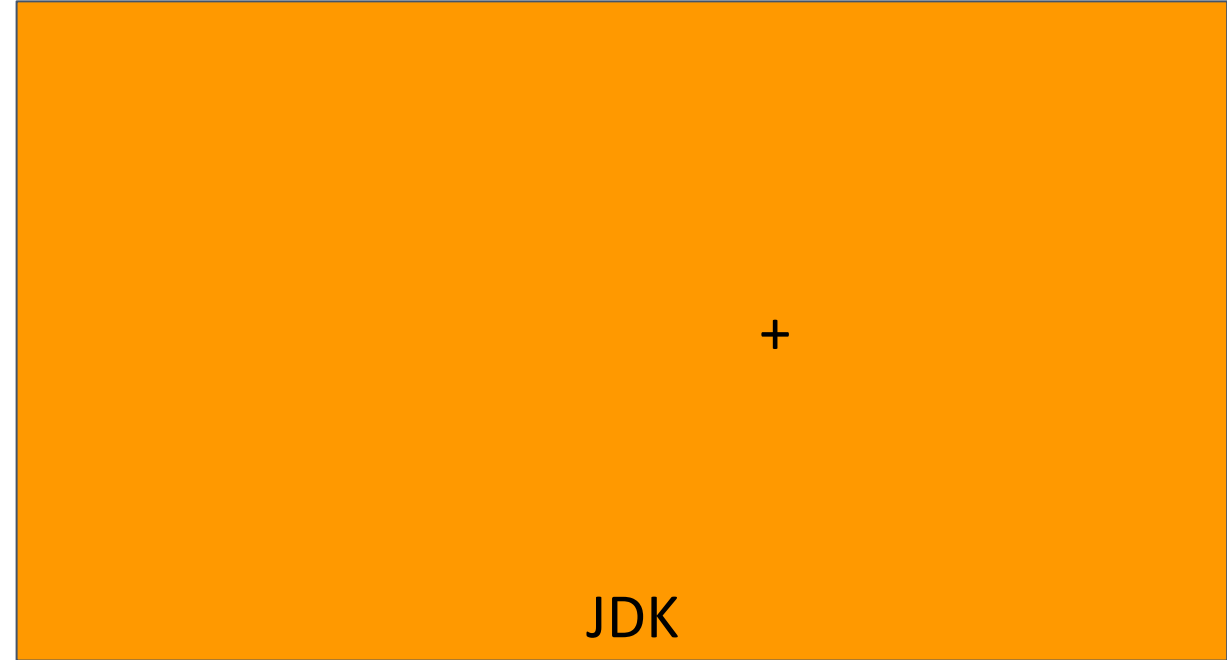
Java Runtime Environment



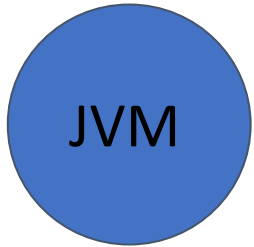
Java Virtual Machine



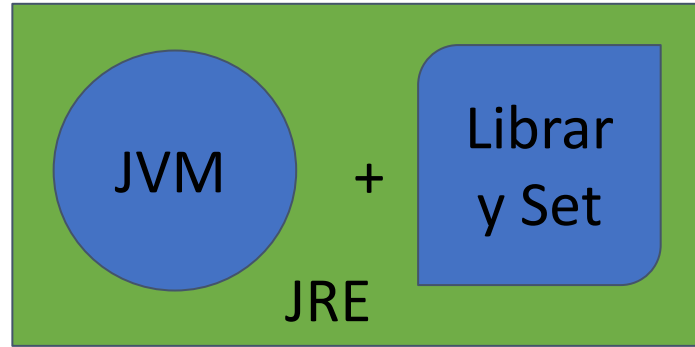
Java Runtime Environment



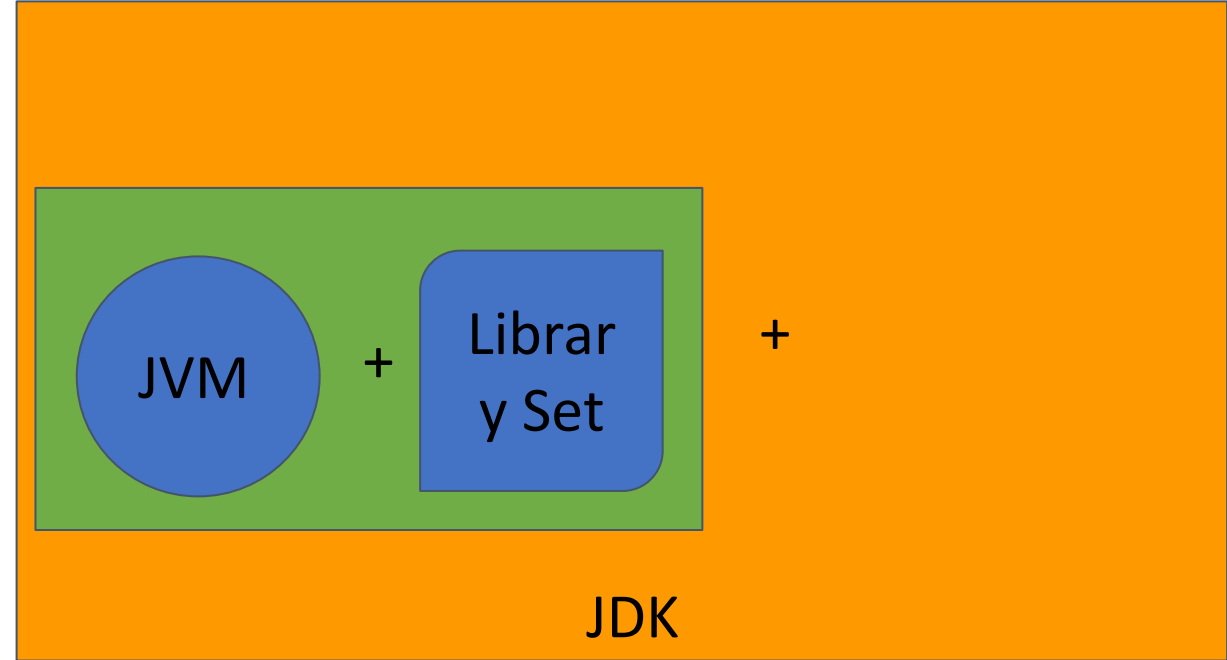
Java Development  
Kit



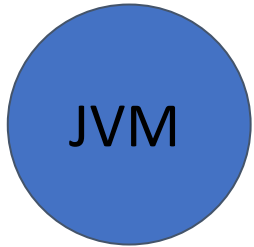
Java Virtual Machine



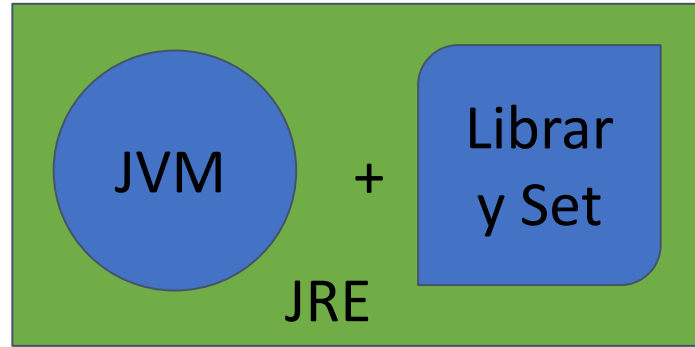
Java Runtime Environment



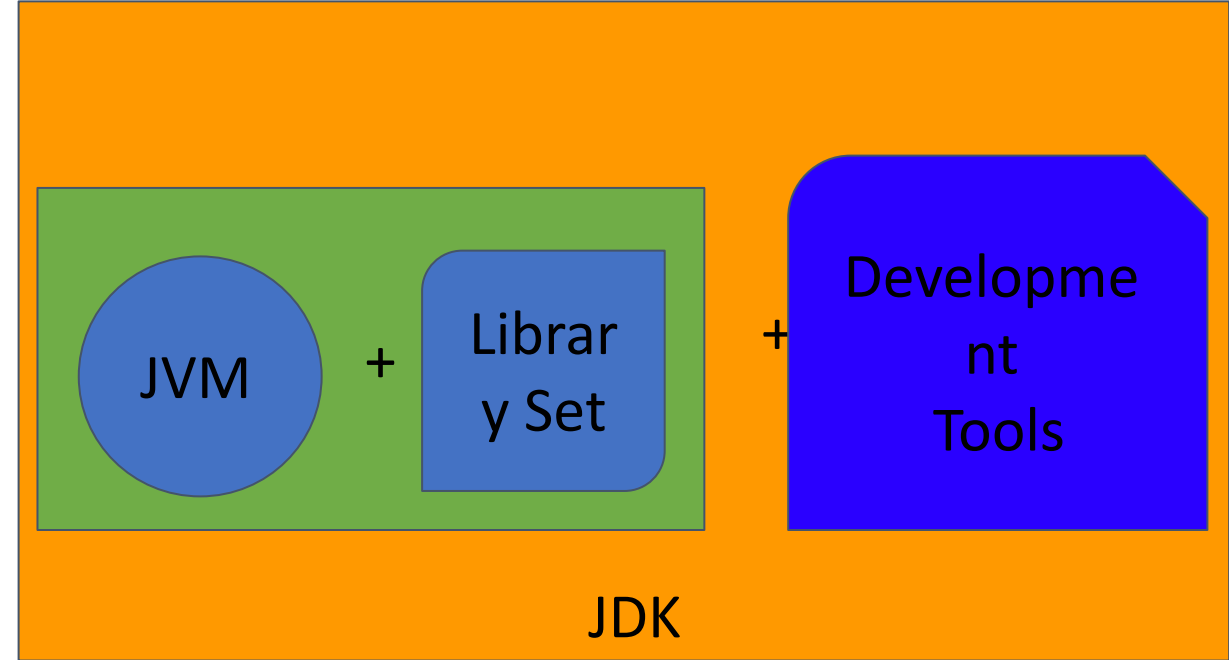
Java Development  
Kit



Java Virtual Machine



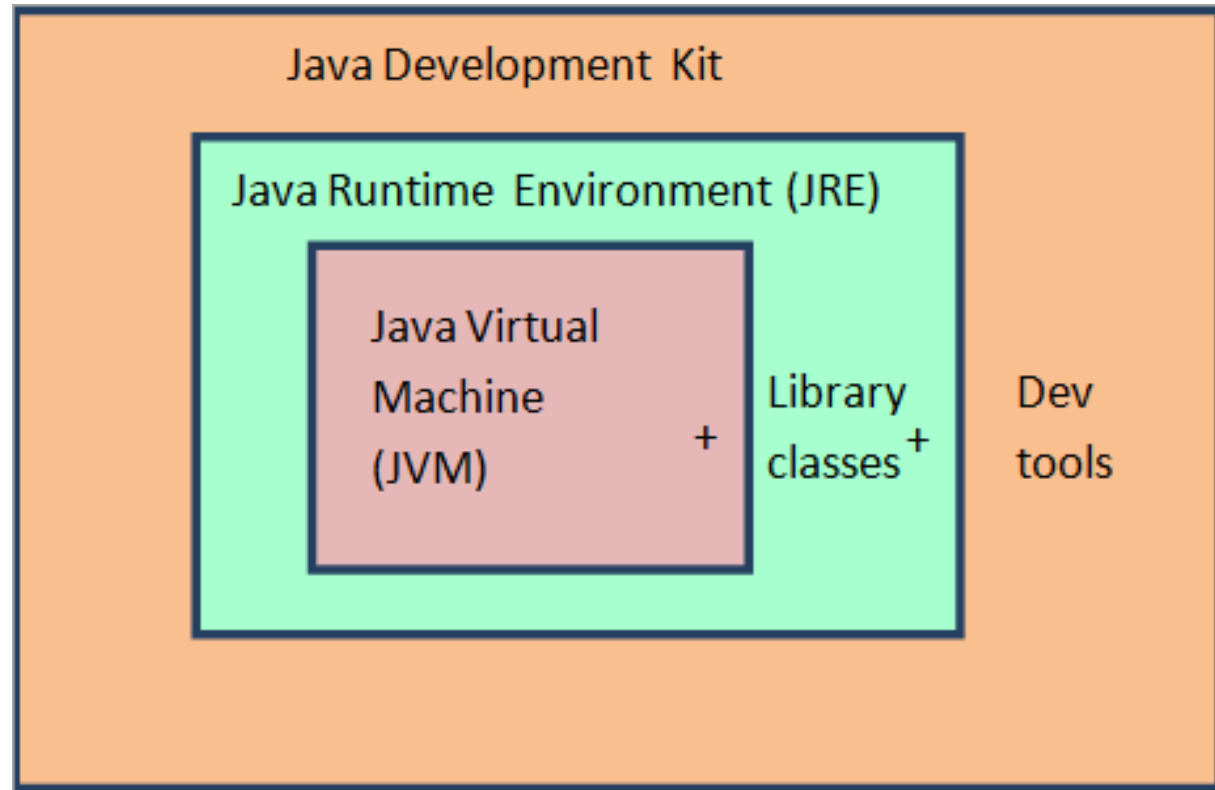
Java Runtime Environment



Java Development Kit



# JAVA Platforms



*JRE = JVM + library classes.*

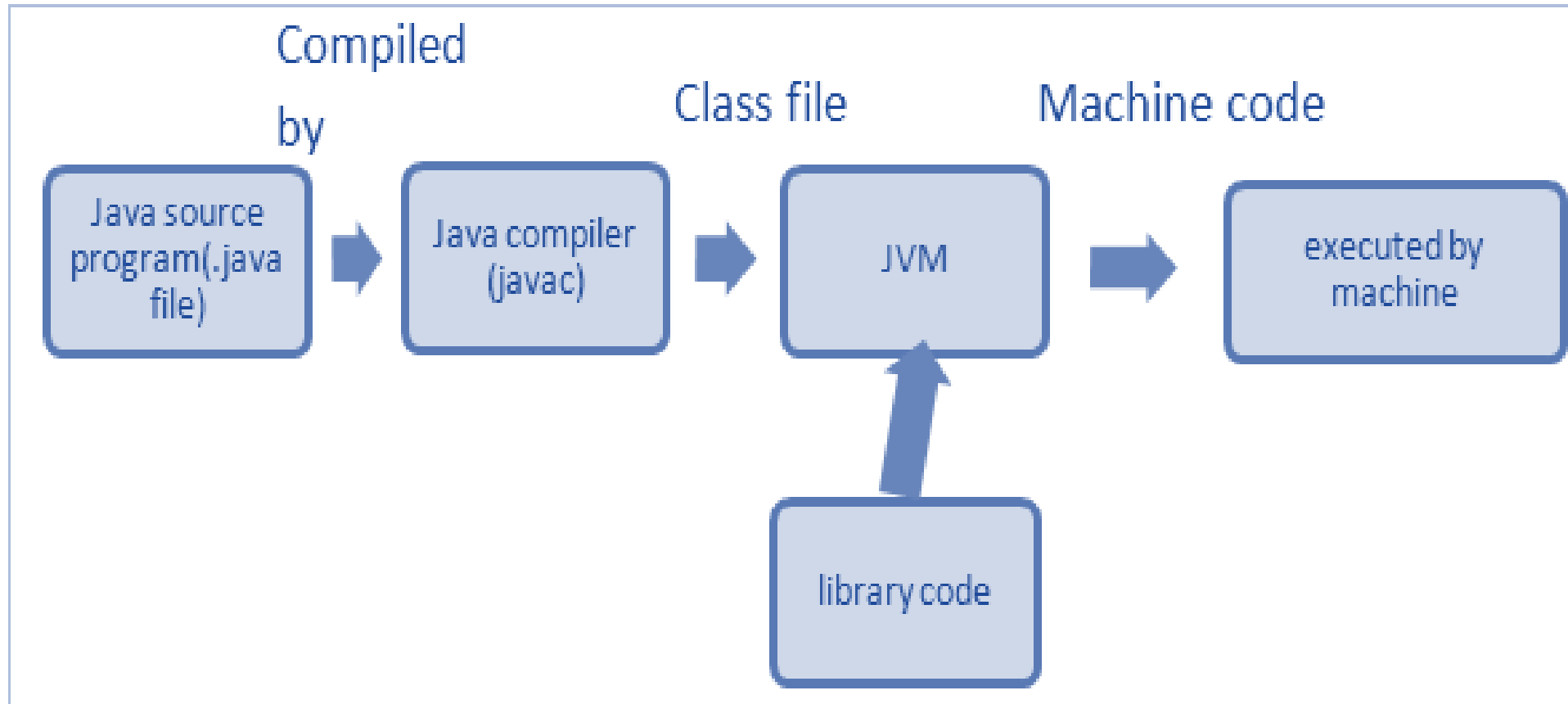
*JDK = JRE + Developer tools.*

<https://www.softwaretestinghelp.com/java-components-java-platform-jdk/>

This ppt is created as a reference material (only for the academic purpose) for the students of PICT. It is restricted only for the internal use and any circulation is strictly prohibited.

Department of Electronics & Telecommunication Engg.

