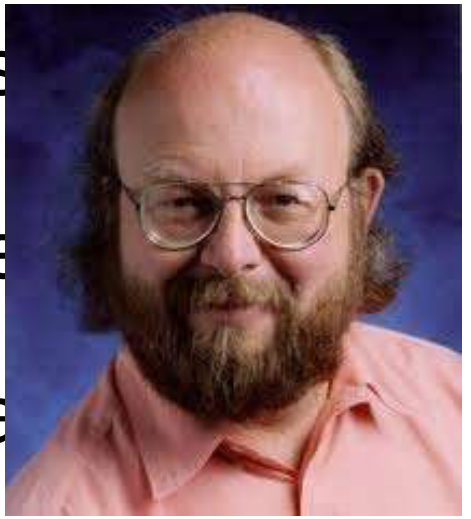
Introduction to Java

Introduction to Java

- Java is a purely an object – oriented language.
- In 1991, James Gosling and Patrick Naughton developed named “OAK” at Sun Micros



- Java is simple and platform-independe
- In 1995, this language was renamed a
- **Java history** is interesting to know. The history of java starts from Green

Introduction to Java

- Java team members (also known as **Green Team**), initiated a revolutionary task to develop a language for digital devices such as set-top boxes, televisions etc.
- For the green team members, it was an advance concept at that time. But, it was suited for internet programming. Later, Java technology as incorporated by Netscape.
- Currently, Java is used in internet programming,

Introduction to Java - History

- There are given the major points that describes the history of java.
- **James Gosling**, **Mike Sheridan**, and **Patrick Naughton** initiated the Java language project in June 1991. The small team of sun engineers called **Green Team**.
- Originally designed for small, embedded systems in electronic appliances like set-top boxes.
- Firstly, it was called **"Greentalk"** by James Gosling

Introduction to Java – History – Why Oak name?

- After that, it was called **Oak** and was developed as a part of the Green project.
- **Why Oak?** Oak is a symbol of strength and choosen as a national tree of many countries like U. S.A.,France, Germany, Roman
- In 1995, Oak was renamed as **"Java"** because it was already Technologies.

Introduction to Java – History – Why Java name?

- **Why they choosed java name for java language?**

  The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", " Silk", "jolt", "DNA" etc.

- They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell and fun to say.

- According to James Gosling "Java was one of the top choices along with **Silk**". Since java was so

Introduction to Java – History

- Java is an island of Indonesia where first coffee was produced called java coffee.
- Notice that Java is just a name not an acronym.
- Originally developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995.
- In 1995, Time magazine called **Java one of the Ten Best Products of 1995**. JDK 1.0 released in(January 23, 1996).

Introduction to Java – Versions

- Sun Microsystems has been releasing new versions of JDK. The originally version of Java released in 1995, was called JDK1.0 version.
- The versions are JDK1.1, JDK1.2, JDK1.3, JDK1.4 and JDK1.5, JDK1.6, JDK1.7 and JDK1.8.
- **Homework** - History of different versions and its features

Introduction to Java - JVM

- CPUs are capable of executing only the machine instructions.
- Different types of CPUs have different sets of machine instructions.
- To enable Java programs to be executed on multiple CPUs without modifications, java programs are compiled into machine instructions that can be understood and executed by an idealized CPU. Such a CPU is called **Java Virtual Machine (JVM)**.

Introduction to Java – Bytecode, JIT

- A Java program is first compiled into the machine code that is understood by JVM. Such a code is called **Byte Code**.
- Although the details of the JVM will differ from platform to platform, they all interpret the **Java Byte Code** into executable native machine code.
- The **Just In Time (JIT)** compiler compiles the byte code into executable machine code in real time, on a piece-by-piece demand basis.

Introduction to Java – JDK

- The JIT cannot compile the entire byte code of a Java program into executable machine code all at once, because Java Runtime Environment carries out various run time checks that can be performed only at run time.
- The collection of the three entities name the ***Java language***, the ***Java language packages*** and the set of ***Java development tools*** is called the **Java Development Kit (JDK)**.

Introduction to Java - JDK Tools

- The important development tools included in JDK are Java compiler, Java interpreter, Java disassembler, Java debugger, tool for C header files, tool for creating HTML documents and tool for viewing Java applets.
- The purpose of each one of these development tools is briefly described in the following table.

Introduction to Java – JDK Tools

Tool in JDK	Purpose
appletviewer	This tool helps us to execute a special type of Java program known as applets.
java	This tool is the Java ***interpreter***. It interprets the bytecode into machine code and then executes it
javac	This tool is Java ***compiler***. It compiles the Java source code into the bytecode, which can be understood by JVM.
javadoc	This tool is used to create documentation in HTML.
javah	This tool produces C header files for use with a special type of Java methods, known as native methods.

Introduction to Java – Types of applications

- Three types of programs, can be developed using Java, Namely *applets, servlets* and *stand-alone* applications.
- *Applet* is a Java program that is developed exclusively for the Internet.
- Applets are embedded in HTML documents that take care of the design of web pages.
- *Servlets* are used to create dynamic web content.
- *Stand-alone* applications can be executed without the internet connectivity. These applications are executed

Features of Java

Features of Java

- There is given many features of java. They are also known as java buzzwords. The Java Features given below are simple and easy to understand.

1. Simple
2. Object-Oriented
3. Platform independent
4. Secured
5. Robust
6. Architecture neutral
7. Portable
8. Dynamic
9. Interpreted
10. High Performance
11. Multithreaded
12. Distributed

Features of Java - Simple

- According to Sun, Java language is simple because:
  - syntax is based on C++ (so easier for programmers to learn it after C++).
  - removed many confusing and/or rarely-used features e.g., explicit pointers, operator overloading etc.
  - No need to remove unreferenced objects because there is Automatic Garbage

Features of Java - Object-oriented

- Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.
- Object-oriented programming(OOPs) is a methodology that simplify software development and maintenance by providing some rules.
- Basic concepts of OOPs are:
  - Object
  - Class
  - Inheritance
  - Polymorphism

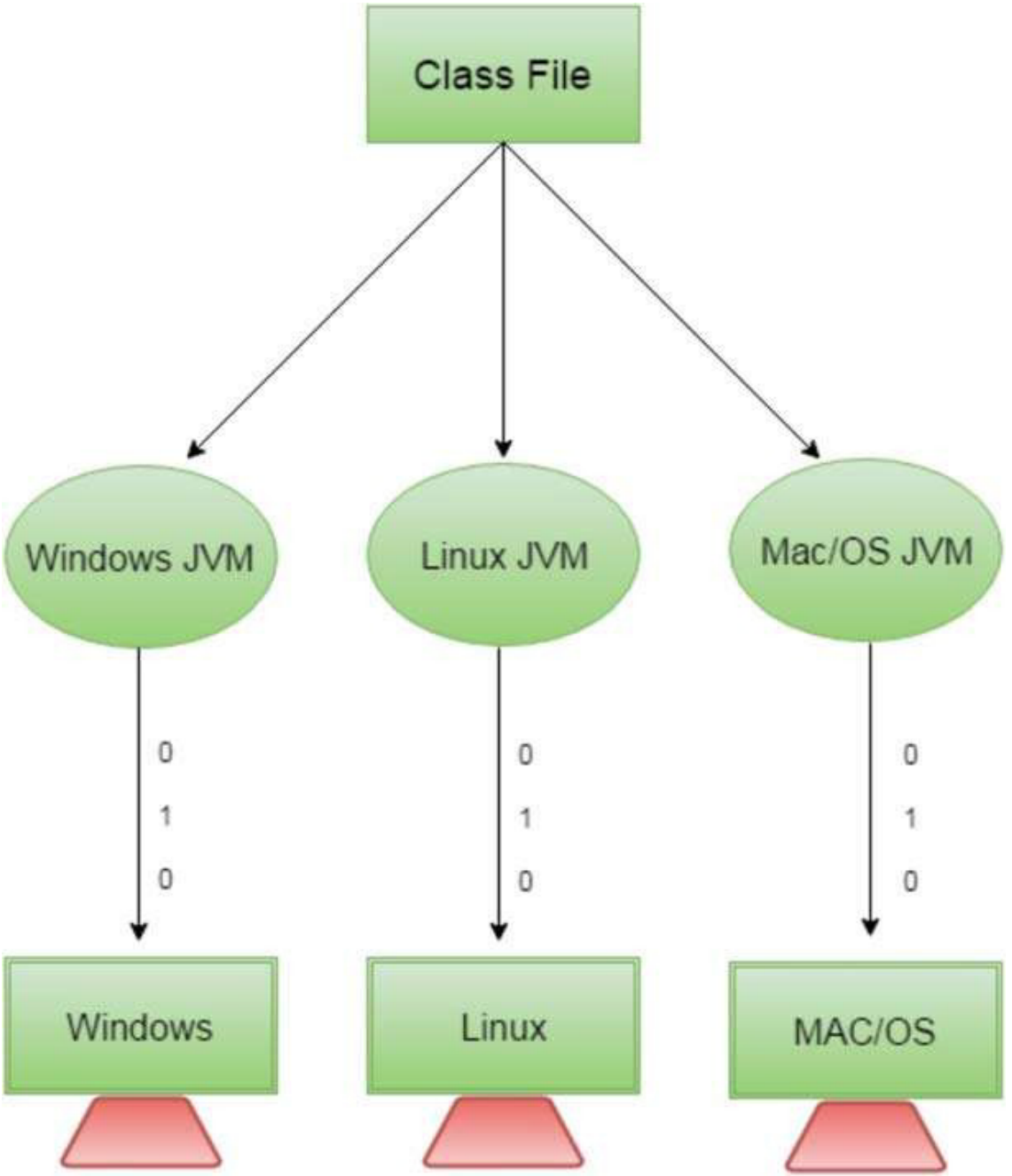
Features of Java - Platform Independent

- A platform is the hardware or software environment in which a program runs.
- There aretwo types of platforms, software-based and hardware-based.
- Java provides software-based platform.
- The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms.
- It has two components:
  1. Runtime Environment
  2. API(Application Programming Interface)

Features of Java - Platform Independent

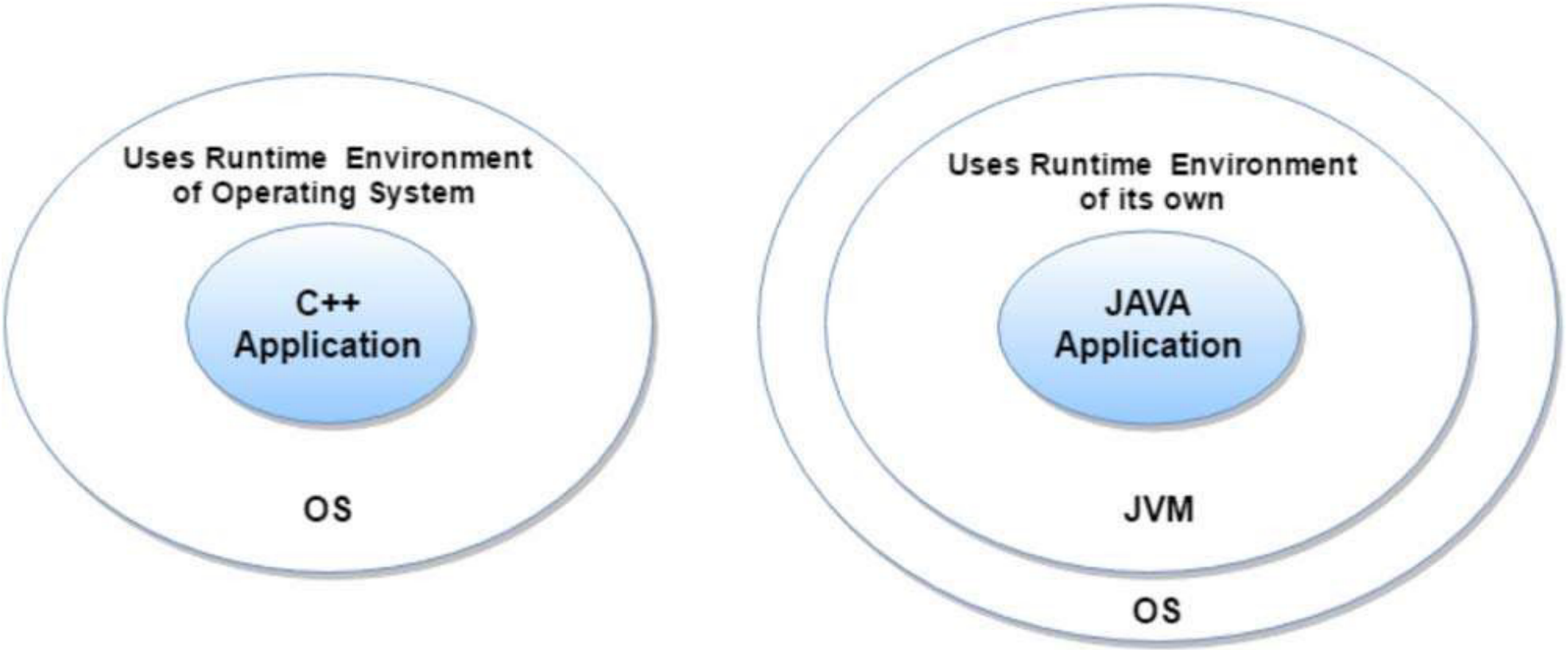
- Javacode can be runon multiple platforms e.g. Windows, Linux, Sun Solaris, Mac/OS etc.
- Java code is compiled by the compiler and converted into bytecode.
- This bytecode is a platform-independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere(WORA).

