Java Programming
UNIT - I
by

Mr. K. Srikar Goud
Asst. Professor
Department of Information Technology

Syllabus

UNIT - I

Object-Oriented Thinking- A way of viewing world – Agents and Communities, messages and methods, Responsibilities, Classes and Instances, Class Hierarchies- Inheritance, Method binding, Overriding and Exceptions

Syllabus

UNIT - I
Object-Oriented Thinking

Object Oriented Thinking

- When computers were first invented, programming was done manually by toggling in a binary machine instructions by use of front panel.
- As programs began to grow, high level languages were introduced that gives the programmer more tools to handle complexity.
- The first widespread high level language is FORTRAN. Which gave birth to structured programming in 1960's.
- The Main problem with the high level language was they have no specific structure and programs becomes larger, the problem of complexity also increases.
- So C became the popular structured oriented language to solve all the above problems.

Syllabus

- However in SOP, when project reaches certain size its complexity exceeds. So in 1980's a new way of programming was invented and it was OOP.
- OOP is a programming methodology that helps to organize complex programs through the use of inheritance, encapsulation & polymorphism.
- OOP is a Revolutionary idea totally unlike anything that has come before in programming
- OOP is an evolutionary step following naturally on the heels of earlier programming abstractions.

Object Oriented Thinking

A way of viewing world – Agents and Communities

Object Oriented Thinking

- It means how we handle the real world situations through OOP and how we could make the computer closely model the techniques.
- Eg: Anil wished to send some flowers to his friend for his birthday & he was living Patancheru.
- He went to local florist and said the kind of flowers.
- He want to send to his friend's address.
- And florist assured those flowers will be sent automatically

Object Oriented Thinking

VISHNU
UNIVERSAL LEARNING

Agents and communities:

- The Structure of OOP is similar to that of a community, that consists of agents interacting with each other.
- These agents are also called as objects.
- An agent or an object plays a role of providing a service or performing an action, and other members of this community can access these services or actions.

Object Oriented Thinking

VISHNU
UNIVERSAL LEARNING

Agents and communities:

- Consider an eg, Anil and sugnan are good friends who live in two different cities far from each other.

- If Anil wants to send flowers to Sugnan, he can request his local florist ‘Ravi’ to send flowers to Sugnan by giving all the information along with the address.

- Ravi works as an agent (or object) who performs the task of satisfying Anil’s request.
- Ravi then performs a set of operations or methods to satisfy the request which is actually hidden from Sugnan, Ravi forwards a message to Sugnan’s local florist.
- Then local florist asks a delivery person to deliver those flowers to Sugnan.

Object Oriented Thinking

Messages and methods, Responsibilities:

Responsibilities:

A fundamental concept in OOP is to describe behavior in terms of responsibilities. A Request to perform an action denotes the desired result. An object can use any technique that helps in obtaining the desired result and this process will not have any interference from other object. The abstraction level will be increased when a problem is evaluated in terms of responsibilities. The objects will thus become independent from each other which will help in solving complex problems. An Object has a collection of responsibilities related with it which is termed as ‘protocol’

The Operation of a traditional program depends on how it acts on data structures.
Whereas an OOP operates by requesting data structure to perform a service.

Object Oriented Thinking

VISHNU
UNIVERSAL LEARNING

Messages and methods, Responsibilities:

Messages & Methods:

- When a message is passed to an agent (or object) that is capable of performing an action, then that action will be initiated in OOP.
- An object which receives the message sent is called ‘receiver’.
- When a receiver accepts a message, it means that the receiver has accepted the responsibility of processing the requested action.
- It then performs a method as a response to the message in order to fulfil the request.

Object Oriented Thinking

- Classes and Instances

- A Receiver's class determines which method is to be invoked by the object in response to a message.

- When similar messages are requested then all the objects of a class will invoke the same method.

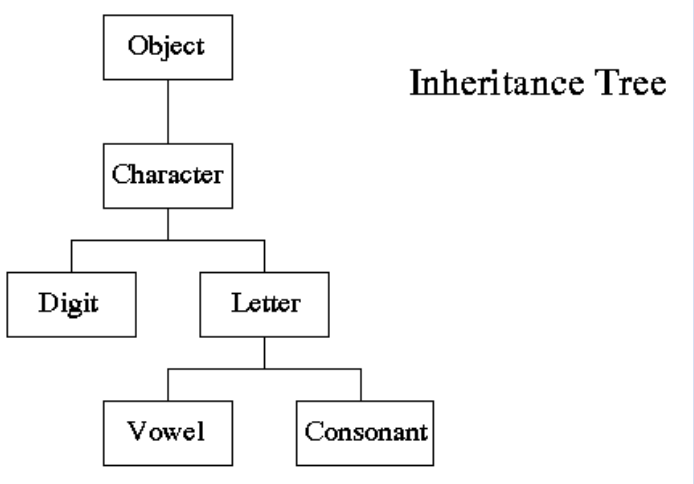
- All objects are said to be the instances of a class.

- For e.g., If 'flower' is a class then Rose is its instance

Object Oriented Thinking

Class Hierarchies- Inheritance

- It is possible to organize classes in the form of a structure that corresponds to hierarchical inheritance.
- All the child classes can inherit the properties of their parent classes.
- A parent class that does not have any direct instances is called an abstract class.
- It is used in the creation of subclasses.



Inheritance Tree

Method binding

When the method is super class have same name that of the method in sub class, then the subclass method overridden the super-class method.

The program will find out a class to which the reference is actually pointing and that class method will be binded.

Object Oriented Thinking

```
class parent {
        void print() {
           System.out.println("From Parent");
} }
class  child extends parent
{
void print() {
   System.out.println("From Child");
} }
class  Bind {
public static void main(String arg[]) {
child ob=new child();
ob.print();
} }
```

o/p: From Child

The child's object 'ob' will point to child class print() method thus that method will be binded.

Object Oriented Thinking

Overriding:

When the name and type of the method in a subclass is same as that of a method in its super class.

Then it is said that the method present in subclass overrides the method present in super class.

Calling an overridden method from a subclass will always point to the type of that method as defined by the subclass, where as the type of method defined by super class is hidden.

E.g. (above ’method binding’ example)

Object Oriented Thinking

VISHNU
UNIVERSAL LEARNING

Exceptions:

Exception is a error condition that occurs in the program execution.

There is an object called ‘Exception’ object that holds error information.

This information includes the type and state of the program when the error occurred.

E.g. Stack overflow, Memory error etc

Object Oriented Thinking

Summary of Object-Oriented concepts.