# JAVA Execution Flow



<https://www.softwaretestinghelp.com/java-components-java-platform-jdk/>

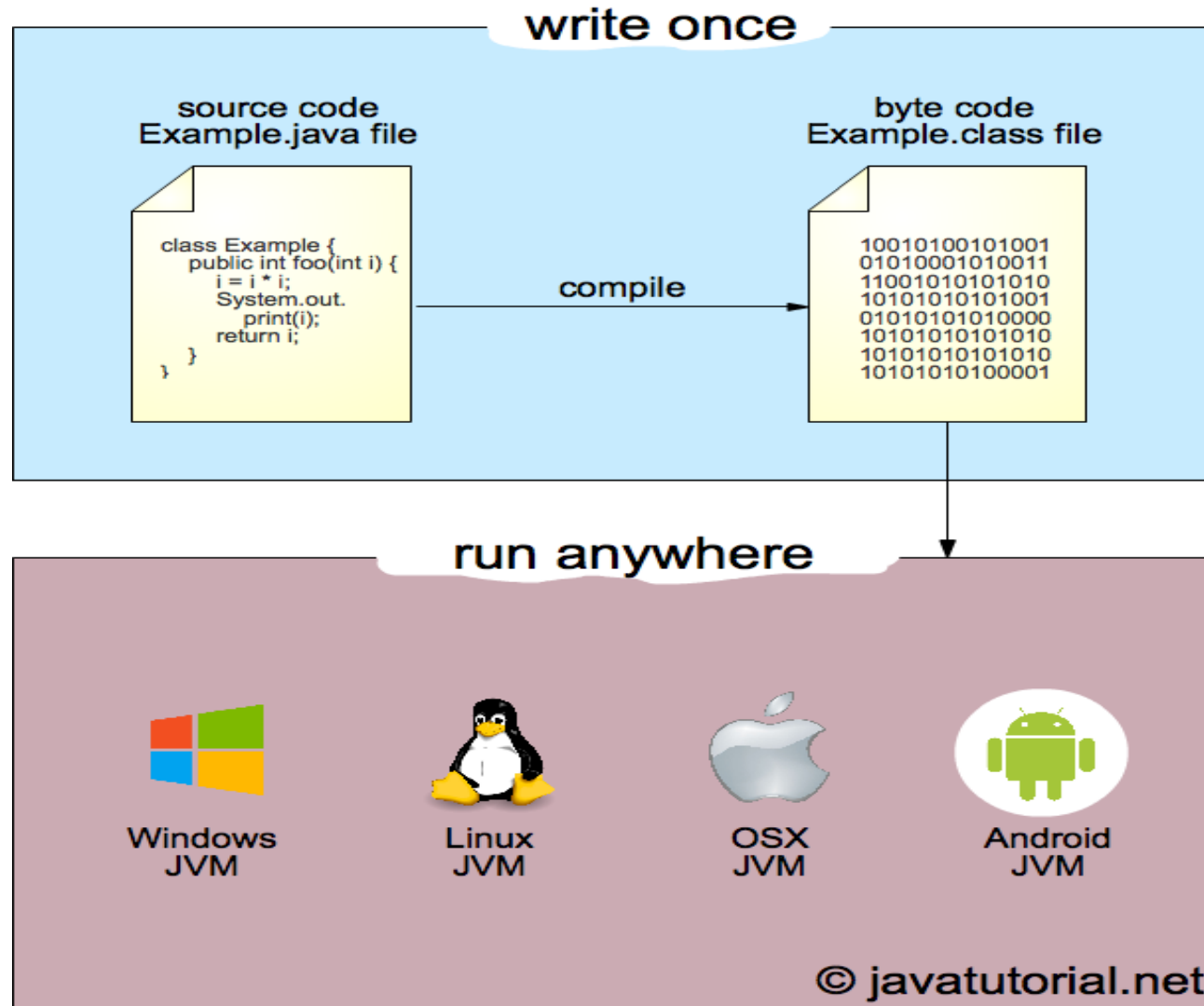
This ppt is created as a reference material (only for the academic purpose) for the students of PICT. It is restricted only for the internal use and any circulation is strictly prohibited.

Department of Electronics & Telecommunication Engg.

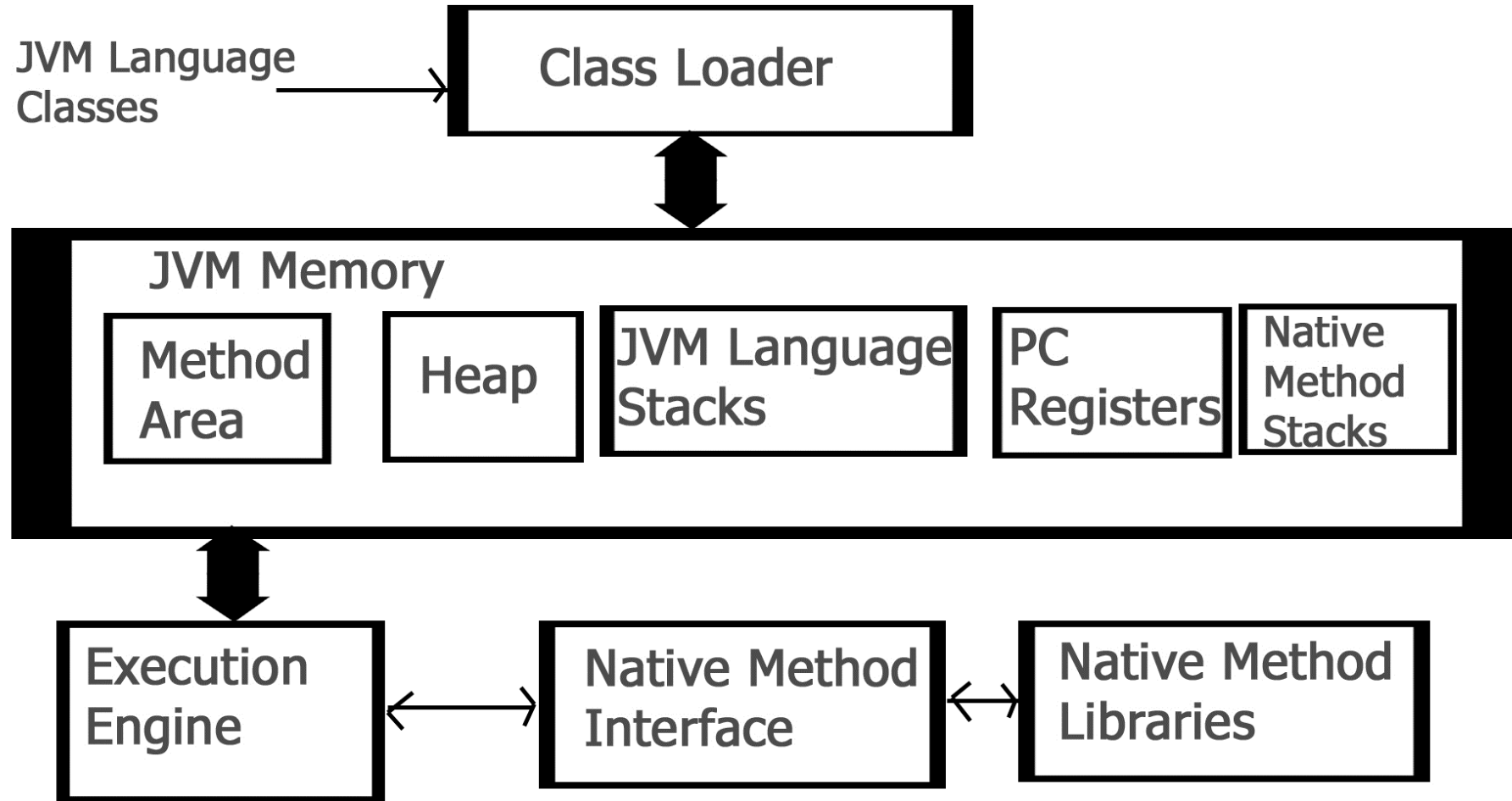


# Java Virtual Machine (JVM)

- Java Virtual Machine is a program that runs pre-compiled Java programs, which mean JVM executes .class files (byte-code) and produces output.
- The JVM is written for each platform supported by Java included in the Java Runtime Environment (JRE).
- JVM allows Java portability to execute within platform and hardware-independent applications.
- It's a big part of the "write once, run anywhere (WORA)" philosophy.
- It is the JRE (JVM plus base classes) that enables Java bytecode to run on any platform.



# JVM Architecture



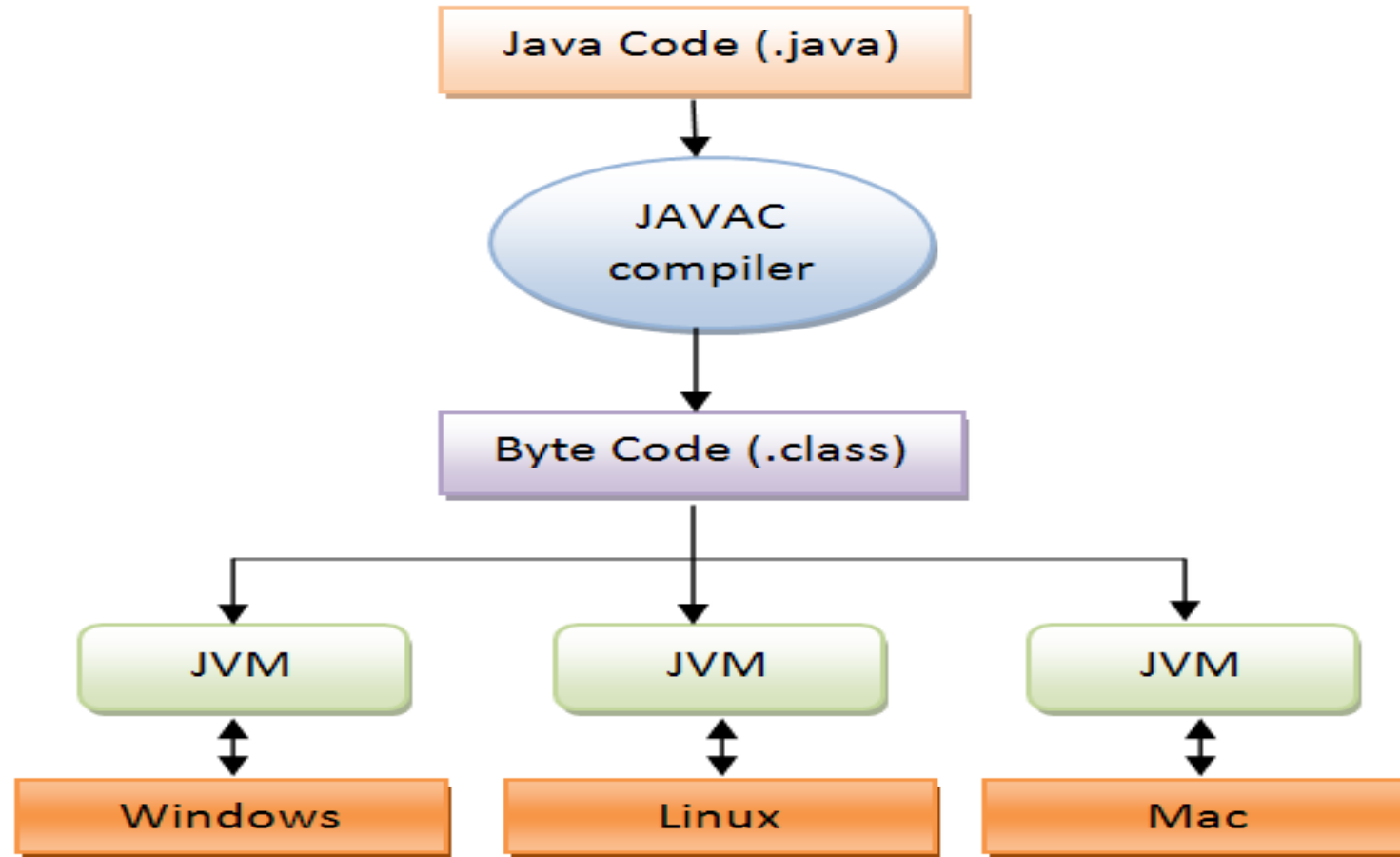


# Java Virtual Machine

- Bytecodes, which are interpreted by the JVM, simply call classes found in the JRE when they need to perform actions, they cannot do by themselves.
- Some actions, like those directed to the underlying hardware or the operating system, are performed by the JVM.
- Bytecodes lack functionality by themselves and need the JVM to do many tasks for them.
- This perceived limitation is an advantage.
  - First, it allows Java programs to be very small compared to other executable programs.
  - Second, and more importantly, it allows them to be very portable.



# Java Virtual Machine: Code Execution





# Java Virtual Machine

- Since each JVM is tailor-made for a specific platform, a Java program cannot run on it unless:
  - An appropriate JVM has been created for it, and;
  - That JVM has been installed on it.
- The portability of Java programs are therefore fully dependent on the presence of a specific JVM.
- Communication between an application and each underlying platform can be very sophisticated, but the JVM handles it well by providing a layer of abstraction between the two.
- As such, developers do not need to be aware of the Complexity involved for each application-platform pair.



# Java Runtime Environment

The JRE includes the following components.

- **Code libraries, property settings, and resource files:** These include files like charsets.jar, rt.jar, etc.
- **DLL files:** Used by Java hotspot client virtual machine and server virtual machine.
- **Java extension files:** For Example, files related to locale specification like localedata.jar
- **Files** required for security management.
- **Applet support classes**



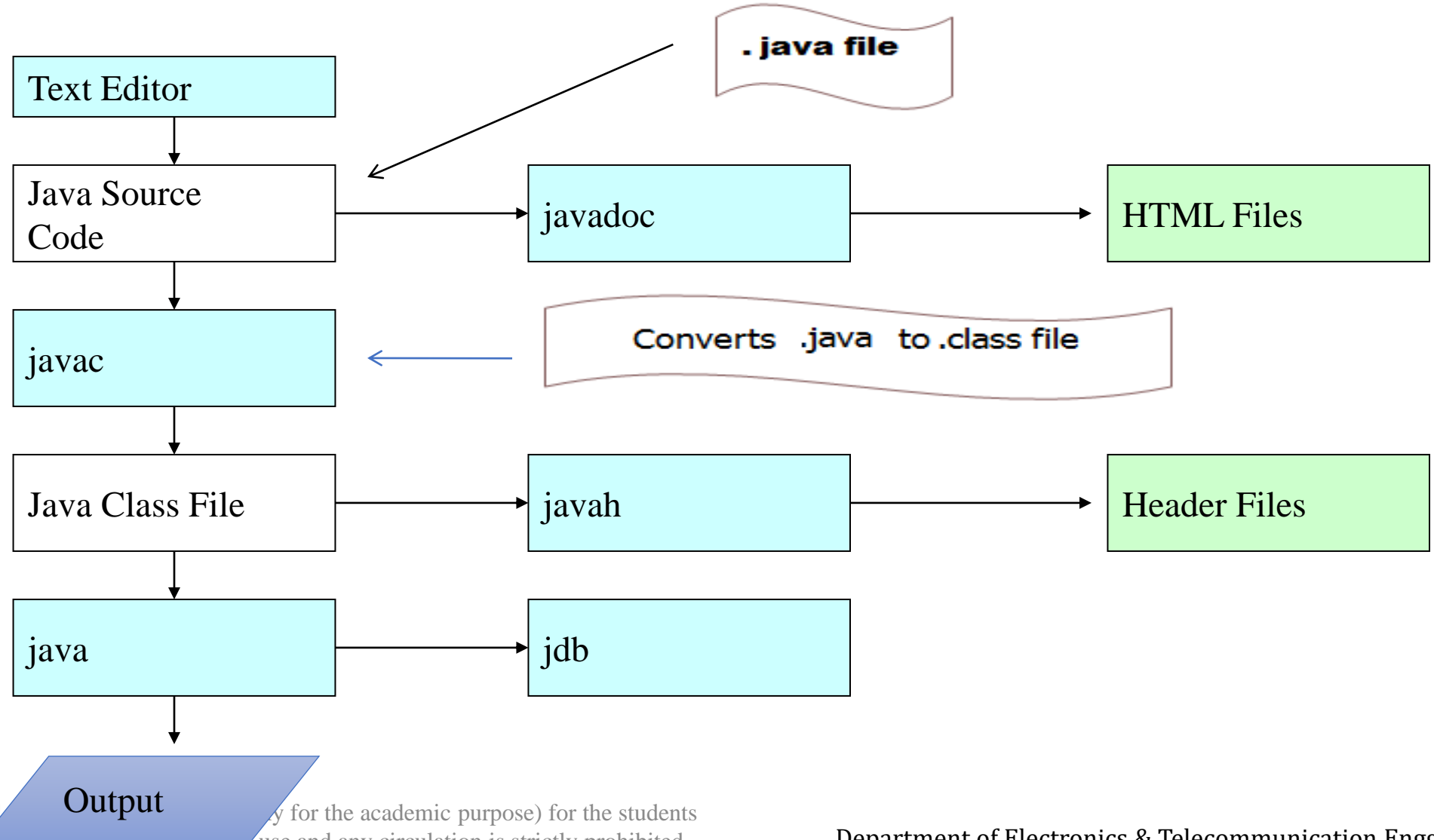
# Java Development Kit

- **javac** - The Java Compiler
- **java** - The Java Interpreter
- **jdb**- The Java Debugger
- **appletviewer** -Tool to run the applets
- **javap** - to print the Java bytecodes
- **javadoc** - documentation generator
- **javah** - creates C header files

# JRE vs JDK

JRE	JDK
JRE stands for Java Runtime Environment.	JDK stands for Java Development Kit.
Mostly used for the execution of Java programs.	JDK is used by developers for developing Java programs.
JRE doesn't have Java compiler so cannot compile programs.	JDK has javac compiler and is responsible for compiling programs.
Contains java class library, the java command, and other infrastructure.	JDK contains tools like Javadoc and archiver that are used to develop Java applications.
JRE can be installed as a standalone program.	JDK is a separate installer and comes bundled with JRE.
Takes the compiled/interpreted Java program as input and generates output.	Compiles Java source program and generates a class file which is then given to JVM.