Features of Java - Platform Independent



Features of Java - Secured

- Java is secured because:
  - No explicit pointer
  - Java Programs run inside virtual machine sandbox



Features of Java - Secured

- **Classloader:** adds security by separating the package for the classes of the local file system from those that are imported from network sources.
- **Bytecode Verifier:** checks the code fragments for illegal code that can violate access right to objects.
- **Security Manager:** determines what resources a class can access such as reading and writing to the local disk.
- These security are provided by java language.

Features of Java - Robust

- Robust simply means strong.
- Java uses strong memory management.
- There are lack of pointers that avoids security problem.
- There is automatic garbage collection in java.
- There is exception handling and type checking mechanism in java.
- All these points makes java robust.

Features of Java - Architecture-neutral

- There is no implementation dependent features

  e.g. size of primitive types is fixed.

- In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture.

- But in java, it occupies 4 bytes of memory for both 32 and 64 bit architectures.

Features of Java - Portable

- We may carry the java bytecode to any platform.

Features of Java – High-performance

- Java is faster than traditional interpretation since byte code is "close" to native code still somewhat slower than a compiled language (e.g. , C++)

Features of Java – Distributed

- We can create distributed applications in java.
- RMI and EJB areused for creating distributed applications.
- We may access files by callingthe methods from any machine on the internet.

Features of Java – Multi-threaded

- A thread is like a separate program, executing concurrently.
- We can write Java programs that deal with many tasks at once by defining multiple threads.
- The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area.
- Threads are important for multimedia, Web applications etc.

Object Oriented Concepts

OOPS and Java

- Java is a purely an object – oriented language.
- The central idea behind object-oriented programming is to divide a program into isolated parts called Objects.
- Each object contain two parts.
  - Data
  - Functions
- The **functions** in an object operate on the **data** involved in that object.

OOPS and Java – Objects and Classes

- Data and routines (functions) required to process it are combined into a single entity called an Object.
- A collection of similar objectsisknown as a class. (Blueprint of Object).
- In order to clearly understand the idea of objects, lets us now consider an example. Suppose we are planning a birthday party and we wish to play some songs available in our compact disk. there is an index card file, in which each card contains a song's name, its author and its playing time in the following format.
  - Song
  - name

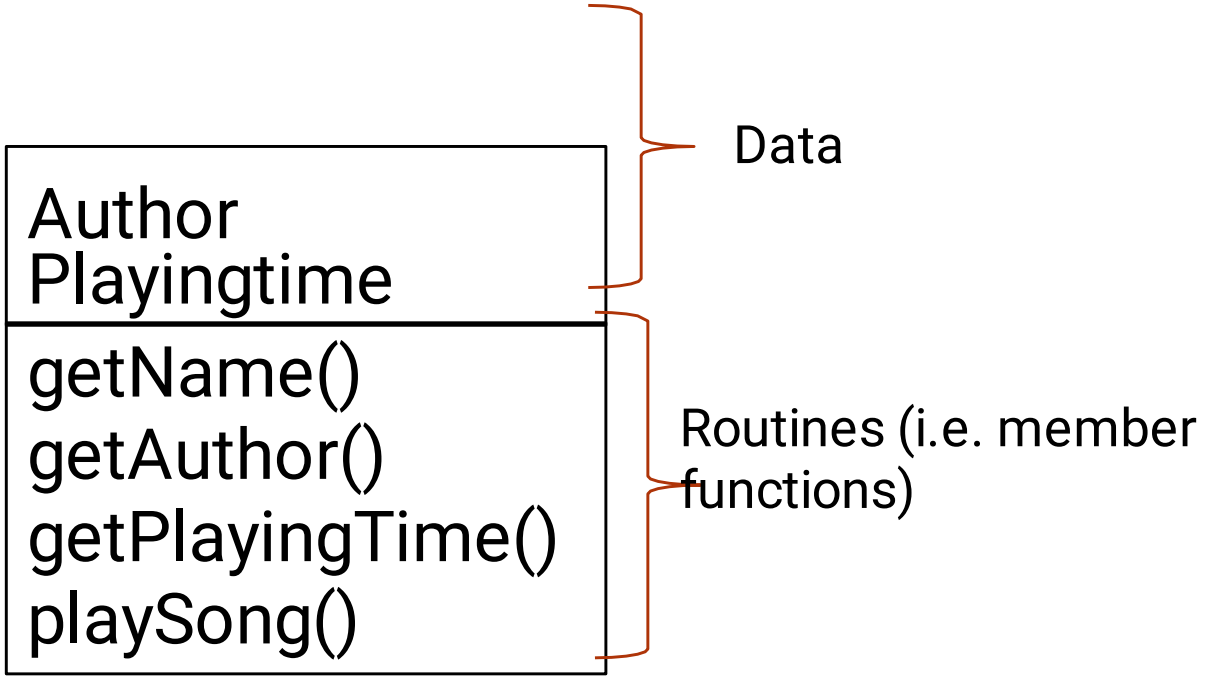
OOPS and Java - Objects and Classes

- The above mentioned **data** in the index card file can be used to get the following types of information about a particular song of interest.
  - Name of the song
  - Name of the Author,
  - Playing time of the song.
- Name the routines (funtions) to get the above mentioned three types of information from the index card file as **getName()**, **getAuthor()**, and **getPlayingTime()** and the routine for instructing the computer to play a song as **playSong()**.

OOPS and Java - Objects and Classes

- It should be carefully note here that all the above mentioned four routines are concerned with the three data items.

- The data and the related routines are combined to together to form



OOPS and Java - Objects and Classes

- Once we define a class, we can declare a group of objects for that class.
- Each object in such a group contains the data and member functions that have been defined in that class.
- In other words, each object in such a group of objects imitates the exact characteristics of the class, which serves as a plan or template.

OOPS and Java - Encapsulation

- One of the most powerful and important points of OOP is that access to a class is strictly regulated.
- If we wish to modify or access the data in an object, we have to issue a command to that object, which will use its member function to retrieve the requested data and communicate them.
- The above mentioned mode of retrieval means that the data are wantonly hidden, so that they are not directly accessible to the user. Here, the data and its related

OOPS and Java - Inheritance

- Inheritance is one the most powerful capabilities of object-oriented programming.
- Using this concept, a new class of objects can be derived from an old one. This process is called **inheritance** because the new class inherits the characteristics of the original class.
- The new class is called a **derived class** of the original class and the original class is called the **base class** of the new class.

OOPS and Java - Inheritance

- The derived class can called as sub class or child class.
- The base class can called as super class or parent class.
- There are five types of Inheritance
  - Singe Inheritance – One base class, one derived class.
  - Multilevel Inheritance – The derived class canact as base class.
  - Multiple Inheritance – More than one base class, one

OOPS and Java – Polymorphism

- We can build up subclasses from a base class by using the concept of inheritance.
- We can specify a unique behavior to each subclass by using the concept of polymorphism.
- Thus, we can design the derived classes in such a way that they respond to a message that a base class responds to, but at the same time they can perform different functions by making use of concepts of polymorphism.

OOPS and Java – Polymorphism

- If we redefine the functionality of a base class member function in a subclass, we say that we override its base class functionality.
- Overloading an important feature of OOP is that process of attaching different functionalities to a function or a operator depending on the different contexts.
- Two types.
  - Runtime Polymorphism – Virtual Functions

A Simple Java Program
(Creating and Executing)

A Simple Java Program (Creating & Executing)

- File name is ***Hello.java***

```
class Hello
{
        public static void main(String[] args)
        {
                System.out.println("Hello,  Welcome to
Java Application");
        }
}
```

- Type the above program in notepad editor.
- Save it as ***Hello.java*** in bindirectory of java software.
- Compile theprogram using command ***javac Hello.java***

A Simple Java Program (Creating & Executing)

- In java, a source program is first compiled and interpreted. After the compilation is over, the compiled version of the source file is stored in special format, namely the bytecode format.
- The source file is stored under the name *Hello.java*
  - The compiled code is stored under the name *Hello.class* During the interpretation stage, *Hello.class* file is converted into the machine code

A Simple Java Program (Creating & Executing)

- Hello.class file contains the byte code corresponding to the code that we have written in Hello.java.
- The Java interpreter, name java, loads the bytecode in Hello.class file, starts the execution of our program, and loads the necessary library files.

Syntax: *java class name* Example: *java Hello*

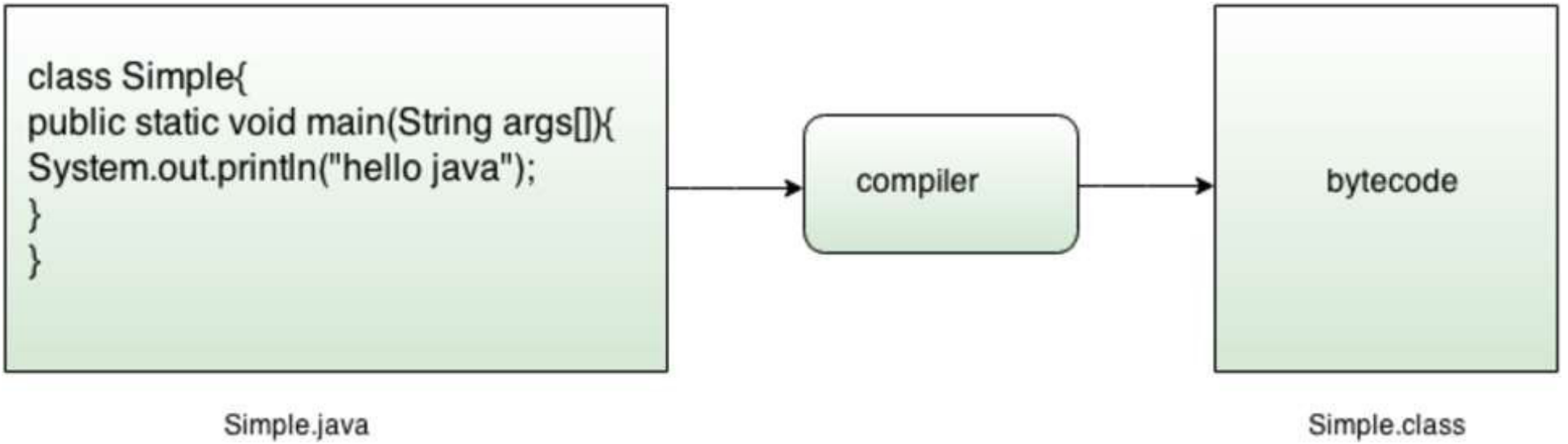
- ***Ouput will be***

Hello, Welcome to Java Applicaiton

What happens at Compile Time?

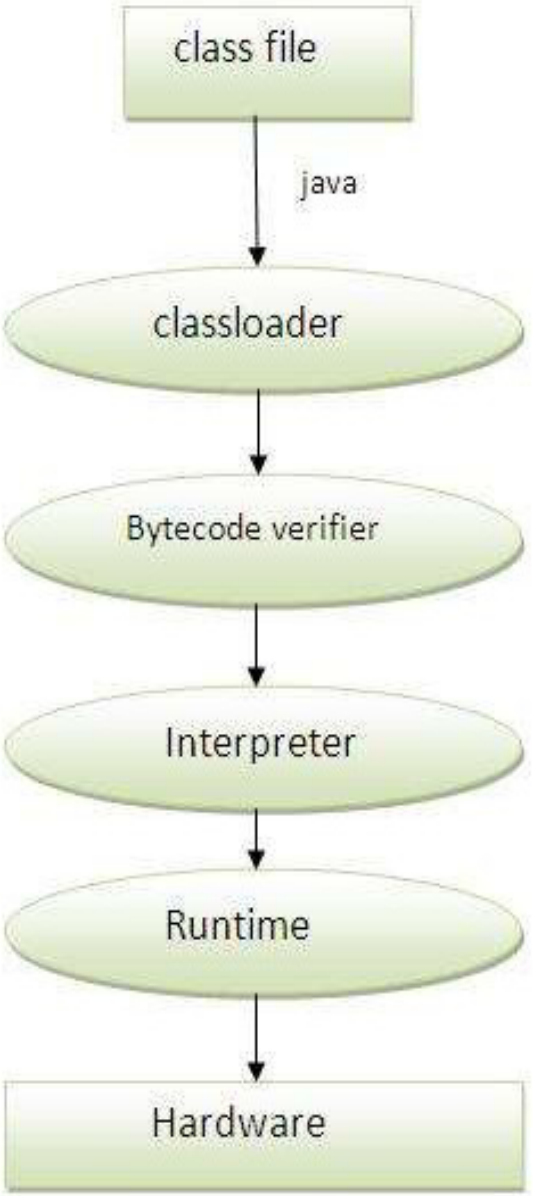
time?