- Everything is an object
- Computation is performed by objects communicating with each other, requesting that other objects perform actions. Objects communicate by sending & receiving *messages*. A message is a request for an action bundled with whatever arguments may be necessary to complete the task.
- Each object has its own *memory*, which consists of other objects.
- Every Object is an *instance* of class. A class simply represents a grouping of similar objects, such as integers or lists.
- The class is the repository for *behavior* associated with an object. That is all objects that are instances of same class can perform the same actions.
- Classes are organized into a singly rooted tree structure, called *inheritance hierarchy.*

Summary of Object-Oriented concepts

VISHNU
UNIVERSAL LEARNING

Definition of OOP Concepts in Java

OOP concepts in Java are the main ideas behind Java's Object Oriented Programming.

They are an

- **Abstraction**,
- **Encapsulation**,
- **Inheritance** and
- **Polymorphism**.

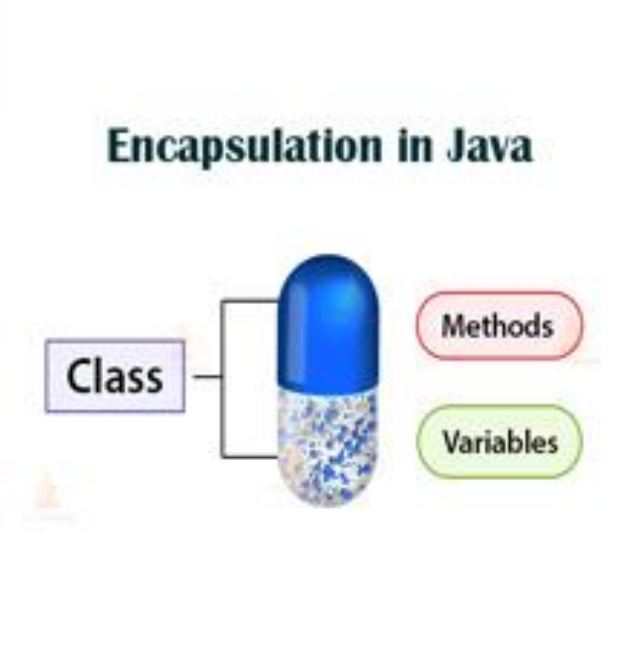
Grasping them is key to understanding how Java works.

Summary of Object-Oriented concepts

Abstraction. Abstraction means using simple things to represent complexity. We all know how to turn the TV on, but we don't need to know how it works in order to enjoy it. In Java, abstraction means simple things like **objects**, **classes**, and **variables** represent more complex underlying code and data. This is important because it lets avoid repeating the same work multiple times.

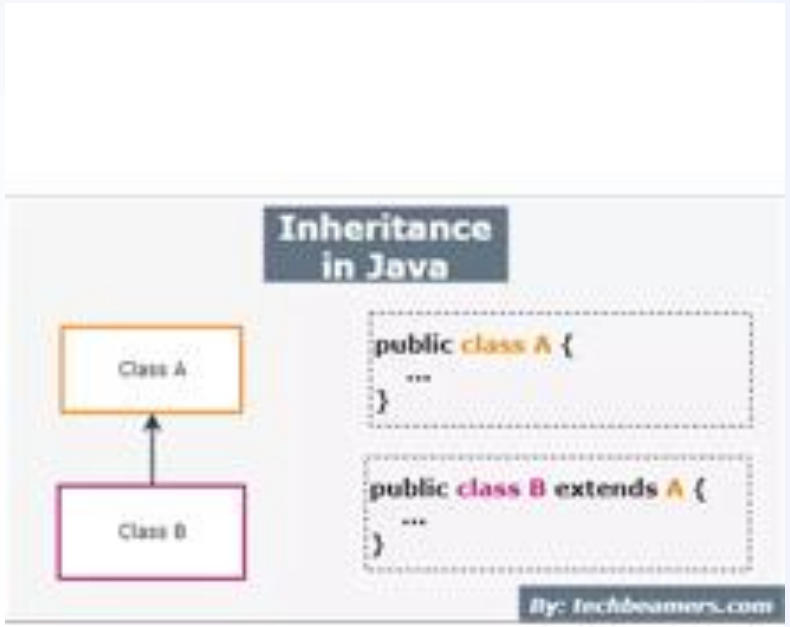
Summary of Object-Oriented concepts

Encapsulation. This is the practice of keeping fields within a class private, then providing access to them via public methods. It's a protective barrier that keeps the data and code safe within the class itself. This way, we can re-use objects like code components or variables without allowing open access to the data system-wide.



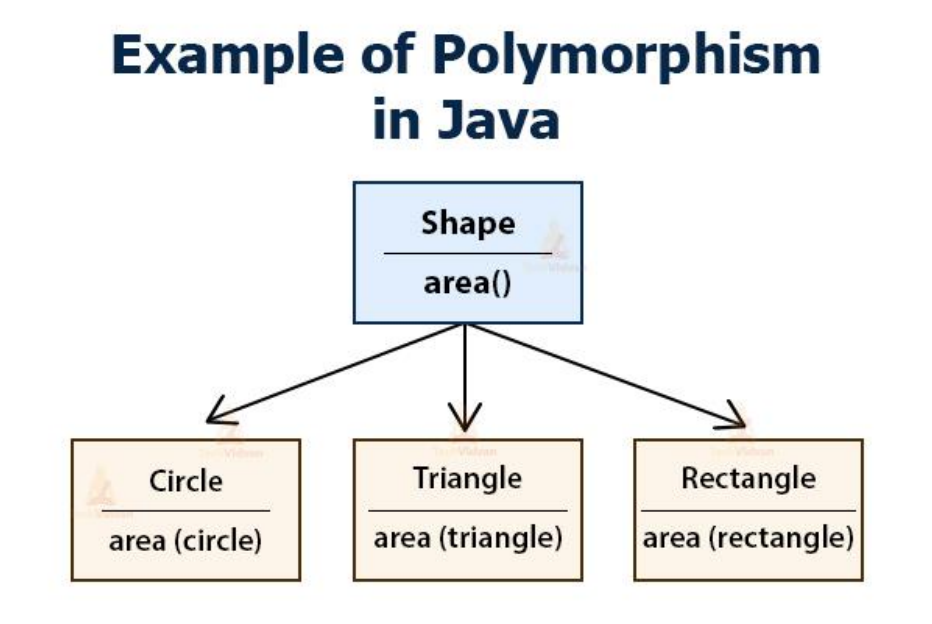
Summary of Object-Oriented concepts

Inheritance. This is a special feature of Object Oriented Programming in Java. It lets programmers create new classes that share some of the attributes of existing classes. This lets us build on previous work without reinventing the wheel.