# Process of Building and Running Java Programs





# Application Programming Interface

- Also called JSL ( Java Standard Library)
- It includes many classes and methods grouped into several packages.
  - 1) **Language support package**: Collection of classes and methods required for implementing basic features of java
  - 2) **Utilities package**: provides utility functions such as date and time
  - 3) **Input/ output package**: required for input/output manipulation
  - 4) **Network package**: collected of classes for communicating with other computer via internet



# Java Applications

- Stand-alone applications
- Web applications (applets)





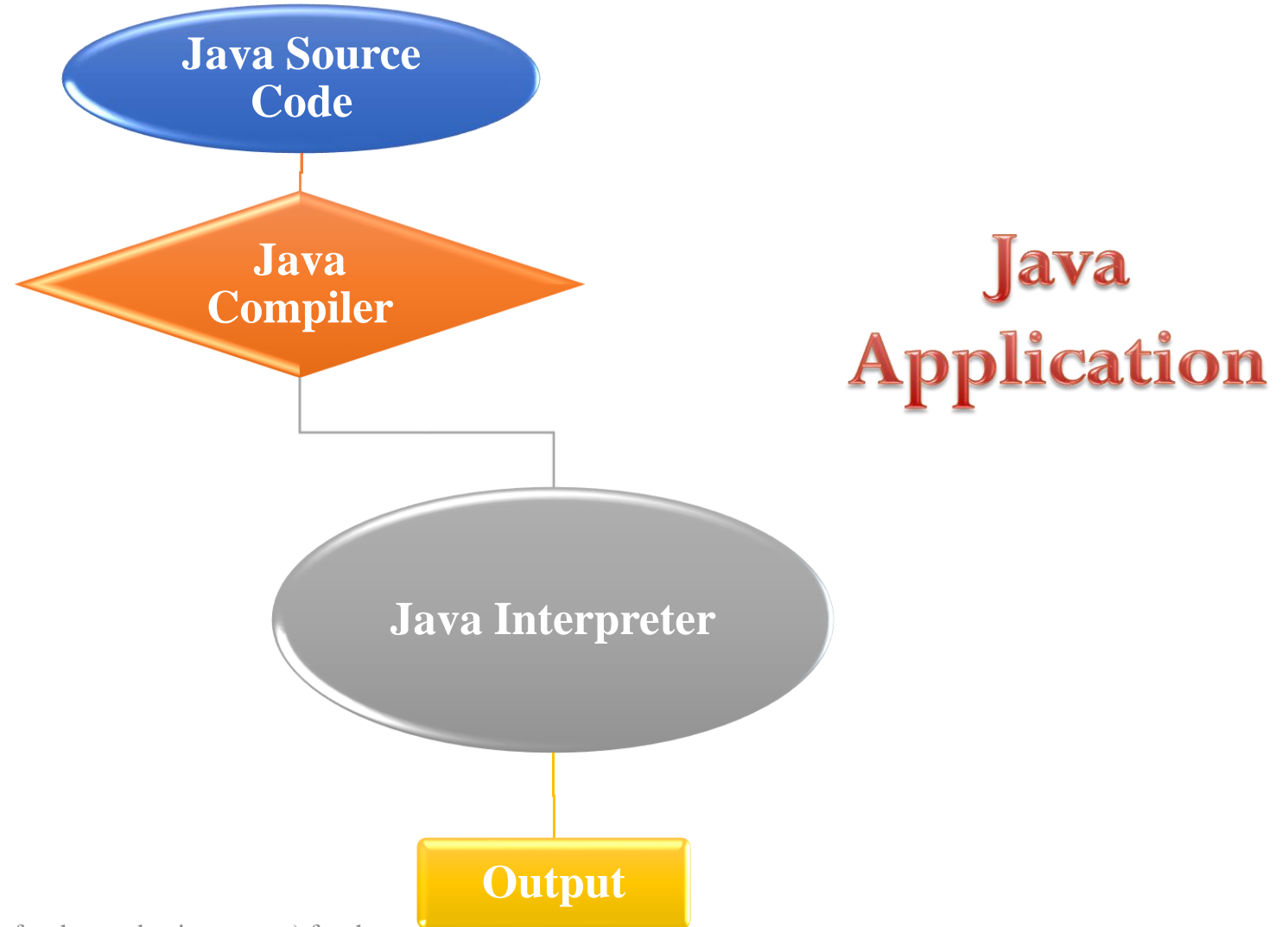
# Application vs Applet

- Different ways to run a Java executable are
- **Application-** A stand-alone program that can be invoked from command line . A program that has a “main” method.
- **Applet-** A program embedded in a web page , to be run when the page is browsed . A program that contains no “main” method.
- Application –Executed by the Java interpreter.
- Applet- Java enabled web browser.



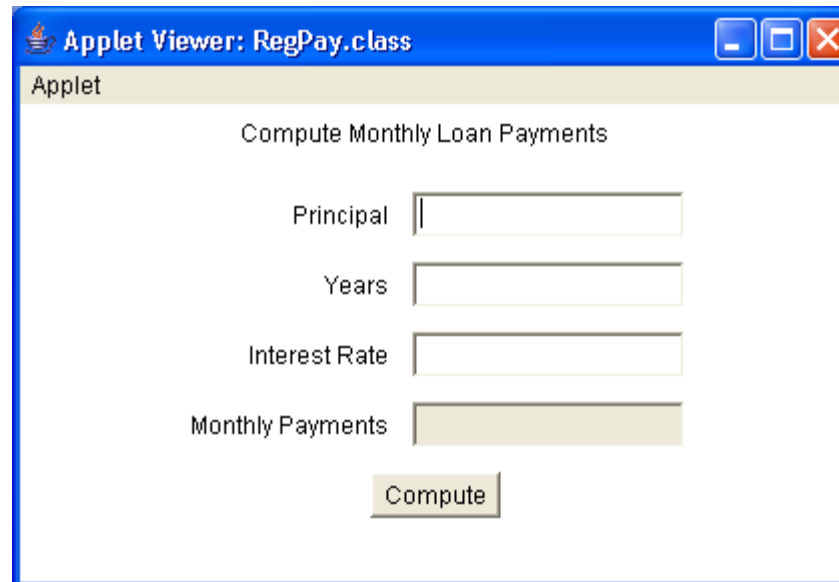
# Standalone Application

- Stand alone applications are programs written in Java to carry out certain tasks on standalone local computer.
- In fact, Java can be used to develop programs for all kind of application.
- Standalone Java program involves two steps:
- Compiling source code into byte codes using javac compiler.
- Executing the byte code program using java interpreter.



# Web Applets

- Applets are small programs developed for internet applications.
- Applet located on distant computer (server) can be downloaded via internet and executed on local computer using a Java web browser.
- Applets can be developed for doing everything from simple animated graphics to complex games utilities.





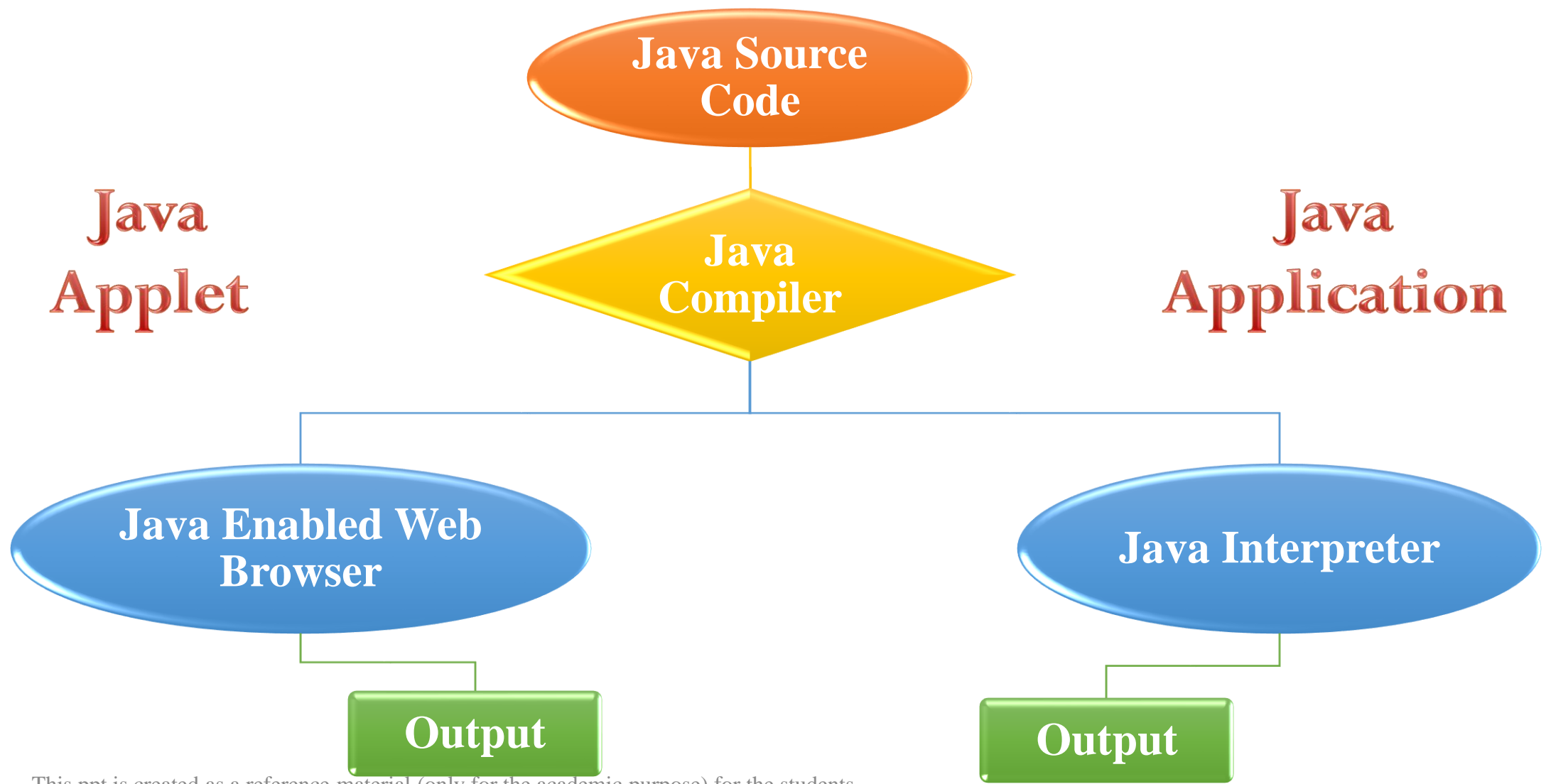
# Java Applet

- An applet is a special kind of Java program that is designed to be transmitted over the Internet and automatically executed by a Java-compatible web browser.
- Furthermore, an applet is downloaded on demand, without further interaction with the user.
- If the user clicks a link that contains an applet, the applet will be automatically downloaded and run in the browser.
- Applets are intended to be small programs.
- They are typically used to display data provided by the server, handle user input, or provide simple functions, such as a loan calculator, that execute locally, rather than on the server.
- In essence, the applet allows some functionality to be moved from the server to the client.



# Java Applet

- All applets are sub-classes (either directly or indirectly) of *java.applet.Applet* class.
- Applets are not stand-alone programs. Instead, they run within either a web browser or an applet viewer. JDK provides a standard applet viewer tool called applet viewer.
- In general, execution of an applet does not begin at `main()` method.
- Output of an applet window is not performed by `System.out.println()`. Rather it is handled with various AWT methods, such as `drawString()`.





# The First Program

```
// hello.java: Hello world program
class HelloWorld
{
    public static void main(String args[])
    {
        System.out.println("Hello World");
    }
}
```





# Execution of Program

- Compilation

```
# javac HelloWorld.java
```

**results in HelloWorld.class**

- Execution

```
# java HelloWorld
```

**Hello World**

# Simple Java Program

```
class sampleone
```

```
{
```

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

```
    {
```

```
        System.out.println("Java is Programming Language");
```

```
    }
```

```
}
```



**1) Class Declaration**



**3) Output Line**

# Class Declaration

- First line

**class sampleone**

declares a class, which is object-oriented construct.

- Java is truly object-oriented language and therefore everything must be placed inside the class.
- sampleone is a java identifier that specifies a name of class.

## Opening Brace

- Every class definition in Java begin with an opening brace { and end with closing brace }, similar to C++ class construct but in C++ class ends with semicolon.

## **public static void main (string args[ ])**

- Defines a method named main.
- This is similar to the main() function in C++.
- Every Java application program must include the main () method.
- It is starting point for the interpreter to begin the execution of the program.
- Java application can have many number of classes but only one of them must include a main method to initiate the execution (Note that Java applets will not use the main method at all).



# Main Line

- **Public:** Keyword public is an **access specifier** that declares the main method as unprotected and accessible to all other classes.
- **Static:** Next appears the keyword static, which declares this method as one that belongs to the entire class and not a part of any object of the class.
- Main method always declared as static since the interpreter uses this method before any object are created.
- **Void:** type modifier void states that main method does not return any value.
- All parameters to method are declared inside a pair of parentheses.
- String arg[] declares a parameter name arg, which contains an array of objects of the class type string.



# Output Line

- The only executable statement in the program is

`System.out.println("java is programming language");`

- Similar to the `printf()` statement of C or `cout<<` construct of C++.
- Java is true object oriented language, every method must be part of an object.
- `Println` method is a member of the `out` object, which is static data member of `system` class.
- `Println` always appends a newline character to the end of the string.

Class Name

File Name: `Simple.java`

Main method

Comment

```
class Simple {  
    // This is my first java program.  
    public static void main(String args[])  
    {  
        System.out.println("Hello World!");  
    }  
}
```

access modifier/  
specifier

Method return type

Body of Main method



# Valid Java Main Method Signature

- **public static void** main(String[] args)
- **public static void** main(String []args)
- **public static void** main(String args[])
- **public static void** main(String... args)
- **static public void** main(String[] args)
- **public static final void** main(String[] args)
- **final public static void** main(String[] args)





# Invalid Java Main Method Signature

- **public void** main(String[] args)
- **static void** main(String[] args)
- **public void static** main(String[] args)
- **abstract public static void** main(String[] args)



# Installing and Configuring Java

## IDE:

An **I**ntegrated **D**evelopment **E**nvironment (**IDE**) is a software application that provides comprehensive facilities to computer programmers for developing software with ease.

An IDE normally consists of,

- **Source code editor** – Used for Creating/editing source code.
- **Compiler and/or Interpreter** – Converts high level source code into low level assembly language.
- **Build Automation tools** – Automation of packaging & deployment tools.
- **Debugger** – Used to identify run time errors in the source code.



# Implementing Java Programs

- Creating the program
- Compiling the program
- Running the program

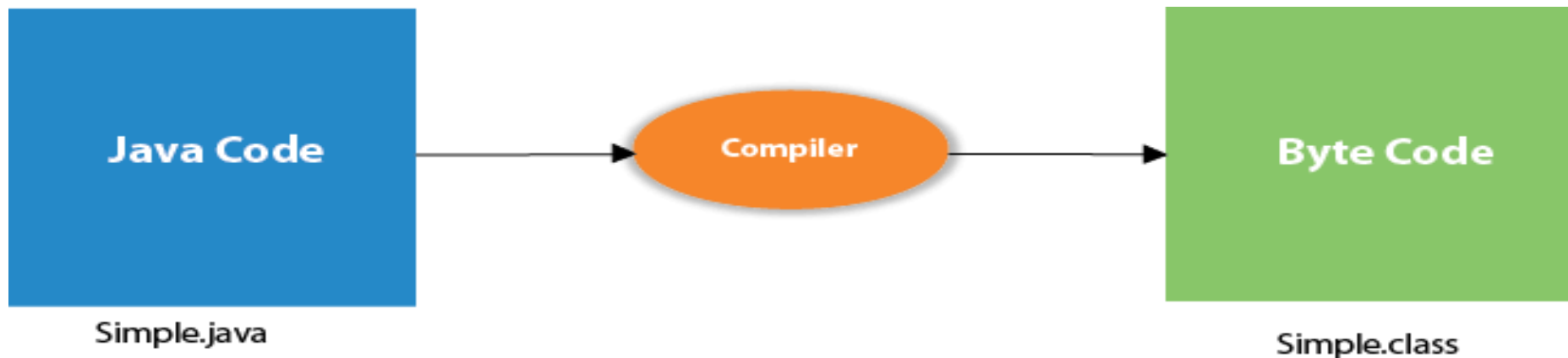


# Creating the Program

- Save file as class name e.g. Simple.java
- Same name as that of class name.
- Called as source file.
- May have multiple classes.
- One of them must contain the 'main method'.
- The file name must be the classname of class containing the 'main method'.

# Compiling and Running the Program

- javac (java compiler) is responsible for compiling the source code.
- converts .java to .class (Byte code).
- Byte code are machine independent, can be run on any machine.
- java (java interpreter) is responsible for running the compiled code.
- Reads the byte code file and translate them into machine code for specific machine.
- Interpreter is specially written for each type of machine.
- Interpreter looks for the 'main method' in the program and begin execution from there.





# Java Program Editing

- Any text editor can be used to write Java Program  
e.g. In windows :- Notepad, Edit, Wordpad, MS-Word etc.  
In Unix :- vi, emacs, gedit etc.
- Save the Program
  - Save the program in a file with the name  
Simple.java

# Implementing Java Code

1. To write the simple program, you need to open notepad by **start menu -> All Programs -> Accessories -> notepad** and write a simple program and save as `classname .java`
2. To compile and run this program, you need to open the command prompt by **start menu -> All Programs -> Accessories -> command prompt**.
3. To compile and run the above program, go to your current directory first; e.g. my current directory is `c:\Java Programming`.