- At compile time, java file is compiled by Java Compiler (It does not interact with OS) and converts the java code into bytecode.



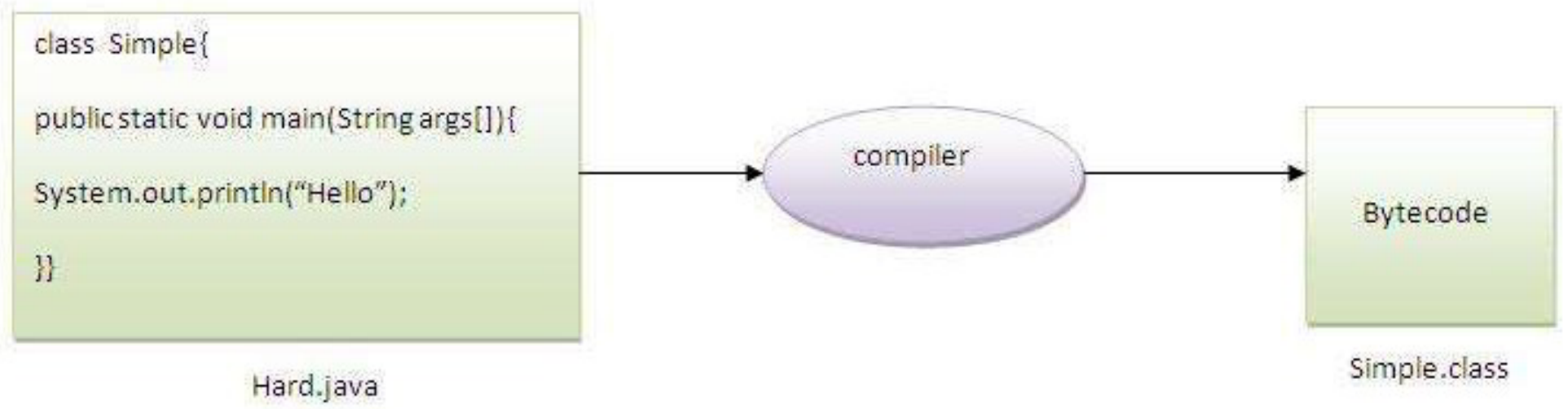
time?

- **Classloader:** is the subsystem of JVM that is used to load class files.
- **Bytecode Verifier:** checks the code fragments for illegal code that can violate access right to objects.
- **Interpreter:** read bytecode stream then



by other name than the class name?

- Yes, if the class is not public. It is explained in the figure given below:
- **To compile:** javac Hard.java
- **To execute:** java Simple



A Simple Java Program
(Understanding)

A Simple Java Program (Understanding)

- File name is *Hello.java*

```
class Hello
{
        public static void main(String[] args)
        {
                System.out.println("Hello,   Welcome to
Java Application");
        }
}
```

- As Java is a pure OOP-based language, each simple Java program is written in the form of a class.
- In java, a class consists of certain **data members** and the related **methods.**

A Simple Java Program (Understanding)

- Since the Hello.java program is a very simple one involving any input or output variables, it doesn't contain any data members.
- It contains only one method, namely the main(), which must be present in every Java stand-alone application. Since this method is not expected to return any value, the return type of this method is indicated in the above program as void.
- In java, we usually create an object of a class and

A Simple Java Program (Understanding)

- However, in certain cases, we prefer to access the methods in a class without creating an object for that class. Such methods are known as static methods.
- It shall be noted here that main() is a static method.
- As is apparent from the above program, the elements of an array of String type, called args, constitute the parameters of the main() method.

A Simple Java Program (Understanding)

- But it is not necessary to always provide the values for these parameters.
- The first periodin ***System.out.println*** means "locate the ***out*** object in the System class".
- The Second periodmeans "apply the***println()*** method to that object".
- The System class, the ***out*** object and the ***println()*** method have all been predefined by the Java developers.

Character Set and Tokens

Character Set

- The set of characters allowed in Java constitutes its character set. All the constituents of a java program, such as the constants, variables, expressions and statements are written only by using characters in the following character set:

  Numerals : 0-9

  Alphapets : a-z (Lower Case), A-Z (Upper Case)

  Special characters: + - * / % = < > ( ) . \ ; , ‘ ’ : ? # $! ^

  & ~ [ ] { }

- Java uses the uniform 16-bit coding scheme called Unicode to represent characters internally. The Unicode can

Tokens - Keywords

- The smallest individual entities in a Java program are known as tokens.
- Five important types of tokens are listed below:
  - Reserved keywords, identifiers, literals, operators, separators.
- Around 60 words have been reserved as the keywords in Java. They have predefined specific meanings. They should be written in lower-case letters. Here are some valid keywords.
  - break, continue, switch, private, while

Tokens – Rules for defining identifiers

- The names that the programmer assign for the classes, methods, variables, objects, packages and interfaces in a java program are known as **identifiers**. They should follow the following rules.
  - They shall contain digits, alphabets, underscore and dollar sign characters.
  - No white spaces are allowed.
  - A keywords should not be used as a identifier.
  - The first character should not be a digit.
  - They shall have any number of characters.

Tokens – Valid & Invalid Identifiers

- Some valid identifiers are furnished below
  - ***rectangle, perimeter$, count, r2, shape2d, tv_Channel***
- Some invalid identifiers are furnished below
  - ***rect angle, 2day, tv&Channel, good morning, wel@com***

Tokens – Literals

- A sequence of valid characters that represents a constant value is called a literal. Literals are also know as constants.
- Five major types of literals available in Java are as follows
  - Integer
  - Floating point
  - Character
  - String

Tokens – Operator and Separator

- A symbol that represents an operation, such as multiplication or division, is called an **operator**.
- Some of the well-known operators are available in Java are as follows.
  - + - * / %< > && || ! ++ -- += -= *= /=
- A symbol that is used to separate one group of code from another group is called a **separator**.

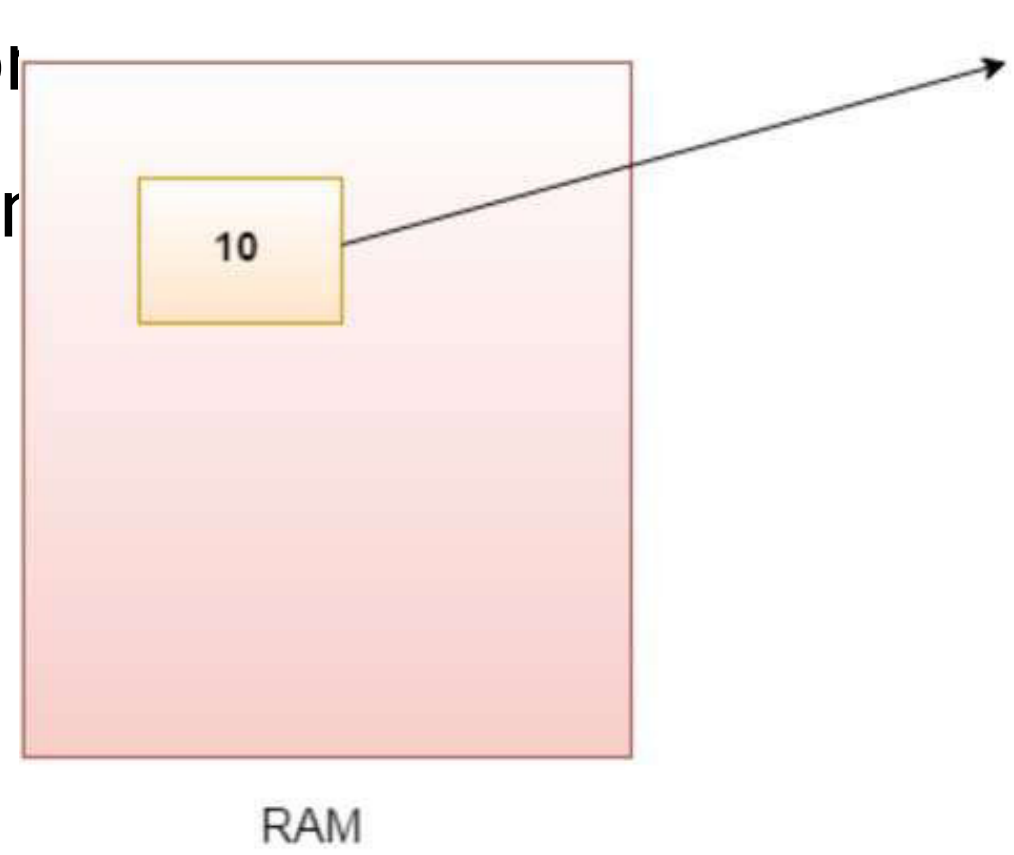
Some of the well-known separators are listed below

( ) [] ; , ‘ ’ “ ”

Variables and Constants

Variables

- **Variable** is name of *reserved area allocated in memory*. In other words, it is a *name of memory location*.
- It is a co s value ca



Variables - Rules

- The first character in a variable name should not be digit.
- A variable name may consist of alphabets, digits, the underscore character and the dollar character.
- A keyword should not be use as a variable name.
- White spaces are not allowed within a variable name.
- A variable name can be of any length

Variables

- Every variable has a **data type**.
- Every variable has a **value**.
- Every variable is the name of a **storage location.**
- Every variable has its **size**.

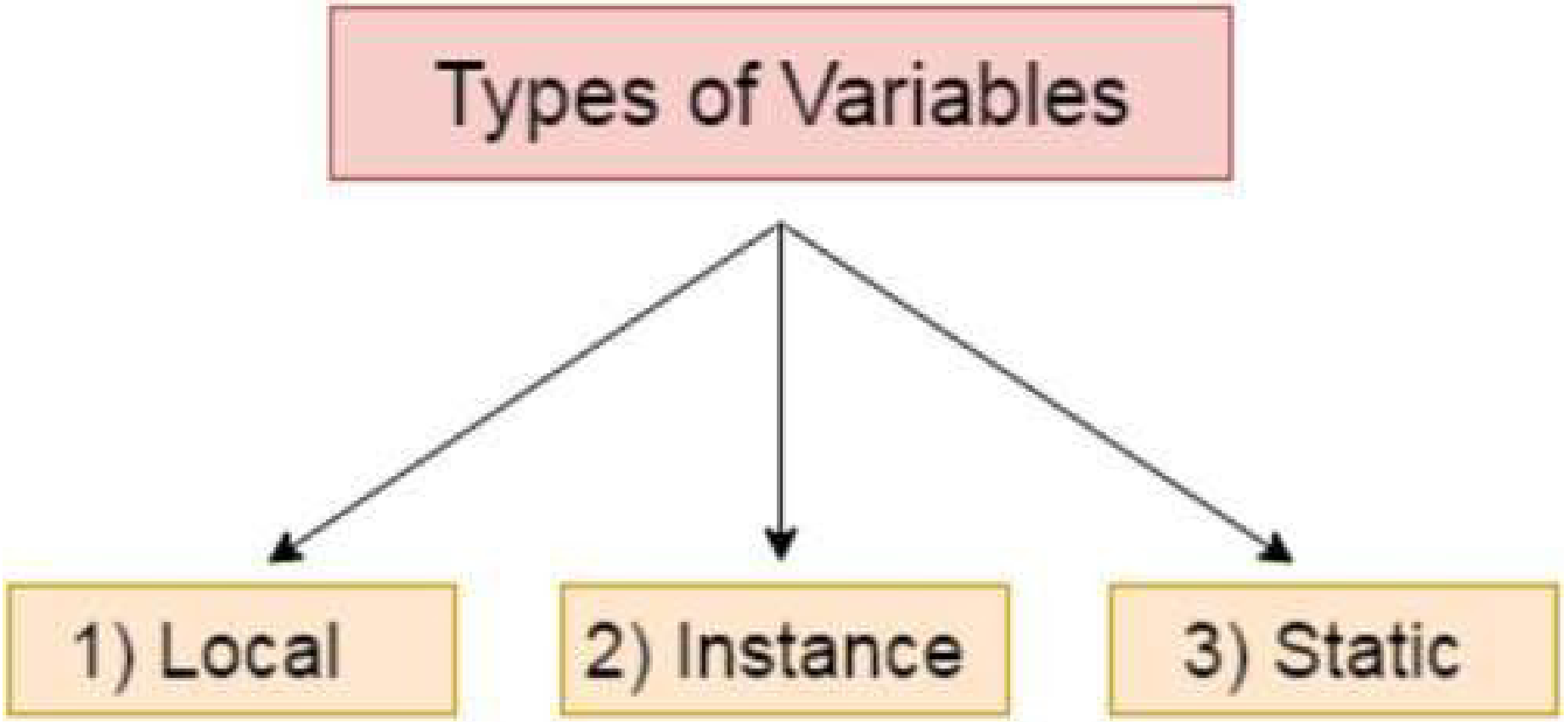
Variables - Declaration

- Variable can declared as follows

  access-specifier data-type variable-name

- Here,
  - ***access-specifier*** specifies which methods can access the variable. Most variables are declared as private.
  - ***data-type*** refers to the type of the variable. Here is a valid type declaration statement.

