

JAVA @11

Introduction to OOPs

Objective

After completing this session you will be able to understand,

- Introduction to OOPs
- OOPs concept – Encapsulation
- OOPs concept – Inheritance
- OOPs concept - Polymorphism

What is a Procedural Language

Procedural programming language helps the programmers to develop applications as series of instructions to achieve the desired functionality.

Disadvantages:

- Not easy to maintain for larger applications.
- Not Flexible for changes.
- Complex to reuse a common programming logic.
- Programmers find it complex to understand the program and fix bugs.

What is OOPs?

The disadvantages of procedural language gave birth to Object Oriented Programming Language (OOPs)

Programmers develop application by breaking them into small objects.

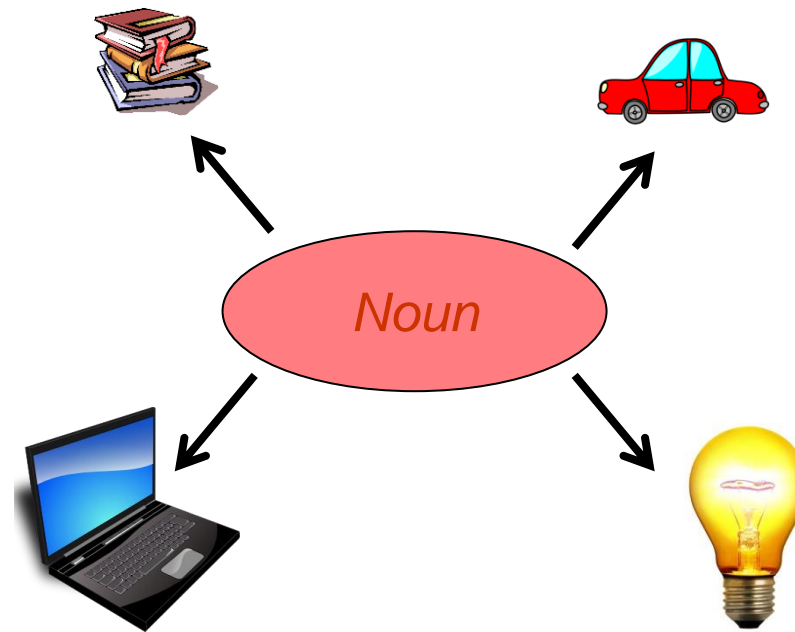
Advantages:

- The code is easy to maintain as the objects are self contained, cohesive units developed for specific functionalities.
- The objects can be reused to perform similar logics.
- The system is flexible to change.

What is Noun?

Definition: As a simple definition, a noun is any word that describes a person, place, or thing.

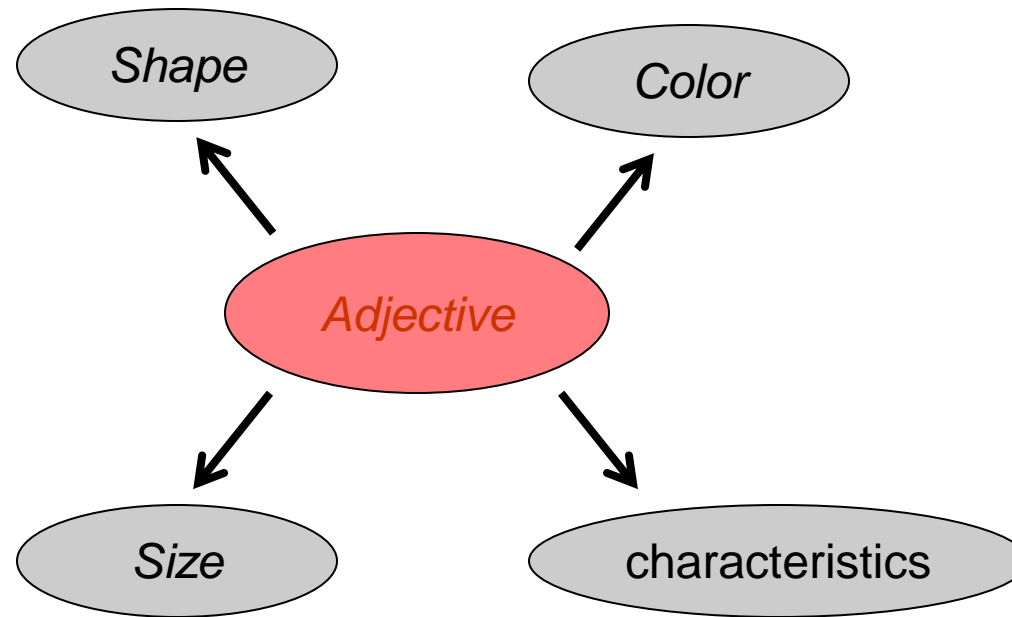
Example: The Man read a Book.



What is a Adjective?

Definition: Adjectives are describing words.

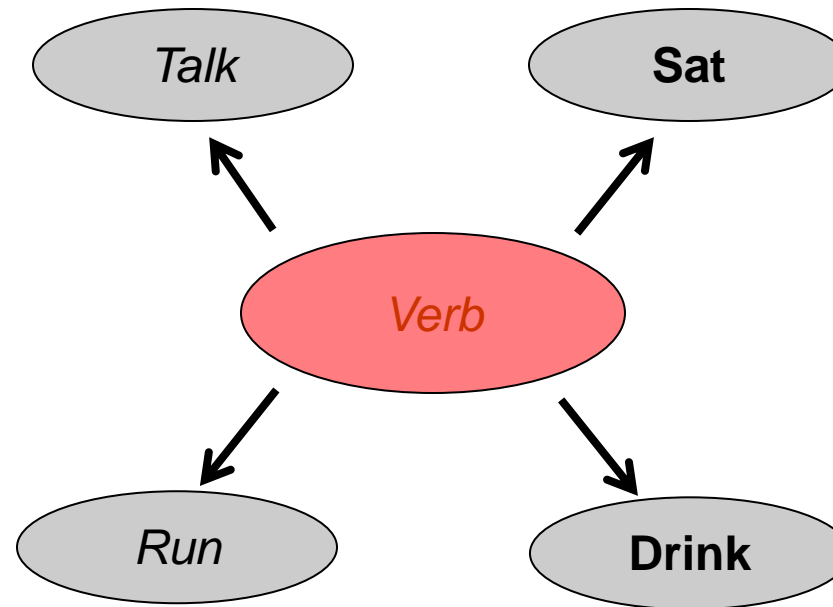
Example: The Man read a Java Book.



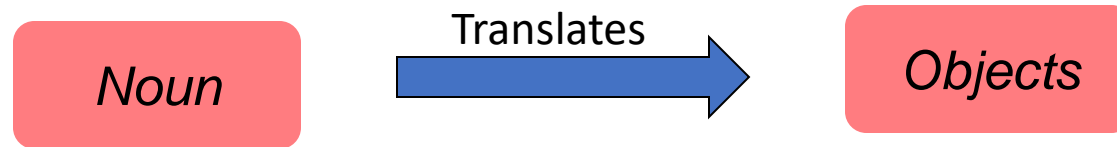
What is a verb

Definition: A verb is a word used primarily to indicate a type of action.

Example: The Man read a Java Book.



Analog to Software world

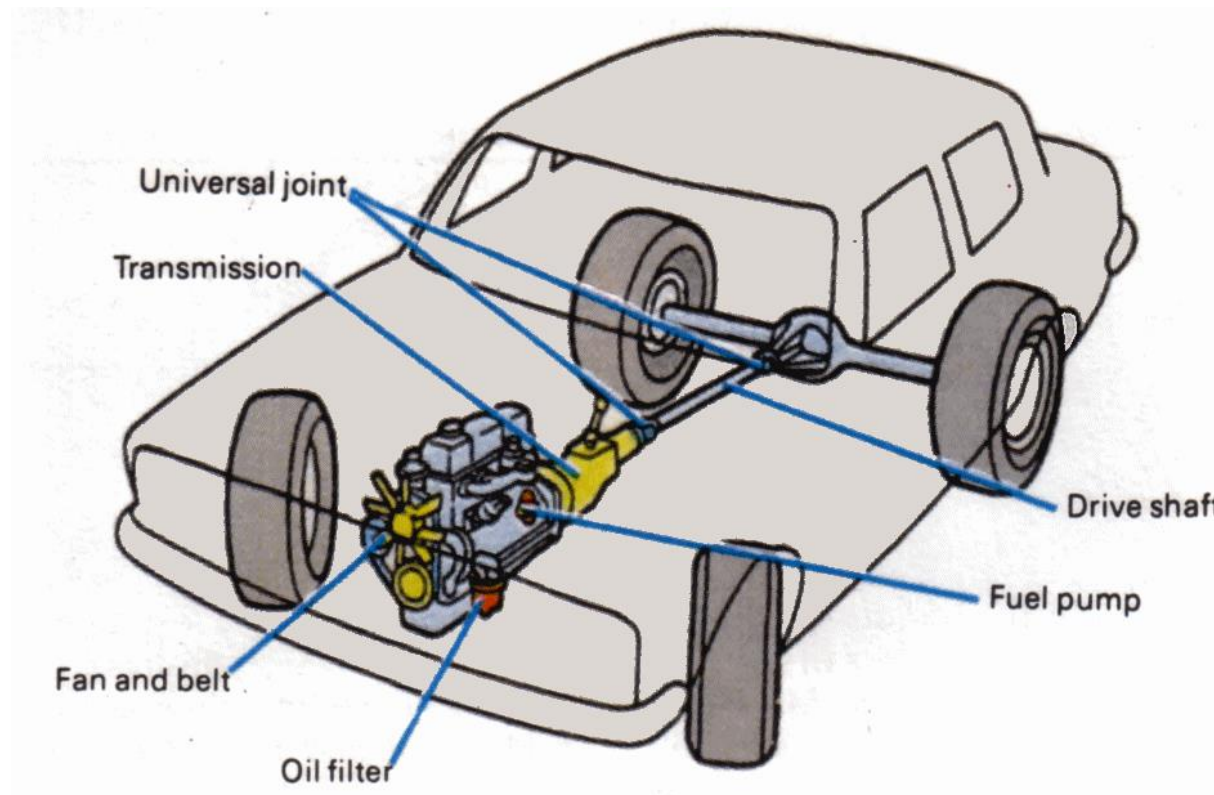


Nouns in real world translates into objects in software paradigm.

Real world Object System

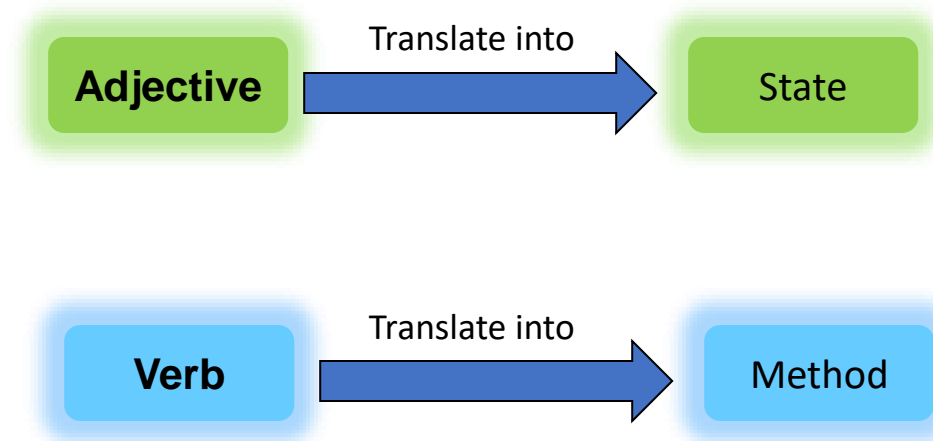
In real world all the entities are made of smaller objects, for example, a car is manufactured by assembling smaller objects like

- Engine
- Tyres
- Fan
- Steering Wheel



Characteristics of Object

Objects has two characteristics,



- **State** – State of the object.
- **Behavior** – Behavior of the object, it changes the state of the object.

Characteristics of Objects Explained with an example

Taking the car example,

- **State** – The speed of the car, fuel indicator which represents the state of the car.
- **Behavior** - Operations such as accelerate the speed, start/stop the car and apply brake are behaviors of the car.

The behavior “*accelerate the speed*” changes the car’s state “*speed*”.

Software System

Software system are collection software objects which interoperate to achieve a desired functionality.

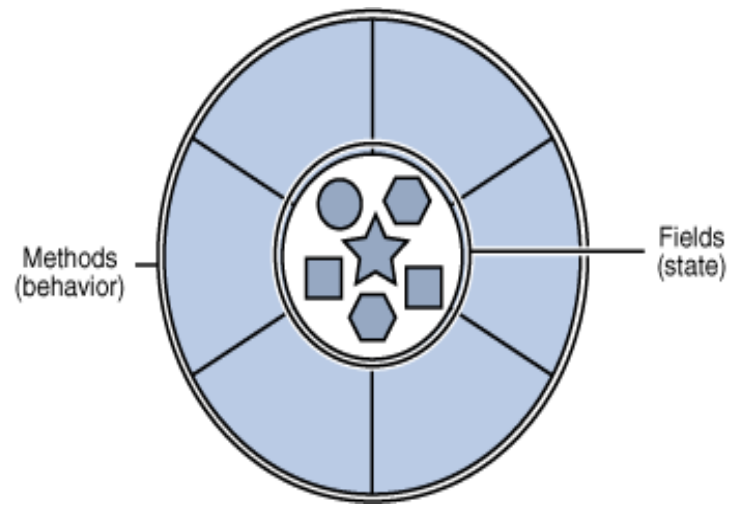
What is a Software Object?

Software objects are conceptually similar to real-world objects, they too consist of state and related behavior.

Software Object

Software objects have the same characteristics,

- **State** - Stored in fields.
- **Behavior** – Exposed through *methods*. Methods changes the state of the objects.



A software example

Lets take a net banking application where user can,

- Check account status.
- Check account balance.
- Transfer amount.

The system will be represented with a “*Account*” object.

Account Objects Characteristics:

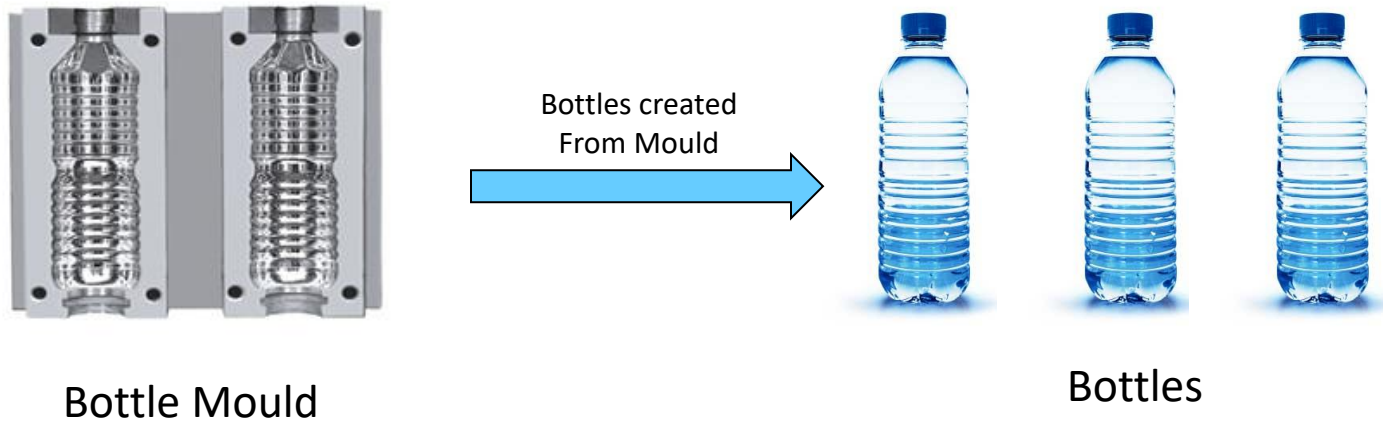
Methods (Behaviors)	Fields (States)
Check Account Status	Account Status
Transfer Amount Check Account Balance	Account Balance

What is Class?

What Is a Class?

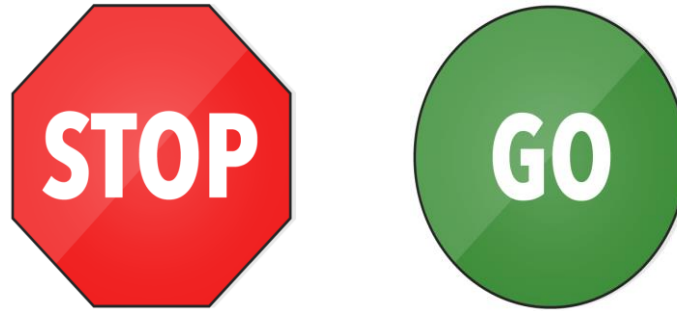
A *class* is the blueprint from which individual objects are created.

Class defines the states and behavior of an object.



The mold here refers to the “*Class*”, the bottles created out of the mold are “*Objects*”

Time To Reflect



Trainees to reflect the following topics before proceeding.

- What is OOPs?
- What are Classes?
- What are Objects?
- What are the characteristics of objects? Explain with the Car example?

OOPS Concepts

The following are the OOPS concepts,

- Encapsulation
- Inheritance
- Polymorphism

We will learn about each in the next slides.!!

Encapsulation

What is Encapsulation?

Encapsulation is a process that allows selective hiding of properties and methods in a class.

Example #1:

The speed of the car is controlled by the method “**accelerateSpeed**”

The state “**speed**” is hidden from direct access and can be only controlled using the `accelerateSpeed()`.

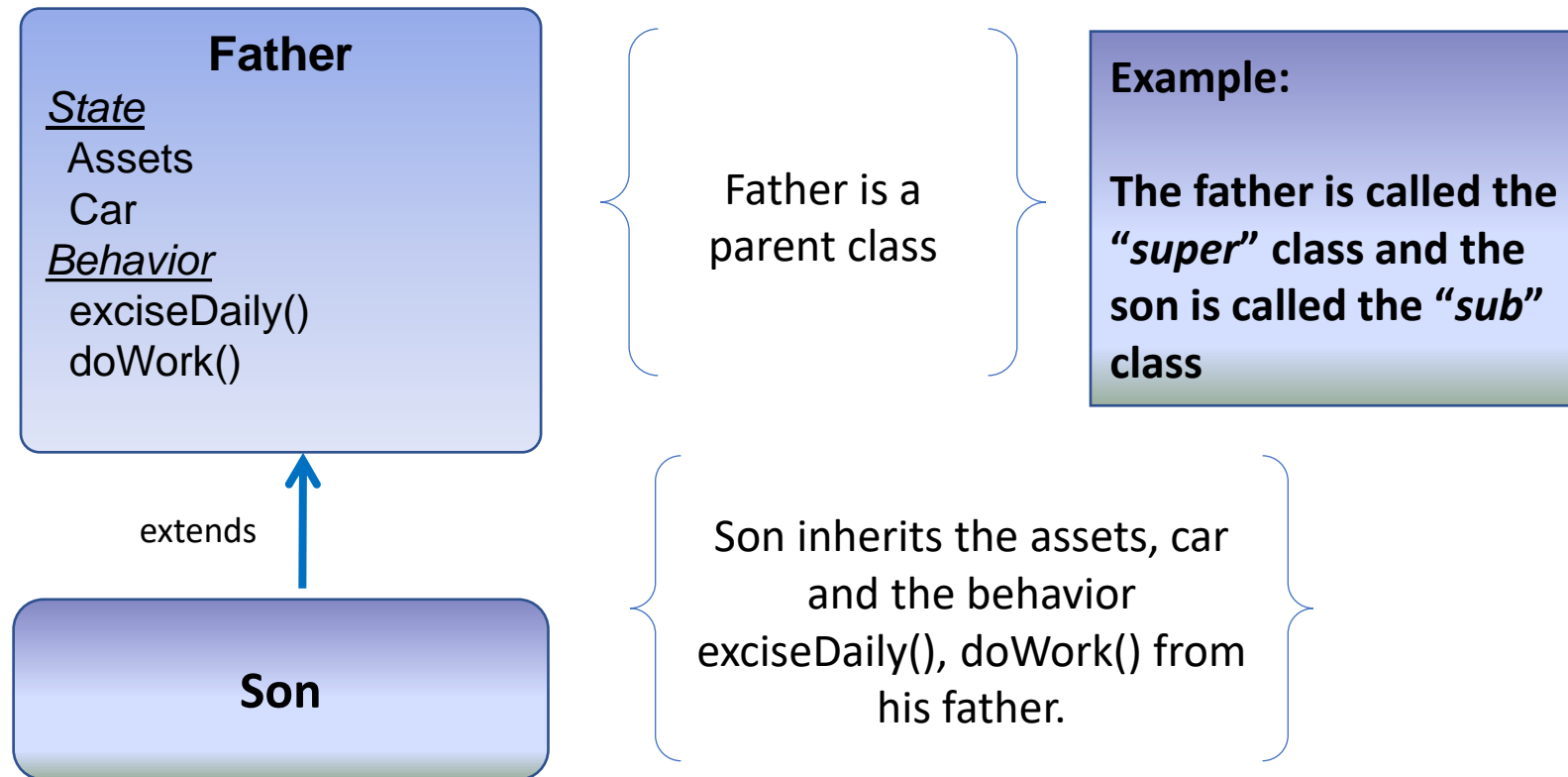
Example #2:

In the net banking application the “**accountBalance**” is hidden and can be accessed only through the “**accountTransfer**” method.