**To compile:** `javac filename.java`

**To execute:** `java filename`

# Resolving an error "javac is not recognized as an internal or external command"?

- To solve this problem, set path.
- Since DOS doesn't know javac or java, we need to set path.
- The path is not required in such a case if the program is inside the JDK/bin directory.  
However, it is an excellent approach to set the path.
- If the Java file is outside the JDK/bin folder, it is necessary to set the path of JDK.
- There are two ways to set the path in Java:
  - Temporary
  - Permanent





# Setting the temporary path

- To set the temporary path of JDK:
  - Open the command prompt
  - Copy the path of the JDK/bin directory
  - Write in command prompt: `set path=copied_path`



# Setting the permanent path

For setting the permanent path of JDK:

Go to My Computer properties -> advanced tab -> environment variables -> new tab of user variable -> write path in variable name -> write path of bin folder in variable value -> ok -> ok -> ok

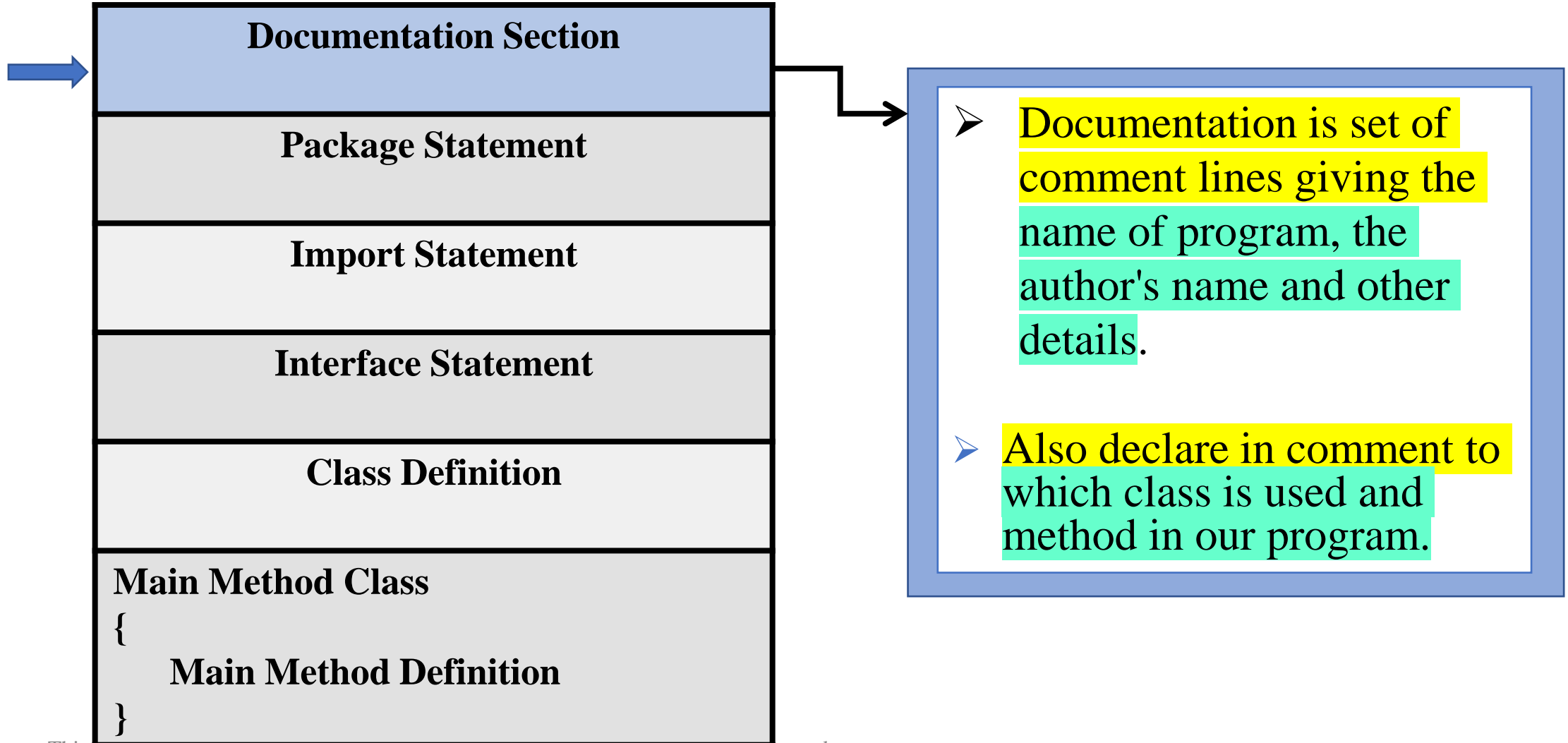


# Java Program Structure

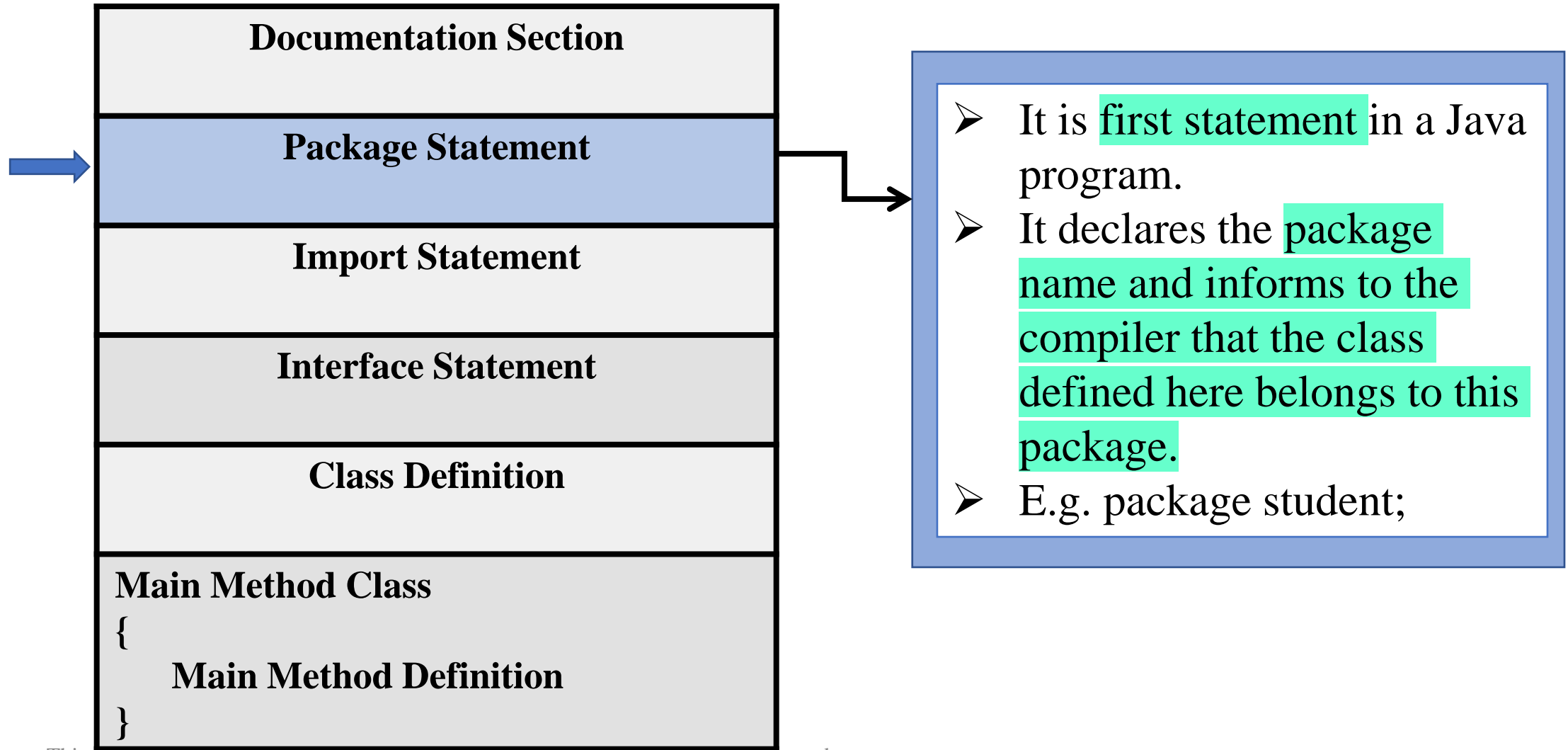
<b>Documentation Section</b>
<b>Package Statement</b>
<b>Import Statement</b>
<b>Interface Statement</b>
<b>Class Definition</b>
<b>Main Method Class</b> { <b>Main Method Definition</b> }

This ppt is created as a reference material (only for the academic purpose) for the students of PICT. It is restricted only for the internal use and any circulation is strictly prohibited.

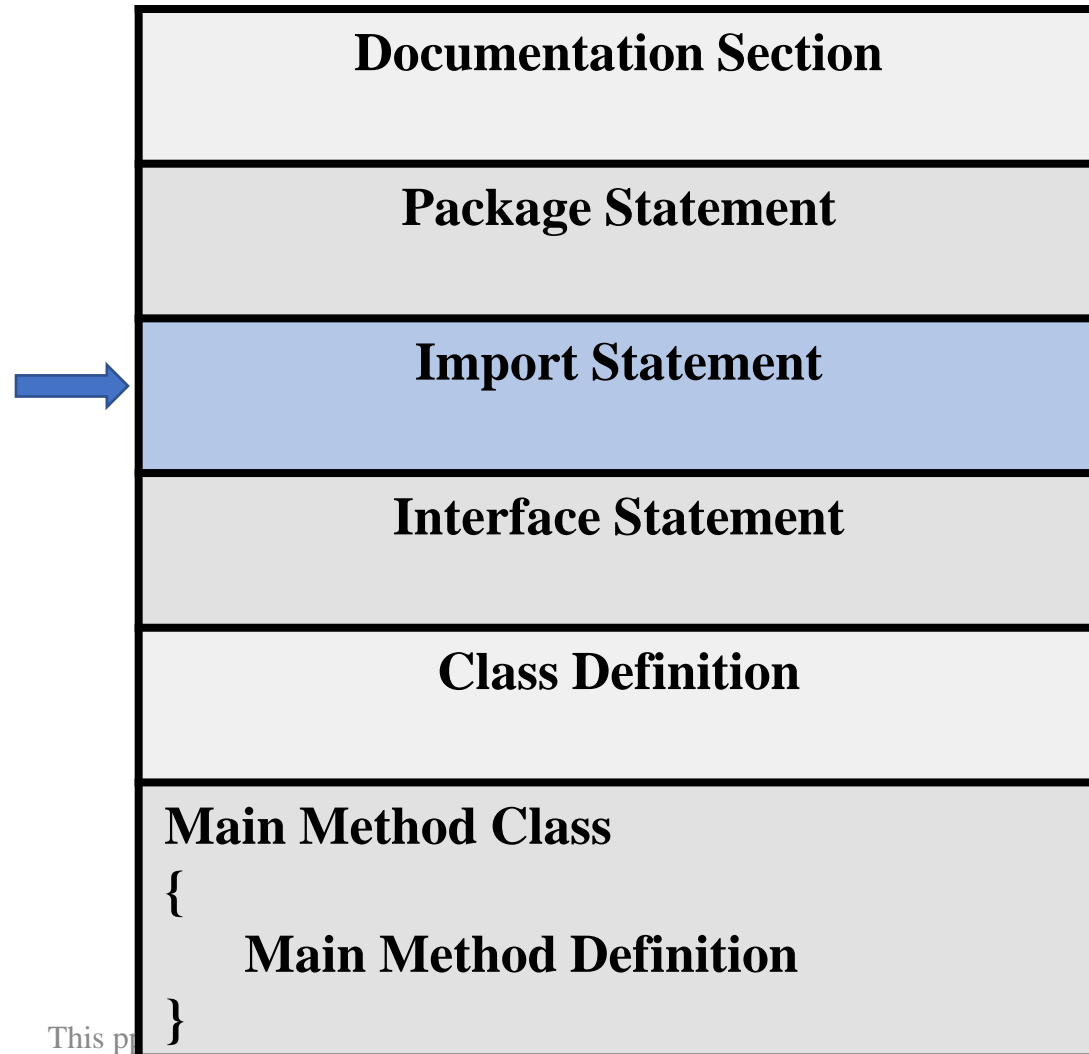
# Java Program Structure



# Java Program Structure

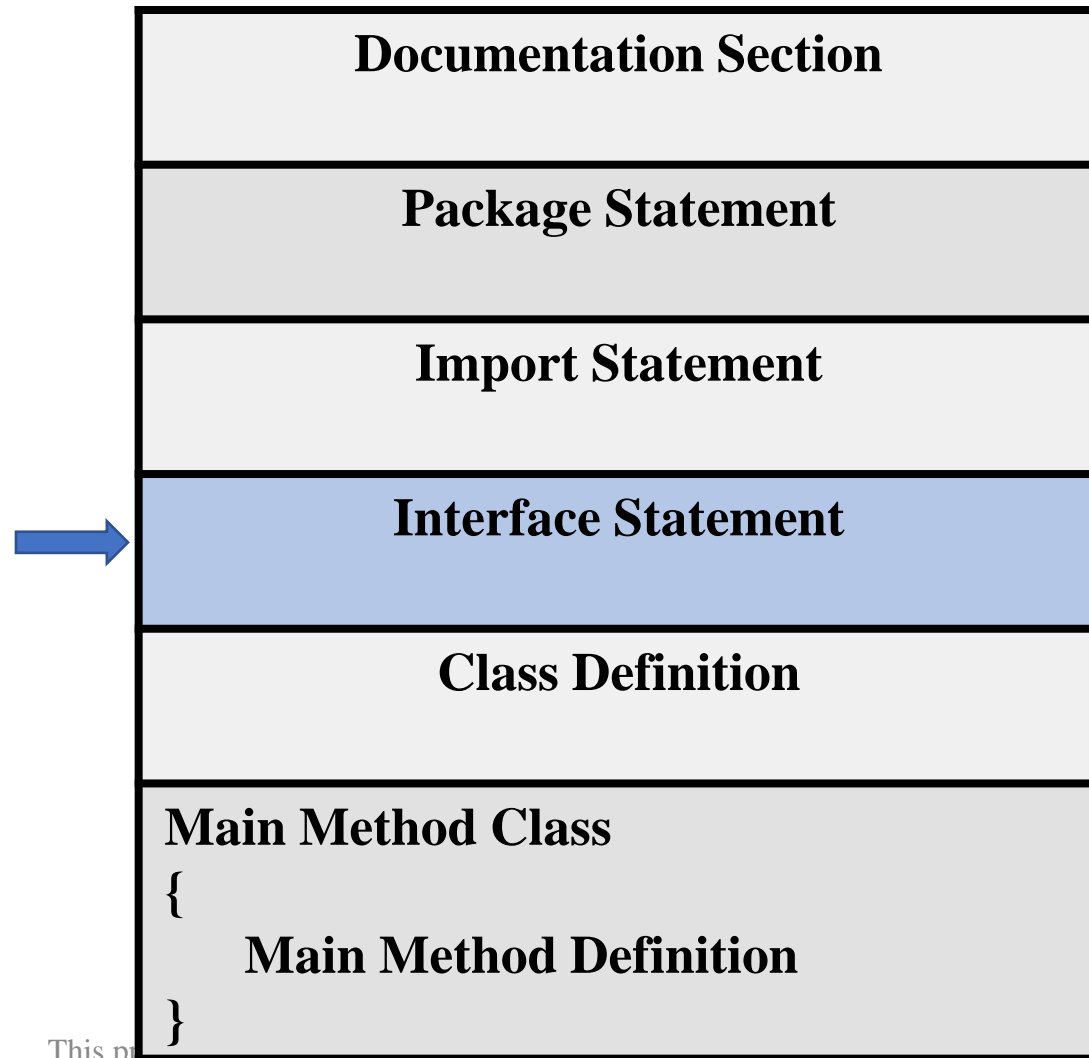


# Java Program Structure



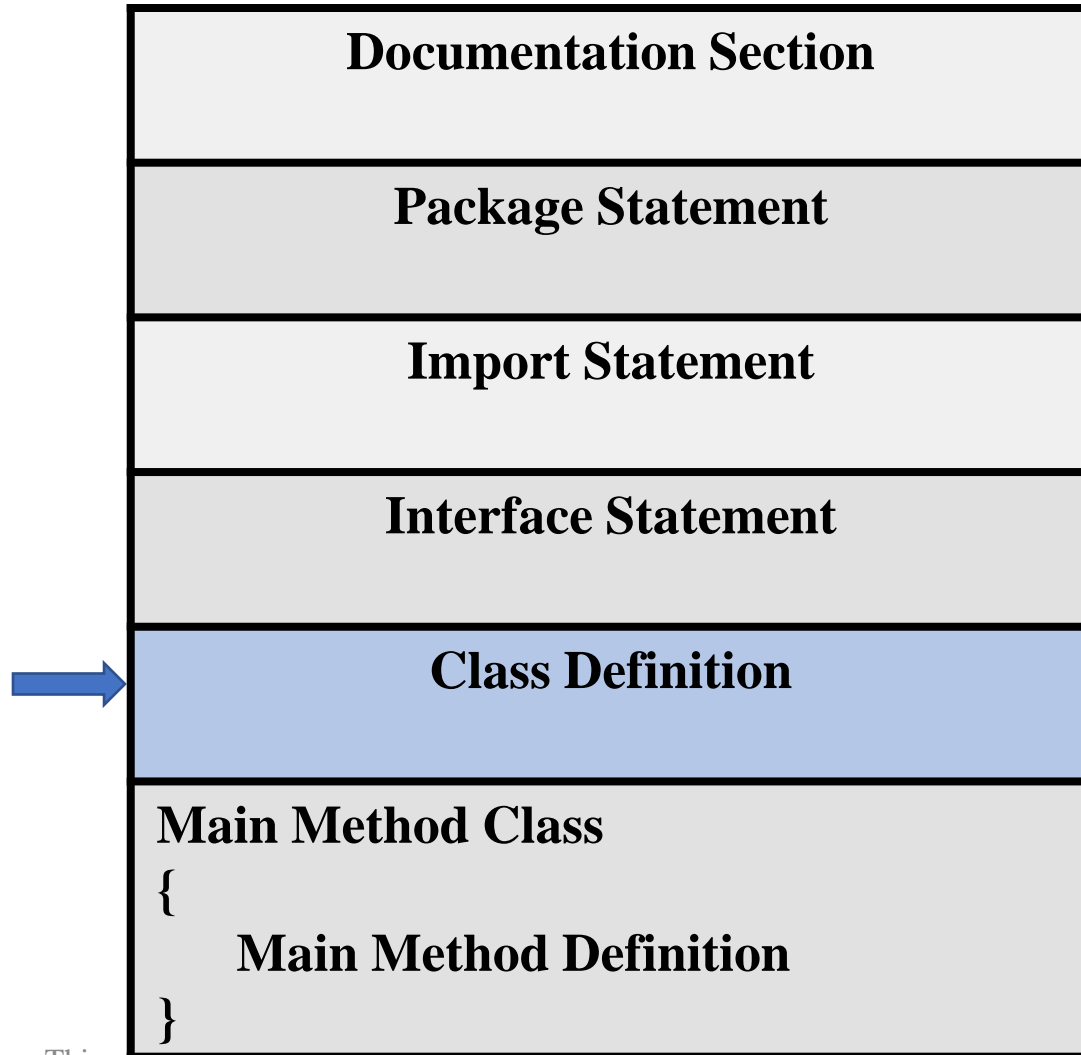
- Next thing after a package statement may be number of import statements .
- Similar to the #include statement in C and C++.
- Instructs the interpreter to load class contained in the particular package.

# Java Program Structure



- Interface is like a class but includes a group of method declarations.
- It is used when we wish to implement the multiple inheritance.