Summary of Object-Oriented concepts

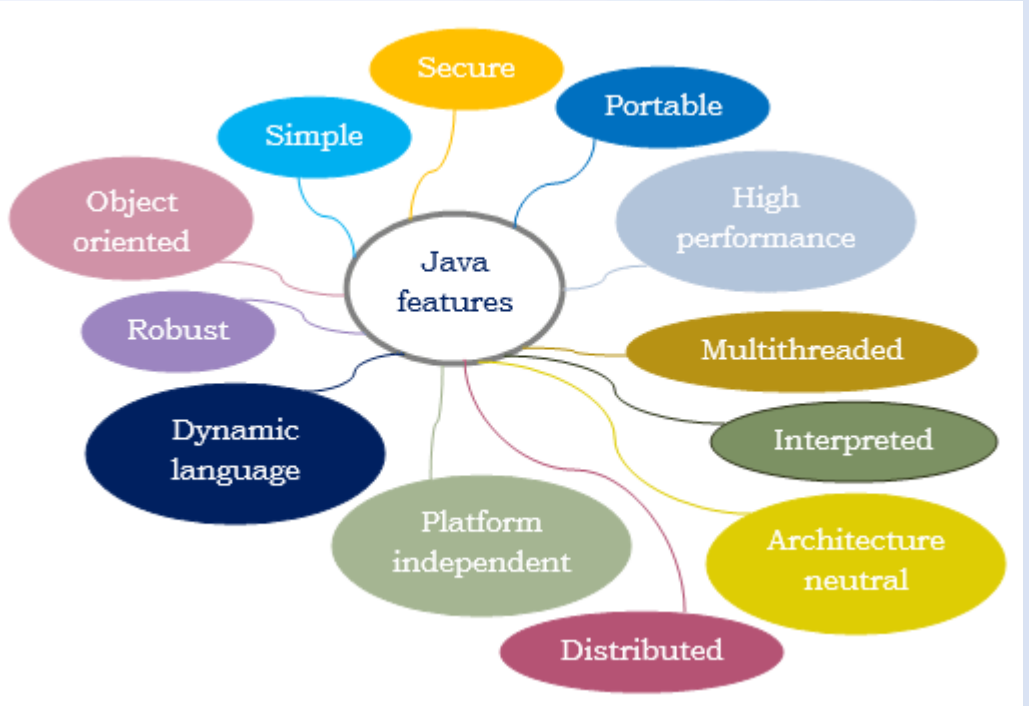
Polymorphism. This Java OOP concept lets programmers use the same word to mean different things in different contexts. One form of polymorphism in Java is **method overloading**. That's when different meanings are implied by the code itself. The other form is **method overriding**. That's when the different meanings are implied by the values of the supplied variables. See more on this below.



Java buzzwords

Apart from being a system independent language, there are other reasons too for the immense popularity of Java language

1. Simple
2. Object Oriented
3. Distributed
4. Robust
5. Secure
6. System independence
7. Portability
8. Interpreted
9. High Performance
10. Multithreaded
11. Scalability
12. Dynamic



Java buzzwords

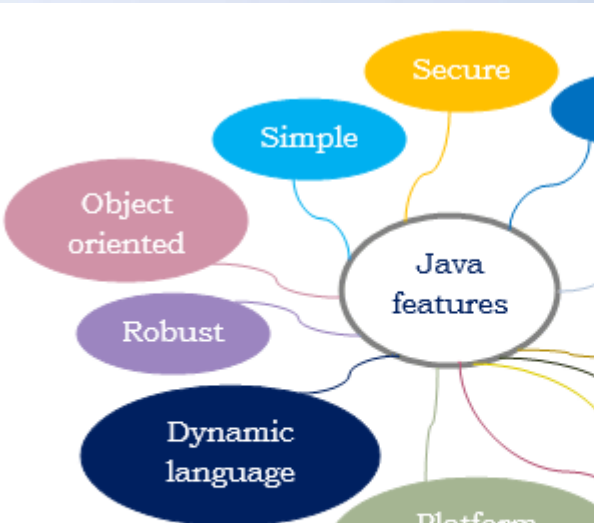
- Simple

Java is a simple programming language. Rather than saying that this is the feature of Java, we can say that this is the design aim of Java.

Now, the question is how Java is made simple? First of all, the difficult concepts of C and C++ have been omitted in Java.

For example, the concept of pointers-which is very difficult for both learners and programmers-has been completely eliminated from Java.

Next, 'JavaSoft (the team who developed Java is called with this name) people maintained the same syntax of C and C++ in Java, so that a programmer who knows C or C++ will find Java already familiar.

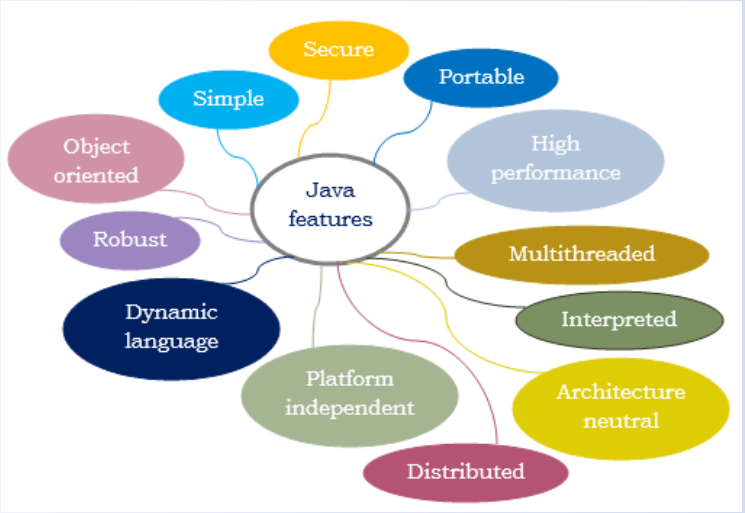


Java buzzwords

Why pointers are eliminated from Java?