Inheritance

What is a Inheritance?

Object Oriented Programming allows classes to inherit the state and behavior from the classes.



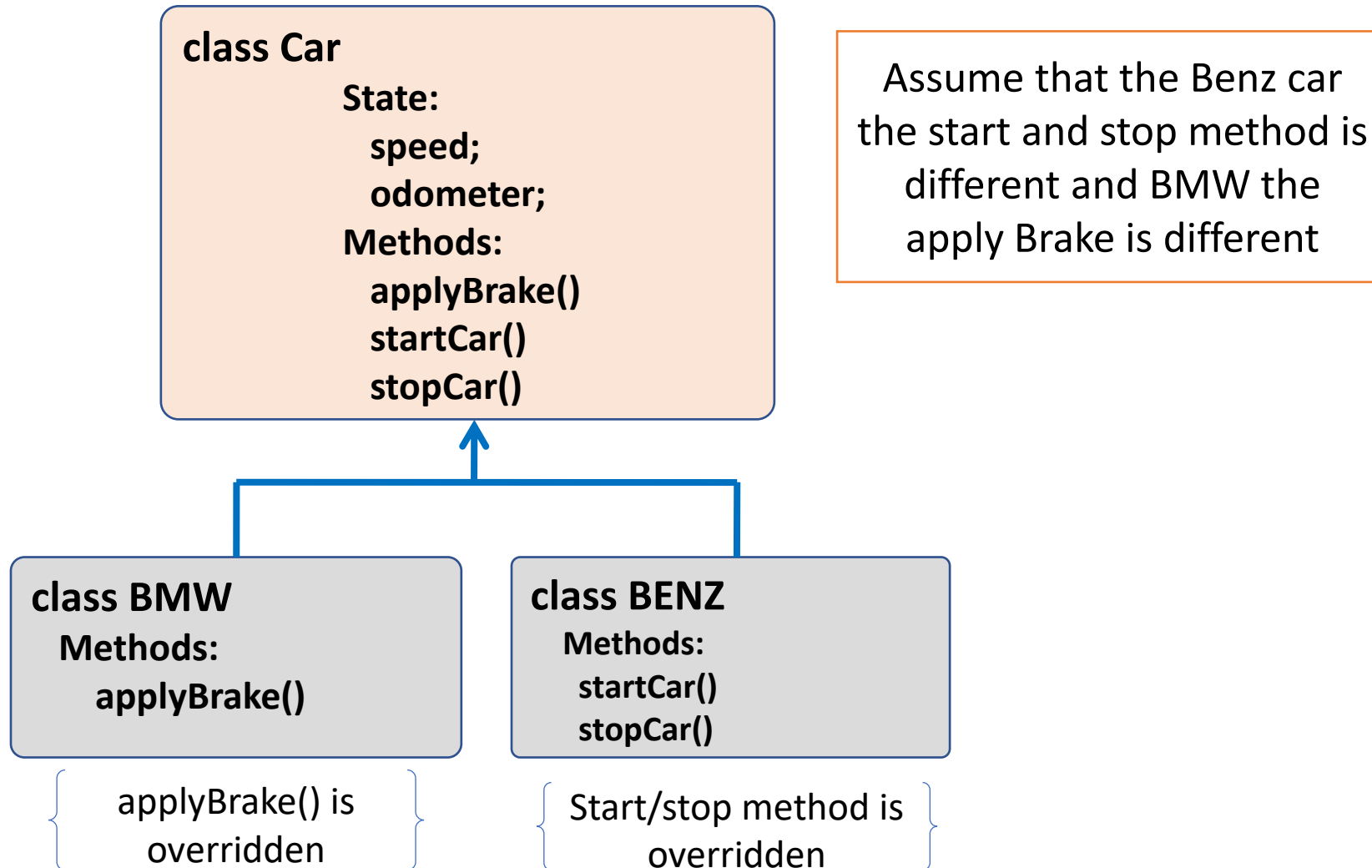
More of Inheritance

- The inheriting class contains all the attributes and behaviours of its parent class.
- Ideally the methods which needs to be reused are placed in the parent class.
- The inheriting class can override the definition of existing methods by providing its own implementation.

Overriding Example:

In the previous example if the son needs to work differently he can override the doWork() and provide a different implementation.

Overriding Example



Polymorphism

What is Polymorphism?

The dictionary definition of ***polymorphism*** refers to a principle in biology in which an organism or species can have many different forms or states.

Polymorphism is the ability to take more than one form. It is extensively used in implementing inheritance.

Polymorphism Types

Polymorphism Types:

Method Overloading - Two versions of the same method available in the same class.

Method Overriding - Methods of a subclass override the methods of a super class.

Dynamic Method Binding - At run time the interpreter will find out which objects methods needs to be executed.

Method Overloading

Method Overloading - Two versions of the same method available in the same class.

This is done by either changing the input parameter or return type.

Lets assume there is a object "*Dog*" with method "*makeSound*". Assume that the dog can make more than one sound based on whether it is normal or injured or other external factor. Now same method will have two implementations.



```
Implementation #1  
makeSound(){  
    //default parameter  
    Bark woof woof  
}
```

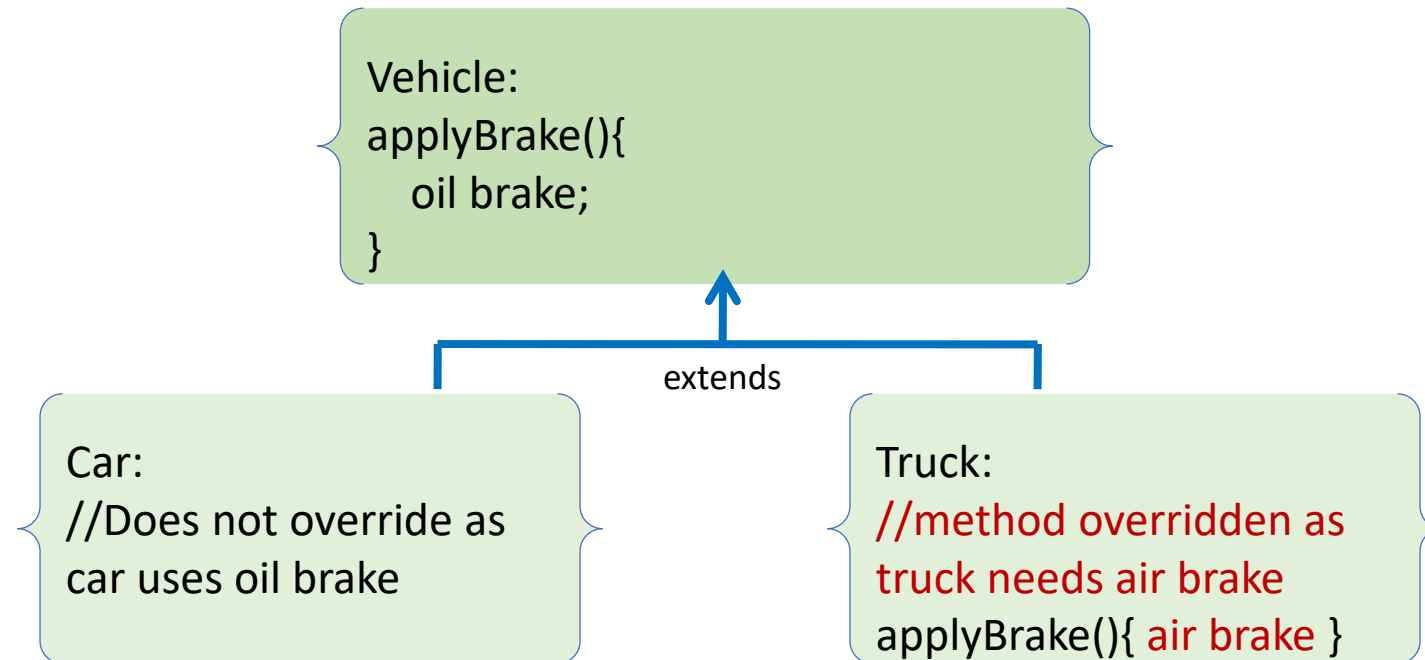
```
Implementation #2  
makeSound(injured){  
    //Added input parameter  
    Make a whining sound  
}
```

Method Overriding

Method Overriding - Methods of a subclass override the methods of a super class.

This is done by redefining the super class method in sub class.

Lets assume there are objects *"Truck"* & *"Car"* which extends *"Vehicle"*



Dynamic Method Binding

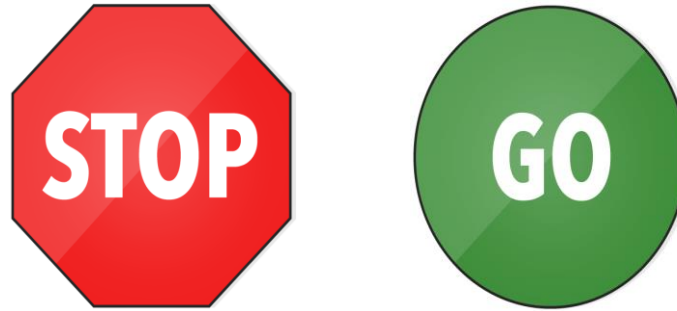
Dynamic Method or Late Binding - At run time the interpreter will find out which objects method needs to be executed.

The method being invoked is based on the type of the object and not based on the reference.

Taking the same truck example, based on what objects is created in run time either the oil/air brake is applied.

You will learn more about overriding and dynamic binding during the inheritance and polymorphism sessions.

Time To Reflect



Trainees to reflect the following topics before proceeding.

- What are the OOPs concepts?
- What is Encapsulation? Explain with a example
- What is Inheritance? Explain with a example
- What is overriding? Explain with a example
- What are the types of overriding?

Thank you

You have successfully completed
Introduction to OOPs

JAVA @11

Introduction to Java

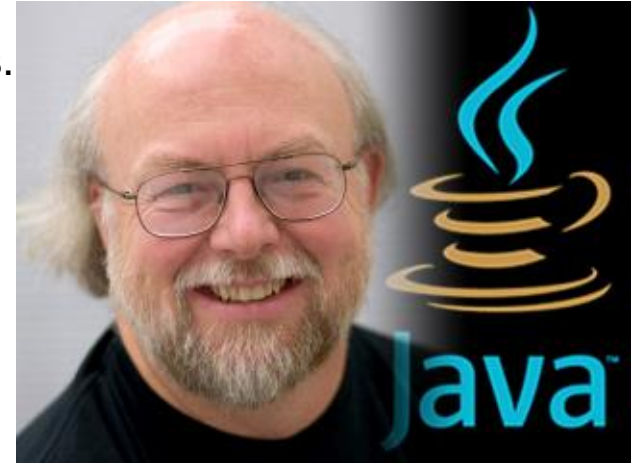
Objective

After completing this session you will be able to understand,

- Evolution of Java
- Features of Java
- How to execute a Java program.

Introduction

- Java was designed by Sun Microsystems in the **early 1990s**.
- Basic aim of java was to solve the problem of **connecting many household machines** together.
- Earlier Name of Java : **OAK**
- Creator of Java: **Mr. James Gosling** (the father of Java)
- As there was another language called Oak , they decided to rename OAK. New name was given to OAK , OAK was renamed Java in **1994**
- Java was publicly released on **May 27, 1995**
- Java was targeted at **Internet development**
- It has syntax similar to c and c++ languages, but has vastly improved features.



Goals

- Java is an object oriented.
- A single representation of a program could be executed on multiple operating systems
- It should fully support network programming
- It should execute code from remote sources securely
- It should be easy to use
- It should be “robust and secure”.
- It should be “architecture-neutral and portable”.
- It should execute with “high performance”.
- It should be “interpreted, threaded, and dynamic”.

Versions of Java

Java Version	Release Date	Year
JDK Beta		1995
JDK 1.0	January 23	1996
JDK 1.1	February 19	1997
J2SE 1.2	December 8	1998
J2SE 1.3	May 8	2000
J2SE 1.4	February 6	2002
J2SE 5.0	September 30	2004
Java SE 6	December 11	2006
Java SE 7	July 28	2011
Java SE 8 (LTS)	March 18	2014

Versions of Java

Java Version	Release Date	Year
Java SE 9	September	2017
Java SE 10	March	2018
Java SE 11 (LTS)	September	2018
Java SE 12	March	2019
Java SE 13	September	2019
Java SE 14	March	2020
Java SE 15	September	2020
Java SE 16	March	2021
Java SE 17 (LTS)	September	2021
Java SE 18	March	2022

Java Editions

The following are the Java frameworks,

- **Java SE** - Java Platform, Standard Edition or Java SE is a widely used platform for programming in the Java language. This is the Java Platform used to deploy portable applications for general use. In practical terms, Java SE consists of a virtual machine, which is used to run Java programs, together with a set of libraries.

Library Examples: [java.lang.*](#), [java.net.*](#), [jav.io.*](#), [java.util.*](#).

- **Java EE** - Java Platform, Enterprise Edition or Java EE is a widely used platform for server programming. The applications developed using JEE stack can be deployed in N tier fashion in appropriate application servers and remotely accessed.

Examples: [EJB](#), [Servlet](#), [JSP](#), [JSF](#) , [JMS](#) etc.

- **Java ME** - Java Platform, Micro Edition, or Java ME, is a Java platform designed for embedded systems such as mobile devices , PDA etc. Target devices range from industrial controls to mobile phones and set-top boxes.

Examples: Connected Limited Device Configuration ([CLDC](#)), Mobile Information Device Profile ([MIDP](#)), Information Module Profile ([IMP](#)).

Java Code

What happens after you develop a Java code?

Java Code is compiled and converted to a byte code rather than a native code.

English is an language which can understood by many people across the world. Similarly byte code is a format that can be run in many platforms Unix, Windows, Linux and also irrespective of hardware's.

Java JDK vs JRE

What is Java JDK?

JDK stands for **J**ava **D**evelopment **K**it is a package used for developing java applications and converting the java code to Byte codes. The conversion is typically done using Java compilers.

For developing and running Java applications go for the bulkier Java JDK.

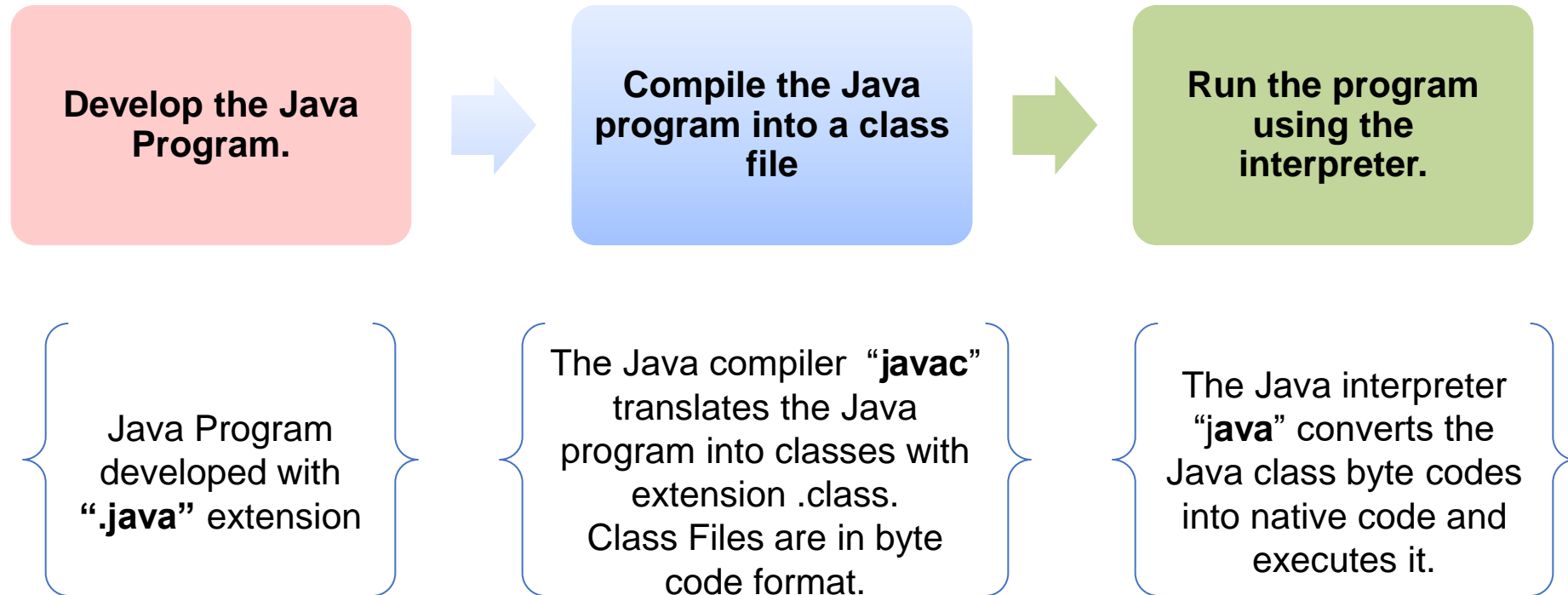
What is JRE?

JRE stands for Java runtime environment is used for executing java applications . It converts the java byte code to the necessary native code based on the underlying platform.

If you want java applications to be executed go for lighter versions JRE.

There are different JRE versions for different platforms such as Linux, Windows, Unix etc.

Basic steps to develop a Java program



Components of Java

The components which makes Java program run are,

- Java API's
- Java Class File
- **Java Virtual Machine (JVM)** – *Heart of Java technology.*

What are Java API's?

They are application interface which developer uses to develop java programs.

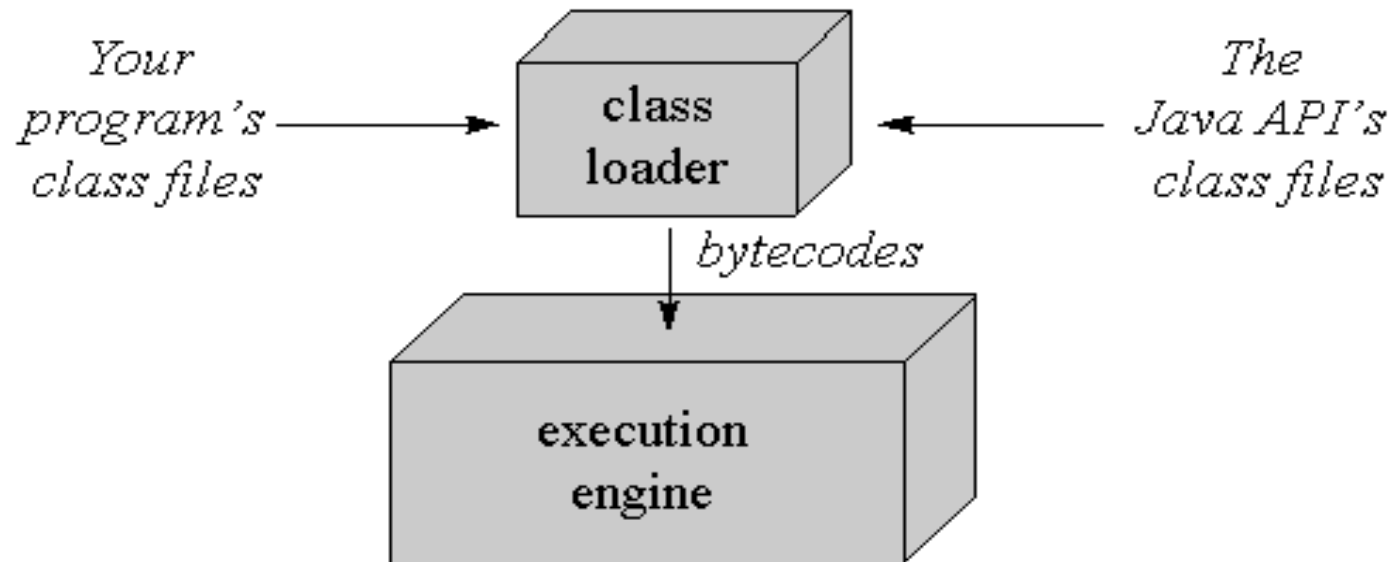
Example: `System.out.println("Hello World");`

Java developers uses `System.out.println` API to print messages on the console.