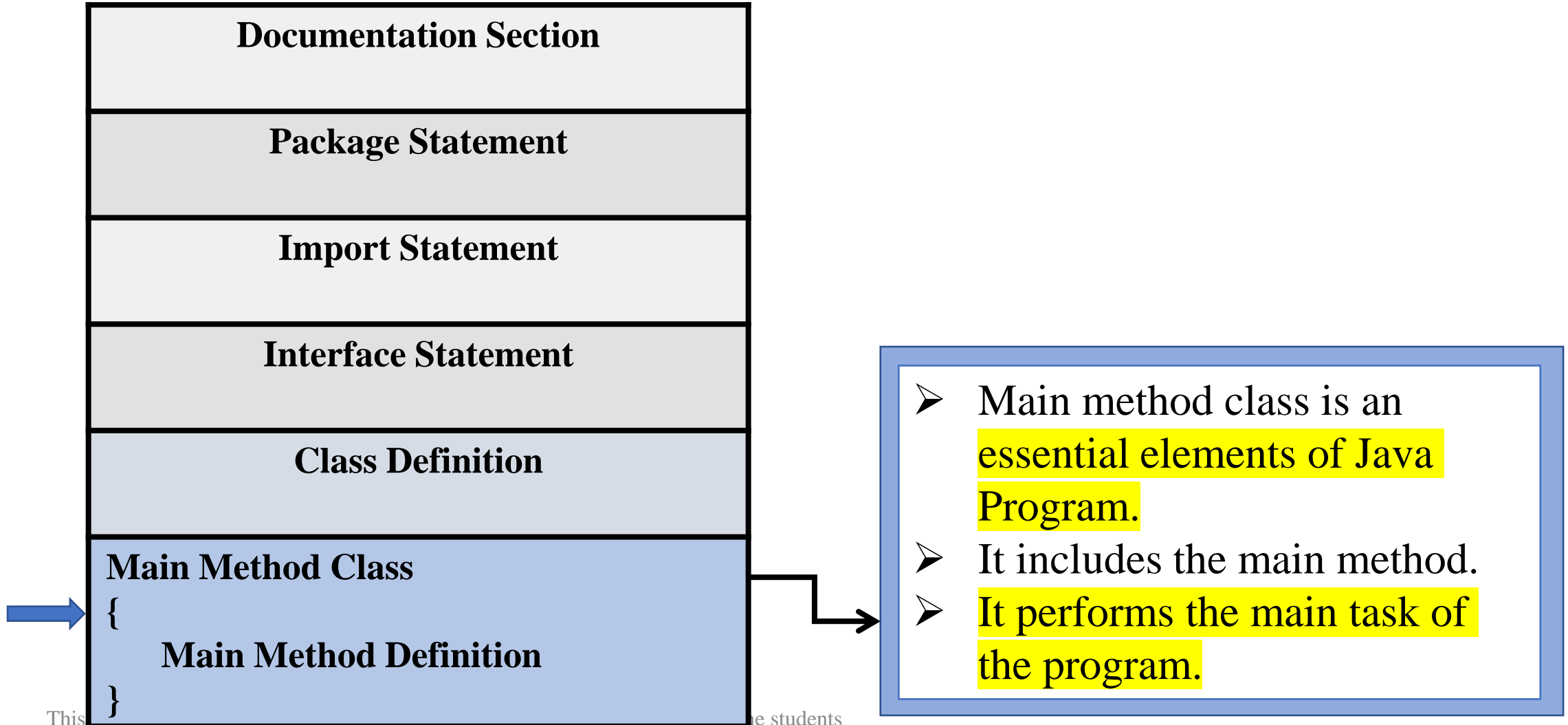
# Java Program Structure



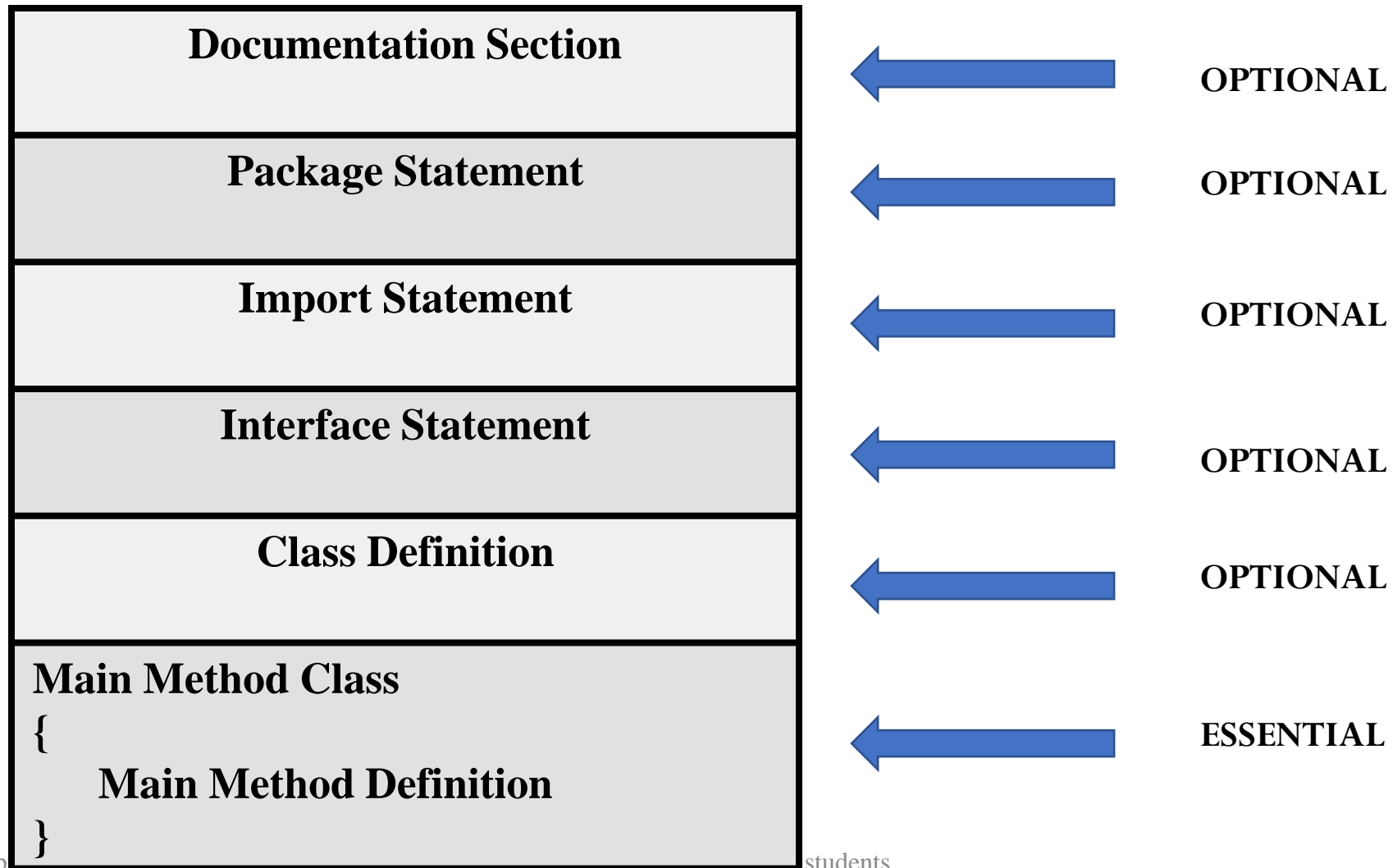
- Classes are primary and essential elements of Java Program.
- It maps the real-world problems.
- No of classes in a program depends on the complexity of the program.



# Java Program Structure



# Java Program Structure



This presentation is the property of PICT. It is restricted only for the internal use and any circulation is strictly prohibited.



# Sample Questions

```
public class PICT
{
    public static void main (String args[])
    {
        System.out.println(100 + 100 + "PICT");
        System.out.println("E-Learning" + (100 + 100));
    }
}
```



# Sample Questions

```
public class Question1{  
    public static void main(String args[]){  
        for(int a=1;a<3;a+=3){  
            System.out.print(--a);  
        }  
    }  
}
```

# Sample Questions

```
public class Question{  
    public static void main(String args[]){  
        int f = 0, g = 1;  
        for (int i = 0; i <= 5; i++)  
        {  
            System.out.println(f);  
            f = f + g;  
            g = f - g;  
        }  
    }  
}
```



# JAVA Tokens

A token is the smallest element of a program that is meaningful to the compiler. Tokens can be classified as follows:

- Keywords
- Identifiers
- Constants
- Special Symbols
- Operators



# Keywords

- Keywords are pre-defined or reserved words in a programming language.
- Each keyword is meant to perform a specific function in a program.
- Since keywords are referred names for a compiler, they can't be used as variable names because by doing so, we are trying to assign a new meaning to the keyword which is not allowed.
- Java language has reserved 50 words as keywords.
- All keywords are written in lower case letters.

# Keywords

<b>abstract</b>	<b>assert</b>	<b>boolean</b>
<b>break</b>	<b>byte</b>	<b>case</b>
<b>catch</b>	<b>char</b>	<b>class</b>
<b>const</b>	<b>continue</b>	<b>default</b>
<b>do</b>	<b>double</b>	<b>else</b>
<b>enum</b>	<b>exports</b>	<b>extends</b>
<b>final</b>	<b>finally</b>	<b>float</b>
<b>for</b>	<b>goto</b>	<b>if</b>
<b>implements</b>	<b>import</b>	<b>instanceof</b>
<b>int</b>	<b>interface</b>	<b>long</b>
<b>module</b>	<b>native</b>	<b>new</b>
<b>open</b>	<b>opens</b>	<b>package</b>
<b>private</b>	<b>protected</b>	<b>provides</b>
<b>public</b>	<b>requires</b>	<b>return</b>
<b>short</b>	<b>static</b>	<b>strictfp</b>
<b>super</b>	<b>switch</b>	<b>synchronized</b>
<b>this</b>	<b>throw</b>	<b>throws</b>
<b>to</b>	<b>transient</b>	<b>transitive</b>
<b>try</b>	<b>uses</b>	<b>void</b>
<b>volatile</b>	<b>while</b>	<b>with</b>





# Identifiers

- Identifiers are used as the general terminology for naming of variables, functions and arrays.
- These are user-defined names consisting of an arbitrarily long sequence of letters and digits with either a letter or the underscore(\_) as a first character.
- Identifier names must differ in spelling and case from any keywords.
- Keywords can not be used as identifiers; they are reserved for special use.
- Once declared, you can use the identifier in later program statements to refer to the associated value.
- A special kind of identifier, called a statement label, can be used in goto statements.

Valid Examples: myvariable, a, abc123

Invalid Examples: my variable, 123abc, variable-2



# Constants/ Literals

- Constants are also like normal variables.
- The values can not be modified by the program once they are defined.
- Constants refer to fixed values. They are also called as literals.
- Constants may belong to any of the data type.

Syntax:

*final data\_type variable\_name;*

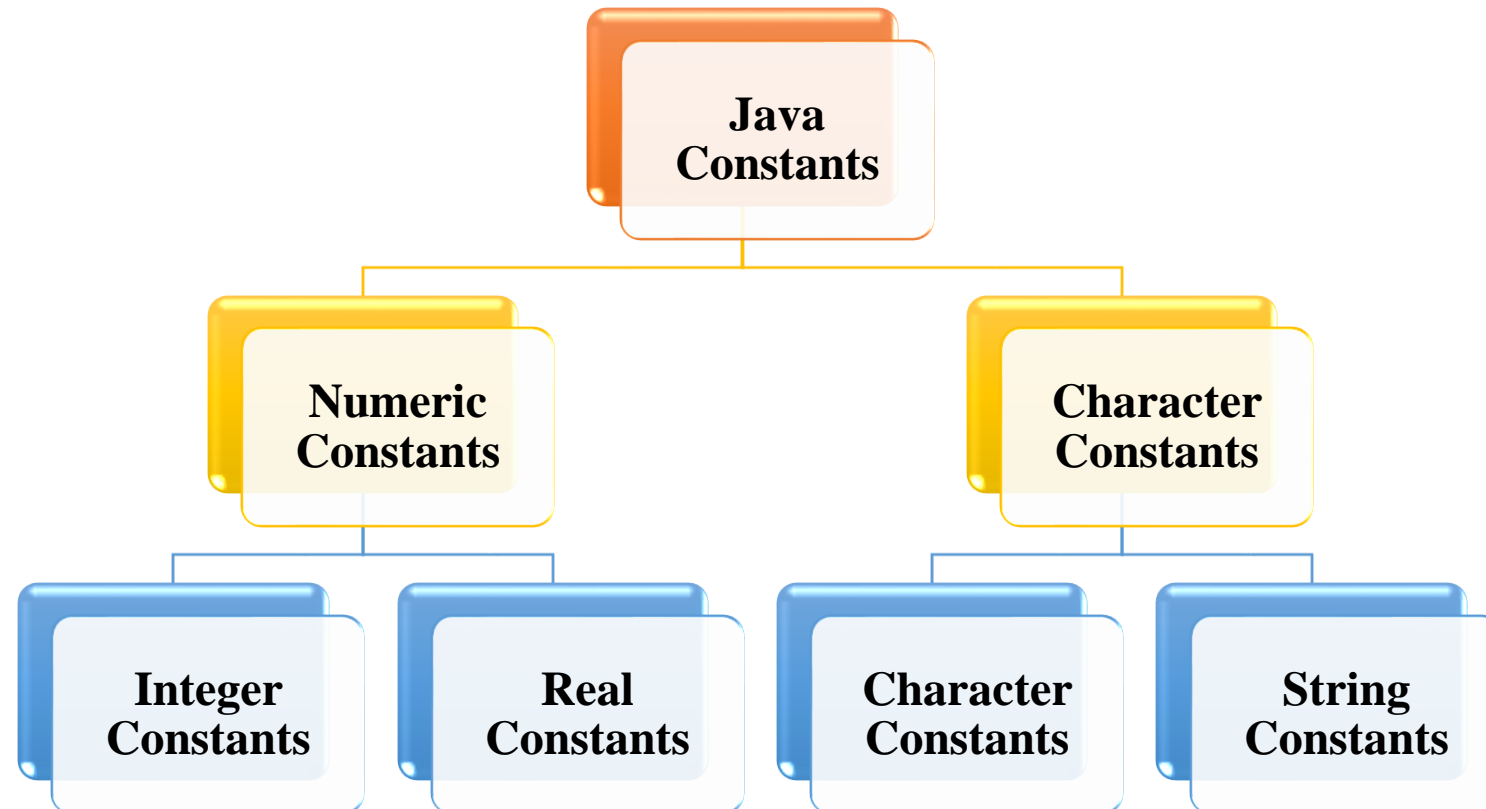
# Separators

- Separators are the special symbols used to indicate where groups of code are divided and arranged.
- They basically define the shape and function of the code.

Name	What it is used for
Parentheses ( )	To enclose parameters in method definition and invocation. For defining priority in expression, containing expressions for flow control, and surrounding cast types.
Braces { }	To define a block of code for classes, methods and local scopes. To contain the values of automatically initialized Array.
Brackets [ ]	To declare array types and for dereferencing array values.
Semicolon ;	To separate statements.
Comma ,	To separate consecutive identifiers in variable declaration.
Period .	To separate package name from sub-packages and classes; also used to separate a variable or method from a reference variable
assignment operator:	To assign values

# Constants

- Constants in java refer to fixed values that do not change during the execution of program.
- Java supports to several types of constants.



This ppt is created as a reference material (only for the academic purpose) for the students of PICT. It is restricted only for the internal use and any circulation is strictly prohibited.



# Constants

- Integer Constants:- Sequence of digit
  - Decimal – 123, -987, 654321
  - Octal – 037, 0435, 0551
  - Hexadecimal – 0x2, 0xbf, 0xff
- Real Constants
  - Real or floating point – 0.009, -45.88, - . 943
  - exponential – 0.65e4
- Character Constants
  - ‘5’            ‘x’            ‘;’            ‘ ‘
- String Constants
  - “Hello Java”            “1987”            “:.)”            “#\$%”

This ppt is provided as reference material (only for the academic purpose) for the students of PICT. It is restricted only for the internal use and any circulation is strictly prohibited.

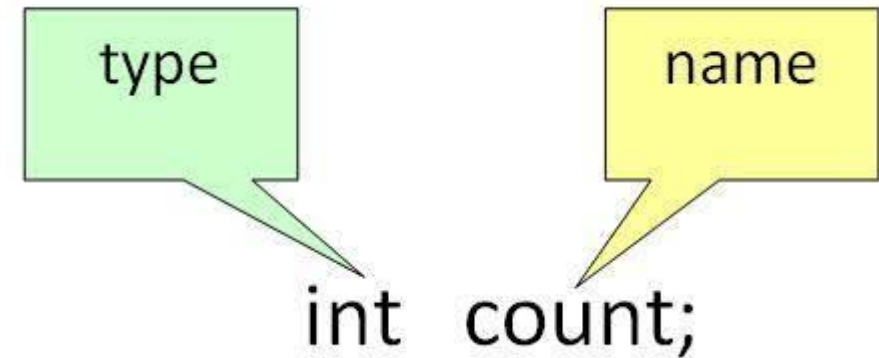
# Constants

- Backslash Character Constants (Escape sequence)

Constant	Meaning
'\b'	Backspace
'\f'	Form feed
'\n'	New line
'\r'	Carriage return
'\t'	Horizontal tab
' ''	Single quote
' "" '	Double quote
' \\'	Backslash

# Variables

- Variable in Java is a data container that saves the data values during Java program execution.
- Every variable is assigned a data type that designates the type and quantity of value it can hold.
- A variable is a memory location name for the data.
- A variable is a name given to a memory location. It is the basic unit of storage in a program.
- The value stored in a variable can be changed during program execution.
- A variable is only a name given to a memory location. All the operations done on the variable affect that memory location.
- In Java, all variables must be declared before use.



The diagram illustrates the components of a variable declaration in Java. It shows the code `int count;`. A green callout box labeled 'type' points to the `int` keyword, indicating the data type. A yellow callout box labeled 'name' points to the `count` identifier, indicating the variable name.



# Variables

- **datatype:** Type of data that can be stored in this variable.
- **data\_name:** Name given to the variable.

In this way, a name can only be given to a memory location. It can be assigned values in two ways:

- Variable Initialization
- Assigning value by taking input





# How to initialize the variables?

It can be perceived with the help of 3 components that are as follows:

- **datatype:** Type of data that can be stored in this variable.
- **variable\_name:** Name given to the variable.
- **value:** It is the initial value stored in the variable.