1. Pointers lead to confusion for a programmer.
2. Pointers may crash a program easily
3. Pointers break security.

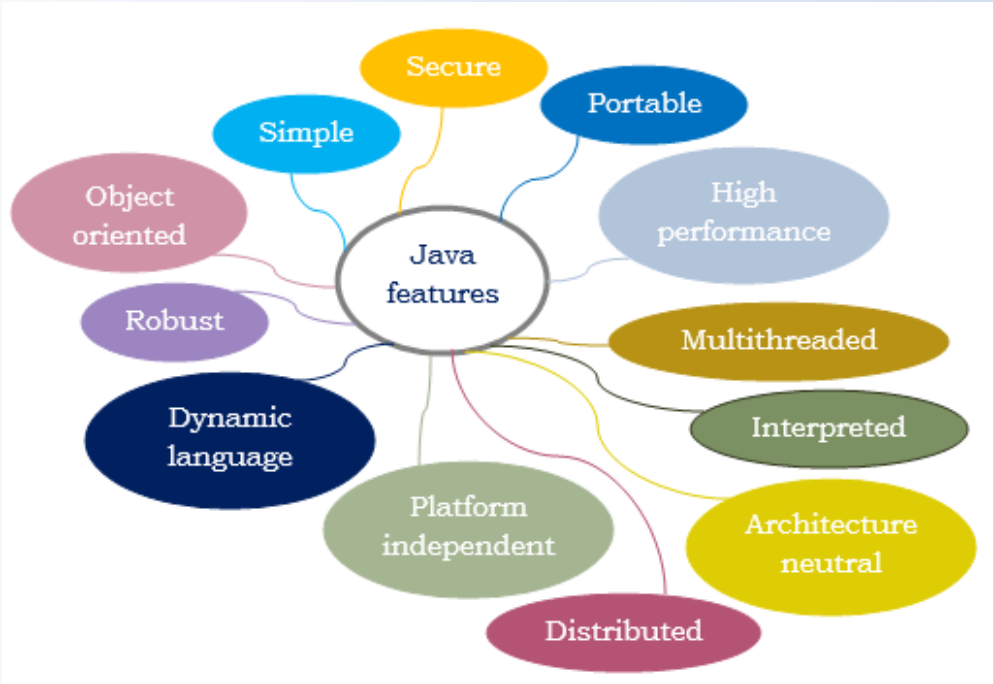
Because of the above reasons, pointers have been eliminated from Java.



Java buzzwords

- Object-oriented:

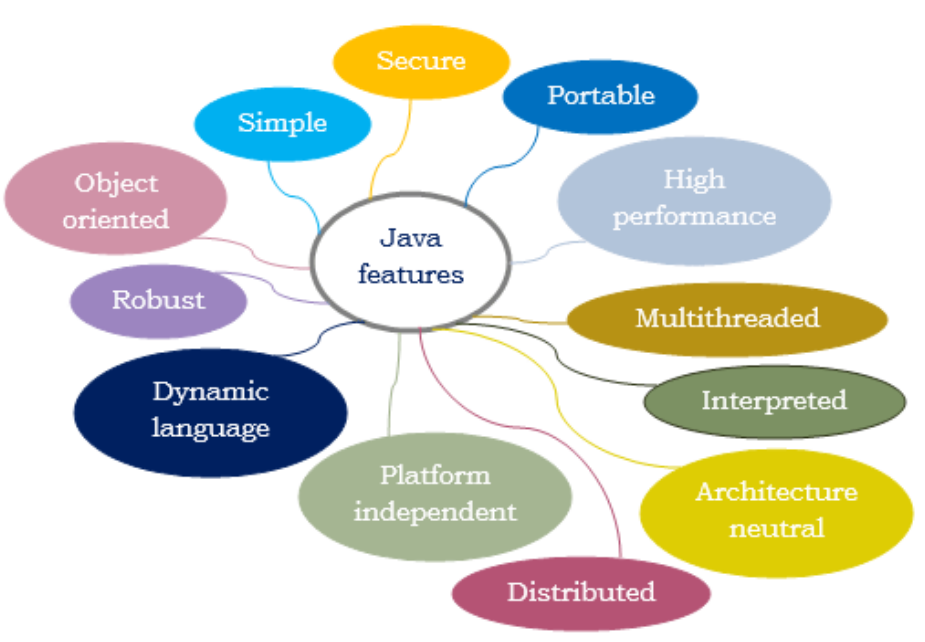
- Java is an object oriented programming language.
- This means Java programs use objects and classes.



Java buzzwords

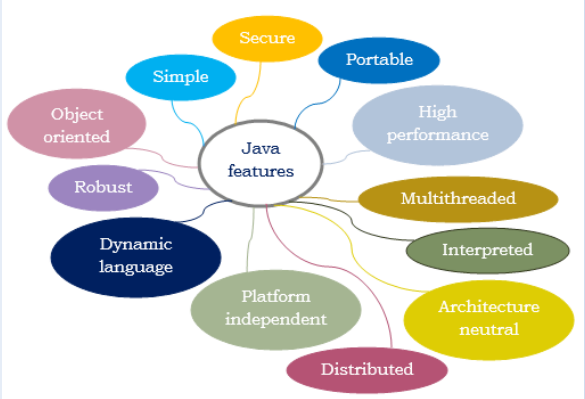
- Distributed:

- Information is distributed on various computers on a network.
- Using Java, we can write programs, which capture information and distribute it to the clients.
- This is possible because Java can handle the protocols like TCP/IP and UDP.



Java buzzwords

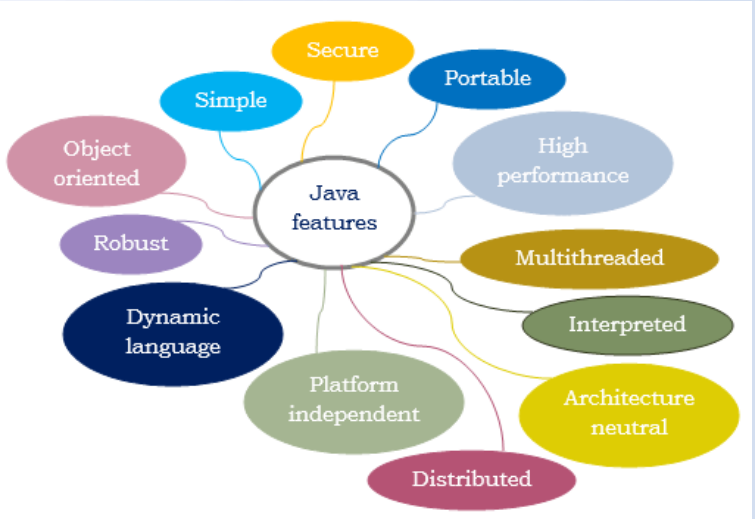
- Secure:
- Java class libraries provide several API that leads to security.
- Security problems like eavesdropping, tampering, impersonation, and virus threats can be eliminated or minimized by using Java on Internet.
- Eavesdropping: This means reading others' data illegally on Internet.
- Tampering: This is another problem on Internet. Not only reading others' data but also modifying it is called tampering.
- Impersonation: A person disguising himself as another person on Internet is called impersonation. Many people hide their original identity and act as somebody else to make transactions. The solution for this problem is digital signature. Digital signature is a file that contains personal identification information in an encrypted format.
- Virus: Virus represents a program that can cause harm to the data, software, and hardware of a computer system.