What is JVM?

Java Virtual Machines is the heart of the java platform. It is a abstract computer which,

- ✓ Loads class files using class loader
- ✓ Executes the class file using the execution engine.

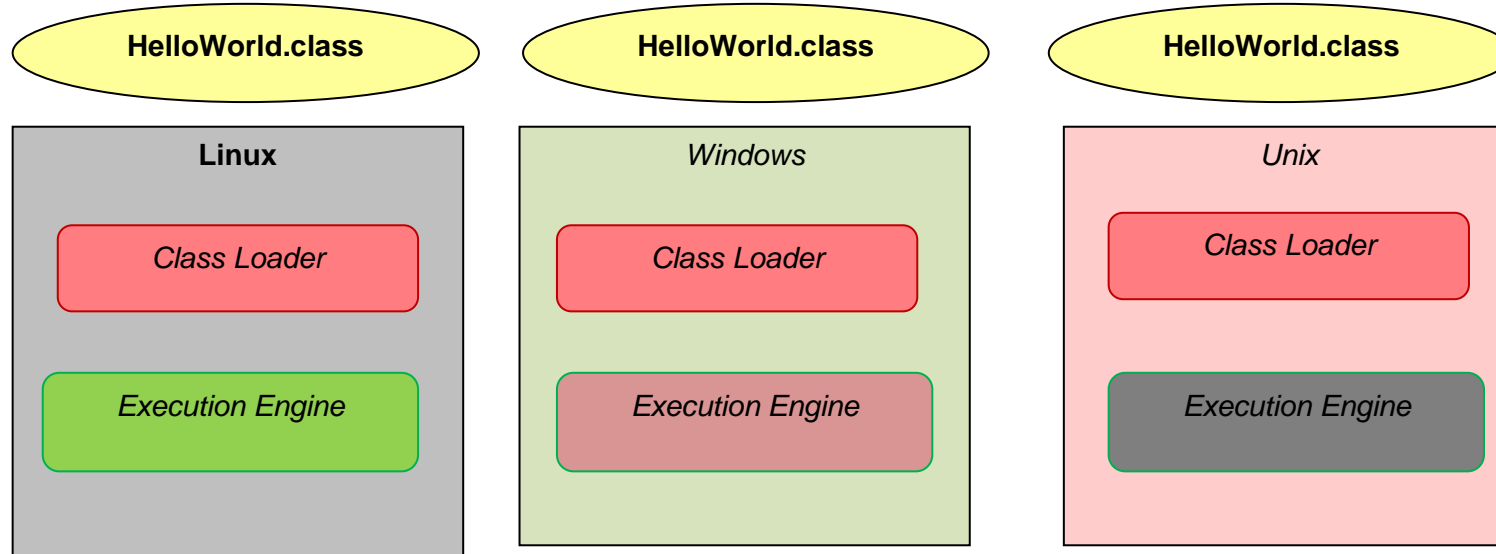


How is Java portable?

Java Platform = Java virtual Machine + Java API

Example: Linux Platform = JVM for Linux + Java API for Linux.

In the below example the same “**helloWorld**” program is developed once and run in different operating systems. The execution engine specific to each OS will translate the byte code to respective OS native code.



Example – First Java Program

1. Create a folder “workspace” in D or C drive.
2. Open notepad and type the following program in the notepad.

```
class HelloWorld{  
    public static void main(String[] args){  
        System.out.println("Hello World");  
    }  
}
```

3. Save the notepad file with the file name HelloWorld.java.
4. Open a command window, type CMD in command window.
5. Set the path and class path variable.

IMPORTANT NOTE: The file should be saved under the same name as of the class name.

Java path settings

What is Path?

Path represents the folders to be searched for running the java or javac commands. Needed for compiling Java program.

How to set path: `set path =%path%; C:\Users\hp\Links\jdk\jdk-11.0.2\bin;`

Where `%path%` - To ensure that the new path is appended with existing path variables set.

NOTE: Assuming Java home is `C:\Users\hp\Links\jdk\jdk-11.0.2`

What is class path?

Class path is the path where the class and Java API's are loaded. Needed for executing class files.

How class path is set: `set classpath =%classpath% ;`

Where `%classpath%` - To ensure that the new class path does not to override the existing class path variables set.

Example – First Java Program

6. Compile the program : In command prompt go to the folder “JavaWorks” and compile the program as follows.

```
javac HelloWorld.java
```

Where `javac` is the command line tool for compiling java programs.

7. Run the program: From the same folder run the program as follows,

```
java HelloWorld
```

Where `java` is the command line tool used for running the program.

8. Latest Java 11 version can proceed the single command for compile and execute,

```
java HelloWorld.java
```

9. In the console you can see the message printed Hello World.

10. Change the message in the program as “Hello <Your Name>” and repeat steps 6 through 8 and see the program displaying a different output.

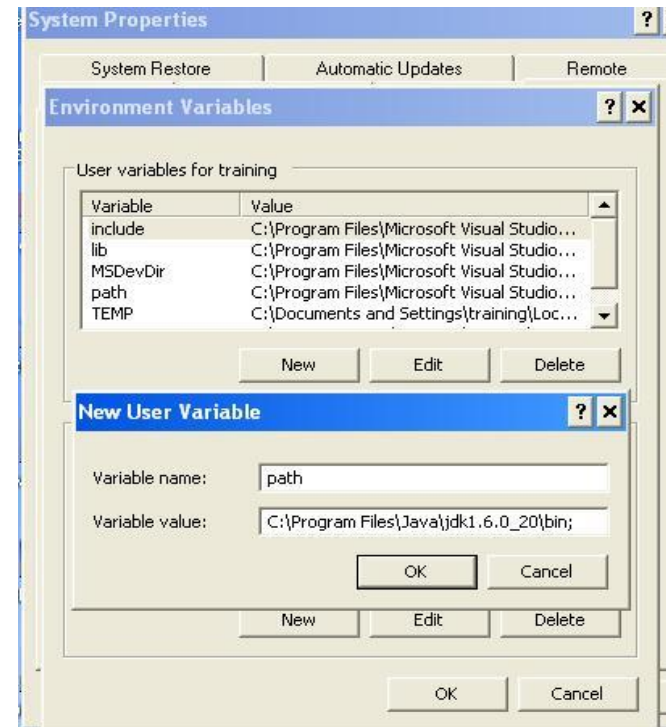
Alternate for path settings

Java path can be set in the environment variables of system properties

Right click My Computer → Click Properties → Click Advanced → Click Environment Variables → Click New(User Variables) → Add a new variable as mentioned below

- Variable Name = path
- Variable Value = %JAVA_HOME%\bin;

NOTE: Assuming Java home is C:\Users\hp\Links\jdk\jdk-11.0.2.
This variable will be reflected across all the applications in the desk top.
On setting this the variable in environment parameter it gets reflected in all the command windows.



Lets Analyze the Code

- The main method is the starting point of any java application. Any java application to be executed using “Java” command needs a class with main method.
- The application cannot run without a main method.
- Once the Helloworld.class file is executed, the interpreter searches the main method and invokes it.

Syntax:

```
public static void main(String [] args)
{
    // The program implementation
    // goes here
}
```

Lets Analyze the Code

In the program you would have noticed a statement

```
System.out.println("Hello World");
```

This is a java API used for printing messages on the console. This prints messages with a line break.

The other variant of this method is

```
System.out.print("Hello World"); //This prints messages without a line break.
```

Example: `System.out.print("A");`

```
System.out.print("B");  
System.out.print("C");
```

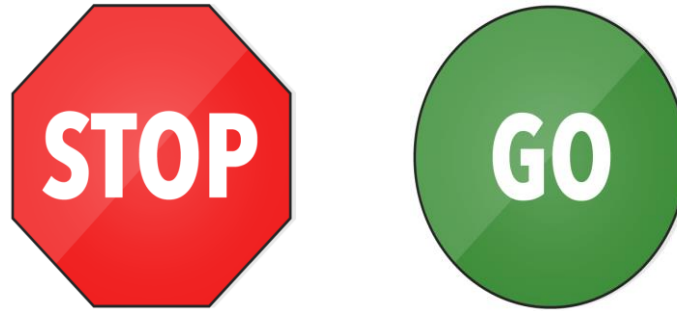
This displays the message
"ABC" in the console.

```
System.out.println("F");  
System.out.println("G");  
System.out.println("H");
```

This displays the message as

F
G
H

Time To Reflect



Trainees to reflect the following topics before proceeding.

- What is the different between Java JDK and JRE?
- What is Java ME & Java EE?
- What are the types of Application Developed using Java?
- Which is responsible for the memory management in Java?

What are Java classes?

Classes are the fundamental building blocks of a Java program. Java application are built using one or more java classes, a class contains data and application business logics.

- Data are represented as variables in classes.
- Application business logic are implemented as methods in classes.
- A class is a blue print for making objects.

Classes Example: Employee, Department

NOTE:

- A single physical .java file can have more than one classes.
- The java file should be named after the class which is declared public.
- There cannot be two classes defined as public in the same java file.

Structure of a Class

The Structure of the class is as follows,

```
class <ClassName>
```

```
{
```

← { This is the class declaration. }

```
// The class implementation goes here
```

```
}
```

{ This is the class body implemented within the curly braces }

The body of the class contains,

- **Variables** – This is a container for storing class data.
- **Methods** – Application behavior implemented and this changes the data (variables values).

A Sample Java Class

class Employee

```
{
    long empSalary=0;
    int empId=1025;

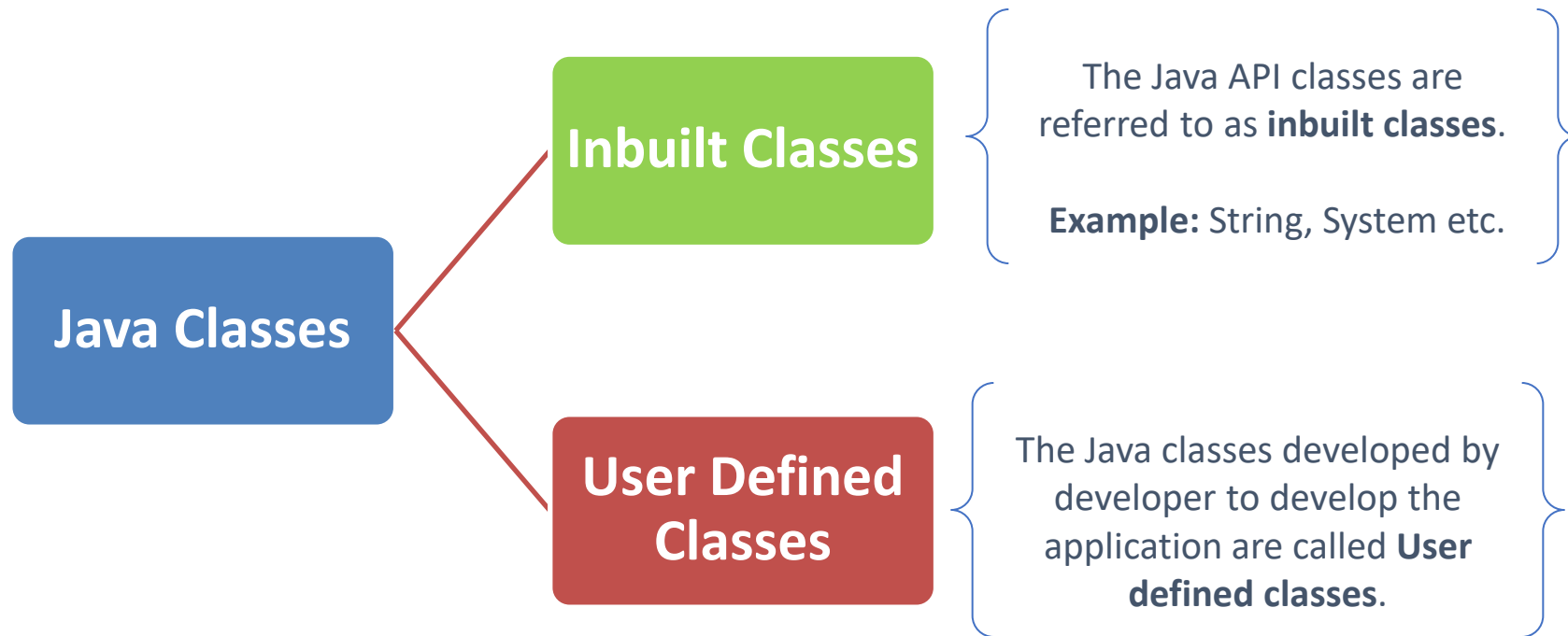
    void calculateSalary()
    {
        // logics of salary
        //calculation goes in here
        empSalary= 45000;
        system.out.println("Salary =" + empSalary);
    }
}
```

These are variables declared to store the employee id and the employee salary. The variable has a declaration and data type like integer (or) String.

This is the method definition. The salary is stored as 10000 in the variable empSalary and printed in the console.

NOTE: Java is case sensitive. For example, “Employee” and “employee” are two different classes.

Types of Classes

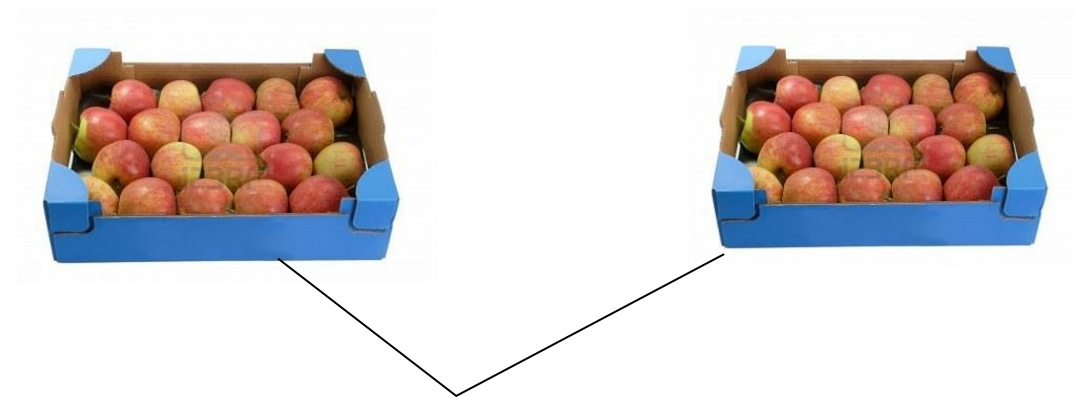


Packages

Packages are used for logically grouping the classes together into a single unit.

In the Java API, classes are grouped into packages.

Example:



Both the apple looks similar, If you mix the apples you will end up with a confusion to differentiate the varieties. The box where they are placed helps us to differentiate them. Similarly though the java classes have the same name using the packages (box) where they reside the classes can be differentiated and used.

Benefits of using packages

- Packages helps to organize classes into a folder structure which will be easy to maintain.
- Two different packages can have classes with the same name. If there is a naming clash, then classes can be accessed with their fully qualified name.
- Packages provide a level of security, because you can restrict the class usage, which you develop in such a way that only the classes in the same package can access it.





Creating an Object

Object creation is a three step process,

- **Declaration** - Giving a name to the object to be created.
- **Instantiation** - The new keyword and constructor creates a instance of the object.
- **<Initialization>** - The values will be initialized using the constructor.

You will learn more about the
constructors and objects in the next sessions.

Main class for Employee

```
class EmployeeProgram {  
    public static void main (String[] args) {  
        Employee empObject =null;  { Declares a employee object. }  
        empObject = new Employee();  { Instantiates and initialize the  
        empObject.empSalary = 20000;  { Sets the employee salary as 20000. }  
        empObject.calculateSalary();  { Triggers the calculate salary method  
    } in the employee object. }  
}  
}
```

This program creates a employee object and invokes the calculateSalary method on the object

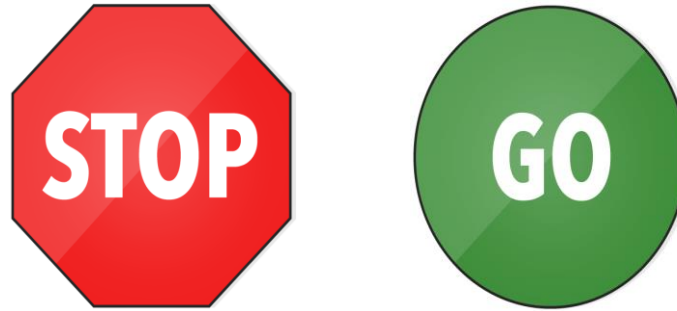
Example – Creating an Object

1. Create a java class “Student” add a integer variable “regId”.
2. Create a method “displayRegId” which will print the “regId” of the student in the format
 - “The student registration id is <RegId>”
3. Create a java class “StudentMain” add a main method which will
 - Create a object instance of the Student class
 - Set the registration id to value “1290”
 - Invoke the method “displayRegId”.
4. The message needs to be displayed in the console.
 - Expected Output: “The student registration id is 1290”

Example – Creating an Object

1. Create a Class “Calculator” with two integer variables “op1” & “op2”.
2. Create a method “displayOperand” and add the logic of displaying the variable values in the below format,
 - “The value of operand 1 is <op1 Value>”
 - “The value of operand 2 is <op2 Value>”
3. Create a main class “CalculatorMain” implement the following logic
 - Set the value of “op1” as 98 and “op2” as 43.
 - Invoke the method displayOperand.
4. The message should be displayed as mentioned in point # 2.

Time To Reflect



Trainees to reflect the following topics before proceeding.

- How can java class be logically grouped?
- What are the benefits of java packages.
- What is the keyword used for creating objects?.

Thank you

You have successfully completed
**Introduction to
Java**

JAVA @11

Language Fundamentals and Operators

Objective

After completing this session you will be able to understand,

- Java Keywords
- Primitive data types
- Variables & Literals
- Casting primitives
- Operators

Keywords

What are Java Keywords?

Keywords are reserved identifiers that are predefined in java language.

Keywords cannot be used as names for variables, methods and classes.

Few Points on Java Keywords

- Keywords use letters only, they do not use special characters (such as \$, _, etc.) or digits.
- All keywords are in lowercase letters and incorrect usage results in compilation errors.

Table of Keywords

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

Primitive Data Types

Java, like other programming languages such as C and C++, supports basic built-in data types, which are also called primitive data types.

There are eight primitive data types in Java they are,

Data Type	Size In Bits	Range	Default Value
boolean	1	true or false	false
byte	8	-2^7 to $2^7 - 1$	0
short	16	-2^{15} to $2^{15} - 1$	0
int	32	-2^{31} to $2^{31} - 1$	0
long	64	-2^{63} to $2^{63} - 1$	0L
float	32	-2^{31} to $2^{31} - 1$	0.0f
double	64	-2^{63} to $2^{63} - 1$	0.0d
char	16	0 to $2^{16} - 1$	'\u0000'

Variables in Java

Dictionary Definition of Variables:

- **A quantity that can assume any of a set of values.**
- **Something that is likely to vary; something that is subject to variation.**

Variable analogy:

The box is used as a container to hold white papers.



What is a Java variable?

Similar to the box, Java variables are the basic unit of storage in a Java program. It is used to store the state of objects.

How are variables declared?

Variables declaration:

```
<data type> <name> [=initial value];
```

Where,

Data Type – Indicates the type of value that variable can hold.

Name – Name by which the variable is identified.