Examples:

```
int a=1, b=20;
```

```
float c;
```



# Declaration of Variables

- Variable:
  - The names of storage locations.
  - Must be declared before it is used.
- Declaration does three things
  - It tells compiler what the variable name is.
  - It specifies what type of data the variable will hold.
  - The place of declaration (in the program) decides the scope of variable.

# Types of Variables

## Local

- A variable defined within a block or method, or constructor is called a local variable.
- The scope of these variables exists only within the block in which the variables are declared, i.e., we can access these variables only within that block.
- Initialization of the local variable is mandatory before using it in the defined scope.

## Instance

- Instance variables are non-static variables and are declared in a class outside of any method, constructor, or block.
- As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed.
- Initialization of an instance variable is not mandatory. Its default value is 0.
- Instance variables can be accessed only by creating objects.

# Types of Variables

## Static

- The static variables are declared using the static keyword within a class outside of any method, constructor or block.
- Static variables are created at the start of program execution and destroyed automatically when execution ends.
- Initialization of a static variable is not mandatory. Its default value is 0.
- If we access a static variable without the class name, the compiler will automatically append the class name.



# Scope of Variables

- Instance variable
  - Declared inside the class
  - Created when objects are declared, associated with objects
  - Different values for different objects



# Scope of Variables

- Class variable (Static Variable)
  - Declared inside the class
  - Global to class
  - Belongs to entire set of objects
  - Only one memory location is created for each



# Scope of Variables

- Local variable
  - Declared inside the method
  - Not available for use of outside the method definition
  - Can also be declared inside blocks that are defined between an opening { and closing brace}.
  - These variables are visible to the program only from the beginning of its program block to the end of the program block.
  - The area of program where the variable is accessible is called its scope.
  - Program block within another program block is called nesting



# Rules to declare Variables

- A variable name can consist of Capital letters A-Z, lowercase letters a-z digits 0-9, and two special characters such as \_ underscore and \$ dollar sign.
- The first character must not be a digit.
- Blank spaces cannot be used in variable names.
- Java keywords cannot be used as variable names.
- Variable names are case-sensitive.
- There is no limit on the length of a variable name but by convention, it should be between 4 to 15 chars.
- Variable names always should exist on the left-hand side of assignment operators.



# Question 1

```
class Main {  
    public static void main(String args[]) {  
        int t;  
        System.out.println(t);  
    }  
}
```

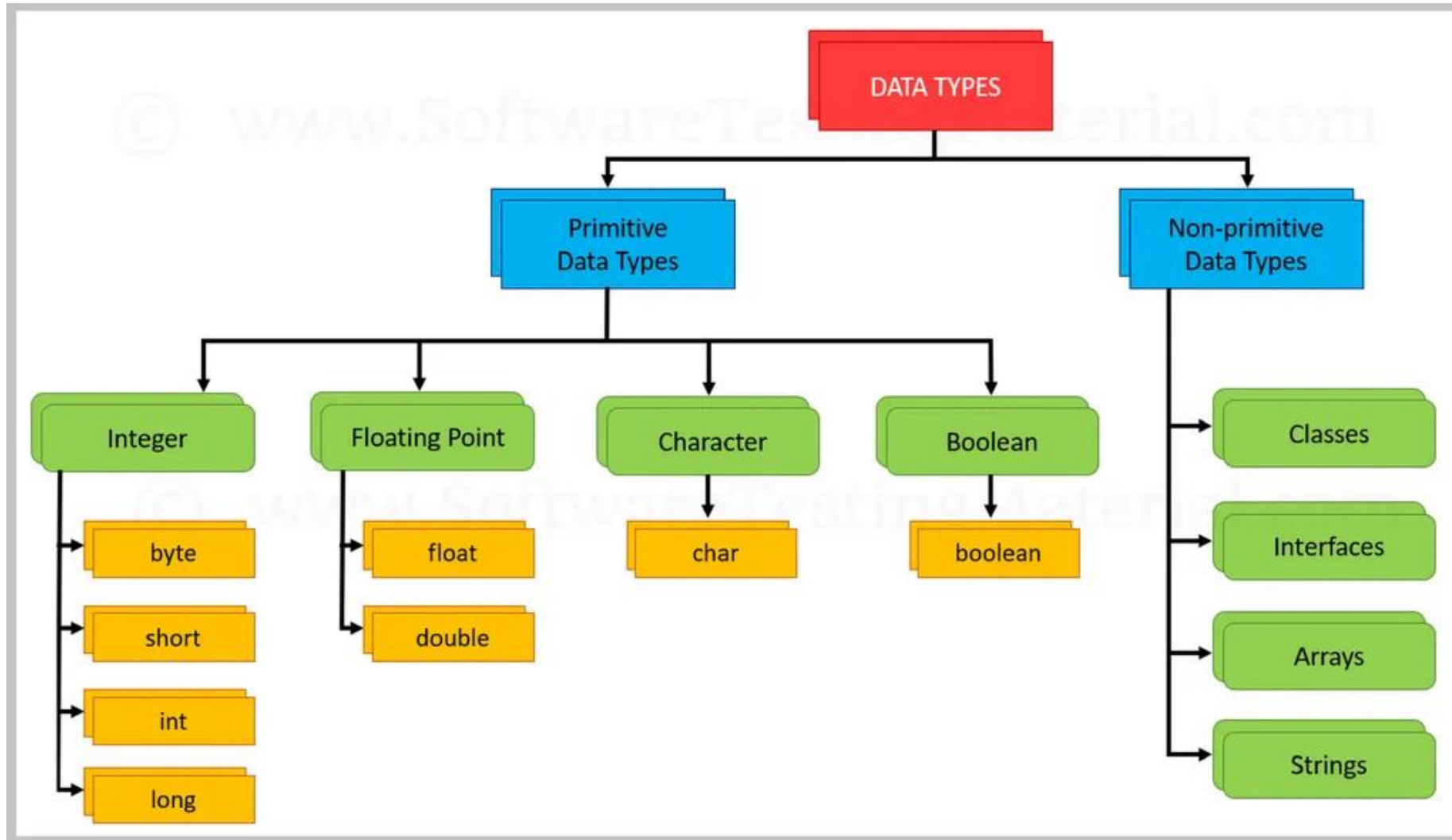
- A. 0
- B. Garbage value
- C. Compiler error
- D. Runtime error

## Question 2

```
class Test {  
    public static void main(String[] args) {  
        for(int i = 0; 0; i++)  
        {  
            System.out.println("Hello");  
            break;  
        }  
    }  
}
```

- A. Hello
- B. Empty output
- C. Compiler error
- D. Runtime error

# Data Types



This ppt is created as a reference material (only for the academic purpose) for the students of PICT. It is restricted only for the internal use and any circulation is strictly prohibited.

<https://www.softwaretestingmaterial.com/data-types-in-java/>  
Department of Electronics & Telecommunication Engg.



# Primitive Data Types

The primitive types are also commonly referred to as simple types, Java defines eight primitive types of data: byte, short, int, long, char, float, double, and boolean.

1. **Integers** This group includes byte, short, int, and long, which are for whole-valued signed numbers.
2. **Floating-point numbers** This group includes float and double, which represent numbers with fractional precision.
3. **Characters** This group includes char, which represents symbols in a character set, like letters and numbers.
4. **Boolean** This group includes boolean, which is a special type for representing true/false values.

# Primitive Data Type: Integer

Name	Width	Range
long	64	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
int	32	-2,147,483,648 to 2,147,483,647
short	16	-32,768 to 32,767
byte	8	-128 to 127

# Primitive Data Type: Integer

```
// Compute distance light travels using long variables.
class Light {
    public static void main(String args[]) {
        int lightspeed;
        long days;
        long seconds;
        long distance;

        // approximate speed of light in miles per second
        lightspeed = 186000;

        days = 1000; // specify number of days here

        seconds = days * 24 * 60 * 60; // convert to seconds

        distance = lightspeed * seconds; // compute distance

        System.out.print("In " + days);
        System.out.print(" days light will travel about ");
        System.out.println(distance + " miles.");
    }
}
```

This program generates the following output:

In 1000 days light will travel about  
160704000000000 miles.

# Primitive Data Type: Floating Point

- Also known as real numbers, are used when evaluating expressions that require fractional precision.
- Example, calculations such as square root, or transcendental functions such as sine and cosine, result in a value whose precision requires a floating-point type.

Name	Width in Bits	Approximate Range
double	64	4.9e−324 to 1.8e+308
float	32	1.4e−045 to 3.4e+038

# Primitive Data Type: Floating Point

```
// Compute the area of a circle.
public class Area
{
    public static void main(String args[])
    {
        double pi, r, a;
        r = 10.8; // radius of circle
        pi = 3.1416; // pi, approximately
        a = pi * r * r; // compute area
        System.out.println("Area of circle is " + a);
    }
}
```





# Primitive Data Type: Character

- In Java, the data type used to store characters is char.
- A key point to understand is that Java uses Unicode to represent characters.
- Unicode : Fully international character set that can represent all of the characters found in all human languages. It is a unification of dozens of character sets, such as Latin, Greek, Arabic, Cyrillic, Hebrew, Katakana, Hangul, and many more.
- At the time of Java's creation, Unicode required 16 bits. Thus, in Java char is a 16-bit type.
- C/C++ - char – ASCII - 8 bit
- The ASCII character set occupies the first 127 values in the Unicode character set.

# Primitive Data Type: Character

```
// Demonstrate char data type.
class CharDemo
{
    public static void main(String args[]) {
        char ch1, ch2;
        ch1 = 88; // code for X
        ch2 = 'Y';
        System.out.print("ch1 and ch2: ");
        System.out.println(ch1 + " " + ch2);
    }
}
```

This program displays the following output:

ch1 and ch2: X Y

This ppt is created as a reference material (only for the academic purpose) for the students of PICT. It is restricted only for the internal use and any circulation is strictly prohibited.

# Primitive Data Type: Character

- Although char is designed to hold Unicode characters, it can also be used as an integer type on which you can perform arithmetic operations.
- For example, you can **add two characters together**, or **increment** the value of a character variable.

//char variables behave like integers.

```
class CharDemo2 {  
    public static void main(String args[]) {  
        char ch1;  
        ch1 = 'X';  
        System.out.println("ch1 contains " + ch1);  
        ch1++; // increment ch1  
        System.out.println("ch1 is now " + ch1);  
    }  
}
```

The output generated by  
this program is shown here:  
ch1 contains X  
ch1 is now Y

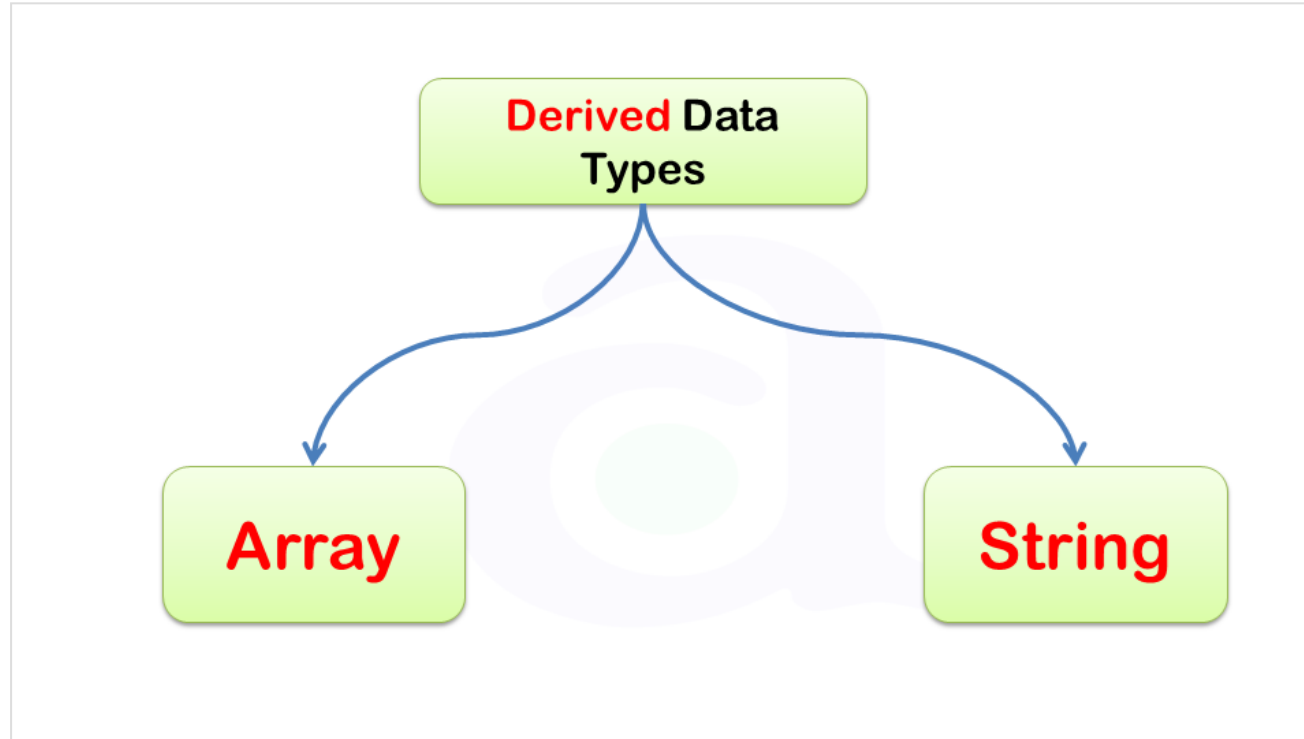
# Primitive Data Type: Boolean

- For logical values – returns true or false.
- This is the type returned by all relational operators, as in the case of  $a < b$ .
- Boolean is also the type required by the conditional expressions that govern the control statements such as if and for.

```
class Main
{
    public static void main (String args[])
    {
        boolean b;
        b = false;
        System.out.println ("b is " + b);
        b = true;
        System.out.println ("b is " + b);
        // a boolean value can control the if statement
        if (b)
            System.out.println ("This is executed.");
        b = false;
        if (b)
            System.out.println ("This is not executed.");
        // outcome of a relational operator is a boolean value
        System.out.println ("10 > 9 is " + (10 > 9));
    }
}
```

```
b is false
b is true
This is executed.
10 > 9 is true
```

# Non-primitive Data Type: Derived



<http://www.atnyla.com/library/images-tutorials/>

This ppt is created as a reference material (only for the academic purpose) for the students of PICT. It is restricted only for the internal use and any circulation is strictly prohibited.

Department of Electronics & Telecommunication Engg.



# Non-primitive Data Type: Strings

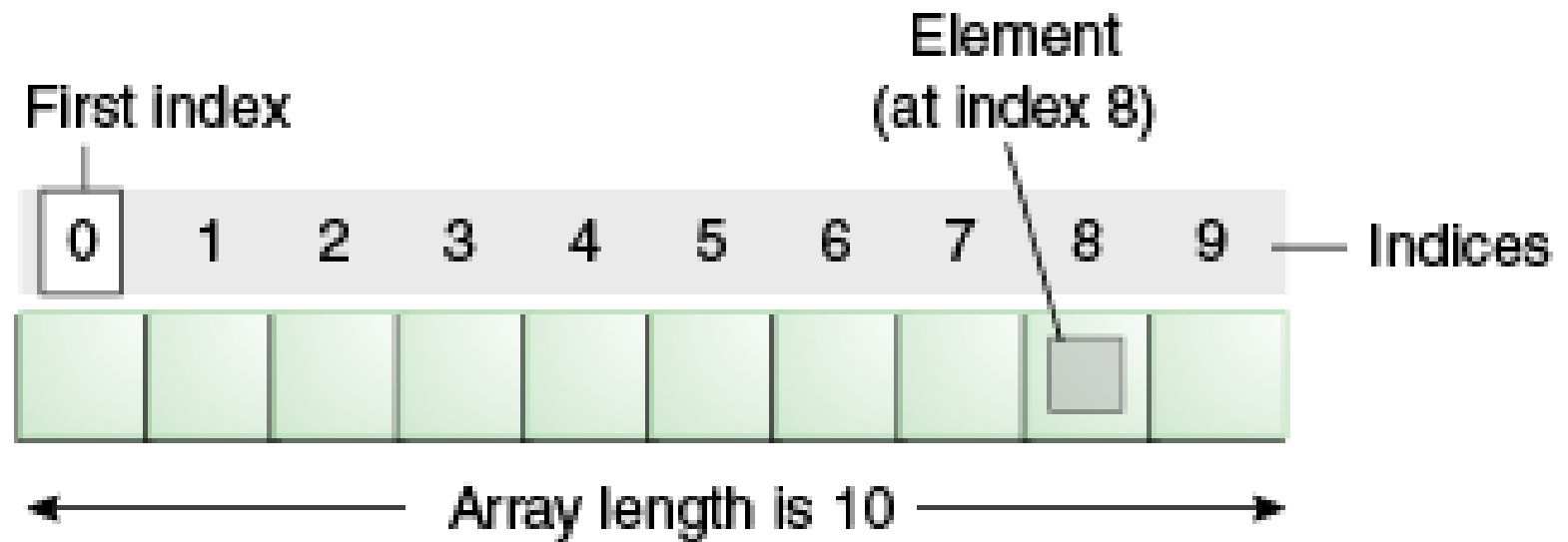
- Strings are defined as an array of characters.
- The difference between a character array and a string in Java is, that the string is designed to hold a sequence of characters in a single variable whereas, a character array is a collection of separate char type entities.
- Unlike C/C++, Java strings are not terminated with a null character.

Example:

*<String\_Type> <string\_variable> = "<sequence\_of\_string>";*

# Non-primitive Data Type: Arrays

- An array is a group of like-typed variables that are referred to by a common name.
- Arrays of any type can be created and may have **one or more dimensions**.
- A specific element in an array is accessed by **its index**.
- Arrays offer a convenient means of **grouping related information**.