Variables – Types of Variables



Variables – Types of Variables

1) **Local Variable**

A variable which is declared inside the method is called local variable.

2) **Instance Variable**

A variable which is declared inside the class but outside the method, is called instance variable . It is not declared as static.

3) **Static variable**

A variable that is declared as static is called static variable. It cannot be local.

Variables – Examples to understand types of Variables

```
class A
{
        int data=50;//instance variable
        static int m=100;//static variable

        void method()
        {
                int n=90;//local variable
        } //end of method.
}//end of class
```

Constants

- ***Constants*** means it can not be changed, that means fixed value. Declaration as follows:

  const const-name=value;

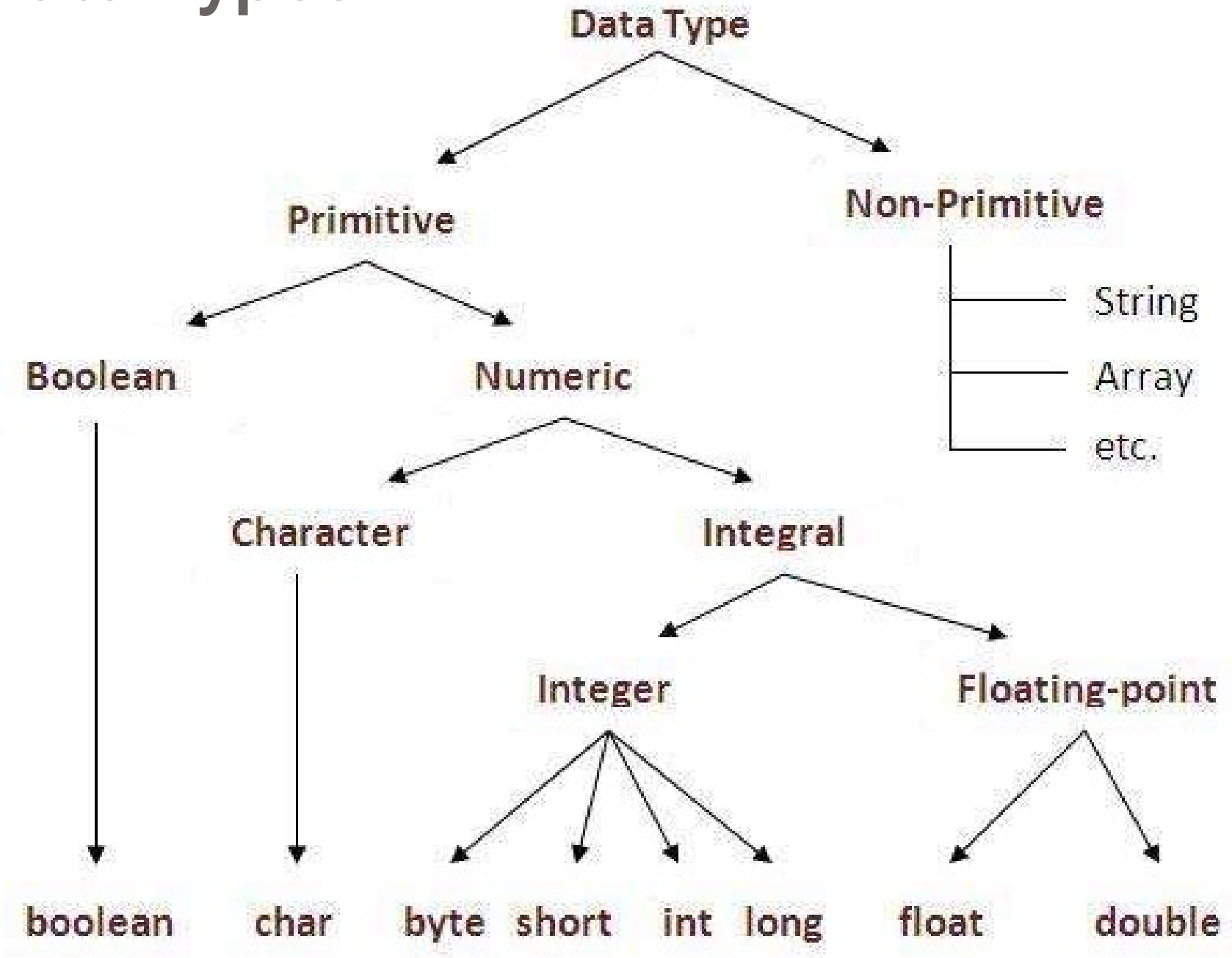
- ***Types are***
  - Integer constant
  - Floating point constant
  - Character constant
  - Boolean constant
  - Backslash character constant or Escape Sequence Character

Constants – Backslash Character Constants

Backslash character constant	Meaning
\b	Backspace
\n	New line
\t	Tab
\r	Carriage return
\f	Form feed
\\	Backslash
\’	Single quote
\”	Double quote

Data Types

Data Types


Data Type
Primitive
Non-Primitive
String
Array
etc.
Boolean
Numeric
Character
Integral
Integer
Floating-point
boolean
char
byte
short
int
long
float
double

Data Types – Default Values

Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

HomeWork: Listthe data types with range

Operators and Expressions

Operators and Expressions

- **Operator** in java is a symbol that is used to perform operations.
- There are many types of operators in java such as unary operator, arithmetic operator, relational operator, shift operator, bitwise operator, ternary operator and assignment operator.
- These operators are classified into three
  - Unary Operator – works with one operand
  - Binary Operator – works with two operands.

Operators and Expressions

- Combination of operators and operands are called expressions.
- When we assign an expression to a variable, the types of the variable and the expression should be compatible.

double sum = 10.1; int total = sum;

We must convert the floating point expression to integer by using a cast, as shown below

Library Methods

Library Methods

- Library methods are normally used to write the expressions in a compact manner. For example, the expression

$$x = \frac{\sqrt{b \pm b^2 - 4ac}}{2a}$$

can be compactly represented as shown below, by using the sqrt() library methods:

```
( -b + Math.sqrt(b*b-4*a*c))/(2*a)
```

Library Methods

Library Method	Purpose
Math.sqrt(x)	To find square root of x (>=0)
Math.pow(x,y)	To find X^y (x >0 and x =0 and y>0 or x<0 and y is an integer)
Math.sin(x)	To find Sine of x (x in radians)
Math.cos(x)	To find Cosine of x
Math.tan(x)	To find Tangent of x
Math.asin(x)	To find Arc sine of x
Math.acos(x)	To find Arc cosine of x
Math.atan(x)	To find Arc tangent of x
Math.toRadians(x)	To convert x radians to degrees

Library Methods

Library Method	Purpose
Math.toDegrees (x)	To convert x degrees to radians
Math.exp(x)	To find e^x
Math.log(x)	To find natural log
Math.round(x)	To find closest integer to x
Math.ceil(x)	To find smallest integer >=x
Math.floor(x)	To find largest integer <=x
Math.abs(x)	To find the absolute value |x|

Strings

Strings

- ***string*** is the most important data type used in programs.
- A ***string*** is a sequence of characters. In, Java a string is enclosed in quotation marks, as in the case of the example “Don Bosco College”.
- The ***String*** keyword, which represents the ***string*** data type, starts with an uppercase letter. This is because of the fact that the keyword ***String*** is the name of a predefined class.

Strings – *length()* Method

- The number of characters in a string is provided by the length() method of the String class, as illustrated below

```
String collegeName = “Don Bosco College”;  int wordLength =
collegeName.length();
```

- Since the string “Don Bosco College” consists of 17 characters, the *length()* method will return the value 17.
- A string contains no is called an

Strings – Concatenation Operator (+)

- Two strings shall be concatenated by using the + operator. This operator is a powerful operator. If one of the two operands, either to the left or the right of the + operator is a string, then the other operand is automatically converted into a string and both strings are concatenated. Here is an example.

```
String s = “Price:”;  int price = 83;
```

Strings

- Here, the expression s + price has strings operand and an integer operand. They will be concatenated into the single string “Price : 83” and assigned to the variable result.

Input and Output Statements (Scanner and System Class)

Input and Output Statements

- Every program needs some data as input. Similarly, every program provides some processed data as output.
- While the input operations enable us to provide input data to a program, the output operations help us to get the output displayed in a desired format.
- Java provides a rich set of I/O functions for performing a variety of operations.

I/O statement – Console Output

- The *out* object in the *System* class represents the standard output stream.
- The *out* object has two methods for sending the output to the Console screen. They are *print()* and the *println()* methods.
- While the *print()* method displays the output without a newline character, the *println()* method displays the output with a newline character.
- Since both the *print()* and the *println()* methods are heavily overloaded methods, they can output many different types of data, such as ***int, double, char,***

I/O statement – Console Output

System.out.print("Welcome to Java."); System.out.println("Have a nice time.");

- Here, the *System.out.print()* method displays the message Welcome to Java on the screen and retains the cursor on the same line.
- The *System.out.println()* method displays the string Have a nice time and moves the cursor to the beginning of the next line. Here is a sample output:

Welcome to Java.Have a nice time.

I/O statement – Console Output

- Sometimes, we may wish to have a blank line between two sets of output values. We shall print a blank line by using *println()* method, as shown below:

```
System.out.println("Value of A = 1000");  System.out.println();
System.out.println("Value of B = 2000");
```

- The above snippet will render the output in the following form:

```
Value of A = 1000
```

I/O statement – Console Input

- In java, any input is read in as a string.

```
System.out.print("Enter Your Name :");  InputStreamReader reader =
new
                                        InputStreamReader(System.in);
BufferedReader in = new BufferedReader( reader);
String name = in.readLine();
```

- In the above program, the name that is keyed in by the user is received by the system and stored in the buffer System.in terms of bytes.

I/O statement – Console Input

- When the second statement in the above snippet is executed, the bytes in the buffer *System.in* are converted into characters and stored in the object *reader*.
- An object *reader* has *InputStreamReader*, can read characters, but it cannot read a while string at time.
- Inorder to overcome this limitation, we turn the input stream reader into a *BufferedReader* object,

I/O statement – Console Input

- When the fourth statement in the following snippet is executed, the *readLine()* method reads in a string (i.e. a name) and stores it in the *String* type variable name.
- Suppose that we provide 1000 as the input to the following snippet.
- This value will be initially read in as the string “1000” and stored as the value of a string type variable text.
- The *parseInt()* method, which is available in the

I/O statement – Console Input

```
System.out.print("Enter the amount in dollars:");
InputStreamReader reader = new
                                InputStreamReader( System.in );
BufferedReader in = new BufferedReader( reader );  String text = in.
readLine();
int dollar = Integer.parseInt( text );
```

- Here, *parseInt()* method is used to convert string into int.

I/O - Console Input using Scanner Class

- There are various ways to read input from the keyboard, the java.util.Scanner class is one of them.
- The **Java Scanner** class breaks the input into tokens using a delimiter that is whitespace by default.
- It provides many methods to read and parse various primitive values.
- Java Scanner class is widely used to parse text for

Methods

Method	Description
public String next()	it returns the next token from the scanner.
public String nextLine()	it moves the scanner position to the next line and returns the value as a string.
public byte nextByte()	it scans the next token as a byte.
public short nextShort()	it scans the next token as a short value.
public int nextInt()	it scans the next token as an int value.
public long nextLong()	it scans the next token as a long value.

I/O - Scanner Class Example

```
import java.util.Scanner;

class ScannerTest

{

public static void main(String args[])
{

        Scanner sc=new Scanner(System.in);  System.out.

        println("Enter your rollno");  int rollno=sc.nextInt();

        System.out.println("Enter your name");  String

        name=sc.next();
```

I/O - Scanner Class Example

```
        System.out.println("Enter your fee");

        double fee=sc.nextDouble();

        System.out.println("Rollno:"+rollno+"name:  "+name+"
fee:"+fee);

        sc.close();

}

}
```

Output:

```
Enter your rollno  111
Enter your name  Vijay
Enter  450000
Rollno:111 name:Vijay fee:450000
```

Control Statements (Conditional, Looping, Jumping)

Control Statements

- The control statements are used to control over the **order of execution of the statements.**
- The two major types of control statements available in Java are the **decision-making statements** and the **repetitive statements**.
  - **Decision-making Statements** – if, if-else, switch
  - **Repetitive statements** – for, while, do-

Control Statements (Conditional or Decision-making)

Control Statements

- The Java *if statement* is used to test the condition.

  It checks boolean condition: *true* or *false*.