Values enclosed in <> are mandatory attributes

Values enclosed in [] are optional attributes

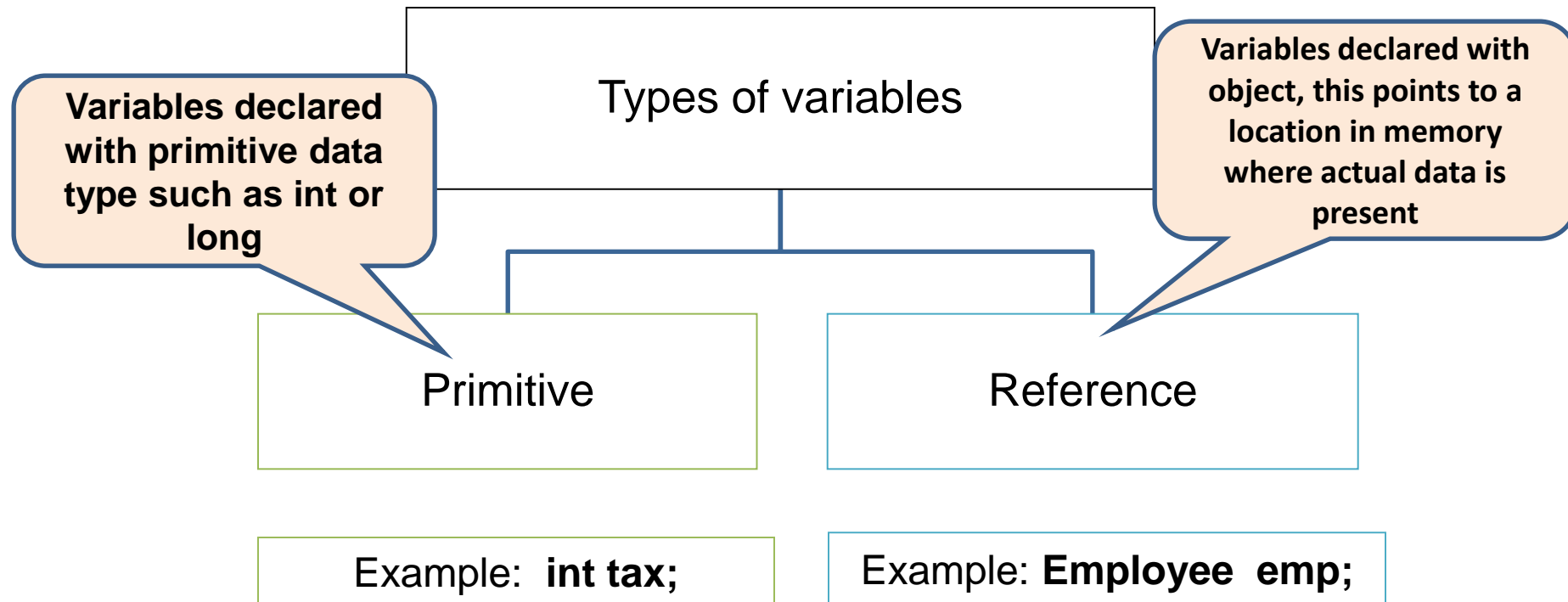
Example:

```
int salary; // Declared a variable salary with data type int.
```

(or)

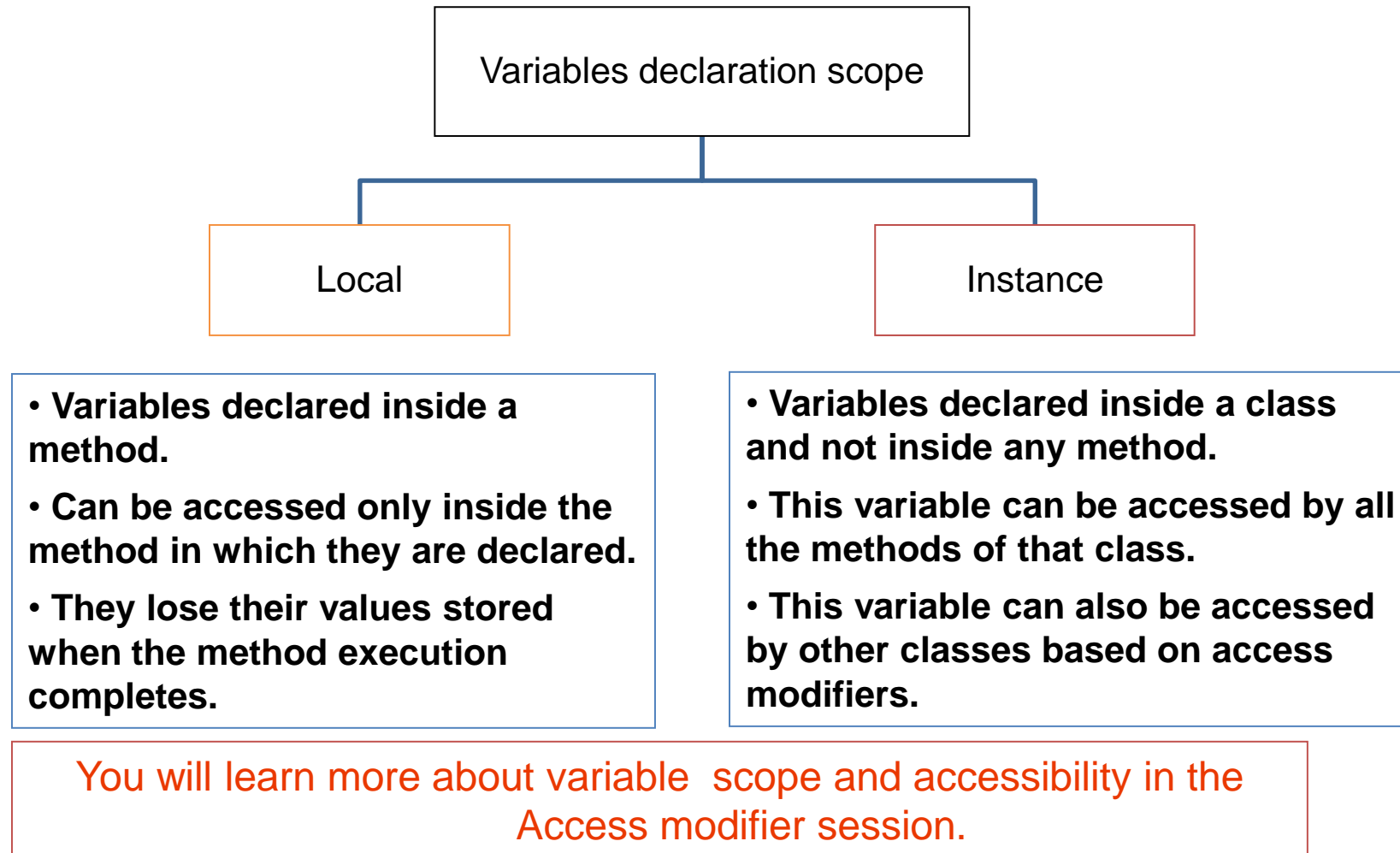
```
int salary =18000; // Salary declared as int with initial value 18000
```

Types of Variables in Java



NOTE: A primitive variable holds the value of the data item, while a reference variable holds the memory address where the object values are stored.

Scope of the Variables



Variable Declaration - Example

Example:

```
public class Student {  
    int regId=1201;  
    void makeMarkList() {  
        int deptId=2011;  
        //code block  
    }  
}
```

}

Instance Variable – Declared
inside the body of the class.

}

Local Variable – Declared
inside the body of the method.

Naming Rules

- Variable names are **case-sensitive**.
- A variable's name **can be any legal identifier**.
- Length of Variable name can be any number.
- Its necessary to use Alphabet at the start (however we can use underscore , but do not use it)
- Some auto generated variables may contain '\$' sign. But try to avoid using Dollar Sign.
- **White space** is not permitted.
- **Special Characters** are not allowed.
- **Digit at start** is not allowed.
- Variable name must not be a **keyword or reserved word**.

Naming Rules

If the variable name has only one word, then spell that word in all lowercase letters. If it consists of more than one word, then capitalize the first letter of each subsequent word.

Example: `employeeName`, `employeeAccountNo`

- Avoid abbreviations, the variable name should be meaningful.

Don't: `empSal`, `empAdd`.

Do: `empSalary`, `empAddress`.

Java Literals

What is a literal?

- A literal is a value assigned to a variable in a class.

A literal is a value and not a variable

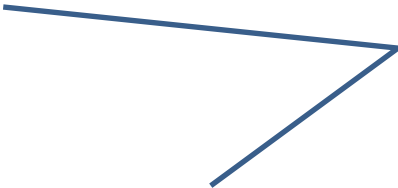
- Literals appears on the right hand side of the variable declaration.
- Literals value can be assigned for any of the primitive data type.

Example:

`int tax = 20;`

(or)

`float salary = 45000.0;`



**Where “20” and “45000.0”
are int and float literals
respectively.**

Java Literals Types

Literal Type	Description	Formats and Example
Integer Literals	Integral literals may be represented by decimal, octal, or hexadecimal numbers. The octal number is prefixed with 0, and hexadecimal with 0x or 0X.	Decimal :12 Hexadecimal:0xC Octal:014
Floating Literals	A floating-point literal is represented by a floating-point number, and is used for both float and double data types.	Float:1234.5f Double:12.65, 3.56E2, 23d
Boolean Literals	A variable of boolean type can have only one of two possible values, true or false. Therefore, these two values are the only available boolean literals	true, false

Java Literals Types

Literal Type	Description	Formats and Example
Char Literals	A char literal may be represented by a single character enclosed in single quotes.	'a', '\u4567'
String Literals	String literals represent multiple characters and are enclosed by double quotes.	"Hello World"
Escape Sequences	Java supports the escape sequences for denoting special characters.	\n :a new line \r :a return \t :a tab \b :a backspace \f :a form feed ' :a single quote " :a double quote \\ :a backslash

Example – Variables Declaration

1. Create a java class “**Employee**” with the following variables created,

Variable Name	Data Type	Default value
empId	long	1025
empSalary	double	45000
empTax	float	14.5
empDaysOfWork	int	24

2. Create a method named **calculatePF()** which will have a local float variable named “**pfRate**” with value as 10.5 and the following message should be printed.

“The PF Rate Of The Employee is <pfRate>”

3. Create a java class “**VariableDemo**” add a main method which will Create a object instance of the Employee. Access the instance variables and display the default values of variable as follows.

“The Id of the Employee is <empId> ”

“The Salary of The Employee is <empSal> ”

“The Percentage of Tax The Employee needs to Pay is <empTax>”

“The Total Number of Working Days is <daysOfWork>”

4. Invoke the calculatePF() method .

Variables Declaration - Solution

```
class Employee{
    long empId = 1025L;
    double empSalary = 45000d;
    float empTax = 14.5f;
    int empDaysOfWork = 24;
    void calculatePF() {
        float pfRate = 10.5f;
        System.out.println("The PF rate of the Employee is "+pfRate);
    }
}

public class VariableDemo {
    public static void main(String[] args) {
        Employee e = new Employee();
        System.out.println("Id of the Employee is "+e.empId);
        System.out.println("The Salary of the Employee is "+e.empSalary);
        System.out.println("The Percentage of Tax the Employee needs to pay is "+e.empTax);
        System.out.println("The Total number of Working Days is "+e.empDaysOfWork);
        e.calculatePF();
    }
}
```

Casting Primitives

What is casting?



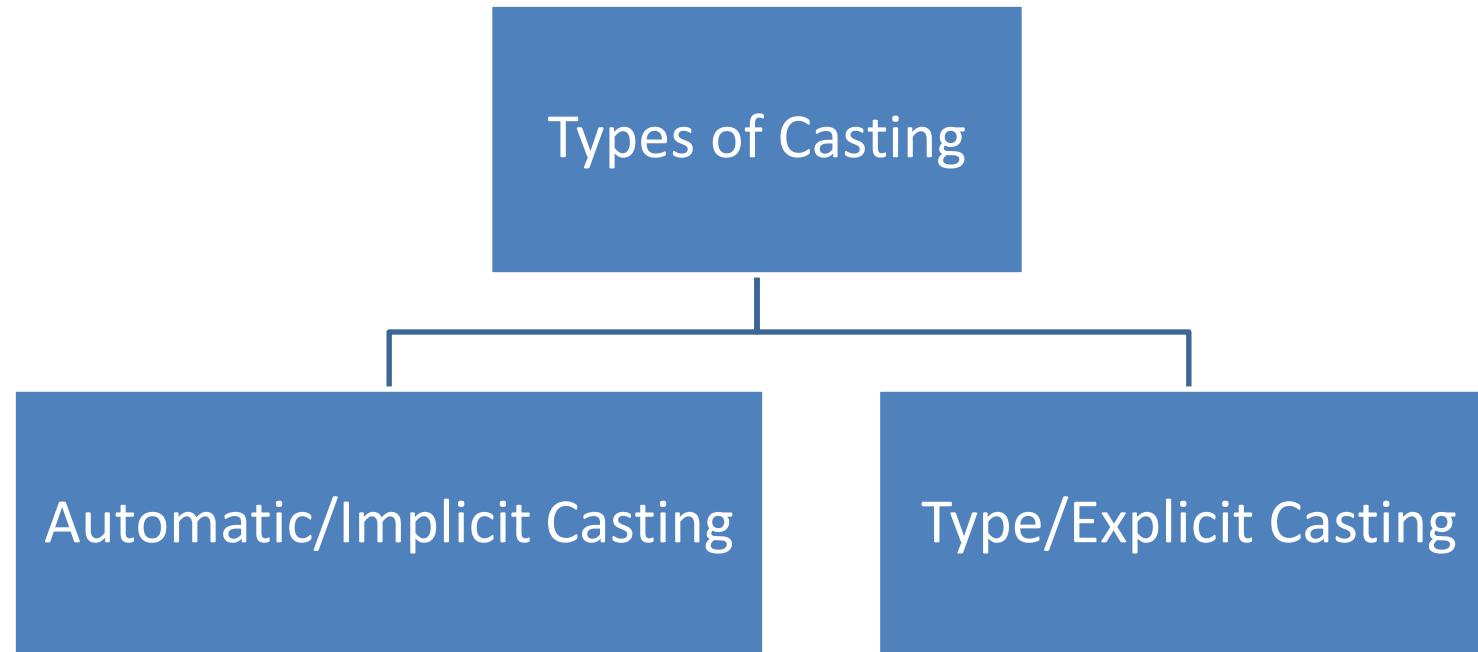
Casting is the process by which a liquid is poured into a mold and is allowed to solidify. In this process, it attains the shape of the mold

Similarly in Java, a value of one primitive type is assigned to a variable of another primitive type. This is called primitive type casting.

Example: A variable int is casted into float. Here float is the mold and int variable is casted as a float.

- **Casting is done with any of the primitive data types**
- **Boolean type variable cannot be casted**

Types of Primitive Casting



Implicit Casting

Java will performs automatic conversion when:

1. The two data types are compatible.
2. The destination data type is larger than the source data type.

Example: An int value (source) to a long variable (destination).

```
int mySal=500;  
long sal=mySal;
```

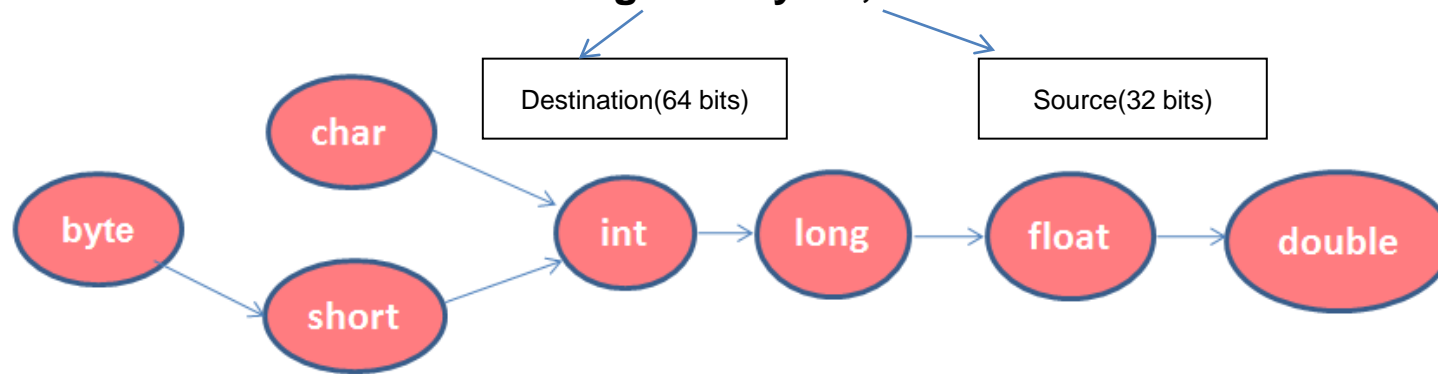


Figure depicts the allowed conversions for primitive data types (arrows indicate paths of allowed conversion).
Example: int can be type casted to long. Long can be type casted to float. etc.

Explicit Casting

Explicit Casting will be done by the programmers when the following condition is met.

The destination type is smaller than the source type.

Example:

```
double pf=12.67;  
float myPf=pf;           // compile-time error.  
float myPf=(float) pf;   // Explicit Casting.
```

A different type of conversion named “*truncation*” will occur when a floating-point value is assigned to an integer type.

Example: if the value 1.43 is assigned to an integer, the resulting value will be 1. The 0.43 will have been truncated.

Operators

What is an operator?

Operators are special symbols that perform specific operation on one, two or more operands and return a result.

Example:

$$a + b = c$$

Here,

- a and b are variables
- the data in a and b are operands
- '+' is the operator
- c contains the result

Navigate to the next slide for more explanation

Operators in Java

- Operators operate on the data represented by a *variable*.
- The data that is being operated on is called an operand.
- Operators operate on their operands in a variety of ways
- Operators change the values of their operands.
- Operators can produce a new value without changing the values of the operands.
- Operators can compare the values of two operands.

Types of Operators in Java

Types of data operators based on the number of operands,

- Unary operators: Require only *one* operand.

Example:

++ increments the value of its operand by 1.

-- decrement the value of its operand by 1.

- Binary operators: Require *two* operands.

Example:

+ adds the values of its two operands.

- Ternary operators: Operate on three operands(?:).

Unary Operators

Unary Arithmetic Operator:

The unary operators require only one operand. They perform various operations such as incrementing/decrementing a value by one, negating an expression, or inverting the value of a boolean.

Operator	Use	Description
-	int a =-b;	Negates the operand b and stores the value in a
++	int b=a ++	Increments the value of a by 1
--	Int b=a --	Decrements the value of a by 1

Unary Operators

Example of Increment & Decrement Operators:

Initial Value of x	Code Statement	Final Value of y	Final value of x
7	y=++x;	8	8
7	y=x++;	7	8
7	y=--x;	6	6
7	y=x--;	7	6

Ternary Operator

The ? operator consists of three operands and is used to evaluate boolean expressions.

The goal of the operator is to decide which value should be assigned to the variable.

Syntax:

variable x = (expression) ? value if true : value if false

Example:

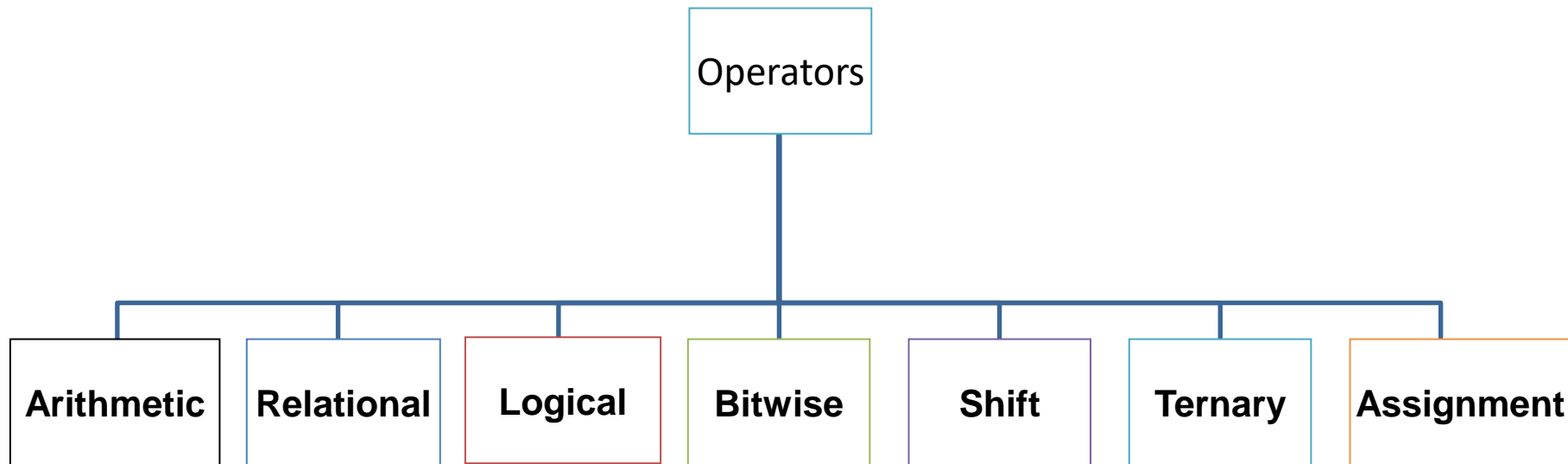
```
public class TernaryDemo {  
    public static void main(String[] args) {  
        int num1 = 35;  
        int num2 = (num1>15)?75:90;  
        System.out.println("The Value of num2 is "+num2);  
    }  
}
```

In the above example since the num1 is greater than 15 the value 75 will be assigned to num2.