# Non-primitive Data Type: Arrays

## Advantages

**Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.

**Random access:** We can get any data located at an index position.

## Disadvantages

**Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.



# Non-primitive Data Type: Arrays

**Single Dimensional Array:** The general form of a one-dimensional array declaration is

```
type var-name[ ];    // establishes the fact that var-name is an array variable, no  
array actually exists.
```

```
array-var = new type [size];    //allocate memory for arrays.
```



# Non-primitive Data Type: Arrays

## Operations on Arrays in Java:

- Access the Elements of an Array
- Change an Array Element
- Array Length
- Loop Through an Array

# Non-primitive Data Type: Arrays

An array declaration has two components: the type and the name.

```
int abc[];  
or int[] abc; // both are valid declarations
```

To link abc with an actual, physical array of integers,

```
abc = new int[10]; // allocating memory to array or Instantiating an Array
```

## Combining declaration and instantiation

```
int[] abc = new int[10]; or int abc[] = new int[10]; // declaration and  
instantiation
```

```
Or int[] abc = {1,2,3,4,5,6,7,8,9,10}; // Declaring array literal or  
//declaration, instantiation and initialization
```

```
String languages[] = {"Prolog", "Java"};
```

# Non-primitive Data Type: Arrays

```
// Demonstrate a one-dimensional array.
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.");
    }
}
```

This ppt is created as a reference material (only for the academic purpose) for the students of PICT. It is restricted only for the internal use and any circulation is strictly prohibited.

# Non-primitive Data Type: Arrays

An improved version of the previous Array program:

```
// An improved version of the previous program.
class AutoArray
{
    public static void main(String args[])
    {
        int month_days[] = { 31, 28, 31, 30, 31, 30, 31,
                             31, 30, 31, 30, 31 };
        System.out.println("April has " + month_days[3] +
                           " days.");
    }
}
```

# Non-primitive Data Type: Arrays

- A multidimensional array is an array containing one or more arrays, also known as Jagged Arrays.
- A multidimensional array is created by appending one set of square brackets ([]) per dimension.

Examples:

```
int[][] intArray = new int[10][20]; //a 2D array or matrix
```

```
int[][][] intArray = new int[10][20][10]; //a 3D array
```

```
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
```

# Non-primitive Data Type: Arrays

```
// Demonstrate a two-dimensional array.
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();
        }
    }
}
```

This program generates the following output:

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



# Symbolic Constants

- Requirement of certain unique constants in a program. e.g.
  - value of pi – 3.142
  - calculation of mean, std dev etc we require total
- For this we may use symbolic names to such constants, e.g. PI, TOTAL, PASSING\_MARKS etc.
- A Constant is declared as:
  - **final** int STRENGTH=100;
  - **final** int PASS\_MARKS=50;
  - **final** float PI = 3.14159



# Standard Default Values

Type of variable	Default Value
Byte	0
Short	0
Int	0
Long	0
Float	0.00f
Char	Null char
Boolean	False



# Type Casting

Assigning a value of one primitive data type to another type.

In Java, there are two types of casting:

**Widening Casting (automatically)** - converting a smaller type to a larger type size

**Narrowing Casting (manually)** - converting a larger type to a smaller size type



# Widening Casting

```
public class Main {  
    public static void main(String[] args) {  
        int myInt = 9;  
        double myDouble = myInt; // Automatic casting: int to double  
  
        System.out.println(myInt);    // Outputs 9  
        System.out.println(myDouble); // Outputs 9.0  
    }  
}
```

[https://www.w3schools.com/java/java\\_type\\_casting.asp](https://www.w3schools.com/java/java_type_casting.asp)



# Narrowing Casting

```
public class Main {  
    public static void main(String[] args) {  
        double myDouble = 9.78d;  
        int myInt = (int) myDouble; // Manual casting: double to int  
  
        System.out.println(myDouble); // Outputs 9.78  
        System.out.println(myInt);    // Outputs 9  
    }  
}
```

[https://www.w3schools.com/java/java\\_type\\_casting.asp](https://www.w3schools.com/java/java_type_casting.asp)

This ppt is created as a reference material (only for the academic purpose) for the students of PICT. It is restricted only for the internal use and any circulation is strictly prohibited.

Department of Electronics & Telecommunication Engg.



# Operators

- Arithmetic operators
- Relational operators
- Logical operators
- Assignment operators
- Increment and decrement operators
- Conditional operators
- Bitwise operators
- Special operators

# Arithmetic Operators

Operator	Meaning
+	Addition or unary plus
-	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo division(remainder)

## • Integer Arithmetic:

➤ When both the operands in an arithmetic expression are integers, then expression is called an *integer expression*.

➤ Operation in *integer expression* is called *integer arithmetic*.

Ex:  $a+b$ ,  $a-b$ ,  $a*b$ ,  $a/b$ ,  $a\%b$

$10/5$ ,  $10\%5$ ,  $-10\%5$ ,  $-10\%-5$ ,  $10\% -5$

## • Real Arithmetic:

➤ An arithmetic operation involving only real operands is called *real arithmetic*

➤ Modulus operator % can be applied to floating point data as well.

## • Mixed-mode Arithmetic: (Ex: $15/10.0 = 1.5$ )

This presentation is prepared (Ex: for the academic purpose) for the students of PICT. It is restricted only for the internal use and any circulation is strictly prohibited.

# Relational Operators

- Used to compare two quantities and take decision depending on their relation.

ex: age of persons, price of two items

Operator	Meaning
<	Is less than
<=	Is less than or equal to
>	Is greater than
>=	Is greater than or equal to
==	Is equal to
!=	Is not equal to

- Expression containing relational operator is termed as a *relational expression*.
- Value of any relational expression is either true or false.
- Syntax: 

ae-1 relational operator ae-2
-------------------------------
- Arithmetic operators have a higher priority over relational operator.**

This ppt is created as a reference material (only for the academic purpose) for the students of PICT. It is restricted only for the internal use and any circulation is strictly prohibited.

# Logical Operators

- When we want to form compound condition by combining two or more relational operators, Logical operators are used.
- Example :  $7 > 6 \ \&\& \ x == 9$

Operator	Meaning
$\&\&$	Logical AND
$\parallel$	Logical OR
$!$	Logical NOT

- The expression which combines two or more relational expression is called as *logical expression or compound relational expression*.



# Assignment Operators

- This operator is used to assign the value of an expression to a variable.
- Example of this operator is '='.
- In addition , java has a set of 'shorthand' assignment operators.
- Syntax of shorthand operator is: `v op= expression;`

Statement with simple = operator	Statement with shorthand operator
<code>a=a+1</code>	<code>a+=1</code>
<code>a=a-1</code>	<code>a-=1</code>
<code>a=a*(n+1)</code>	<code>a*=(n+1)</code>
<code>a=a/(n+1)</code>	<code>a/=(n+1)</code>
<code>a=a%b</code>	<code>a%=b</code>

# Increment and Decrement Operators

- Increment operator ++, adds 1 to the operand.
- Decrement operator --, subtracts 1 from the operand.
- These operators are also called as unary operators.
- Different forms:

++a pre increment , pre decrement --b

a++ post increment , post decrement b --

- ++a is equivalent to a=a+1(or a+=1).
- ++a and a++ mean the same thing when they form statements independently, and they behave differently when they are used in expressions on RHS of assignment operator.

# Conditional Operators

- This operator is the character pair **? :**, also called as **ternary operator**
- Syntax:

**Exp 1 ? Exp 2 : Exp 3**

- Ex: num1 =5;

num2 =10;

result= (num1>num2)? num1: num2;

In this case result will be assigned the value of num2

# Bitwise Operator

- Used to manipulate data at bit level

Operator	Meaning
&	Bitwise AND
!	Bitwise NOT
^	Bitwise EX-OR
~	One's complement
<<	Shift left
>>	Shift right
>>>	Shift right with zero fill

# Assignment Operator

$\ll=$	Left shift AND assignment operator.	$C \ll= 2$ is same as $C = C \ll 2$
$\gg=$	Right shift AND assignment operator.	$C \gg= 2$ is same as $C = C \gg 2$
$\&=$	Bitwise AND assignment operator.	$C \&= 2$ is same as $C = C \& 2$
$\wedge=$	bitwise exclusive OR and assignment operator.	$C \wedge= 2$ is same as $C = C \wedge 2$
$ =$	bitwise inclusive OR and assignment operator.	$C  = 2$ is same as $C = C   2$

```
public class Test {  
    public static void main(String args[]) {  
        int a = 60; /* 60 = 0011 1100 */  
        int b = 13; /* 13 = 0000 1101 */  
        int c = 0;  
        c = a & b;  
        System.out.println("a & b = " + c );  
        c = a | b;  
        System.out.println("a | b = " + c );  
        c = a ^ b;  
        System.out.println("a ^ b = " + c );  
        c = ~a;  
        System.out.println("~a = " + c );  
        c = a << 2;  
        System.out.println("a << 2 = " + c );  
        c = a >> 2;  
        System.out.println("a >> 2 = " + c );  
        c = a >>> 2;  
        System.out.println("a >>> 2 = " + c );  
    }  
}
```

```
public class sample1
{
    public static void main(String args[])
    {
        int a = 10;
        int c = 15;

        c <<= 2 ;
        System.out.println("c <<= 2 = " + c );

        c >>= 2 ;
        System.out.println("c >>= 2 = " + c );

        c &= a ;
        System.out.println("c &= a  = " + c );

        c ^= a ;
        System.out.println("c ^= a   = " + c );

    }
}
```

This ppt is created as a reference material (only for the academic purpose) for the students of PICT. It is restricted only for the internal use and any circulation is strictly prohibited.



# Special Operator

- Java supports special operator called as *instanceof* operator.
- The *instanceof* operator is an object reference operator.
- This operator allows us to determine whether the object belongs to a particular class or not.
- The *instanceof* in java is also known as *type comparison operator* because it compares the instance with type
- Ex:

```
person instanceof student
```

is true if the object person belongs to class student: otherwise it is false.



```
class simple{  
    public static void main(String args[]){  
        simple s=new simple();  
        System.out.println(s instanceof simple);  
    }  
}
```

# Evaluation of Expressions

- Expressions are evaluated using assignment statement as

```
variable = expression;
```

- All the variables used in expression must be initiated before evaluation is attempted.

Ex:

```
x = a*b-c;
```

Here the variables a, b, and c must be defined before they are used in the expression.

# Precedence of Arithmetic Operators

- An arithmetic expression **without** any parenthesis will be evaluated from **left to right** using the rules of precedence of operators.
- Arithmetic operators with priority levels:  
High priority       $*, /, \%$   
Low priority       $+, -$
- Without parentheses, evaluation procedure includes two left to right passes through expression
  - ✓ During the first pass, high priority operators gets executed
  - ✓ During the second pass, low priority operators gets executed
- If parentheses are used, the expression within parentheses gets highest priority.
- If two or more parentheses occurs in expression then, the left most parentheses in an expression gets evaluated first.

ex: **result = (a+b) \* (c/d)**

•  $x = 9 - 12/3 + 3*2 - 1$

First pass

step 1:  $x = 9 - 4 + 3*2 - 1$

step 2:  $x = 9 - 4 + 6 - 1$

Second pass

step 3:  $x = 5 + 6 - 1$

step 4:  $x = 11 - 1$

step 5:  $x = 10$

•  $x = 9 - 12/(3+3)*(2-1)$

First pass

step 1:  $x = 9 - 12/6*(2-1)$

step 2:  $x = 9 - 12/6*1$

Second pass

step 3:  $x = 9 - 2*1$

step 4:  $x = 9 - 2$

Third pass

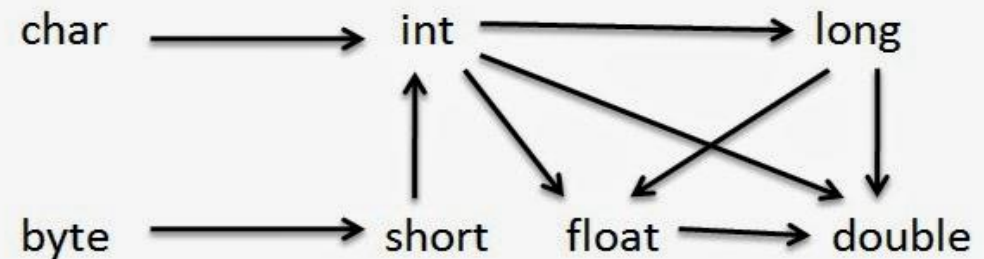
step 5:  $x = 7$

✓ Number of evaluation steps remains same in both the cases equal to 5( *i.e. equal to number of arithmetic operators in expression*)

# Type Conversion in Expressions

## Automatic Type Conversion :-

- Java permits mixing of constants and variables of different types in expression, but during evaluation it follows to strict rules of type conversion
- If operands are of different type then the lower type is automatically converted to the higher type before operation proceeds.



# Comparison

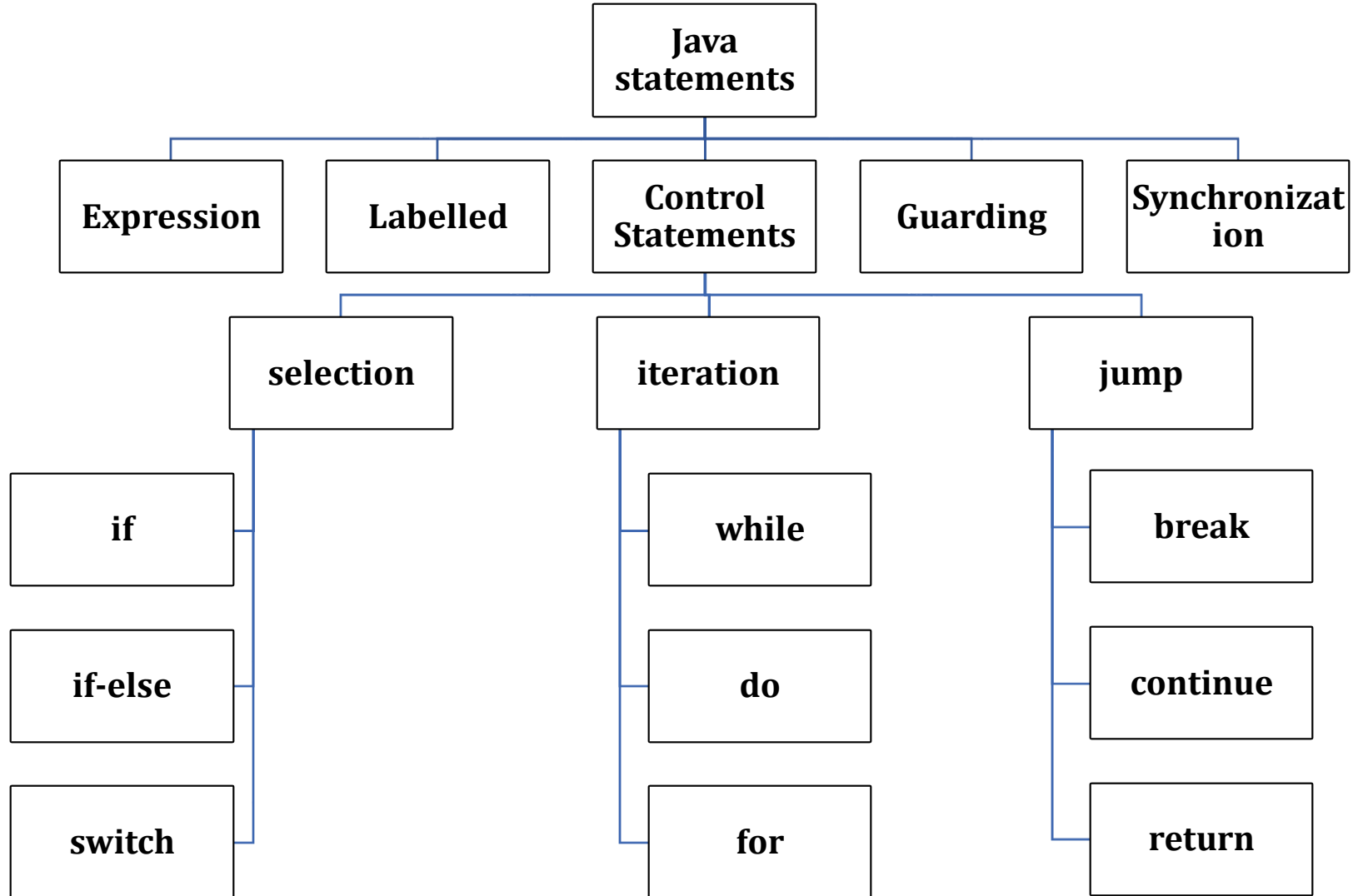
Type Casting	Type Conversion
In type casting, a data type is converted into another data type by a programmer using casting operator.	Whereas in type conversion, a data type is converted into another data type by a compiler.
Type casting can be applied to <b>compatible data types</b> as well as <b>incompatible data types</b> .	Whereas type conversion can only be applied to <b>compatible datatypes</b> .
In type casting, casting operator is needed in order to cast the data type to another data type.	Whereas in type conversion, there is no need for a casting operator.
In typing casting, the destination data type may be smaller than the source data type, when converting the data type to another data type.	Whereas in type conversion, the destination data type can't be smaller than source data type.
Type casting takes place during the program design by programmer.	Whereas type conversion is done at the compile time.
Type casting is also called narrowing conversion because in this, the destination data type may be smaller than the source data type.	Whereas type conversion is also called widening conversion because in this, the destination data type can not be smaller than the source data type.
Type casting is often used in coding and competitive programming works.	Whereas type conversion is less used in coding and competitive programming as it might cause incorrect answer.
Type casting is more efficient and reliable.	Whereas type conversion is less efficient and less reliable.



# Java Statements

- A statement is an executable combination of tokens ending with a semicolon (;) mark.
- Statements are usually executed in sequence in the order in which they appear.
- However, it is possible to control the flow execution, if necessary, using special statements.

# Java Statements



This ppt is created as a reference material (only for the academic purpose) for the students of PICT. It is restricted only for the internal use and any circulation is strictly prohibited.





# Control Statements

A programming language uses control statements to cause the flow of execution to advance and branch based on changes to the state of a program.