Java buzzwords

- System independence:

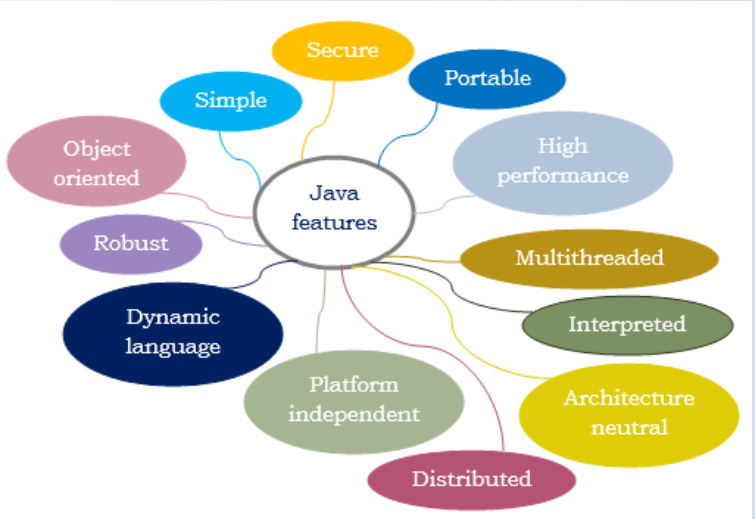
- Java's byte code is not machine dependent.
- It can be run on any machine with any processor and any operating system



Java buzzwords

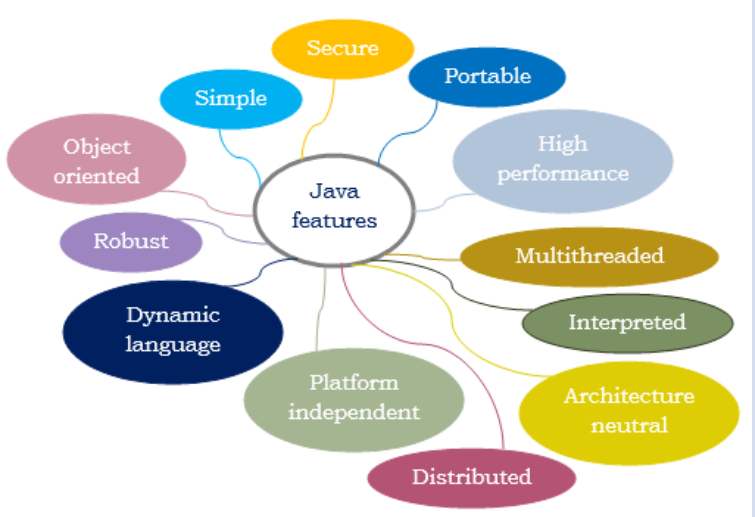
- Portability:

- If a program yields the same result on every machine, then that program is called portable.
- Java programs are portable.
- This is the result of Java's System independence nature.



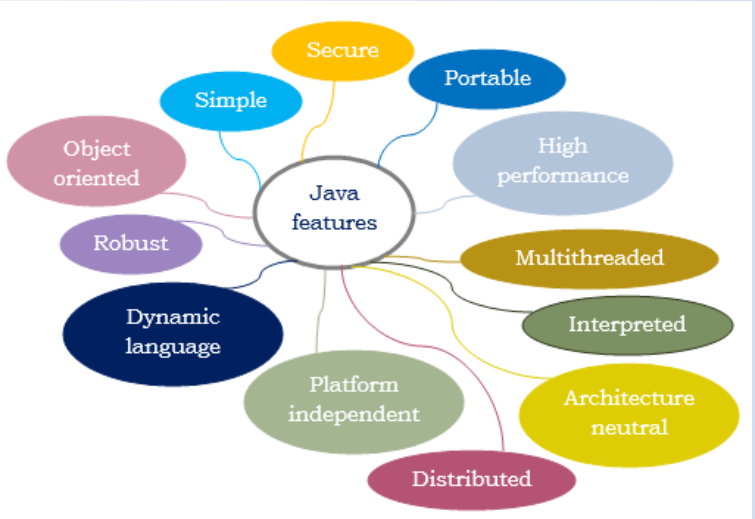
Java buzzwords

- Interpreted:
- Java programs are compiled to generate the byte code.
- This byte code can be downloaded and interpreted by the interpreter in JVM.
- If we take any other language, only an interpreter or a compiler is used to execute the programs. But in Java, we use both compiler and interpreter for the execution.



Java buzzwords

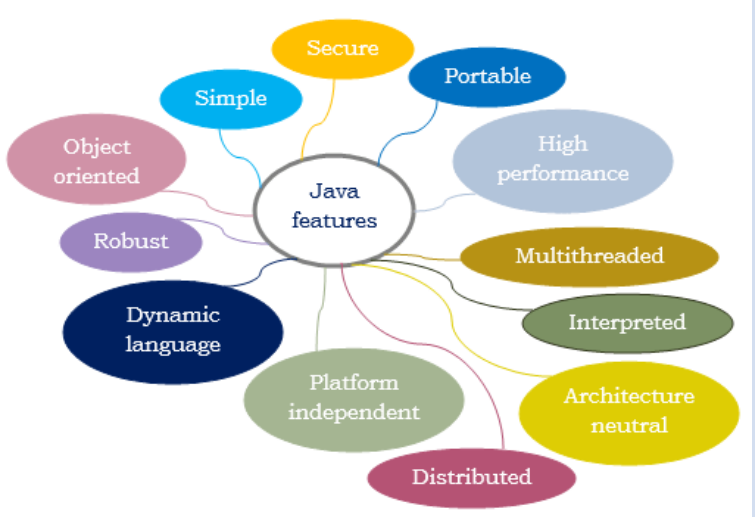
- High Performance:
- The problem with interpreter inside the JVM is that it is slow.
- Because of this, Java programs used to run slow.
- To overcome this problem, along with the interpreter, java has JIT (Just In Time) compiler, which enhances the speed of' execution.
- So now in JVM, both interpreter and JIT compiler work together to run the program.