If num1 is < 15 the value 90 will be assigned to num2.

Types of Operators

Types of Operators – Based on the Operation



You will learn more about operators in the next coming slides.

Arithmetic Operators

Arithmetic Operator:

Arithmetic operators are used for mathematical calculations. The following table lists the arithmetic operators

Operator	Value of a	Value of b	Usage	Value of c
+	6	5	$c=a+b$	11
-	6	5	$c=a-b$	1
*	6	5	$c=a * b$	30
/	6	5	$c=a/b$	1
%	6	5	$c=a \% b$	1

Example - Operators

1. Create a java class "**Arithmetic**" with three instance variable int num1, num2, result.
2. Create three methods "**addition**", "**subtraction**" and "**printSmaller**" which will add ,subtract and print the smallest of two numbers respectively. The methods should display the results as follows,

Addition: "The Sum of the Two numbers <num1> and <num2> is <result>"

Subtraction: "The Subtracted Result of the two numbers <num1> and <num2> is <result>"

Print Smaller: "The Smallest of the Two numbers <num1> and <num2> is <result>"

3. Create a java class "**OperatorDemo**" add a main method which will
 1. Create a object instance of the Arithmetic .
 2. Set the value of num1 and num2 as 24 and 16.
 3. Invoke the methods "**addition**", "**subtraction**" and "**printSmaller**".
4. The message needs to be displayed in the console.

Operators - Solution

```
class Arithmetic{
    int num1;
    int num2;
    int result;
    public void addition() {
        result = num1+num2;
        System.out.println("The Sum of Two numbers "+num1+" and "+num2+" is "+result);
    }
    public void subtraction() {
        result = num1-num2;
        System.out.println("The Subtracted result of two numbers "+num1+" and "+num2+" is "+result);
    }
    public void printSmaller() {
        result = (num1<num2)?num1:num2;
        System.out.println("The Smallest of the Two numbers "+num1+" and "+num2+" is "+result);
    }
}
public class OperatorDemo {
    public static void main(String[] args) {
        Arithmetic arith = new Arithmetic();
        arith.num1 = 24;
        arith.num2 = 16;
        arith.addition();
        arith.subtraction();
        arith.printSmaller();
    }
}
```

Relational Operators

Relational Operator:

A Relational operator, also called a comparison operator, compares the values of two operands and returns a boolean value, true or false.

Operator	Use	Description
>	op1>op2	True if op1 is greater than op2, otherwise false
>=	op1>=op2	True if op1 is greater than or equal to op2, else false
<	op1 < op2	True if op1 is less than op2, otherwise false
<=	op1 <=op2	True if op1 is less than or equal to op2, otherwise false
==	op1 == op2	True if op1 and op2 are equal, otherwise false
!=	op != op2	True if op1 and op2 are not equal, otherwise false

Logical Operators

Logical Operator:

Logical operators are used to evaluate one or more conditions and return a true or false value based on conditions result.

Operator	Operator Type	Use	Description
&&	Binary	op1 &&op2	True if both the operands are true, otherwise false.
	Binary	op1 op2	True if any one of the operand is true , otherwise false.
!	Unary	! op1	True if operand is false and vice versa

Bitwise Operators

Bitwise Operator:

Bitwise operators that are used to manipulate bits of an integer (byte, short, char, int, long) value.

Operator	Type	Use	Description
&	Binary	op1 & op2	The AND operator & returns a value 1 bit if both operands are 1, a zero is produced in all other cases. It is equivalent to multiplying both the bits.
	Binary	op1 op2	The OR operator , returns a value 1 if one of the bits in the operands is a 1. Returns zero if both the operands has a value zero. It is equivalent to adding both the bits.
~	Unary	~ op1	The unary NOT operator, ~ , inverts all of the bits of its operand., converts the 1 to zero and vice versa.
^	Binary	op1 ^ op2	The XOR operator, ^ , It represents the inequality function, i.e., the output is HIGH (1) if the inputs are not alike otherwise the output is LOW (0).

Bitwise Operators

Example of Bitwise Operators:

Initial Value of x	Initial Value of y	Code Statement	Final Value of z
15(0000 1111)	5(0000 0101)	Z=x & y;	5 (0000 0101)
15(0000 1111)	5(0000 0101)	y=x y;	15 (0000 1111)
7(0000 1111)	-	z=~x;	-8 (~1111 0000)
15(0000 1111)	5(0000 0101)	z=x ^ y;	10 (0000 1010)

Example - Operators

1. Create a java class "**Bitwise**" add three integer instance variables num1, num2 and result.
2. Create four methods which will perform bitwise **AND**, **OR**, **XOR** and **unaryNOT** for the two numbers and print in following the format

"The Bitwise AND of the Two numbers <num1> and <num2> is <result>"

"The Bitwise OR Result of the two numbers <num1> and <num2> is <result>"

"The Bitwise XOR of the Two numbers <num1> and <num2> is <result>"

"The Unary NOT Result of the number <num1> is <result>"
3. Create a java class "**BitwiseOpDemo**" add a main method which will
 - Create a object instance of the **Bitwise**.
 - 1. Set the value of num1 and num2 as 15 and 5.
 - 2. Invoke the four methods.
4. The message needs to be displayed in the console.

```

class Bitwise{
    int num1, num2, result;
    public void bitwiseAND() {
        result = num1&num2;
        System.out.println("The Bitwise AND of the Two numbers "+num1+" and "+num2+" is "+result);
    }
    public void bitwiseOR() {
        result = num1|num2;
        System.out.println("The Bitwise OR result of the two numbers "+num1+" and "+num2+" is "+result);
    }
    public void bitwiseXOR() {
        result = num1^num2;
        System.out.println("The Bitwise XOR of the Two numbers "+num1+" and "+num2+" is "+result);
    }
    public void unaryNOT() {
        result = ~num1;
        System.out.println("The Unary NOT result of the numbers "+num1+" is "+result);
    }
}

public class BitwiseOpDemo {
    public static void main(String[] args) {
        Bitwise obj = new Bitwise();
        obj.num1 = 15;
        obj.num2 = 5;
        obj.bitwiseAND();
        obj.bitwiseOR();
        obj.bitwiseXOR();
        obj.unaryNOT();
    }
}

```

Shift Operators

Shift Operator:

Shift operators operate on one or more bit patterns or binary numerals at the level of their individual bits.

Operator	Use	Description
<<	op1 <<op2	The left shift operator, <<, shifts all of the bits in a value to the left a specified number of times.
>>	op1>>op2	The right shift operator, >>, shifts all of the bits in a value to the right a specified number of times.
>>>	op1 >>> op2	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.

Assignment Operators

Assignment Operators:

An assignment operator is used to set (or reset) the value of a variable

Operator	Use	Description
<code>+=</code>	<code>op1 +=op2</code>	Equivalent to <code>op1=op1+op2</code>
<code>-=</code>	<code>op1-=op2</code>	Equivalent to <code>op1=op1-op2</code>
<code>*=</code>	<code>op1*=op2</code>	Equivalent to <code>op1=op1 * op2</code>
<code>/=</code>	<code>op1 /=op2</code>	Equivalent to <code>op1=op1/op2</code>
<code>%=</code>	<code>op1%=op2</code>	Equivalent to <code>op1=op1%op2</code>

Assignment Operators

Operator	Use	Description
&=	op1 &=op2	Equivalent to op1=op1& op2
=	Op1 =op2	Equivalent to op1=op1 op2
^=	op1 ^=op2	Equivalent to op1=op1 ^ op2
<<=	op1 <<=op2	Equivalent to op1=op1<< op2
>>=	op1>>=op2	Equivalent to op1=op1>> op2
>>>=	op1 >>>=op2	Equivalent to op1=op1 >>> op2

Operator Precedence

If in a expression there are more than one operators the order in which they are evaluated would be based on the operator precedence.

Precedence	Operator
1	(),[]
2	++,--
3	*,/,%,+,-
4	<.<=,>,>=
5	==,!=
6	&&,

Operator Precedence Example

Let us analyze some examples to understand the precedence of operators.

Example 1:

```
result = (num1*num2) + num3 / num4;
```

Here is the order on how the above expression is processed

1. num1 * num2
2. num3 / num4
3. Final Result = result of step 1 + result of Step 2

Example 2:

```
result = num1>num2 && num3<=num4 || num5!=num6
```

Here is the order on how the above expression is processed

1. num3<=num4
2. num1>num2
3. num5!=num6
4. Result of Step1 && Result of Step2
5. Result of Step4 || Result of Step3

Thank you

You have successfully completed
**Language Fundamentals &
Operators**

JAVA @11

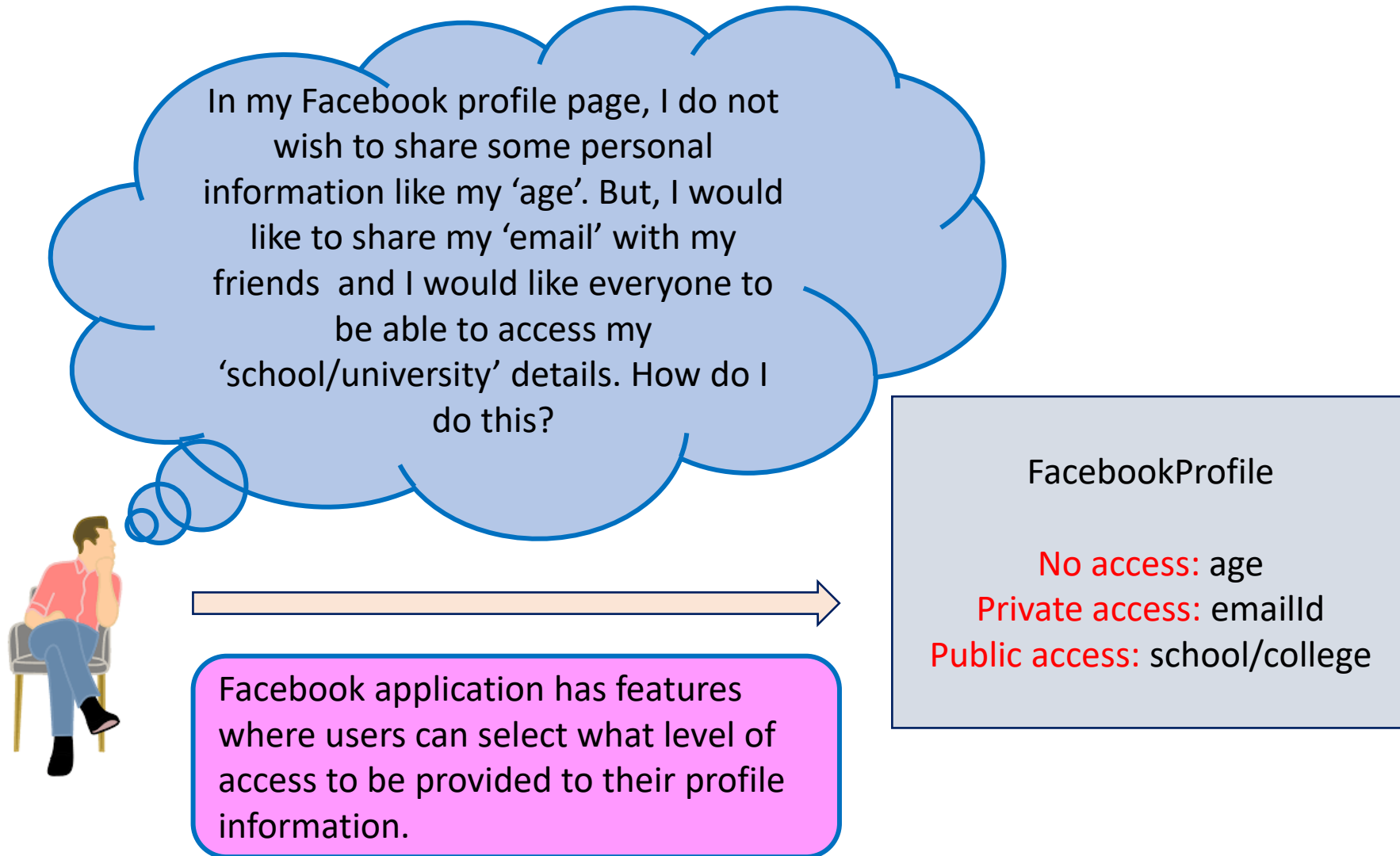
Access Specifier, Constructors, Methods

Objective

After completing this session you will be able to understand,

- Access Modifiers
- Encapsulation
- Method overloading
- Static keyword
- Constructor
- Constructor chaining

Access Modifier



How is access level defined in Java?

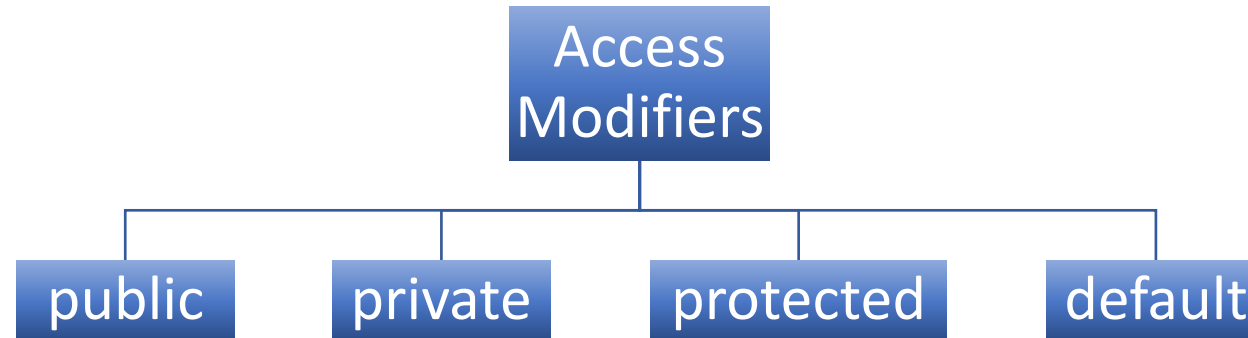
Similarly when developing java application developers needs to secure their methods/variables by defining different access level.

This is achieved using “Access Modifiers”

What is Access Modifier?

What are access modifiers?

The scope of a variable/methods can be defined by using access modifiers.



Analogy to the Facebook example:

- School details can be made **public** (can be accessed by any class)
- Age can be made **private** (to be accessed only by the same class)
- Email Id can be made **default** (to be accessed only by the classes in same package)

Public access

What is public access?

Public access specifies that the class members (variables or methods) are accessible to **anyone**, both inside and outside the class and outside of the package.

Syntax: `public <methods/variable name>`

Example:

public variable:

```
public int x = 0;
```

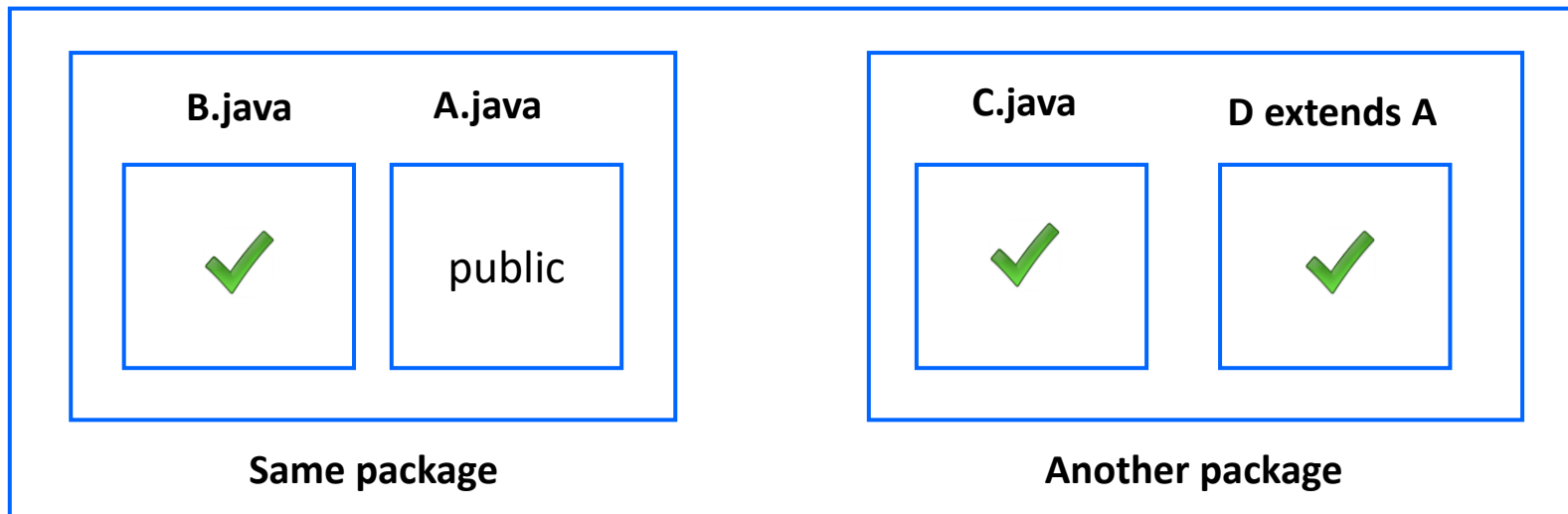
public methods:

```
public addNumbers(int a, int b){  
    //code block  
}
```

public access

Illustration for public access:

- Assume there are 4 java classes A, B, C and D.
- Classes A & B are in the same package where as C & D is in a different package. D is a subclass from A.
- Class A has a public variable. Class B will be able to access the public variable in class A.
- Class C & D will be able to access the public variable in class A



Private access

What is private access?

Private access specifies that the class members (variables or methods) are only accessible by the class in which they are defined. Cannot be accessed by any other class.

Syntax: `private <methods/variable name>`

Example:

private variable:

```
private int x = 0;
```

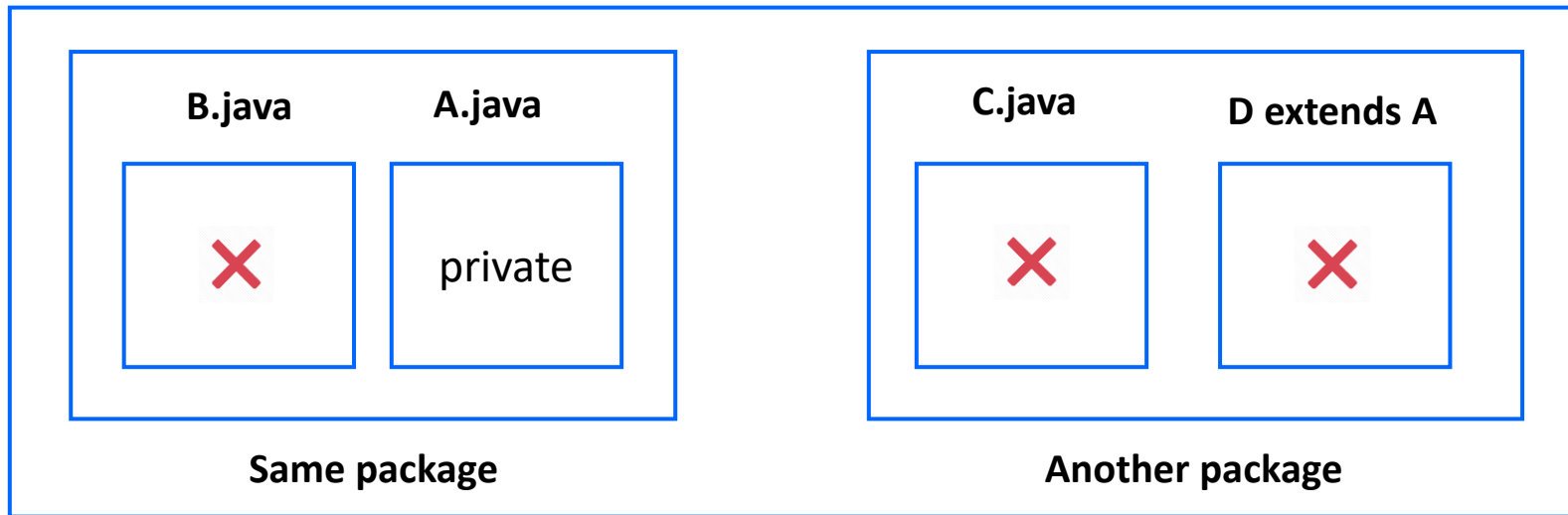
private methods:

```
private addNumbers(int a, int b){  
    //code block  
}
```

private access

Illustration for protected access:

- Class A has a **private** variable.
- Class B will **not** be able to access the private variable in class A
- Class C will **not** be able to access the private variable in class A
- Class D will not be able to access the private since it is a subclass from class A
- Only methods in class A can access the private variable in class A.