- There are various types of if statement in java.
  - if statement
  - if-else statement
  - nested if statement
  - if-else-if ladder
- Switch statement

Control Statements – if statement

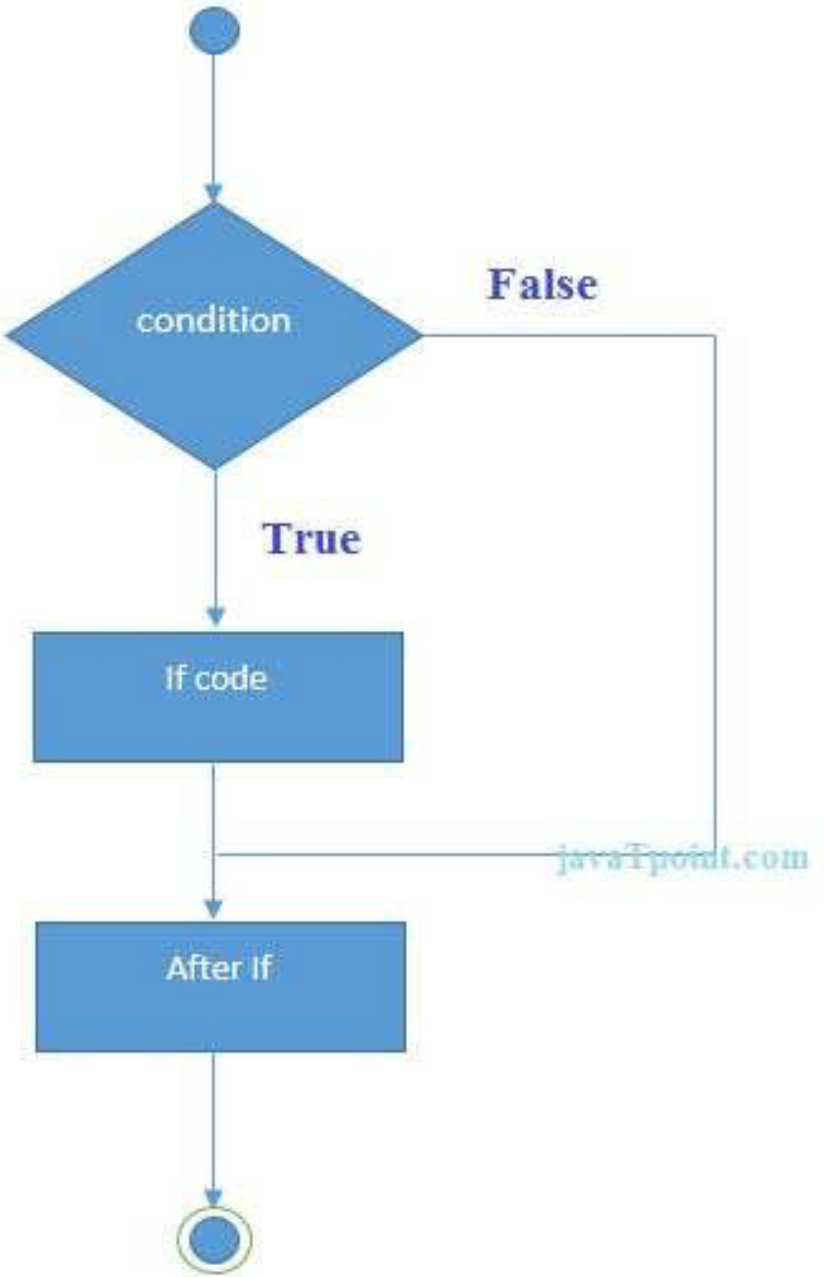
- The Java if statement tests the condition.

  It executes the *if block* if condition is true.

- The syntax as follows

```
if(condition)
{
        //code to be executed
}
```

Control Statements – if statement – flow chart



Control Statements – if statement - example

```
public class IfExample
{
        public static void main(String[] args)
        {
                int age=20;
                if(age>18)
                {
                        System.out.print("Age is  greater than 18");
                }
        }
}
```

Output:

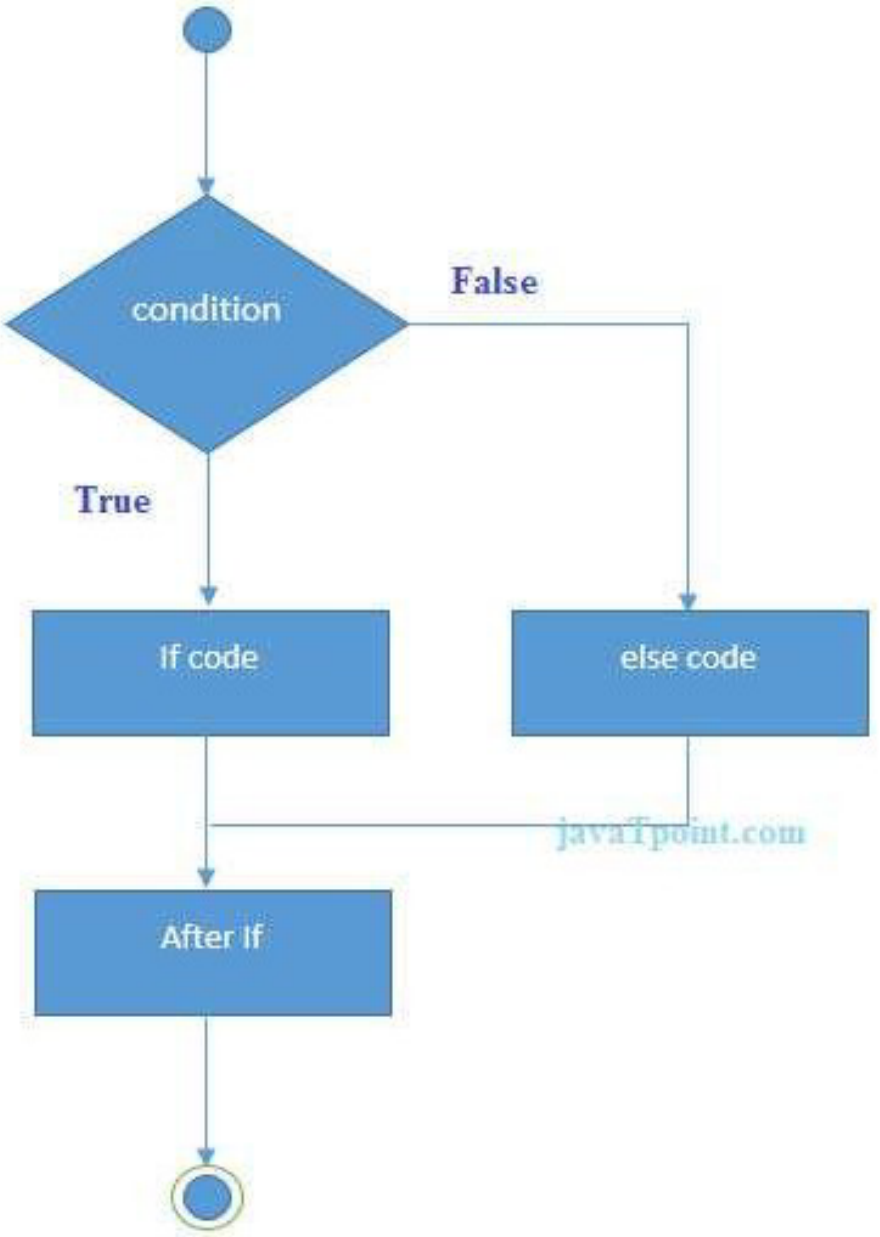
Age is greater than 18

Control Statements – if-else statement

- The Java if-else statement also tests the condition.
- It executes the if block if condition is true otherwise else block is executed.
- Syntax as follow:

```
if(condition)
{
        //code if condition is true
}
else
{
        //code if condition is false
}
```

Control Statements – if-else – flow chart



Control Statements – if-else - Example

```
public class IfElseExample
{
        public static void main(String[] args)
        {
                int number=13;
                if(number%2==0)
                {
                        System.out.println("even
number");
                }
                else
                {
                        System.out.println("odd number");
                }

        }
}
```

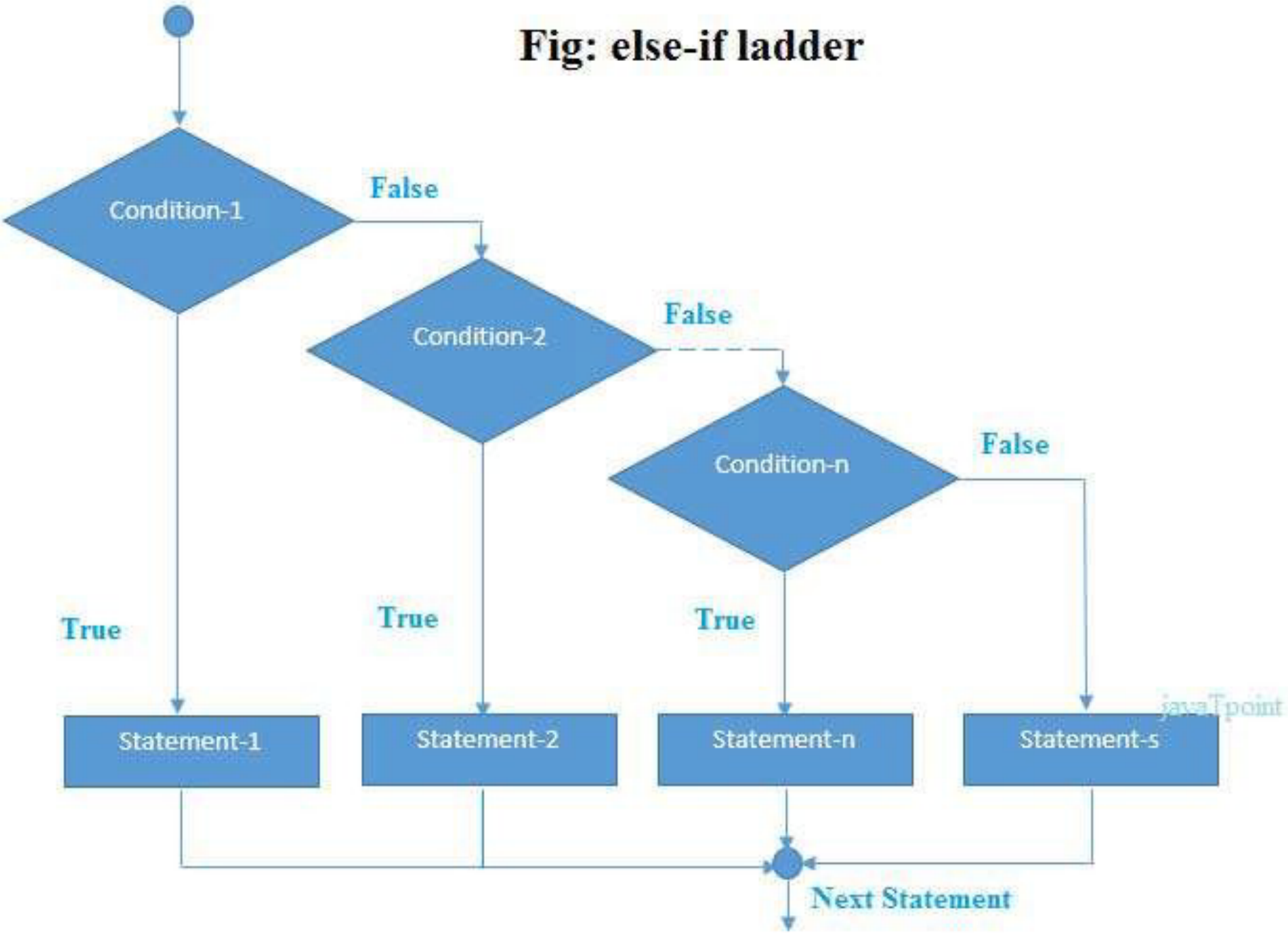
Output:
odd number

Control Statements – if-else-if ladder

- The if-else-if ladder statement executes one condition from multiple statements. The syntax as follows:

```
if(condition1){
        //code to be executed if condition1 is true
}else if(condition2){
        //code to be executed if condition2 is true
}
else if(condition3){
        //code to be executed if condition3 is true
}
...
else{
        //code to be executed if all the conditions are  false
}
```

Control Statements – if-else-if – flow chart



Control Statements – if-else-if Example

```
public class IfElseIfExample
{
        public static void main(String[] args)
        {
                int marks=65;
                if(marks<50)
                {
                        System.out.println("fail");
                }
                else if(marks>=50 && marks<60)
                {
                        System.out.println("D grade");
                }
```

Control Statements – if-else-if Example

```
else if(marks>=60 && marks<70)
{
        System.out.println("C grade");
}
else if(marks>=70 && marks<80)
{
        System.out.println("B grade");
}
else if(marks>=80 && marks<90)
{
        System.out.println("A grade");
}
```

Control Statements – if-else-if Example

```
            else if(marks>=90 && marks<100)
            {
                    System.out.println("A+ grade");
            }
            else
            {
                    System.out.println("Invalid!");
            }
        }
}
```

Output:

C grade

Control Statements: Nested –if:

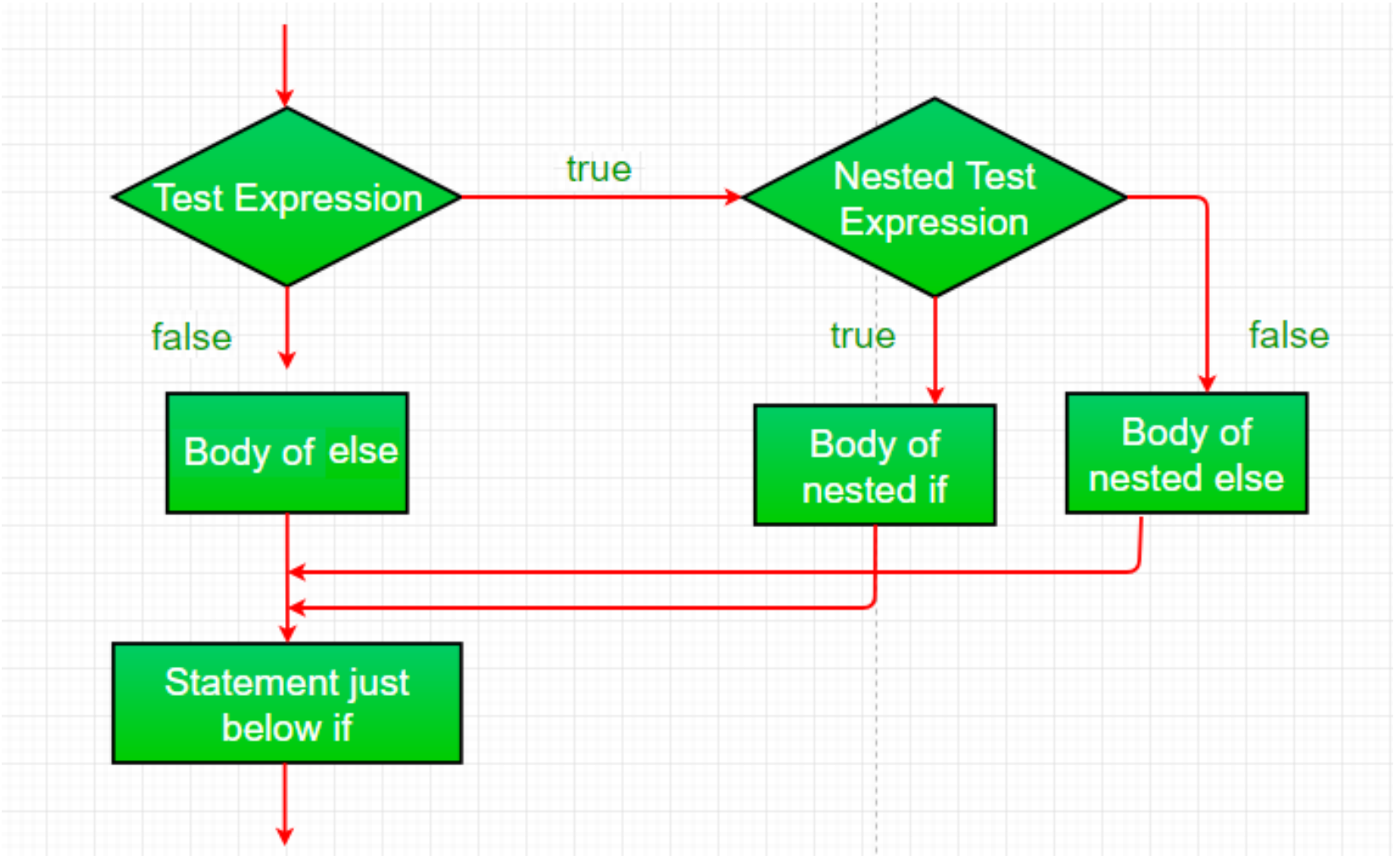
A nested if is an if statement that is the target of another if or else. Nested if statements means an if statement inside an if statement. Yes, java allows us to nest if statements within if statements. i.e, we can place an if statement inside another if statement.

Control Statements: Nested –if :

Syntax:

```
if (condition1)
{
      // Executes when condition1 is true
   if (condition2)
   { // Executes when condition2 is true
   }
         ……
 }
```

Control Statements: Nested –if- Flow chart



Control Statements: Nested –if- Example

```
class NestedIfDemo
{
    public static void main(String args[])
    {
        int  i = 10;

        if (i == 10)
        {
            // First if statement
            if (i < 15)
                System.out.println("i is smaller than 15");

            // Nested - if statement
            // Will only be executed if statement above
            // it is true
            if (i < 12)
                System.out.println("i is smaller than 12 too");
            else
                System.out.println("i is greater than 15");
        }
    }
}
```

Output:

i is smaller than 15
i is smaller than 12 too

Control Statements – switch statement

- The Java *switch statement* executes one statement from multiple conditions. It is like if-else- if ladder statement. The syntax as follows:

```
switch(expression)
{
    case value1:
                    //code to be executed;
                    break; //optional
    case value2:
                    //code to be executed;
                    break; //optional
                    ......
    default:
                    code to be executed if all  cases are not
matched;
}
```

chart

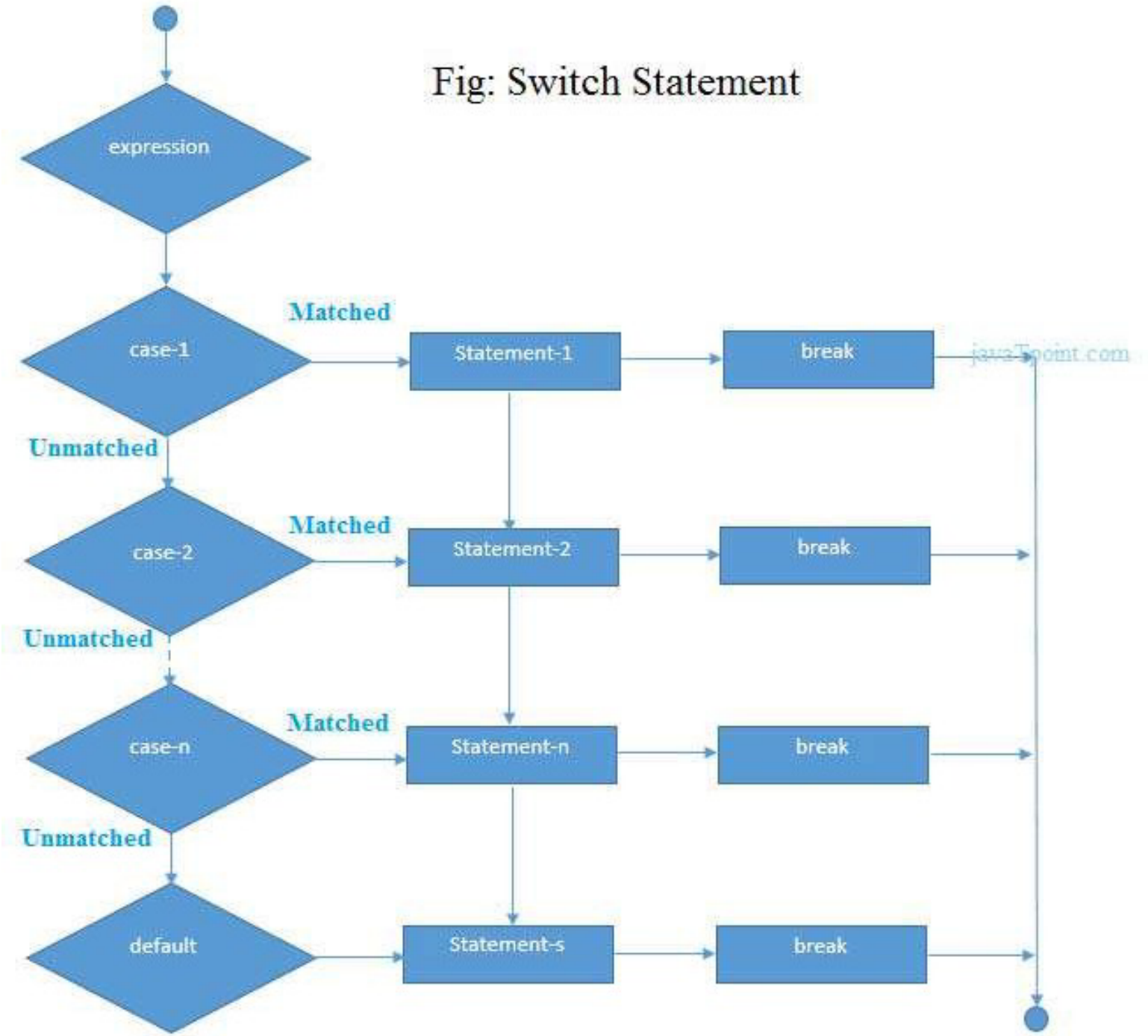


Fig: Switch Statement
expression
case-1
Matched
Statement-1
break
Unmatched
case-2
Matched
Statement-2
break
Unmatched
case-n
Matched
Statement-n
break
Unmatched
default
Statement-s
break
javaTpoint.com

Control Statements – switch – Example

```
public class SwitchExample
{
        public static void main(String[] args)
        {
                int number=20;
                switch(number)
                {
case 10: System.out.println(“Ten"); break;  case 20: System.
out.println(“Twenty"); break;  case 30: System.out.
println(“Thirty"); break;  default: System.out.println("Not in 10,
20 or
30");
                }
        }
}
```

Output

Twenty

Control Statements (Looping or repetitive)

Looping Statements

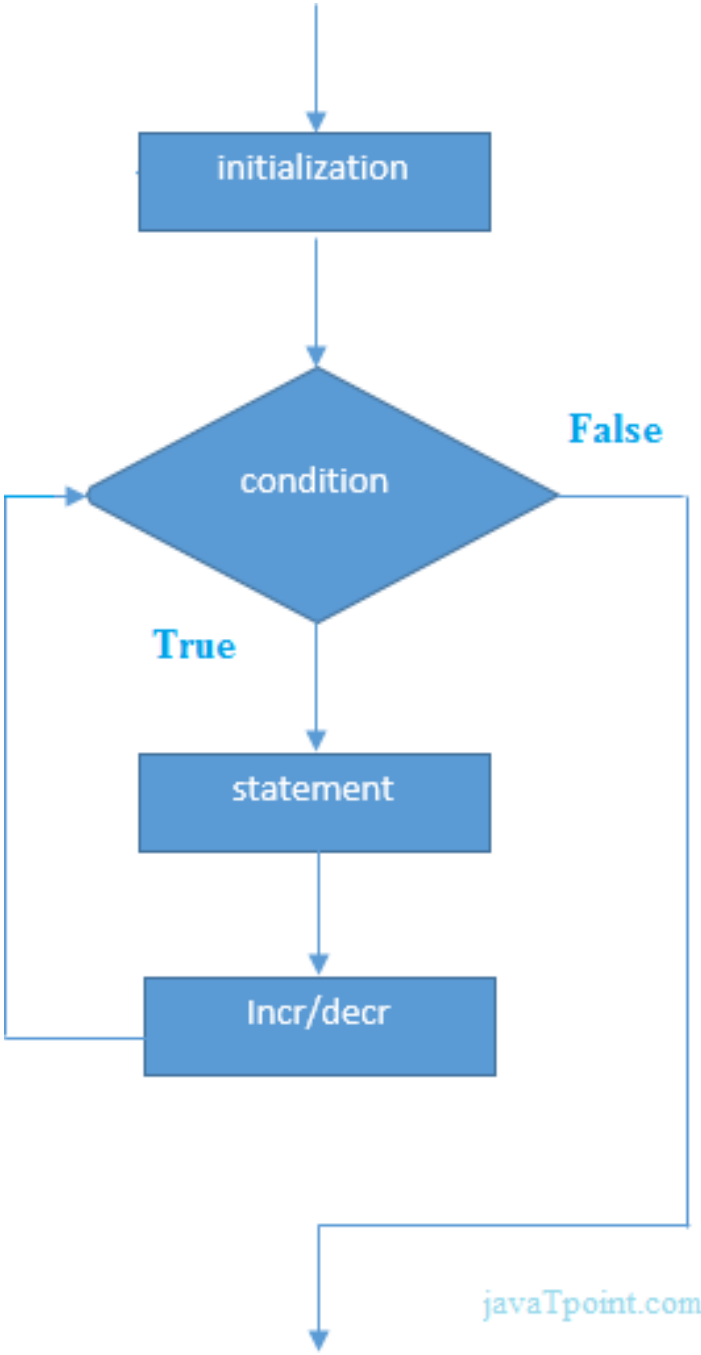
- The Java *for loop* is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.
- There are three types of for loop in java.
  - Simple For Loop
  - For-each or Enhanced For Loop
  - Labeled For Loop

Looping Statements – for loop

- The simple for loop is same as C/C++. We can initialize variable, check condition and increment/ decrement value. The syntax as follows:

```
for(initialization; condition; incr/decr)
{
        //code to be executed
}
```

Looping Statements – for loop – flow chart



Looping Statements – for loop - Example

```
public class ForExample
{
        public static void main(String[] args)
        {
                for(int i=1;i<=10;i++)
                {
                        System.out.println(i);
                }
        }
}
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Looping Statements – for each

- The for-each loop is used to traverse array or collection in java. It is easier to use than simple for loop because we don't need to increment value and use subscript notation.
- It works on elements basis not index. It returns element one by one in the defined variable. The syntax as follows