Java buzzwords

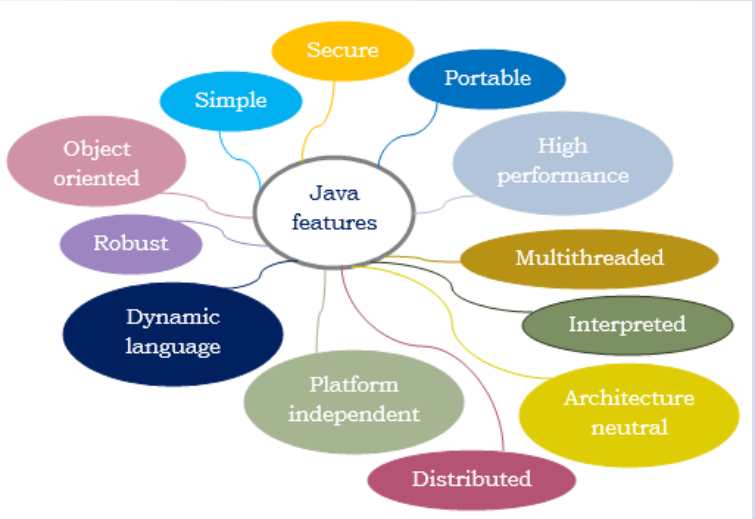
- Multithreaded:

- A thread represents an individual process to execute a group of statements.
- JVM uses several threads to execute different blocks of code.
- Creating multiple threads is called 'multithreaded'.



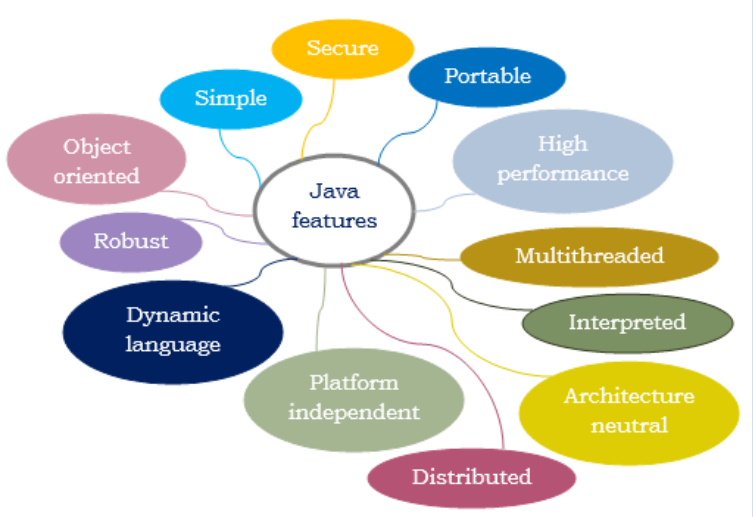
Java buzzwords

- Scalability:
- Java platform can be implemented on a wide range of computers with varying levels of resources from embedded devices to mainframe computers.
- This is possible because Java is compact and platform independent.



Java buzzwords

- Dynamic:
- Before the development of Java, only static text used to be displayed in the browser.
- But when James Gosling demonstrated an animated atomic molecule where the rays are moving and stretching, the viewers were dumbstruck. This animation was done using an applet program, which are the dynamically interacting programs on Internet.



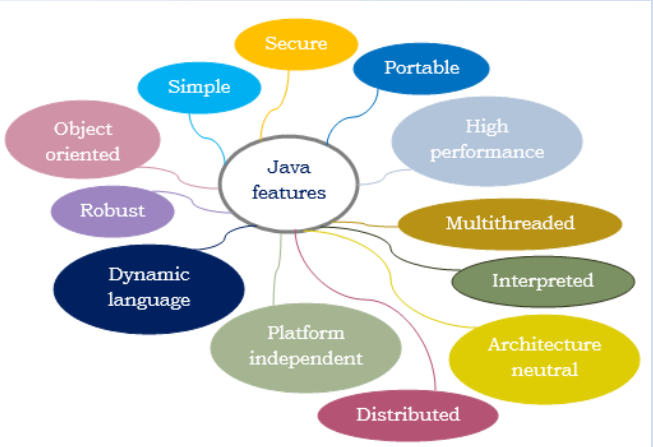
Java buzzwords

Architecture-neutral

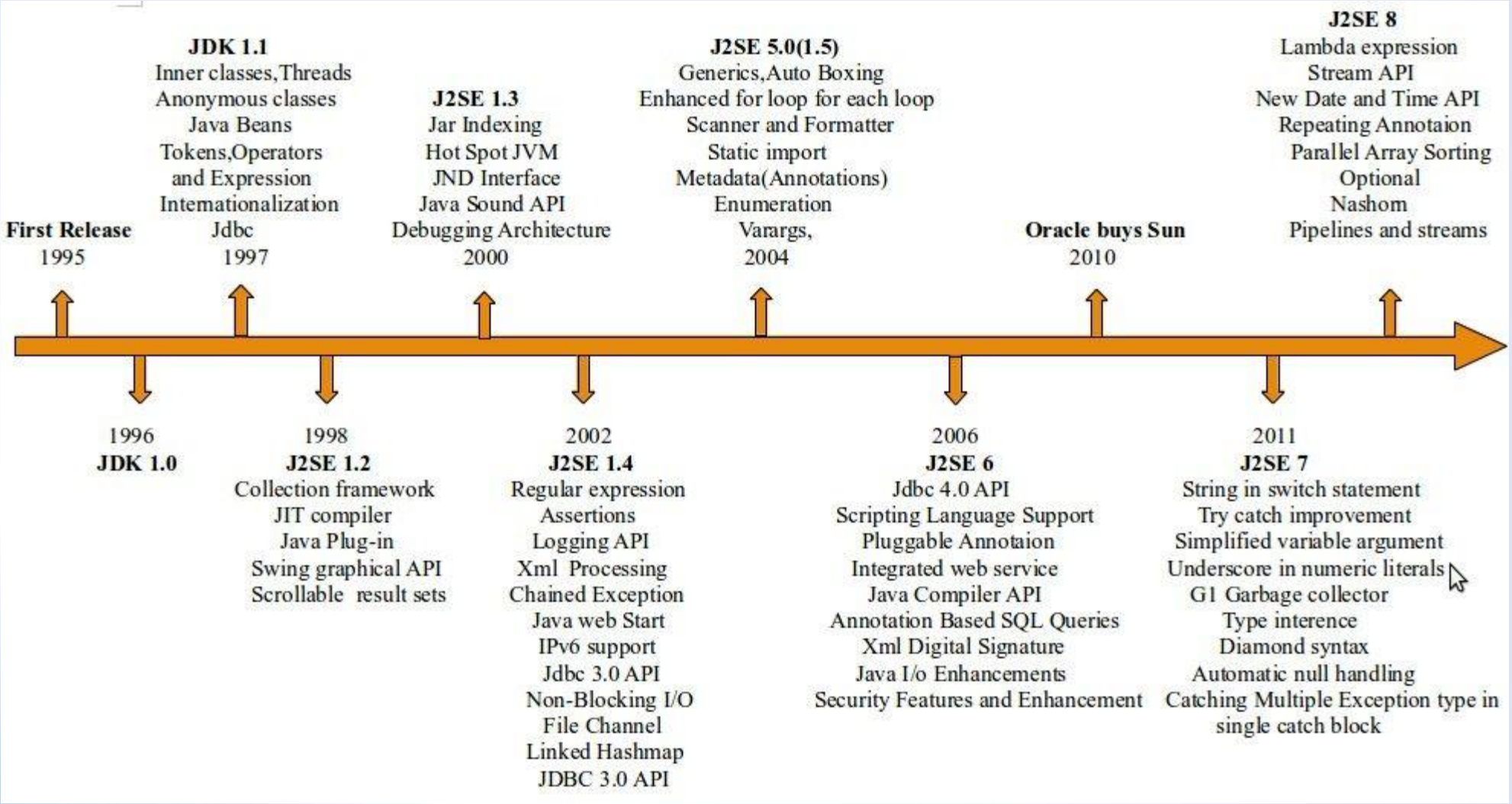
- Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.