Protected access

What is protected access?

Protected access specifies that the class members (variables or methods) are accessible to only the methods in that **class**, classes from **same** package and the **subclasses** of the class. The subclass can be in any package.

Syntax: `protected <methods/variable name>`

Example:

protected variable:

```
protected int x = 0;
```

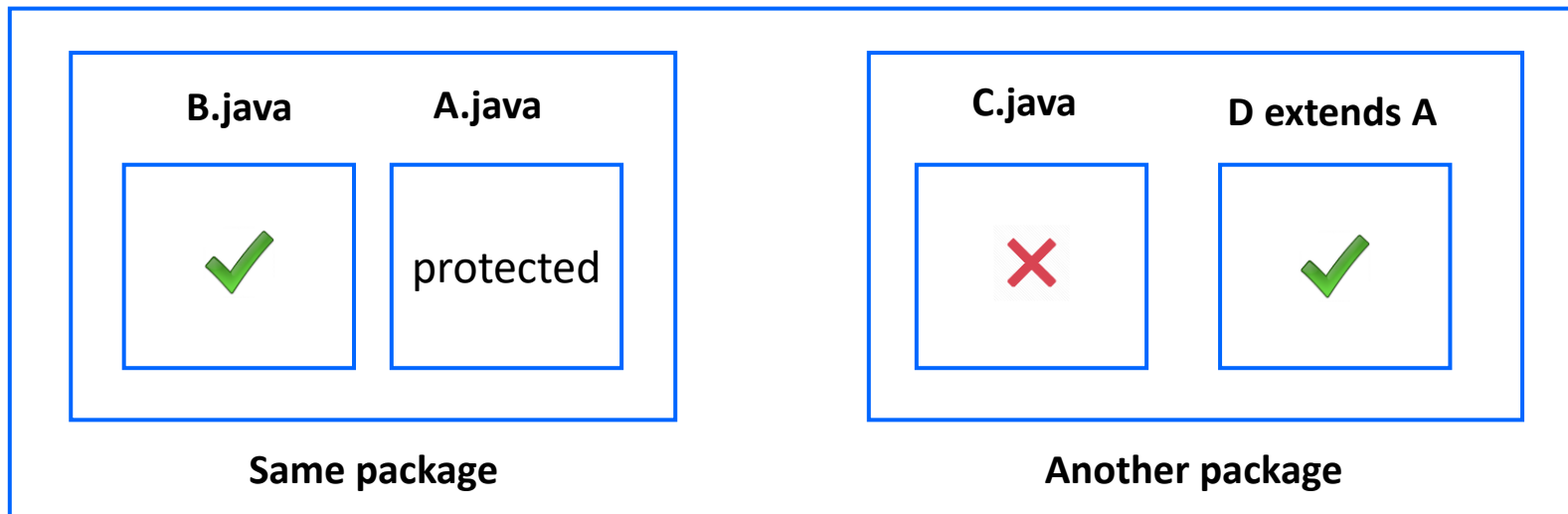
protected methods:

```
protected addNumbers(int a, int b){  
    //code block  
}
```

Protected access

Illustration for protected access:

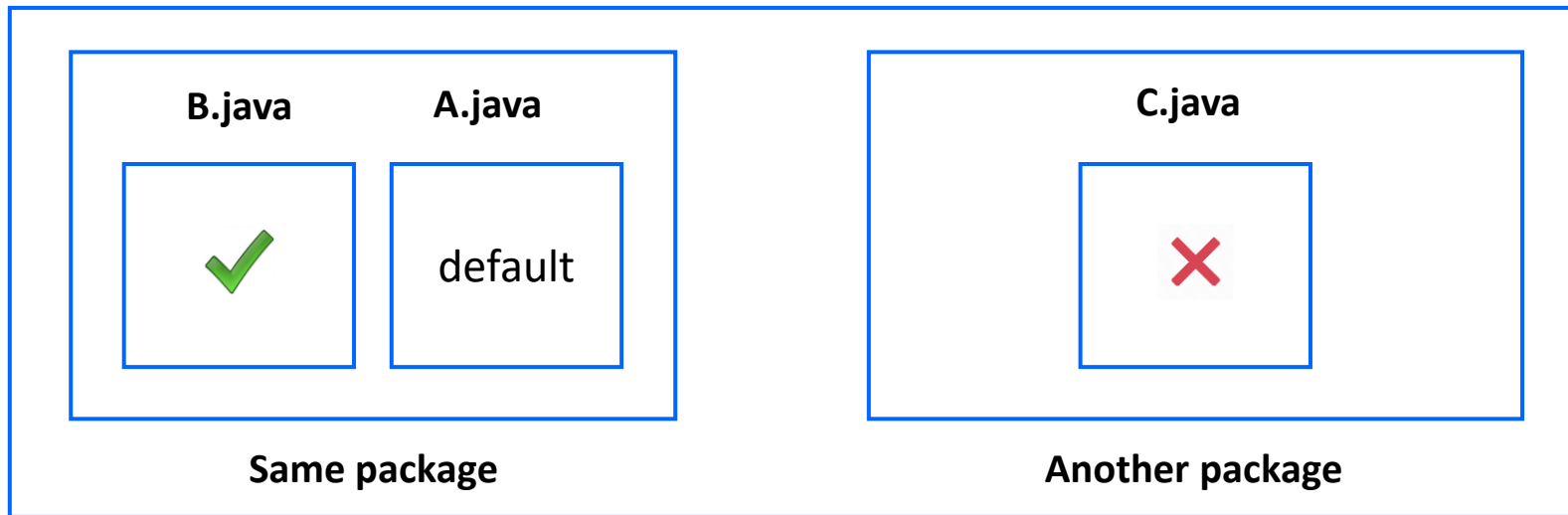
- Class A has a protected variable.
- Protected variable in class A can be accessed by class B in the same package.
- It can be accessed by class D since class D is a subclass of class A
- It cannot be accessed by class C since class C is in a different package and is not a sub class of class A.



Default access

What is default access?

Default/No access specifies that only classes in the ***same package*** can have access to the variables and methods of the other class. No keyword is required for the default modifier and it is applied in the absence of an access modifier.

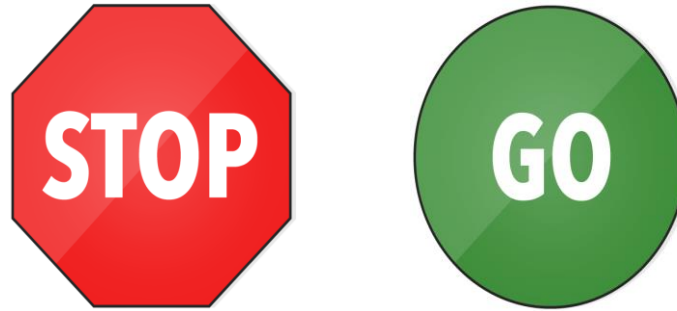


Modifiers in a nutshell

<i>Access Modifier</i>	<i>Same Class</i>	<i>Same Package</i>	<i>Subclass</i>	<i>Other packages</i>
public	Y	Y	Y	Y
protected	Y	Y	Y	N
No access modifier	Y	Y	N	N
private	Y	N	N	N

public and *private* are the commonly used access specifiers in projects

Time To Reflect



Trainees to reflect the following topics before proceeding.

- Why do you need access modifiers?
- What are the types of access modifiers?
- A method declared protected can it be accessed from other class residing in another package?
- A variable declared private can it be accessed from other class residing in the same package?
- What is the access specifiers used to prevent sub classes accessing the methods and variables?

Methods

What is methods?

A method is a set of statements to perform a desired functionality which can be included inside a java class.

This set of statements can be called (invoked) at any point in the program by using the method name.

Example: main method in class A invokes method “add” in class B and method “multiply” in class C

```
class A{  
    public static void main(String[] args){  
        statement1;  
        add();  
        statement2;  
        multiply();  
    }  
}
```

```
class B{  
    public void add(){  
        //code block  
    }  
}
```

```
class C{  
    public void multiply(){  
        //code block  
    }  
}
```


Method declaration

How to declare a method?

Syntax:

```
<modifier><returnType><methodName>(<parameter-list>){  
    <statements>*  
}
```

Example:

```
public int add(int x, int y){  
    int sum = x+y;  
    return sum;  
}
```

Encapsulation

What is encapsulation?



The actual medicine is hidden inside the capsule. The patient is not aware of the contents of the medicine.

Encapsulation

What is Encapsulation?

- Encapsulation is one of the fundamental OOP concepts.
- It is the protective barrier that prevents the data in a class from being directly accessed by the code outside the class.
- Access to the data and code is tightly controlled by using an interface.

Example: In facebook people don't share their age information.

How is Encapsulation achieved?

How is Encapsulation done?

- The fields in class are made ***private*** so that it cannot be accessed by anyone outside the class.
- Hence encapsulation is also called ***“Data Hiding”***.
- The fields can be accessed only by using the methods in the class.
- Encapsulated data is accessed using the “Accessor (getting)” and “Mutator (setter)” methods.
 - **Accessor** – methods to retrieve the hidden data.
 - **Mutators** – methods to change hidden data.

When encapsulation used?

Encapsulation are used to create **value** or **transfer** objects.

Transfer objects are used to logically bundle the related data and transferring it to another objects.

Example: EmployeeVO – variable like **employeeid**, **salary** will be encapsulated and exposed using getters/setters.

VO stands for Value Object.

Exercise - Encapsulation

Let us look at an example of encapsulation,

In our **Facebook** profile, we would like to hide the **age**, **contactNo** and **maritalStatus** information to the external world. This can be done using encapsulation where the fields are made private and can be accessed only by the accessor and mutator methods.

```
public class Facebook {  
    private int age;  
    private long contactNo;  
    private String maritalStatus;  
    public int getAge() {  
        return age;  
    }  
    public void setAge(int age) {  
        this.age = age;  
    }  
    public long getContactNo() {  
        return contactNo;  
    }  
    public void setContactNo(long contactNo) {  
        this.contactNo = contactNo;  
    }  
    public String getMaritalStatus() {  
        return maritalStatus;  
    }  
    public void setMaritalStatus(String maritalStatus) {  
        this.maritalStatus = maritalStatus;  
    }  
}
```

The public methods are access points to the fields of this class. Hence, any class that wants to access the variables should access them through these getters and setters

Exercise - Encapsulation

The variables of the EncapsDemo can be accessed as follows,

```
public class EncapsDemo {  
    public static void main(String[] args) {  
        Facebook fb = new Facebook();  
        fb.setAge(21);  
        fb.setContactNo(9943155523L);  
        fb.setMaritalStatus("Single");  
  
        System.out.println("Marital status: "+fb.getMaritalStatus()+  
                           " Age: "+fb.getAge()+  
                           " Contact Number: "+fb.getContactNo());  
    }  
}
```

The fields are set using the setter (Mutator) methods.

The values are got using the getter (Accessor) methods.

Returning values from a method

How to return from a method?

- A method returns a value using the **return** keyword.

Syntax:

```
return <return Value>;
```

Where,

- **<return Value>** denotes the variable whose value needs to be returned.
- The datatype of the “**return Value**” must match the data type specified in the method declaration.
i.e. You cannot return an integer variable value from a method which is declared to return a boolean.
- **Returning control:** If the method execution needs to be stopped and the control needs to be sent back to the calling method simply use the return keyword without the variable.

Syntax: `return;`

Return from a method

Multiple Return Statements:

- A method can have multiple return statements.
- You can use constants as return values instead of variables below is an example of a code having multiple return statements.

```
public String getNumberInWords(int number) {  
    String defaultNumber = "zero";  
    if (number == 1) {  
        return "one";  
    } else if (number == 2) {  
        return "two";  
    }  
    return defaultNumber;  
}
```

← A constant value is being returned

← Returning a variable value

NOTE: It is **NOT** a good practice to have multiple return statements from a method. Let us look at a solution to fix this best practice violation in the next slide.

Return from a method

Best Practice: A method should have only one return statement.

```
public String getNumbeInWords(int number) {  
    String defaultNumber = "zero";  
    if (number == 1) {  
        defaultNumber = "one";  
    } else if (number == 2) {  
        defaultNumber = "two";  
    }  
    return defaultNumber;  
}
```



Value stored in a variable and
returned from a single point

Exercise for Methods.

Let us all develop a program with a method invocation.

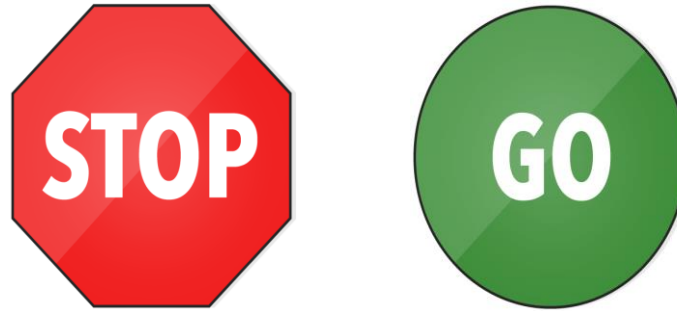
1. Create a Java class "Circle.java" with a method "calculateArea"
2. This method should accept radius as an argument, calculate the area of circle and return the result.
3. The main method should invoke the Circle object "calculateArea" method by passing a value for the radius, say 12.5
4. The main method should also print the area of circle (result variable)

Exercise - Solution

Solution:

```
public class Circle {  
    public static double calculateArea(double radius) {  
        return (Math.PI*radius*radius);  
    }  
    public static void main(String[] args) {  
        double result = calculateArea(12.5);  
        System.out.println("Area of Circle is "+result);  
    }  
}
```

Time To Reflect



Trainees to reflect the following topics before proceeding.

- What is a method?
- What are the building block elements of the method?
- What is encapsulation?
- How is a variable encapsulated?
- How can we transfer the execution control to the calling method?
- Can we have multiple return statements from a method?

Method Overloading

What is method overloading?

Let us recollect what we learnt in the Introduction to OOPs session.



```
Implementation #1  
makeSound(){  
    //default parameter  
    Bark woof woof  
}
```

```
Implementation #2  
makeSound(injured){  
    //Added input parameter  
    Make a whining sound  
}
```

Yes, You got    

If not, go to the next slide for the definition !!

Method Overloading

What is method overloading?

- Two different versions of the same method available in the same class.
- This is done by either changing the input parameter or return type.

Example: The method “**add**” has been overloaded below,

```
void add() {  
  
    System.out.println("No parameters for this method");  
}  
  
void add(int a, int b, int c) {  
  
    int sum = a + b + c;  
    System.out.println("Sum of a+b+c is " + sum);  
}  
  
void add(String s1, String s2) {  
  
    String s = s1 + s2;  
    System.out.println(s);  
}
```

Exercise – Method Overloading

Let us all create 4 overloaded methods for “test()” and invoke all versions of the overloaded methods.!!

```
public class OverloadDemo {  
  
    void test() {  
        System.out.println("No parameters");  
    }  
  
    // Overload test for one integer parameter.  
    void test(int a) {  
        System.out.println("a: " + a);  
    }  
  
    // Overload test for two integer parameters.  
    void test(int a, int b) {  
        System.out.println("a and b: " + a + " " + b);  
    }  
  
    // overload test for a double parameter  
    double test(double a) {  
        System.out.println("double a: " + a);  
        return a * a;  
    }  
}
```


Exercise – Method Overloading

Create another class Overload .java which has a main method to call the overloaded methods in OverloadDemo.java

```
public class Overload {  
    public static void main(String args[]) {  
        OverloadDemo ob = new OverloadDemo();  
        double result;  
        // call all versions of the overloaded method test()  
        ob.test();  
        ob.test(10);  
        ob.test(10, 20);  
        result = ob.test(123.2);  
        System.out.println("Result of ob.test(123.2): " + result);  
    }  
}
```

static keyword

Static keyword:

The static keyword is used before a method or variable (similar to an access modifier).

Examples:

```
variable: private static int x = 0;  
method: public static add(){  
        // some code here  
}
```

Dictionary Definition of static: Changeless, motionless

static keyword

What is a static member?

- Static variables are global variables, which is shared by all the instances of the class.
- Static means that there will only be only one instance of that object in the class.
- It is not required to create an instance of the object to access the static method or variable.

Where is it used?

Static variables are used in singleton implementation. You will learn more about this during the design pattern session.

Exercise – static variable

Let us all understand static variables using an example. Develop the code and look at out static is different from a normal variable.

```
public class StaticVarDemo {  
    private int x = 5;  
    private static int y = 3;  
    public StaticVarDemo() {  
        x++;  
        y+=7;  
        System.out.println("X is "+x+" and Y is "+y);  
    }  
    public static void main(String[] args) {  
        StaticVarDemo s1 = new StaticVarDemo();  
        StaticVarDemo s2 = new StaticVarDemo();  
        StaticVarDemo s3 = new StaticVarDemo();  
    }  
}
```

- The output of the program is below,

```
X is 6 and Y is 10  
X is 6 and Y is 17  
X is 6 and Y is 24
```

Reason:

- Before s1 is created, X = 5 and Y = 3;
- When s1 is created, X = 6 and Y = 10 (as per logic inside constructor)
- When s2 is created, Y = 17 since the value of Y does not change with every instance. The value of Y is shared across all instances.
- But X = 6 (since a new object of X is created and incremented by 1.
- Similarly, when s3 is created, X = 6 and Y = 24

What is a static method?

What is a static method?

A "static" method belongs to the class, not to any particular instance. In other words, static methods can be invoked using the class name

Where is it used?

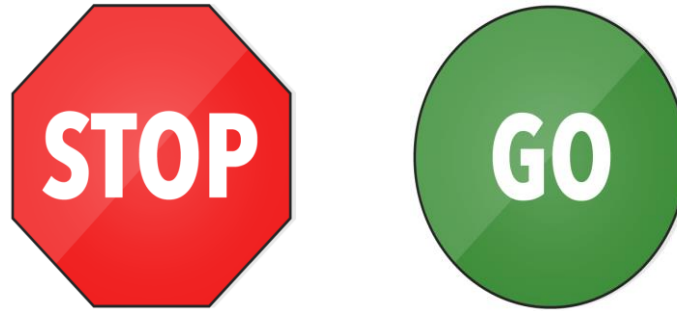
Static methods are used in singleton implementation. You will learn more about this during the design pattern session.

Exercise – static method

Let us develop the following code to get accustomed to static method declaration and invocation. The static method `printMessage()` within class and `printData()` can be accessed directly using the class name.

```
class Test{
    public static void printData() {
        System.out.println("This is static other class.");
    }
}
public class StaticMethodDemo {
    public static void printMessage() {
        System.out.println("This is static within class.");
    }
    public static void main(String[] args) {
        printMessage();
        Test.printData();
    }
}
```

Time To Reflect



Trainees to reflect the following topics before proceeding.

- What is method overloading?
- What are the ways by which you can overload methods?
- What is a static variable?
- How is a static variable invoked?

Constructor

What is a constructor?

A constructor is a special method used for creating an object of a class and initializing its instance variables

- Constructors should have the same name as the class
- It should not have any return, not even void.
- It is not mandatory for the developer to write a constructor.
- If constructor is not defined by developer java will use the default constructor to create objects.

Default Constructor

Default Constructor (no-args constructor):

- A constructor with no parameters
- If the class does not specify any constructors, then an implicit default constructor is automatically called by the JVM.

Syntax:

```
public class Employee {  
    public Employee() {  
        //code block  
    }  
}
```



**Default
Constructor**

Overloading Constructors

What is overloaded constructor?

A default constructor with one or more arguments is called a overloaded constructor

How to overload constructors?

Step 1: Create a method with the same name as the class name

Step 2: Do not provide any return type for the method created

Step 3: Add required number of arguments for the constructor (method) to create an overloaded constructor

Step 4: Any number of overloaded constructors can be created for a class.

Exercise – How to overload constructor?

Lets all create this class and overload the constructors as illustrated.

```
public class Employee {  
    private int empId;  
    private String empName;  
    private String desig;  
    public Employee() {  
        System.out.println("Default constructor.");  
    }  
    public Employee(int empId) {  
        this.empId = empId;  
    }  
    public Employee(int empId, String empName) {  
        this.empId = empId;  
        this.empName = empName;  
    }  
    public Employee(int empId, String empName, String desig) {  
        this.empId = empId;  
        this.empName = empName;  
        this.desig = desig;  
    }  
    public static void main(String[] args) {  
        Employee e1 = new Employee(1201);  
        Employee e2 = new Employee(1202, "Ram");  
        Employee e3 = new Employee(1203, "Hari");  
  
        System.out.println("Id of First employee "+e1.empId);  
        System.out.println("Id of Second employee "+e2.empId);  
        System.out.println("Id of Third employee "+e3.empId);  
    }  
}
```

Default Constructor

Overloaded Constructors

Objects instantiated using different constructors

“this” Reference

The “this” keyword:

- “**this**” refer to the **current object** instance itself.
- It can be used for only instance variables and not for static or class variables
- It is used for also invoking the overloaded constructors.
- It is used to access the instance variables shadowed by the parameters in methods. Typically used with the encapsulated field.