```
for(Type var:array)
{
        //code to be executed
```

Looping Statements – for each - Example

```
public class ForEachExample
{
        public static void main(String[] args)
        {
                int arr[]={12,23,44,56,78};
                for(int i:arr)
                {
                        System.out.println(i);
                }
        }
}
```

Output:

```
12
23
44
56
78
```

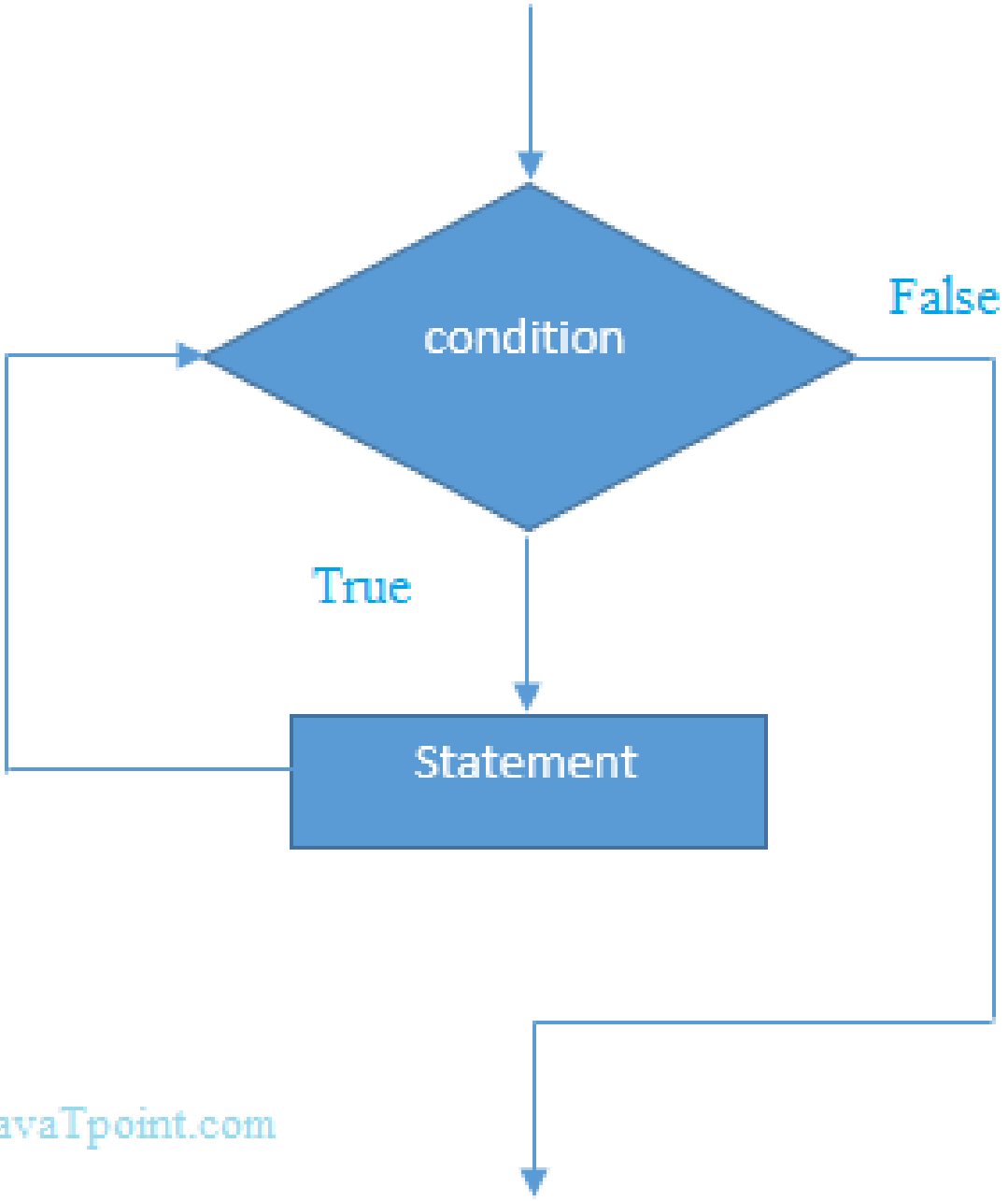
Looping Statements – While loop–Entry controlled

- The Java *while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

Syntax:

```
while(condition)
{
        //code to be executed
}
```

Looping Statements – While loop–Entry controlled



Example

```
public class WhileExample
{
        public static void main(String[] args)
        {
                int i=1;
                while(i<=10)
                {
                        System.out.println(i);  i++;
                }

        }
}
```

Output:

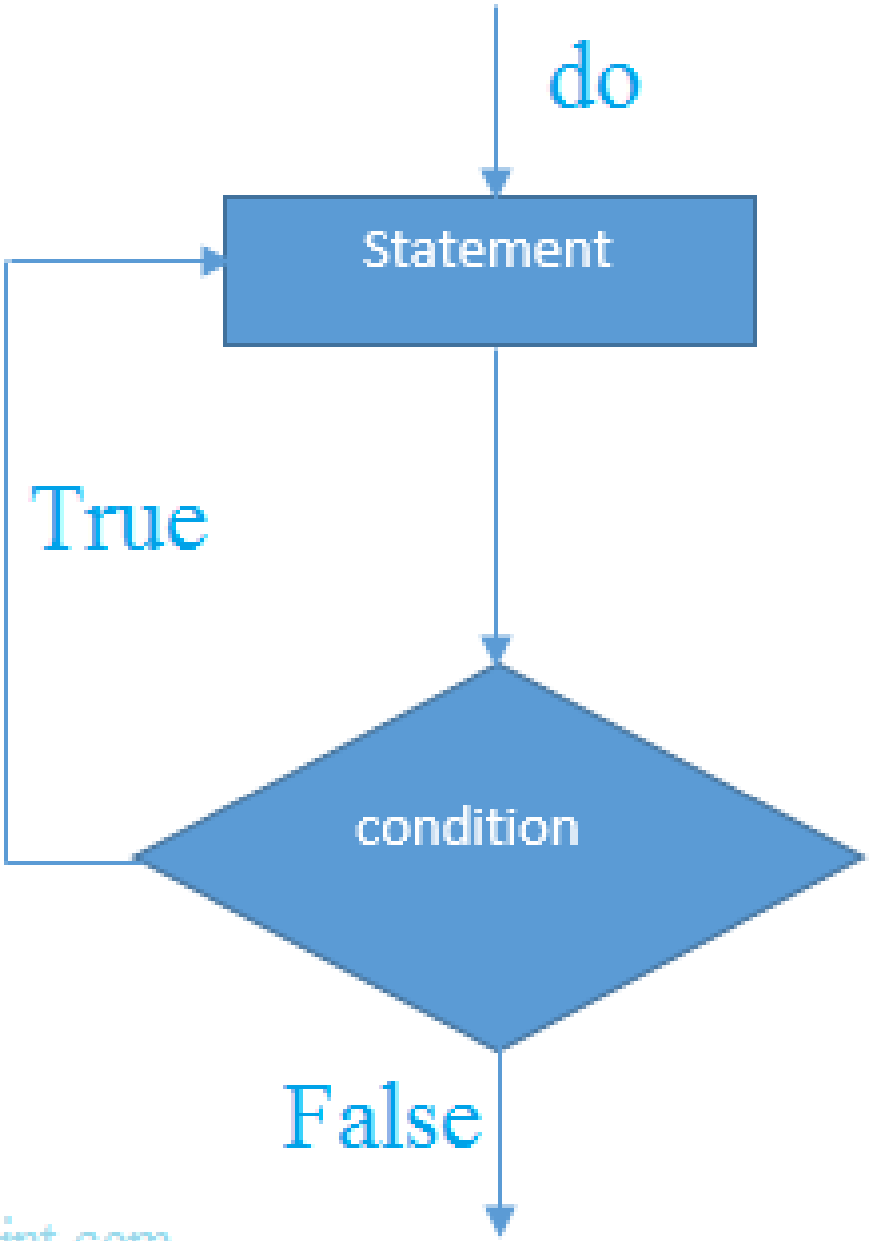
```
1
2
3
4
5
6
7
8
9
10
```

Looping Statements–do-while loop–Exit controlled

- The Java *do-while loop* is used to iterate a part of the program several times.
- If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop.
- The Java *do-while loop* is executed at least once because condition is checked after loop body. The syntax as follows:

```
do
{
```

Looping Statements–do-while loop–Exit controlled



Example

```
public class DoWhileExample
{
        public static void main(String[] args)
        {
                int i=1;
                do
                {
                        System.out.println(i);  i++;
                }while(i<=10);

        }
}
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Control Statements (Jumping)

Jumping Statements – break statement

- The Java *break* is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.

Syntax:

```
jump-statement;
break;
```

Jumping Statements – break statement

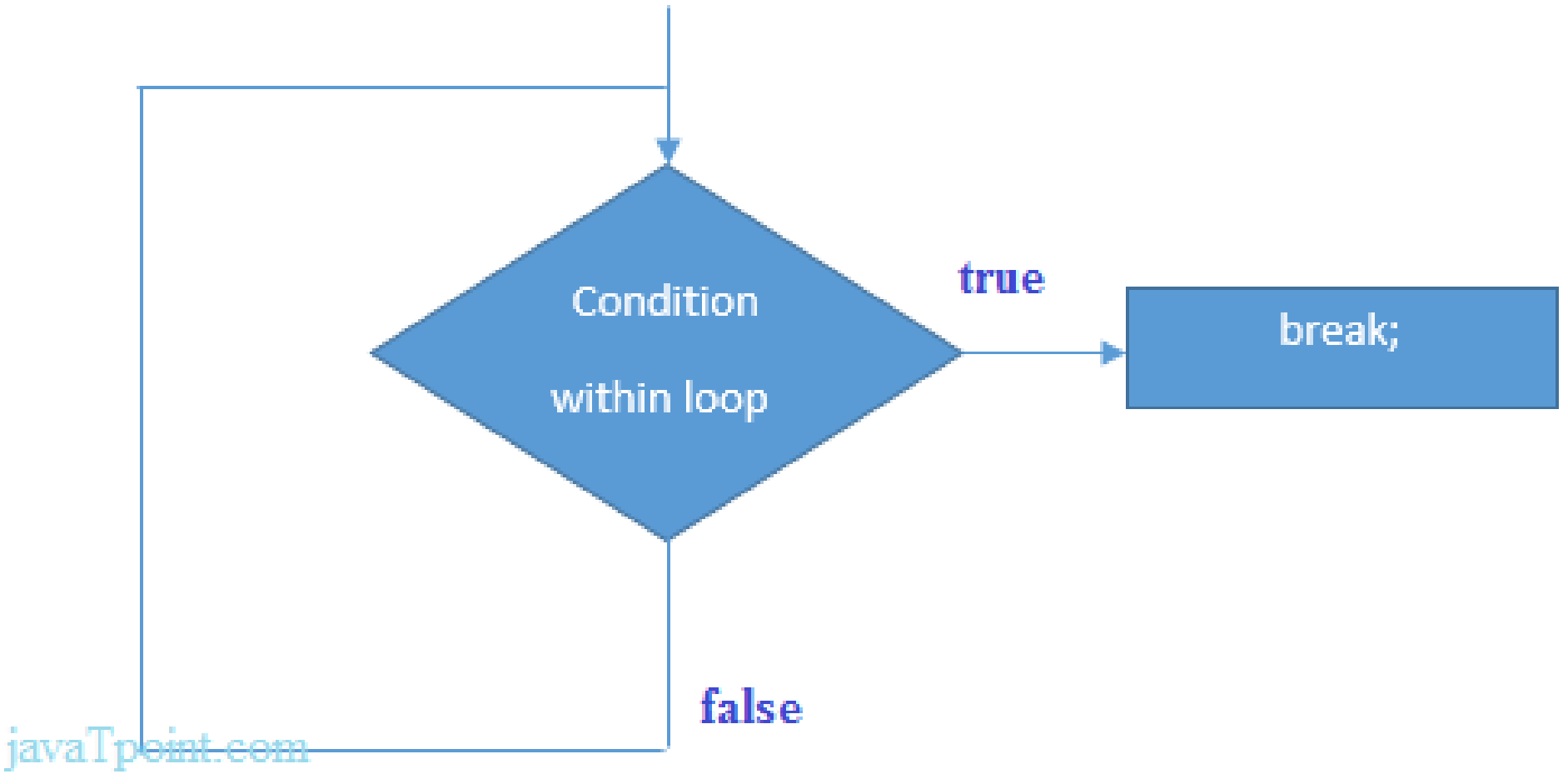


Figure: Flowchart of break statement

Jumping Statements – break statement-example

```
public class BreakExample
{
        public static void main(String[] args)
        {
                for(int i=1;i<=10;i++)
                {
                        if(i==5)
                        {
                                break;
                        }
                System.out.println(i);
                }
        }
}
```

Output:

1
2
3
4

Jumping Statements – continue statement

- The Java *continue statement* is used to continue loop. It continues the current flow of the program and skips the remaining code at specified condition. In case of inner loop, it continues only inner loop.