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

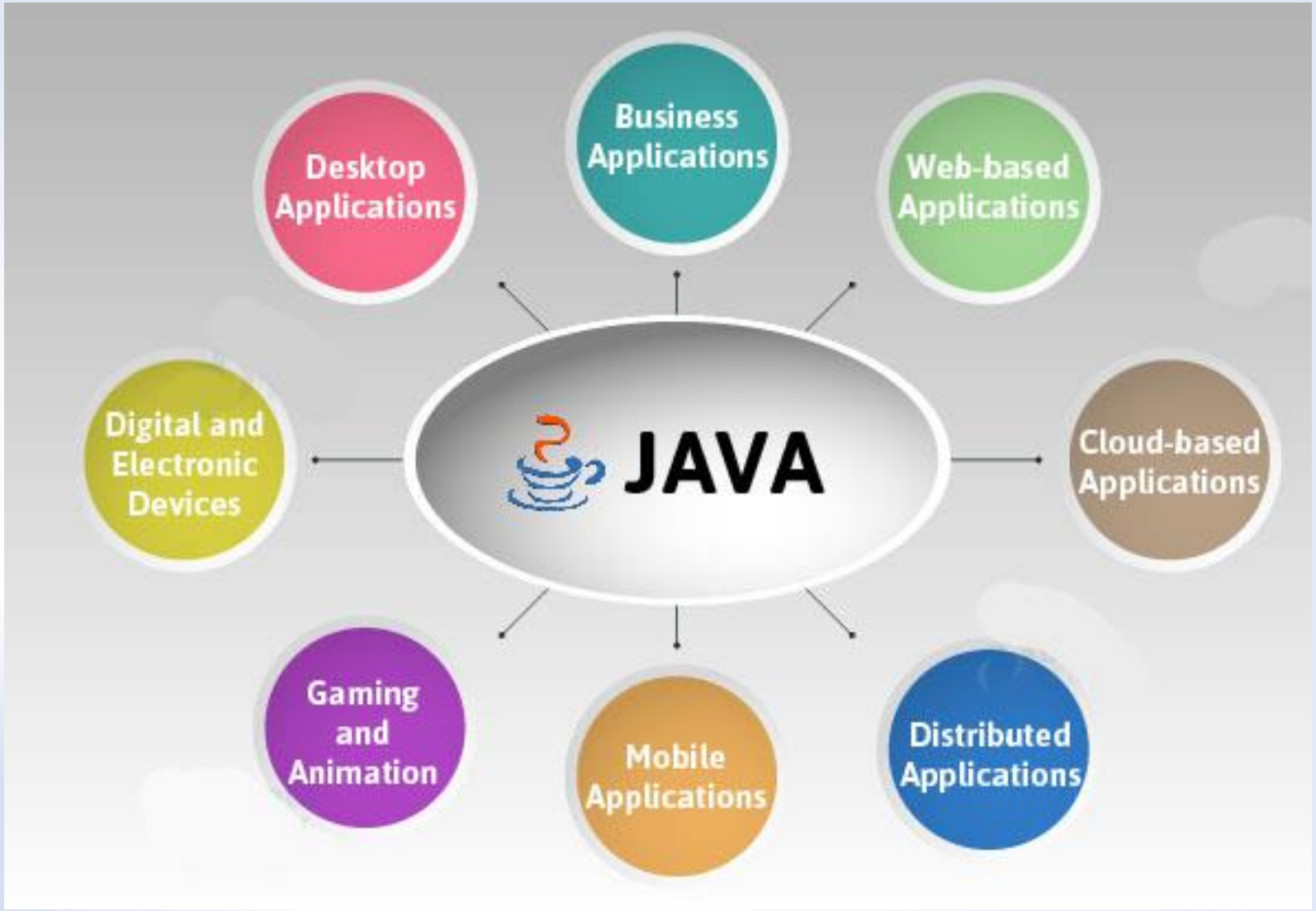
- However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.