Example: A method **calculateTax** has a parameter *salary*, also the class has the instance variable *salary*. In the method if we refer using “**this**” keyword it means we are referring to instance variable.

Syntax:

`this.<name of instance variable>`

`this.<constructor Name><arguments>`

“this” reference - example

```
public class Shape {  
    private int x=0;  
    private int y=0;  
    public Shape() {  
        //code block  
    }  
    public Shape(int x,int y) {  
        this.x=x;  
        this.y=y;  
    }  
}
```

The member variables
are being referred
using the this key word.

Chaining Constructor call using “this()”

Chaining Constructor Calls:

- Constructor calls can be chained, which mean you can call another constructor from a constructor of the same class.
- You can use “this()” for invoking constructors from other constructor of a class.

There are a few things to remember when using this() method constructor call:

- When using “this()” constructor call, it must occur as the first statement in a constructor.
- The “this()” call can then be followed by any other statements.

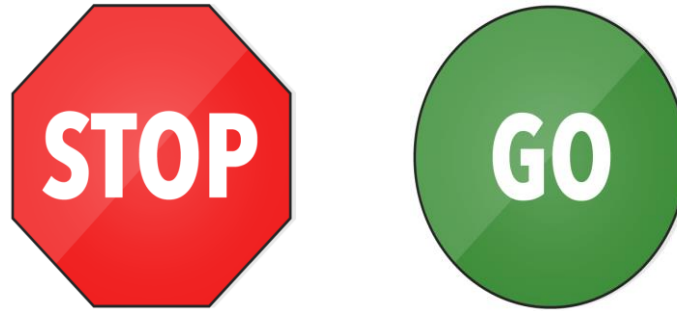
“this()” Constructor call

Let us example of constructor chaining using “this()” reference.

```
public class Employee {  
    private String empName;  
    public Employee() {  
        this("Hari");  
    }  
    public Employee(String empName) {  
        this.empName = empName;  
    }  
    public static void main(String[] args) {  
        Employee e1 = new Employee();  
        System.out.println("Name of the employee is "+e1.empName);  
    }  
}
```

The overloaded constructor is being invoked by the default constructor using this()

Time To Reflect



Trainees to reflect the following topics before proceeding.

- What is a constructor?
- How to overload constructor?
- Can a constructor have return value?
- What is chaining of constructors? How can it be implemented?

Thank you

You have successfully completed **Access
Specifier, Constructors &
Methods**

JAVA @11

Statements

Objective

After completing this session you will be able to understand,

- Statement
- Selection statement
- Iteration statement
- Transfer statement

Java Statements and Blocks

What is a Statement?

A statement is a complete instruction terminated by a semi-colon.

Example: Assignment Statement

```
String name="Greeting";
```

What is a Block?

A block is group of statements enclosed in curly brackets.

Example:

```
{  
    name="Ramesh";  
    age=12;  
}
```

Java executes one statement after the other in the order they are written

Java Control Statements

What are Control Statements?

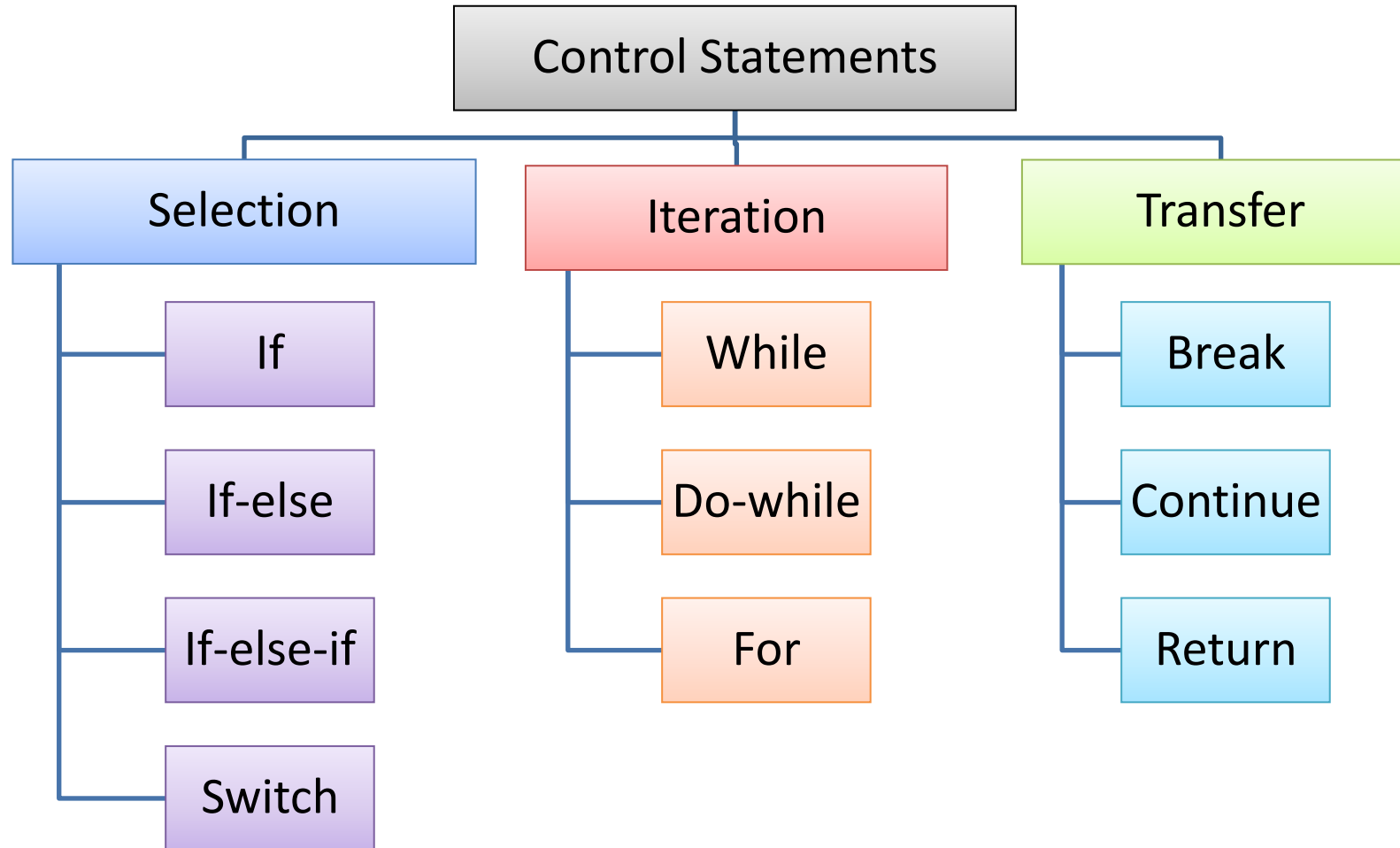
The control statements allows developers to control the flow of program's execution based upon some conditions during run time.

The control statements allows developers to,

Repetitive execution of statements – Executing a statement 'N' number of times

Conditional execution of statements – Execute statements based on some condition.

Control Statements - Categories



Selection Statements

What are Selection statements?

Selection statements allow the program to choose different paths of execution based upon the outcome of an conditional expression or the state of a variable.

Java supports the following selection statements

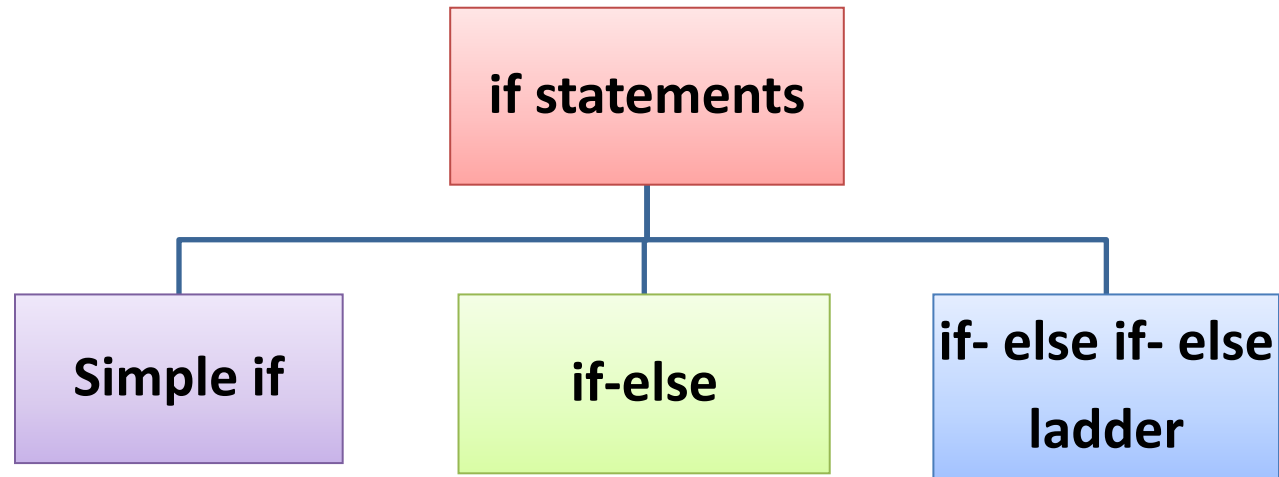
- **if else** statement
- **switch** statement

Selection Statement - IF

If Statement:

The if statements can check **conditions** starting from very simple to quite complex and execute statements.

Conditions are nothing but a single relational expression or the combination of more than one relational expressions with logical operators. If statement comes in **three** different constructs



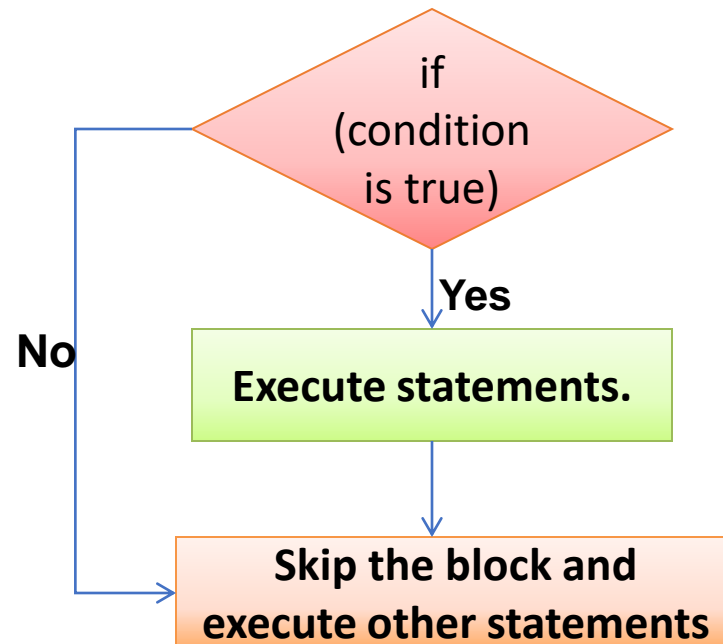
A simple If Statement

Features of Simple If statement:

The simple if statement allows the execution of a **single** statement or a **block** of statements enclosed within curly braces.

The if statement handles a very **simple** situation and executes only when the condition is true otherwise the whole statement body is skipped .

Illustration:



Simple if Statement

Syntax of Simple If statement:

Option I:

```
if(condition1 ) {  
    //statement body;  
}
```

Option II:

```
if(condition1 && condition2){  
    //statement body;  
}
```

The condition can be a **single** relational expression or a **combination** of more than one relational expression separated using a logical operators

if Statement Example

Examples of a simple If statement:

Example1: Without curly braces

```
int a = 10;  
int b = 20;  
If(a>b)  
    System.out.println("Value of a is greater than b");
```

Example2: With curly braces

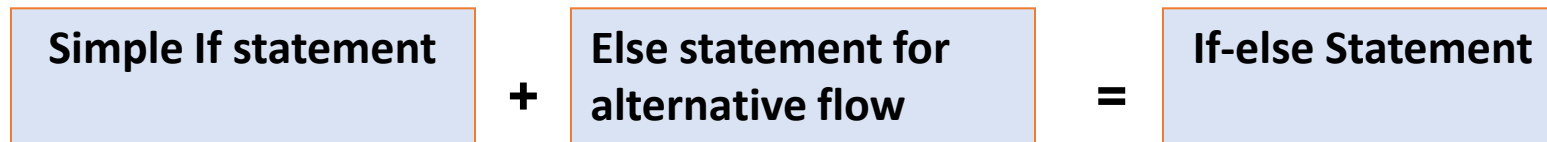
```
int empId=1201;  
int retirementAge=58;  
int empAge=36;  
if((empAge<retirementAge) && (empId==1201)){  
    System.out.println("Calculate Salary ");  
}
```

If both the condition **age** and **id** are satisfied the message will be printed.

It is a good programming practice to use the curly braces regardless of whether there is one or more statements.

if-else Statement

What is an if-else statement?



If-else can handle **two** blocks of code, and only one of those blocks will be executed based on the condition outcome.

Syntax

```
if( <condition1> ) {  
    statements  
}  
else{  
    statements  
}
```

If **condition1** is **satisfied** these statements are executed

If **condition1** is **not satisfied** these statements are executed

if-else Statement Example

Example of an if-else statement:

```
int empId=1201;
```

```
int retirementAge=58;
```

```
int empAge=36;
```

```
if( (empAge<retirementAge) && (empId==1201) ){
```

```
    System.out.println("Calculate Salary ");
```

```
}
```

```
else {
```

```
    System.out.println("Calculate Pension");
```

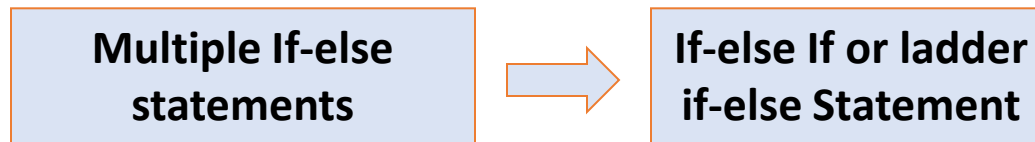
```
}
```

If both the condition **age** and **id** are satisfied the message will be printed.

If the conditions in if block is not satisfied, this block will be fired.

if-else if Statement

What is an if-else if statement?



If-else if can handle **more than two blocks** of code, and only one of those blocks will be executed.

Syntax

```
if( <condition1> ) {  
    statements  
}  
else if(<condition2>){  
    statements  
}  
else{  
    statements  
}
```

If **condition1** is **satisfied** these statements are executed.

If **condition1** is **not satisfied** and **condition2 is satisfied** these statements are executed.

If **both conditions are not satisfied** these statements are executed.

if-else if Statement Example

```
package com.statements.demo;

class IfElseIfExample {

    public static void main(String args[]) {
        int month = 4;
        String season;
        if(month == 12 || month == 1 || month == 2)
            season = "Winter";
        else if(month == 3 || month == 4 || month == 5)
            season = "Spring";
        else if(month == 6 || month == 7 || month == 8)
            season = "Summer";
        else if(month == 9 || month == 10 || month == 11)
            season = "Autumn";
        else
            season = "Bogus Month";
        System.out.println("April is in the " + season + ".");
    }
}
```

If statement

Else If statement

Else statement

Nested if Statements

What is a nested if statement?

The if statement in java can be nested, in other words, an if statement can be present inside another if statement

Example:

The discount % of T.V is calculated based on the below criteria

If T.V is LED, check the screen size

If screen size is 32, discount % = 10

If screen size is 46, discount % = 15

If T.V is LCD, discount % = 5

Nested if statement
If the type of TV is LED,
then size check is done

```
public void checkDiscount() {  
    String typeOfTV = "LED";  
    int sizeofTV = 32;  
    int discount;  
  
    if("LED".equals(typeOfTV)) {  
        if(sizeofTV==32){  
            discount = 10;  
        }else if (sizeofTV == 46){  
            discount = 15;  
        }  
    }else if("LCD".equals(typeOfTV)) {  
        discount = 5;  
    }  
}
```

Lend a Hand – if else if

1. Create a java class “**NumberCheck**” add a method **displayBigNumber** three int parameters “**num1**” ,”**num2**” and “**num3**”.
2. The method **displayBigNumber** will check and print the biggest of the three numbers in the following format

“<result> + is the Biggest Number”
3. Create a java class “MainProgram” add a main method which will
 - Create a object instance of the NumberCheck.
 - Trigger the method **displayBigNumber** by passing values of num1,num2, and num3 as 11,18 and 7.
4. The message needs to be displayed in the console.

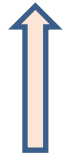
Solution

```
public class Numbercheck {  
  
    void displayBigNumber(int num1,int num2,int num3){  
  
        int biggestNumber;  
  
        if(num1>num2){  
            if(num1>num3){  
                biggestNumber = num1;  
            }else if(num3 > num2){  
                biggestNumber = num3;  
            }else{  
                biggestNumber = num2;  
            }  
        }else if(num2>num3){  
            biggestNumber = num2;  
        }else{  
            biggestNumber = num3;  
        }  
        System.out.println(biggestNumber+ " is the biggest number");  
    }  
}  
  
public class MainProgram {  
  
    public static void main(String[] args) {  
        Numbercheck check = new Numbercheck();  
        check.displayBigNumber(11, 18, 7);  
    }  
}
```

Switches in Real life

In the below illustration the respective switches are used to control the working of the respective electrical appliances.

Example: Switch on Fan use the fan switch and so on....



Switch Statement

Similarly when developing software applications to control the flow of execution in executing a particular block of statements we use the ***switch*** statement.

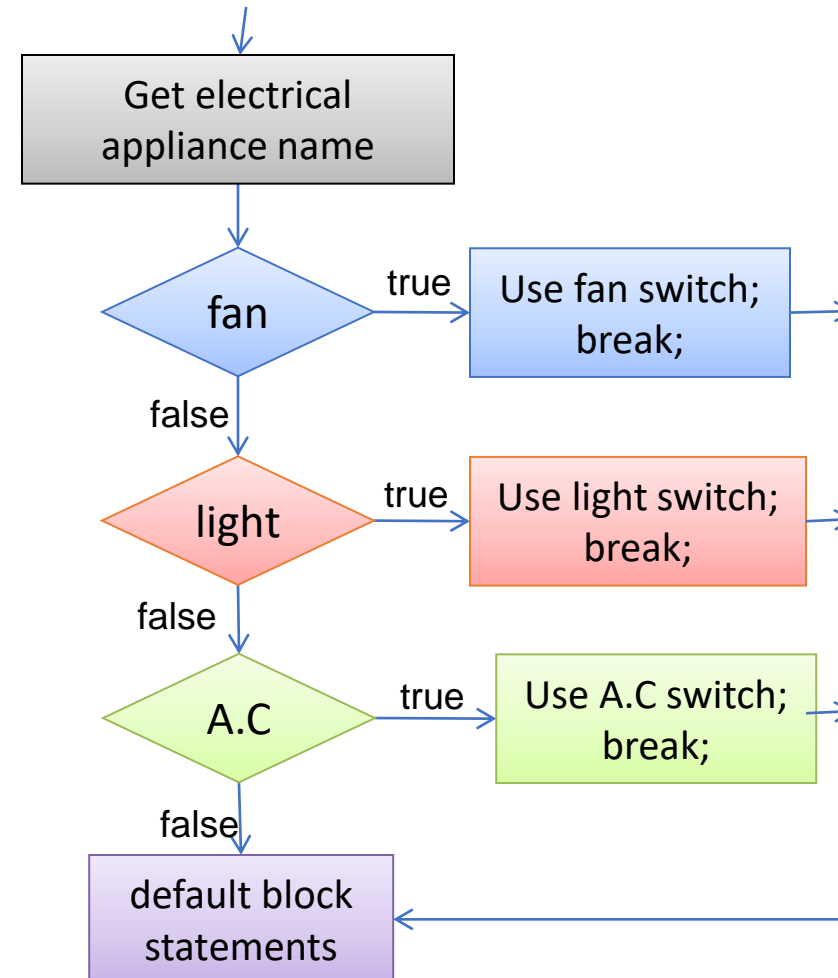
The switch statement allows to **choose** a **block** of statements to run from a number of option available.

This can also be implemented using nested if-else. So what is the difference. We will see the difference soon.