Syntax:

```
jump-statement;
continue;
```

Jumping Statements – continue Statement

```
public class ContinueExample
{
        public static void main(String[] args)
        {
                for(int i=1;i<=10;i++)
                {
                        if(i==5)
                        {
                                continue;
                        }
                        System.out.println(i);
                }
        }
}
```

Output:

1
2
3
4
6
7
8
9
10

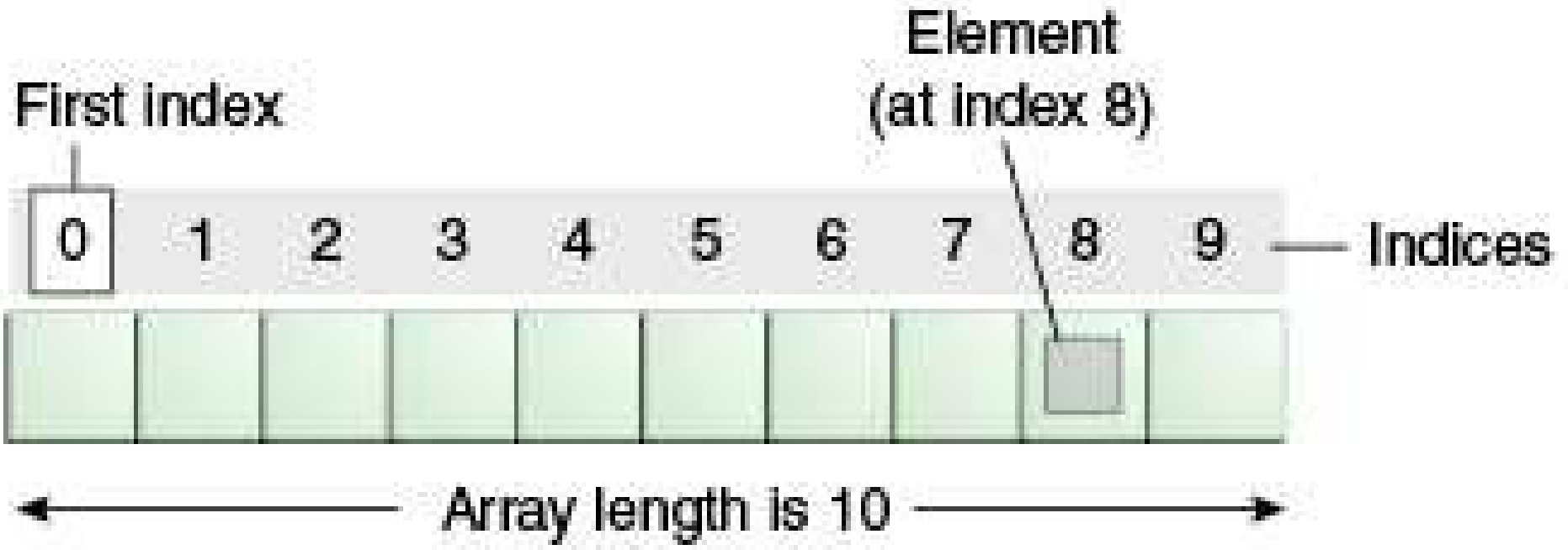
Arrays

Arrays

- Normally, Array is a collection of similar type of elements that have contiguous memory location.
- **Java array** is an object the contains elements of similar data type.
- It is a data structure where we store similar elements.
- We can store only fixed set of elements in a java array.

Arrays

- Array in java is index based, first element of the array is stored at 0 index.



Arrays – Advantage & Disadvantages

- Advantages:
  - **Code Optimization:** It makes the code optimized, we can retrieve or sort the data easily.
  - **Random access:** We can get any data located at any index position.
- Disadvantages:
  - **Size Limit:** We can store only fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection

Arrays

One Dimensional Array

One dimensional arrays –

Syntax to declare

dataType[] arr; (or) dataType []arr; (or) dataType arr[];

- Instantiation of an Array in java

arrayRefVar=**new** datatype[size];

Ex. int marks=new int[5];

One dimensional arrays – Example

```
class Testarray {

public static void main(String args[]){

int a[]=new int[5];//declaration and instantiation
a[0]=10;//initialization  a[1]=20;

a[2]=70;

a[3]=40;

a[4]=50;

//printing array
for(int i=0;i<a.length;i++)//length is the property of array  System.out.println(a[i]);

}}
```

Output:

```
10
20
70
40
50
```

Declaration, Instantiation, and Initialization

- We can declare, instantiate and initialize the java array together by:

```
int a[]={33,3,4,5};//declaration, instantiation and initialization
```

Passing array to method in Java

- We can pass the java array to method so that we can reuse the same logic on any array.

```
class Testarray2{
static void min(int arr[]){
        int min=arr[0];
        for(int i=1;i<arr.length;i++)
                if(min>arr[i])
                                min=arr[i];
                        System.out.println(min);
}
public static void main(String  args[]){
        int a[]={33,3,4,5};
        min(a);//passing array to method    } }
```

Output:3

Arrays

Two Dimensional Array

Two Dimensional Arrays

- In such case, data is stored in row and column based index (also known as matrix form).
- **Syntax to Declare Multidimensional Array in java**

```
dataType[][] arrayRefVar; (or)
dataType [][]arrayRefVar; (or)
dataType arrayRefVar[][]; (or)
dataType []arrayRefVar[];
```

- Example to instantiate Multidimensional Array in java

Example to initialize two dimensional Array in java

```
arr[0][0]=1;

arr[0][1]=2;

arr[0][2]=3;

arr[1][0]=4;

arr[1][1]=5;

arr[1][2]=6;

arr[2][0]=7;

arr[2][1]=8;

arr[2][2]=9;
```

Two Dimensional Array - Example

```
class Testarray3{
public static void main(String args[]){
//declaring and initializing 2D array
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
//printing 2D array  for(int i=0;i<3;i++){
for(int j=0;j<3;j++){
System.out.print(arr[i][j]+" ");
}
System.out.println();
}
}}
```

```
Output:1  2 3
       2  4 5
       4  4 5
```

Methods

General Form of a Method

Methods

- A method is a self-contained entity that carries out a well-defined task of some kind.
- For example, a method named ***rectangleArea()*** may be defined so as to compute the area of rectangle. The methods in class are its basic operation entities.
- Methods are also called functions, member functions (in class), routine or procedures.
- In java we are calling Methods.

Methods – General Form

- Each method carried out well-defined task.
- Each method has a **name** and a list of **parameters**.
- Any valid identifier can be assigned as the name of a method. However, this names of predefined methods and keywords are not to be used as the names of user-defined methods.
- The syntax of a method is as follows

```
access-specifier return-type method-name (parameter  list)
{
                    statement 1;
                    …
                    …
                    statement n;
```

Methods – General Form

- Here,

1. ***access-specifier*** specifies which other methods can call this method. Most methods shall be declared as public.
2. ***Return-type*** is the type of the value that is returned by the method to its caller. If no value returned by a method, then the keyword ***void*** should be specified in the place of ***type***.

Methods – General Form

- Here,

3. ***Method-name*** is the name assigned to the method by the user. It shall be any valid identifier.
4. ***Parameter-list*** is a list of parameters. This list, always enclosed in parentheses contains parameter names and their types. The values that we want to pass on to the method as input

Methods – General Form

- The parameters in the parameter list should be separated by commas.
- In no input data are to be provided to a method, then the method name should be followed by pair of ***empty*** parentheses.
- The ***body*** of the method contains a set of variables and a set of statements.
- The statements in a method describe the operations that are performed by the method.

Methods – General Form

- The variables that are declared in the method are know as ***local variables***.
- The local variables and the parameters associated with a method. They are removed from the memory, when the control comes out of the method.
- Local variables should be always initialized.
- A method may be instructed to return a value after its execution is over. A special statement namely the ***return*** statement of the following form shall be used for this purpose

return expression;

Methods – General Form - Example

- Here, ***data type*** of the value yielded by the ***expression*** should be same as the ***return-type*** of the method.

  Example,

```
int rectAngleArea()
{
        return length * width;
}
```

- Here the value yielded by the expression **length * width** should be *int*, as the ***return type*** of the method *rectangleArea()* is specified as int.
- Method name is rectAngleArea(), return type is int, no parameters list, inside body only one statement that is

Methods

Method of Invoking a Method

Methods – Invoking a Method

- Suppose we define a method named product() for finding the product of two numbers as shown below:

```
int product( int a, int b)
{
        return a * b;
}
```

- Once we define a method, such as the product() method, we shall invoke it any number of times.
- To use a method, we have to invoke it by specifying its name with the values for its

Methods – Invoking a Method

- For example, we shall invoke the *product()* method, as shown below

```
product(10,6);
```

- When the method *product()* is invoked as specified above, the numbers 10 and 6 will be passed to the method *product()*.
- As a result, the product **a * b** will be evaluated as **60** and this value will be returned by the *product()* method

Methods – Invoking a Method

- The value that is returned by a method shall very well be assigned to a variable and then involved in some computation. For example, consider the following **assignment statement.**

int p = product (10, 6);

- Here the value 60, which is returned by the product() method, will be stored as the value of the variable p. The value that is returned by a method shall also be directly displayed on the screen with the help of a statement of the following form:

Methods – Invoking a Method

- Here the value **60**, which is returned by the *product()* method will be displayed on the screen along with thestring *"Product of two numbers = "* as shown below:

Product of two numbers = 60

Methods

Method Overloading

Methods –Method Overloading

- When two methods in a class have the same name but different parameter lists, those two methods are said to be **overloaded**.

- **Advantage of method overloading?**

  Method overloading **increases the readability of the program**.

- There are two ways to overload the method in java, By changing
  1. Number of arguments
  2. By changing the data type

Methods –Method Overloading Example

```
class Calculation{

        void sum(int a,int b){  System.out.println(a+b); }
        void sum(int a,int b,int c){  System.out.println(a+b+c);
}

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

        obj=new Calculation();  obj.sum(10,10,10);



        obj.sum(20,20);

}
}
```

Output:30
40

Methods

Recursion

Methods – Recursion

- A method can invoke another method. It can also invoke itself. Such a method is known as a **recursive** method.
- The definition of a recursive method has the following two parts
  1. The **basis**, which defines the value of a recursive function for the first one or few values of the parameter(s).
  2. The **recurrence relationship**, which expresses

Methods – Recursion

- For example, let us consider the following recursive method *factorial()*.

```
static int factorial (int m)
{
        if( m==1 )
                return 1;
        return m *  factorial ( m – 1);
}
```

Methods – Recursion

- Here the factorial of 1 is 1. This is the **basis**.
- The factorial of m can be found from the expression ***m * factorial (m – 1).*** This is the **recurrence relationship.**
- Suppose we invoke the above recursive method *factorial()* with the argument 3, as shown below.

factorial(3);

- Now, the *factorial()* method will return the following expression:

*3 * factorial(2);*

Methods – Recursion

- So, the next call to the *factorial()* method will be as follows:

factorial(2);

- Now, the *factorial()* method will return the following expression

*2 * factorial(1);*

- So, the next call to the *factorial()* method will be as follows:

factorial(1);

Methods – Recursion

- Since the *factorial()* method is invoked with the argument 1at this stage, this method will return the value 1.
- At this stage, *factorial(2)* will be computed as **2**.
- Then the *factorial(3)* will be computed as **6**.

Random Class

java.util.Random Class

- The **java.util.Random** class instance is used to generate a stream of pseudorandom numbers.

  Following are the important points about Random:

  - The class uses a 48-bit seed, which is modified using a linear congruential formula.
  - The algorithms implemented by class Random use a protected utility method that on each invocation can supply up to 32 pseudorandomly generated bits.

java.util.Random Class Constructors

1. Random()

This creates anewrandom number generator.

2. Random(*long seed)*

This creates anewrandom numbe r generator using a single long seed.

java.util.Random Class Methods

1. **protected int next(int bits)**

This method generates the next pseudorandom number.

2. **boolean nextBoolean()**

This method returns the next pseudorandom, uniformly distributed boolean value from this random number generator's sequence.

3. **void nextBytes(byte[] bytes)**

This method generates random bytes and

java.util.Random Class Methods

4. **double nextDouble()**

This method returns the next pseudorandom, uniformly distributed double value between 0.0 and 1.0 from this random number generator's sequence.

5. **float nextFloat()**

This method returns the next pseudorandom, uniformly distributed float value between 0.0 and 1.0 from this random number generator's sequence.

6. **double nextGaussian()**

This method returns the next pseudorandom, Gaussian ("normally") distributed double value with mean 0.0 and standard deviation 1.0 from this random number generator's sequence

java.util.Random Class Methods

7. int nextInt()

This method returns thenext pseudorandom, uniformly distributed int value from this random number generator's sequence.

8. int nextInt(int n)

This method returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.

java.util.Random Class Methods

9. **long nextLong()**

This method returns the next pseudorandom, uniformly distributed long value from this random number generator's sequence.

10. **void setSeed(long seed)**

This method sets the seed of this random number generator using a single long seed.


Thank you!