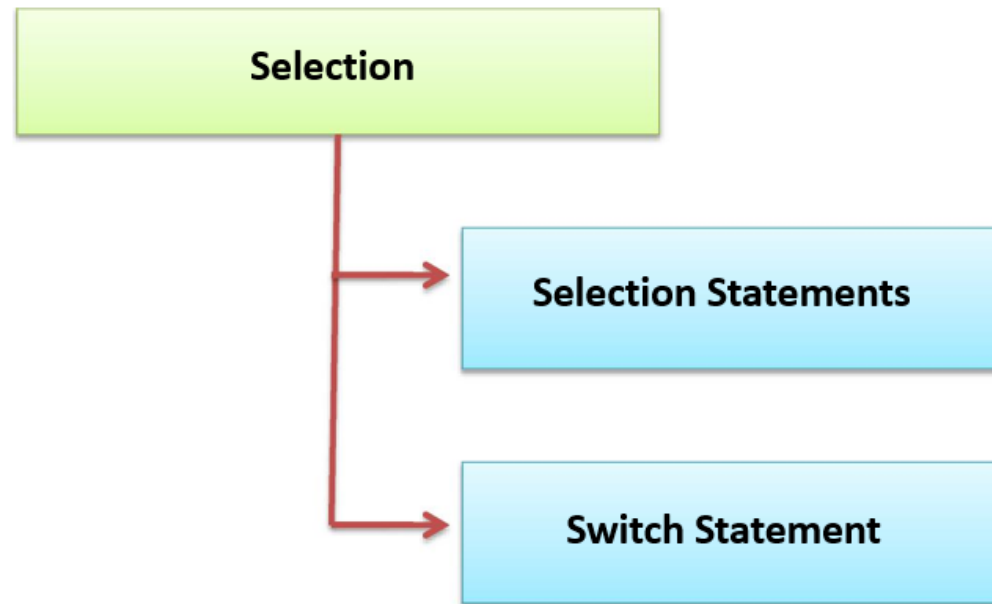
Java's program control statements can be put into the following categories:

selection, iteration, and jump.

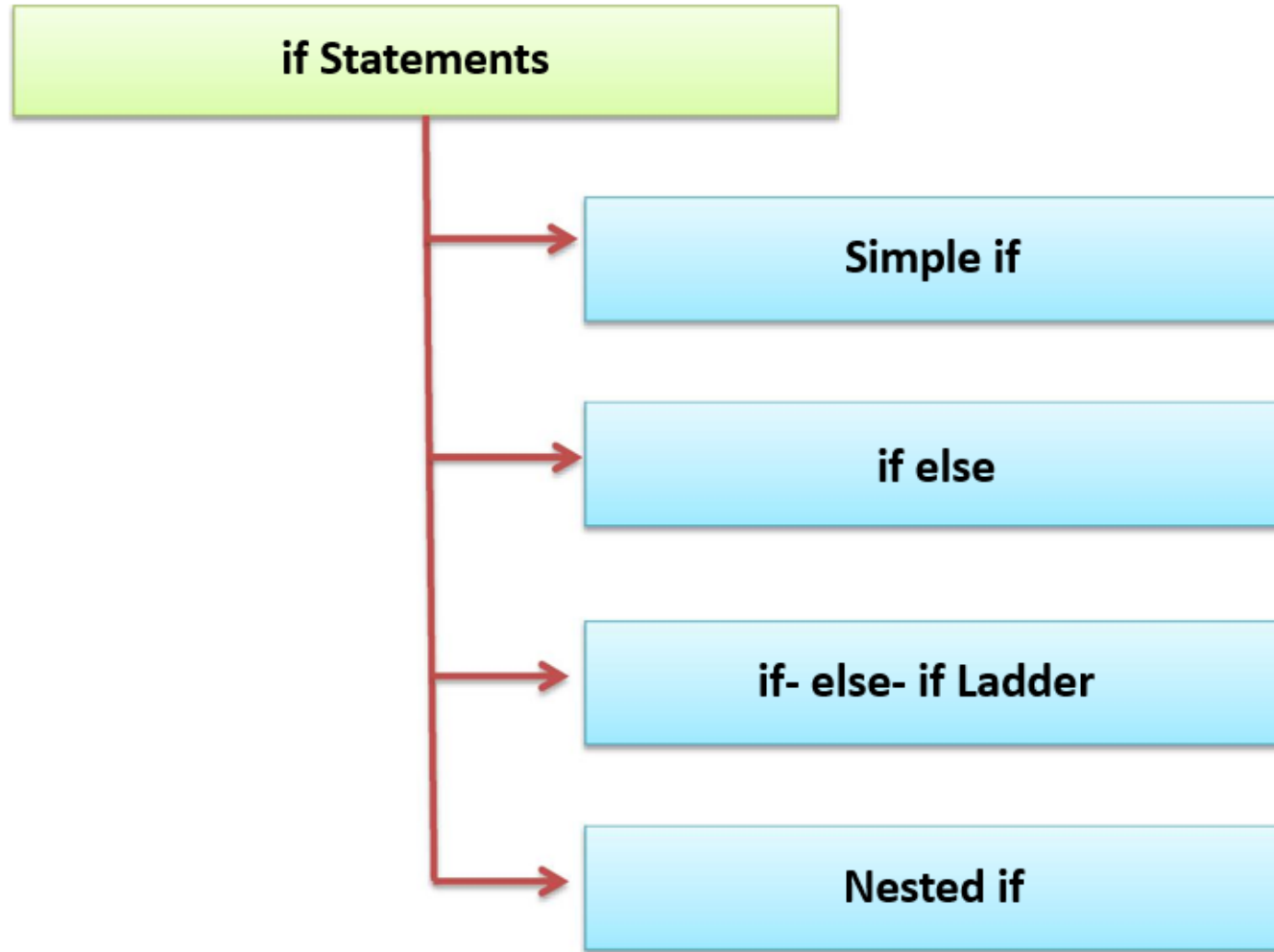
- Selection statements (**if and switch**) allow your program to choose different paths of execution based upon the outcome of an expression or the state of a variable.
- Iteration statements (**for, while, and do-while**) enable program execution to repeat one or more statements (that is, iteration statements form loops).
- Jump statements (**break, continue, and return.**) allow your program to execute in a nonlinear fashion.

# Selection Statements

Selection Statements are also called Decision Making Statements.



# if Statement



# Simple if

Syntax:

```
if (condition)
```

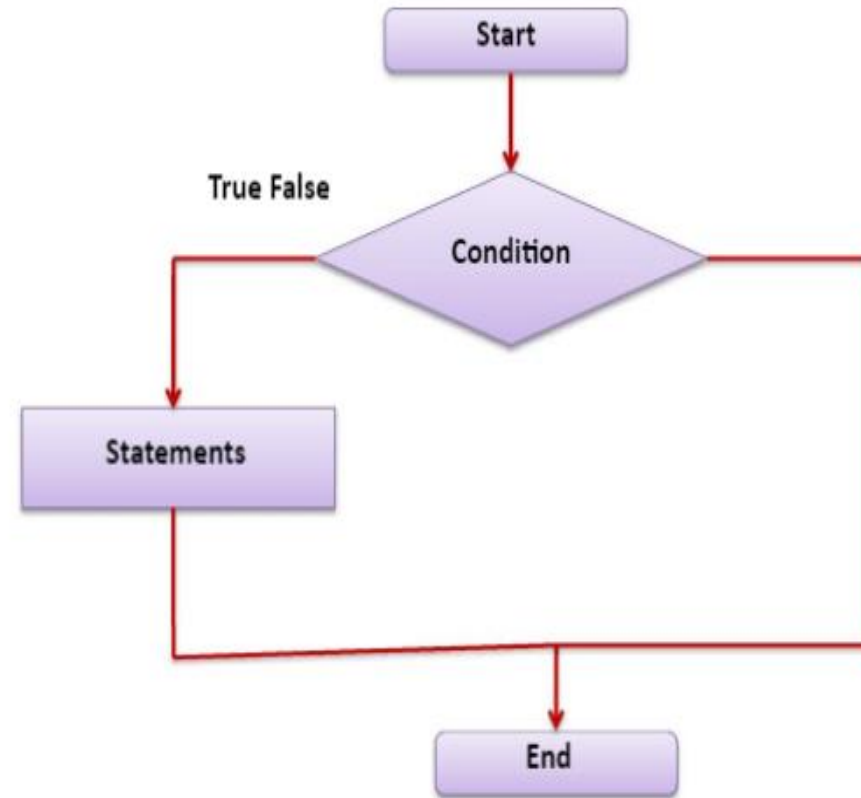
```
{
```

```
    statement1;
```

```
}
```

Purpose: The statements will be evaluated if the value of the condition is true.

Flow Chart:





# if - else

Syntax:

```
if (condition)
```

```
{
```

```
    statement1;
```

```
}
```

```
else
```

```
{
```

```
    statement2;
```

```
}
```

Purpose: The statement 1 is evaluated if the value of the condition is true otherwise statement 2 is true.

# if – else – if Ladder

**Syntax :**

```
if (condition)
    statements;
else if (condition)
    statements;
else if (condition)
    statements;
...
...
else
    statements;
```

This ppt is created as a reference material (only for the academic purpose) for the students of PICT. It is restricted only for the internal use and any circulation is strictly prohibited.



# Nested if

## Syntax :

```
if (condition)
{
    if (condition)
        statements....
    else
        statements....
}
else
{
    if (condition)
        statements....
    else
        statements....
}
```

# switch

**Syntax :**

```
switch (expression)
{
    case value 1 :
        statement 1 ; break;
    case value 2 :
        statement 2 ; break;
    ...
    case value N :
        statement N ; break;
    default :
        statements ; break;
}
```

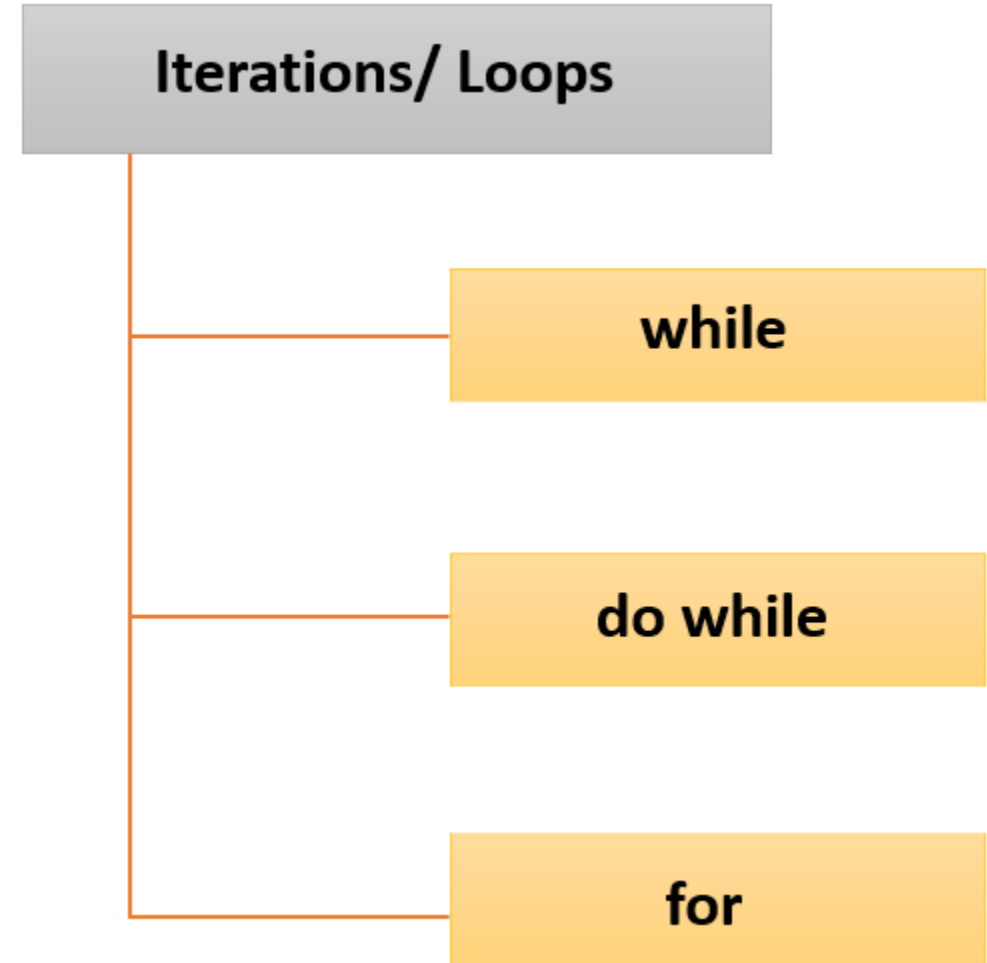
This ppt is created as a reference material (only for the academic purpose) for the students of PICT. It is restricted only for the internal use and any circulation is strictly prohibited.



# Iteration Statements

Each loop has four types of statements :

- Initialization
- Condition checking
- Execution
- Increment / Decrement



# while

## Syntax

```
initialization  
while(final value)  
{  
    statements;  
    increment/decrement;  
}
```

```
m=1  
while(m<=20)  
{  
    System.out.println(m);  
    m=m+1;  
}
```

**Purpose:** To evaluate the statements from initial value to final value with given increment/ decrement.

# do while

## Syntax

```
initialization  
do  
{  
    statements;  
    increment/decrement;  
}  
    while (final value) ;
```

```
m=1  
do  
{  
    System.out.println(m);  
    m=m+1;  
}  
while(m==20);
```

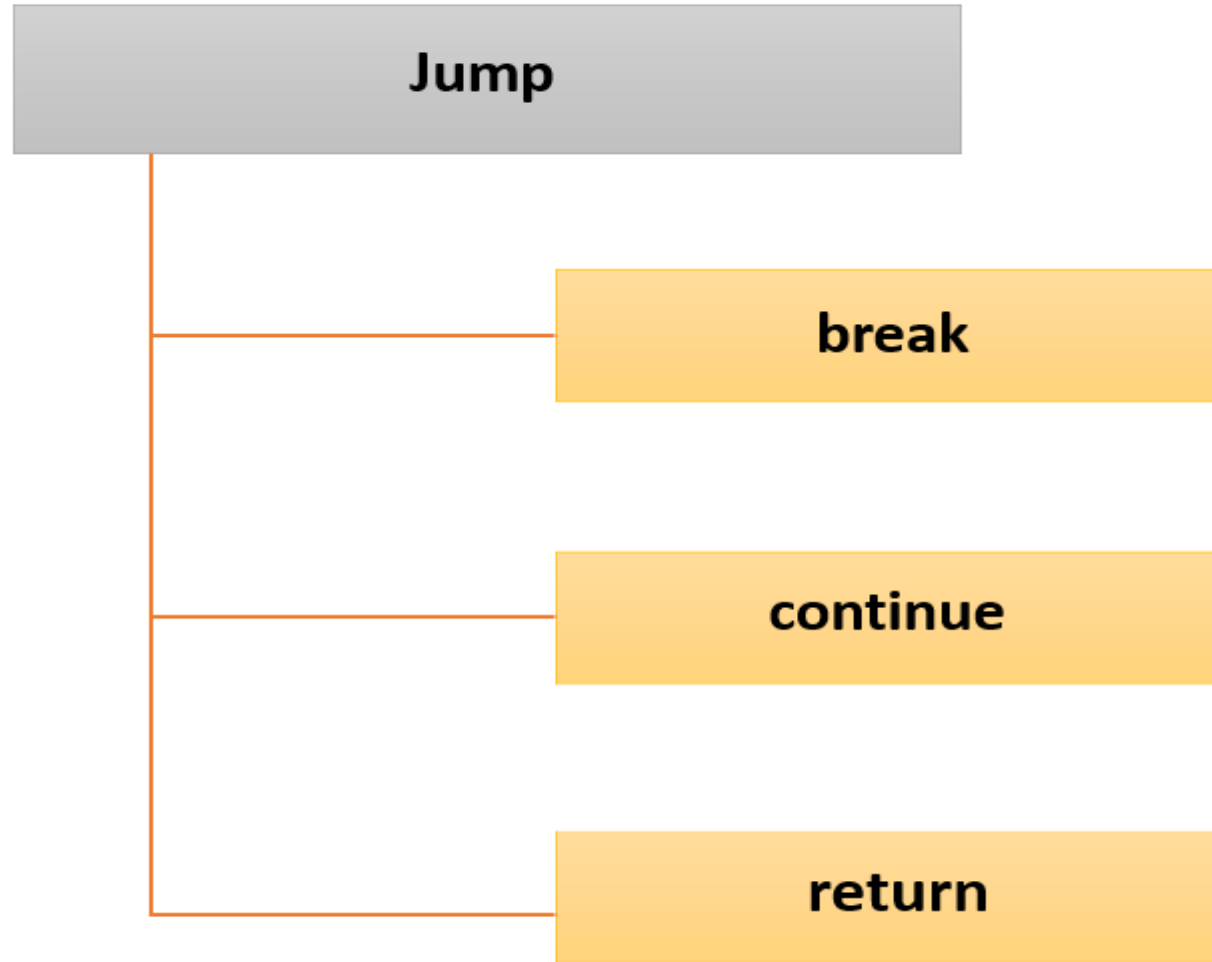


# for

**Syntax:**

```
for (initialization; final  
value; increment/decrement)  
{  
statements;  
}
```

```
for(m=1;m<=20;m=m+1)  
{  
System.out.println(m);  
}
```





# break

- This statement is used to jump out of a loop.
- Break statement was previously used in switch – case statements.
- On encountering a break statement within a loop, the execution continues with the next statement outside the loop.
- The remaining statements which are after the break and within the loop are skipped.
- Break statement can also be used with the label of a statement.
- A statement can be labeled as: **statementName : SomeJavaStatement**
- When we use break statement along with label as, **break statementName;**



# continue

- This statement is used only within the looping statements.
- When the continue statement is encountered, the next iteration starts.
- The remaining statements in the loop are skipped. The execution starts from the top of loop again.



# return

- The last control statement is return. The return statement is used to explicitly return from a method.
- That is, it causes program control to transfer back to the caller of the method.
- The return statement immediately terminates the method in which it is executed.





# References

- E Balagurusamy, “Programming with JAVA”, Tata McGraw Hill, 6th Edition.
- Herbert Schildt, “Java: The complete reference”, Tata McGraw Hill, 7th Edition.
- <https://www.tutorialspoint.com>
- <https://www.geeksforgeeks.org>
- <http://underpop.online.fr/j/java/help/static-fields-and-methods-objects-and-classes.html.gz#:~:text=out%20.,.%20.%3B%20.%20.%20.%20%7D>
- <https://www.w3schools.com>