How to write Switch Statement

Syntax:

```
switch (expression) {  
  case value1:  
    // statement sequence  
    break;  
  
  case value2:  
    // statement sequence  
    break;  
  
  case value N:  
    // statement sequence  
    break;  
  
  default:  
    // default statements  
    break;  
}
```



Switch Statement Example

Example:

```
int x=6%2;
switch (x) {
    case 0:
        System.out.println("The value of x is 0." );
        break;
    case 1:
        System.out.println("The value of x is 1." );
        break;
    default:
        System.out.println("The value of x is default.");
        break;
}
```

The argument of switch() must be one of the types byte, short, char, int

There should be no duplicate case labels i.e., the same value cannot be used twice.

How Switch works?

Lets see how Switch works without break statements,

Assume the value of X is 10

```
switch (x){
```



Switch Statement Executed

```
case 12:
```

```
//Statements
```

```
break;
```

```
case 10:
```



Case 10 condition passes

```
//Statements
```



Case 10 statements executed

```
break;
```



Break statement executed
and control goes outside the
switch block.

```
case 15:
```

```
//Statements
```

```
break;
```

```
default:
```

```
//Statements
```

```
break;
```

```
}
```



Break statement breaks the execution control
flow and control passed outside the switch
block.

How Switch works without break?

Lets see how Switch works without break statements,

Assume the value of X is 10

```
switch (x){
```

```
case 12:
```

```
    //Statements
```

```
case 10:
```

```
    //Statements
```

```
case 15:
```

```
    //Statements
```

```
default:
```

```
    //Statements
```

```
}
```



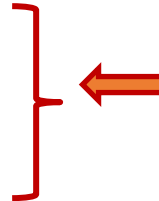
Switch Statement Executed



Case 10 condition passes



Case 10 statements executed



Since case 10 block does not have break the case 15 and the default block statements will be executed .



After all the cases statements are executed the control goes outside the switch blocks.

Switch Statement

Some facts about switch statement:

- Java first evaluates the *switch expression* and jumps to the case which matches the value of the expression
- Once the correct match is found, all statements from that point are executed till a ***break*** statement is encountered
- Once break statement is encountered, the flow jumps to the statements after the switch structure
- If none of the cases are satisfied, default block is executed. The default block does not have to be at the end of the switch.

switch Vs if

If-else	switch
<p>This can test expressions based on ranges of values or conditions.</p> <p>Example: <code>if(a==10 && b=21)</code></p>	<p>This tests expressions based only on a single integer, enumerated value, or String object.</p> <p>Example: <code>switch(i)</code> <code>// where i is an int.</code></p>

Based on the condition to be evaluated developers can either go for **switch** or **if-else**.

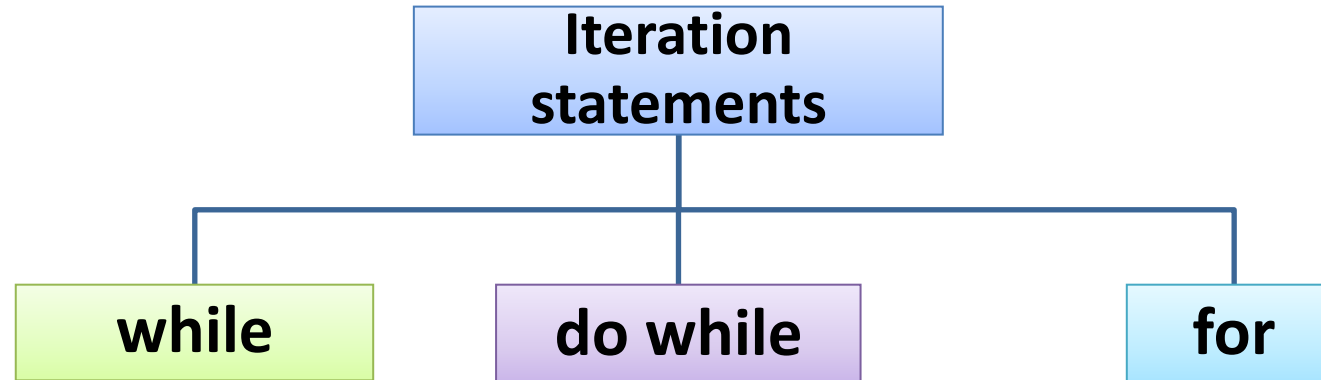
Iteration Statement

What are Iteration statements?

Iteration Statements are used to execute a block of statements **repeatedly** as long as a certain **condition** is **true**.

A single relational expression or the combination of more than one relational expression with logical operators are given as **conditions**.

Java offers three iteration constructs



While Statement

The while loop is Java's most fundamental iteration statement.

Simple Problem statement to understand the usage of while statement:

John has to develop a small java program which needs to **print** a welcome **message** as long as the number of guests is greater than zero.

The above problem statement can be easily done using while loop

Example:

```
while(countOfGuests>0){  
    System.out.println("Welcome to my party");  
    countOfGuests--;  
}
```

While Statement

Facts about while statement:

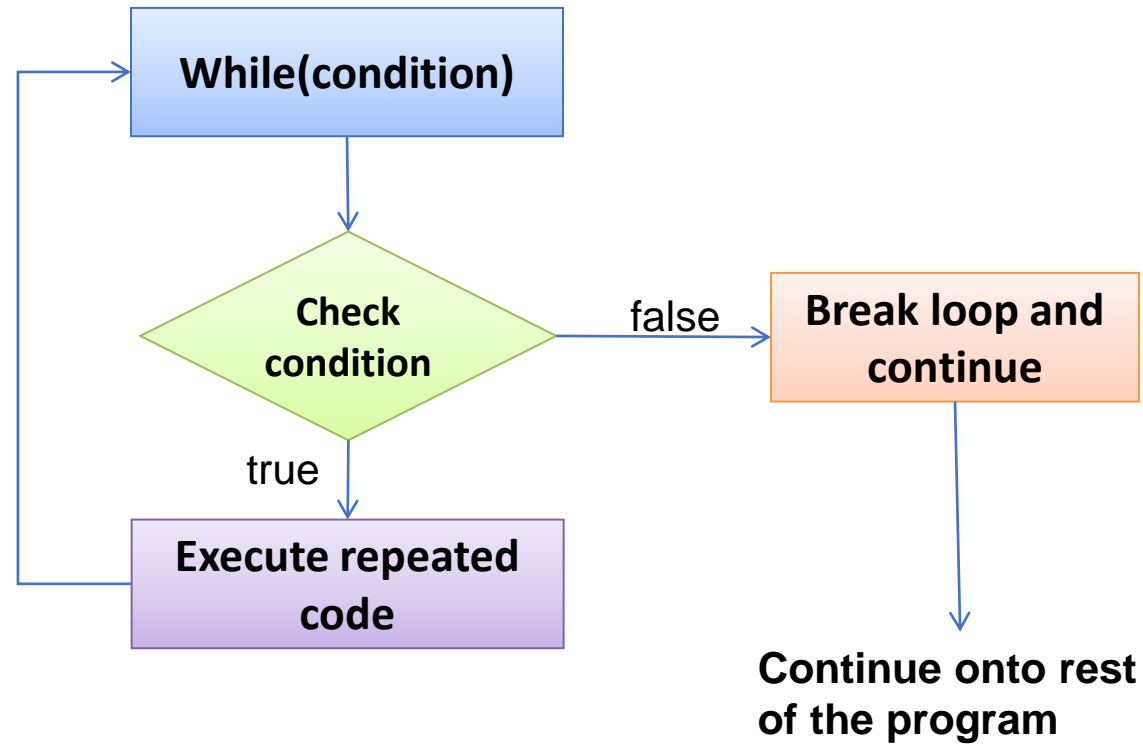
The while loop is a **statement** or **block** of statements that is **repeated** as long as some **condition** is **satisfied**

Syntax

```
while (boolean_expression) {  
    statement1;  
    statement 2;  
    .....  
}
```

The statements in *while* loop are executed as long as the *boolean expression* is true

Illustration of a while statement



Lend a Hand – while

1. Create a java class “WelcomeMessage” and add a method named **printMessage** which would display “Welcome All”.
2. Create a java class “TestProgram” add a main method which will
 - Create an instance of the **WelcomeMessage** and trigger the method **printMessage** five times.
 - The message “Welcome All” should be displayed 5 times.
3. The message needs to be displayed in the console.

Use while Statement

while Statement Example

Develop the code as illustrated below.

```
class WelcomeMessage{
    void printMessage(){
        System.out.println("Welcome All");
    }
}
class TestProgram {

    public static void main(String[] args) {
        int count =5;
        WelcomeMessage message=new WelcomeMessage();
        while(count>0){
            message.printMessage();
            count--; }
    }
}
```

do-while Statement


What is a Do while Loop:

It is **similar** to **while** loop except that the do-while **execute** the block **once**, and then **checks** the **while** condition.

The do-while loop always executes its body at least once, because its conditional expression is at the end of the loop.

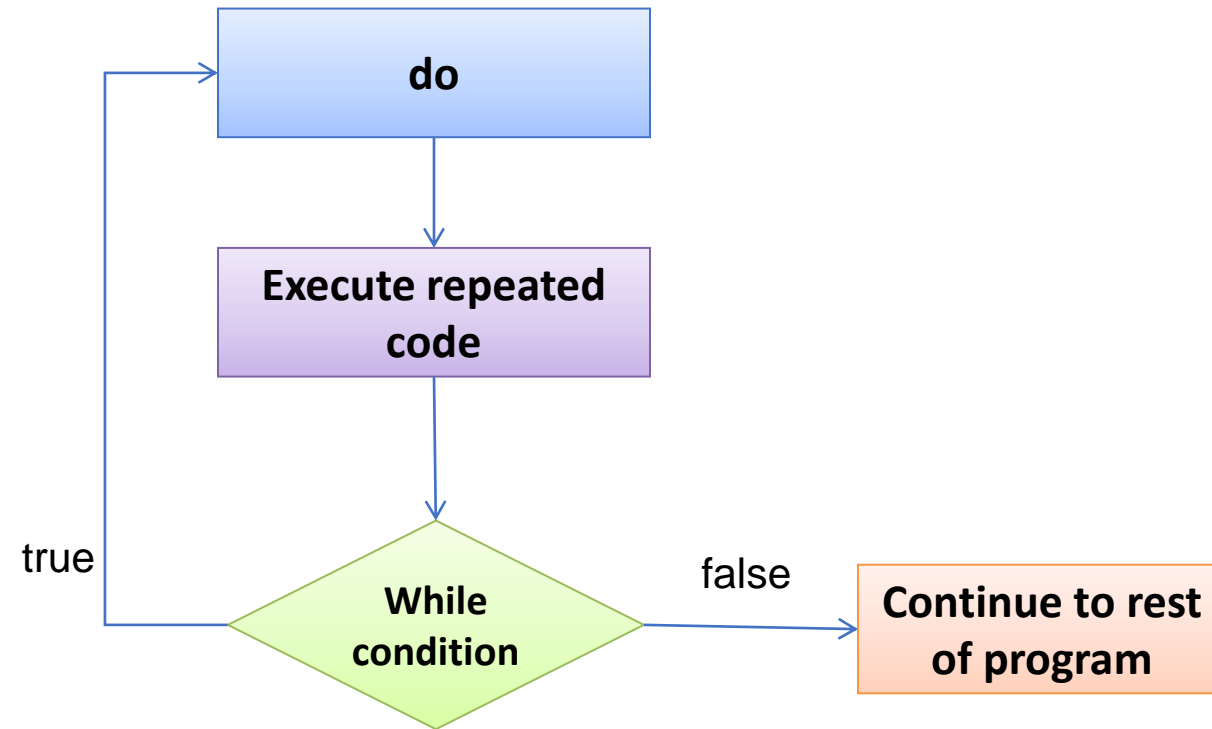
Syntax

```
do {  
    statement1;  
    statement2;  
} while(boolean_expression);
```



Do not forget to
use semicolon
after the while
statement

Illustration of a do while statement



Example of do-while Statement

Example of a do while statement:

```
public class DoWhileExample {  
  
    public static void main(String[] args) {  
        int i = 6;  
        do {  
            System.out.println("i is : " + i);  
            i++;  
        } while (i < 5);  
    }  
}
```

The value of **i** is printed for the first time, even though it does not match the condition **i < 5**

Output:

```
i is : 6
```

Lend a Hand – do while

1. Create a java class “WelcomeMessage” and add a method named **printMessage** which would display “Welcome All”.
2. Create a java class “TestProgram” add a main method which will
 - Create an instance of the **WelcomeMessage** and trigger the method **printMessage** five times.
 - The message “Welcome All” should be displayed 5 times.
3. The message needs to be displayed in the console.

Use do while Statement

do-while Statement Example

Develop the code as illustrated below.

```
class WelcomeMessage{
    void printMessage(){
        System.out.println("Welcome All");
    }
}
class TestProgram {
    public static void main(String[] args) {
        int count =5;
        WelcomeMessage message=new WelcomeMessage();
        do{
            message.printMessage();
            count--; }
        while (count>0);
    }
}
```

for Statement

What is a for loop?

For statement is similar to while loop is used to repeat the execution of the code till a condition is met.

Syntax:

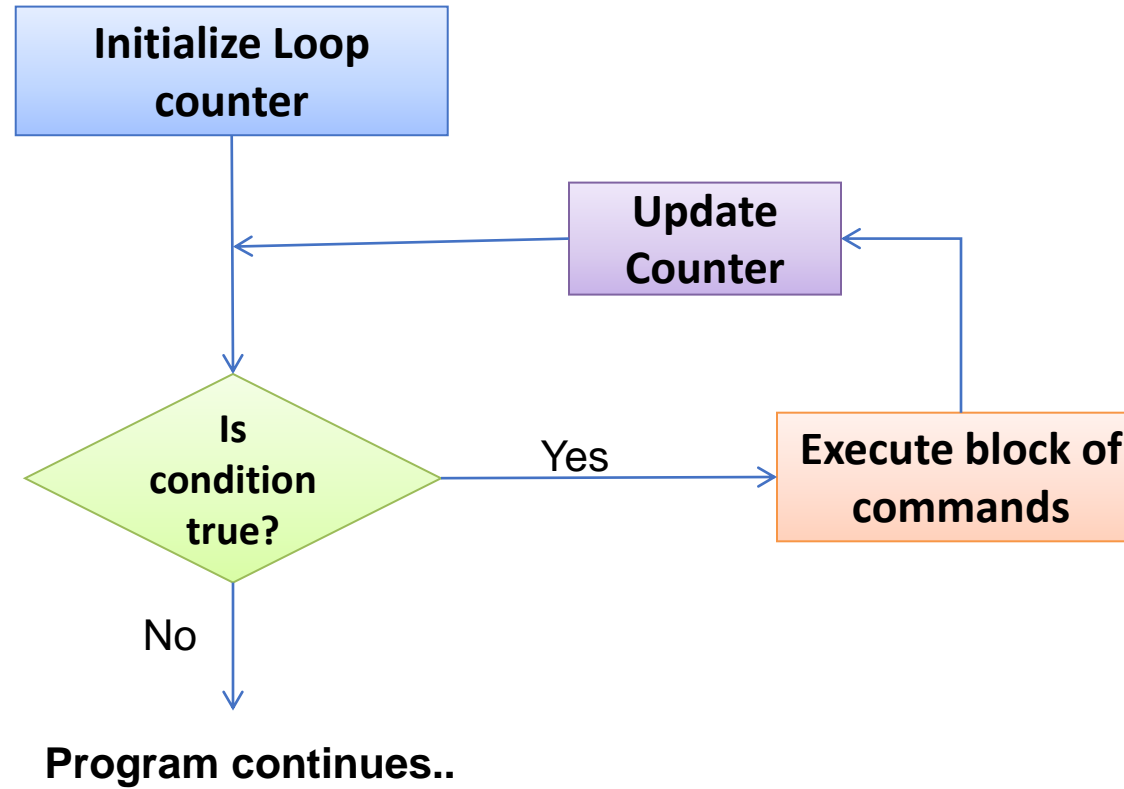
```
for(initialization; loopCondition; iteration) {  
    statements;  
}
```

The **initialization** allows to declare and/or initialize loop variables, and is executed only once.

The **loopCondition** compares the loop variable to some limit value. If the loop condition is not met it is broken.

The **iteration** usually increments or decrements the values of the loop variables

Illustration of a for statement



for Statement Example

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

Loop Value
Initialized to 1.

Condition for loop,
Loop executed till
value is < 10

Loop value
incremented.

Statement
Executed in loop

Lend a Hand – for Loop statement

1. Create a java class “WelcomeMessage” and add a method named **printMessage** which would display “Welcome All”.
2. Create a java class “TestProgram” add a main method which will
 - Create an instance of the **WelcomeMessage**. and trigger the method **printMessage** five times.
 - The message “Welcome All” should be displayed 5 times in the console.
3. The message needs to be displayed in the console.

Use For Loop Statement

for Statement Example

Develop the code as illustrated below.

```
package com.statements.demo;

class DisplayMessage{
    void printMessage(){
        System.out.println("Welcome All");
    }
}

class TestProgram {
    public static void main(String[] args) {
        DisplayMessage message=new DisplayMessage();
        int num=5;
        for(int i=1;i<=num;i++){
            message.printMessage();
        }
    }
}
```

Transfer Statement

What are transfer statements?

The **transfer** Statements in Java alter the normal control flow of the statements. They allow you to redirect the flow of program execution.

Transfer Statements are used to quit either the current iteration of a loop or the entire loop.

Java offers two transfer statements

1. break.
2. continue.
3. return.

break Statement

Break statement used for,

1. Used to terminates a statement sequence in a switch statement.
2. Used to exit loops in Iteration Statement.

Problem statement:

This program iterates through the 100 employees and calculate salary . If one employee is minor age, i.e. age < 18 it should break the loop and stop the execution.

```
while (employeecount<=100) {  
    if(employeeAge <18)  
    {  
        break;  
    }  
    calculateSalary();  
}
```

continue Statement

Continue Statements stops the processing the remaining code in the body of the particular iteration Statement and continue with the next loop.

Problem statement:

This program iterates through the 100 employees and calculate salary . If one employee is minor age, i.e. age < 18 it should SKIP the salary calculation logic for the employee and proceed with other employees.

```
while (employeecount<100) {  
    if(employeeAge <18)  
    {  
        continue;  
    }  
    calculateSalary();  
}
```

continue Statement

- Continue Statement can be used within selection statements which are inside iteration statements .
- In while and do-while statements, a continue statement causes control to be transferred directly to the conditional expression that controls the loop.
- In a for statement, a continue statement skips the current iteration and causes the control to go to the first statement of the loop to proceed with the next iteration.

return Statement

Return statement:

The return statement **exits** from the **current method**, and the control flow returns to where the method was invoked.

The return statement has two forms:

- One that returns a value

- One that doesn't.

Option 1: To return a value, simply put the value or expression that needs to be returned after the return keyword.

return <value/expression>;

Option 2: When the return type of method is void, use the form of return that does not return a value

return;

In this case, the execution of the method is stopped.

Lend a Hand - return statement

```
public class TestProgram {  
  
    public static void main(String[] args) {  
        int count = 5;  
        int i;  
        WelcomeMessage welcome = new WelcomeMessage();  
        for (i=1;i<=count;i++){  
            welcome.printMessage();  
            if(i==3){  
                return;  
            }  
            System.out.println("After if loop "+i);  
        }  
        System.out.println("Final returned value of i is "+i);  
    }  
}
```

Let us use the same WelcomeMessage class that we developed for the previous example

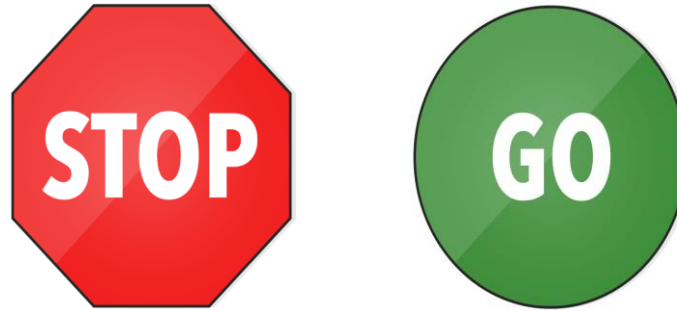
When *i* is equal to 3, the return statement is executed and the execution of the method is stopped

Output

```
Welcome all  
After if loop 1  
Welcome all  
After if loop 2  
Welcome all
```

Try the same example with break and continue statement and see how the program behaves.

Time To Reflect



Trainees to reflect the following topics before proceeding.

1. What statements will you use to execute a block of code repetitively.
2. How to stop a execution of a loop?
3. Difference between `do while` and `while` statement?
4. What is the difference between `switch` and `if` statements?

Thank you

You have successfully completed
Statements