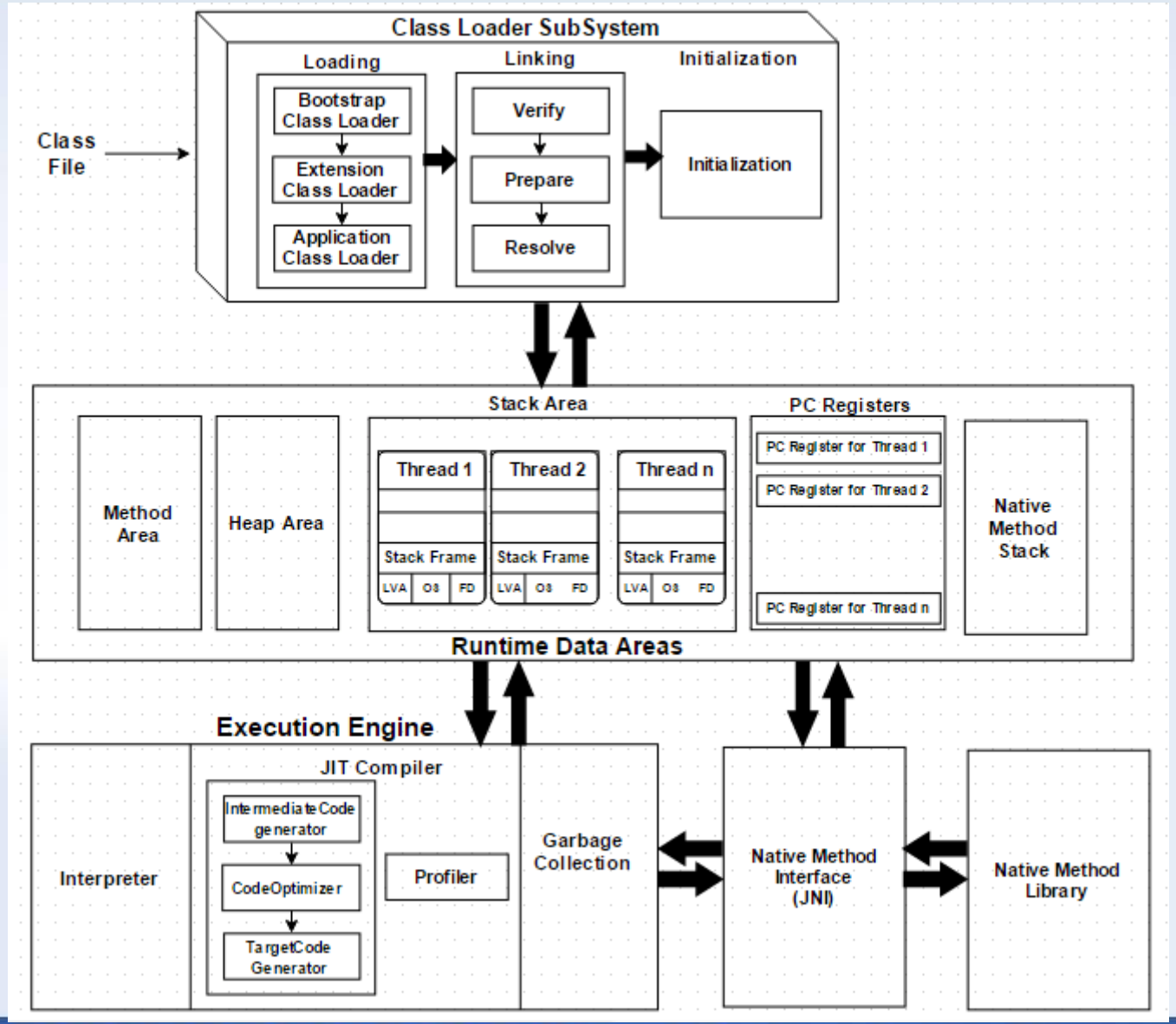
An Overview of Java – History



An Overview of Java – applications



An Overview of Java – JVM Architecture



Next Session

Basics – Naming Conventions, Data types and Variables

An Overview of Java – Difference between c++ and Java

C++	Java
C++ is not a purely object-oriented programming language, since it is possible to write C++ programs without using a class or an object.	Java is purely an object-oriented programming language, since it is not possible to write a Java program using atleast one class.
Pointers are available in C++.	We cannot create and use pointers in Java.
Allotting memory and deallocating memory is the responsibility of the programmer	Allocation and deallocation of memory will be taken care of by JVM.
C++ has goto statement.	Java does not have goto statement.
Automatic casting is available in C++.	In some cases, implicit casting is available. But it is advisable that the programmer should use casting wherever required.
Multiple Inheritance feature is available in C++.	No Multiple Inheritance in Java, but there are means to achieve it.
Operator overloading is available in C++.	It is not available in Java.
#define, typedef and header files are available in C++.	#define, typedef and header are not available in Java, but there are means to achieve them.
There are 3 access specifiers in C++: private, public, and protected.	Java supports 4 access specifiers: private, public, protected, and default.
There are constructors and destructors in C++.	Only constructors are there in Java. No destructors are available in this language.

An Overview of Java - Comments

Comments :

- What are comments?

- There are three types of comments in Java
  - Single line comments: start with double slash symbol / /
  - Multi-line comment: start with / * and end with */
  - Java documentation comments: start with / ** and end with */

An Overview of Java - Naming Conventions

VISHNU
UNIVERSAL LEARNING

Naming Conventions :

- What are Naming Conventions?

- Since Java is a case sensitive programming Language, it recognizes capital and small letters as different.
- But how to know where to use which case-upper or lower?
- For this purpose, certain conventions (rules) are followed while naming
    - variables,
    - classes,
    - methods etc

An Overview of Java - Naming Conventions

VISHNU
UNIVERSAL LEARNING

Naming Conventions in Java:

- What are Naming Conventions?

- Naming conventions specify the rules to be followed by a Java programmer while writing the names packages, classes, methods, etc

- A package represents a sub directory that contains a group of classes and interfaces.
- Names of packages in Java are written in small letters as:
  - java.awt
  - java.io
  - javax.swing

An Overview of Java - Naming Conventions

- A class is a model for creating objects.
- A class specifies the properties and actions of objects.
- An interface is also similar to a class.
- Each word of class names and interface names start with a capital letter as:

    - String
    - DataInputStream
    - ActionListener

An Overview of Java - Naming Conventions

- A class and an interface contain methods and variables.
- The first word of a method name is in small letters; then from second word onwards, each new word starts with a capital letter as shown here:

  - printtln()
  - readLine()
  - getNumberInstance()

An Overview of Java - Naming Conventions

- The naming convention for variables names is same as that for methods as given here:
  - age
  - employteeName
  - Employtee_Net_Sal

What is the difference between variable and method ?

An Overview of Java - Naming Conventions

- Constants represent fixed values that cannot be altered. For example, PI is a constant whose value is 22/7 or 3 . 14159, which is fixed.
- Such constants should be written by using all capital letters as shown here:
    - PI
    - MAX_VALUE
    - Font.BOLD

An Overview of Java - Naming Conventions

- All keywords should be written by using all small letters as follows:

- public
- void
- static

An Overview of Java – Data Types and Variables

An Overview of Java – Data Types and Variables

VISHNU
UNIVERSAL LEARNING

- **Data Types in Java**

Data types specify the different sizes and values that can be stored in the variable.
There are two types of data types in Java:

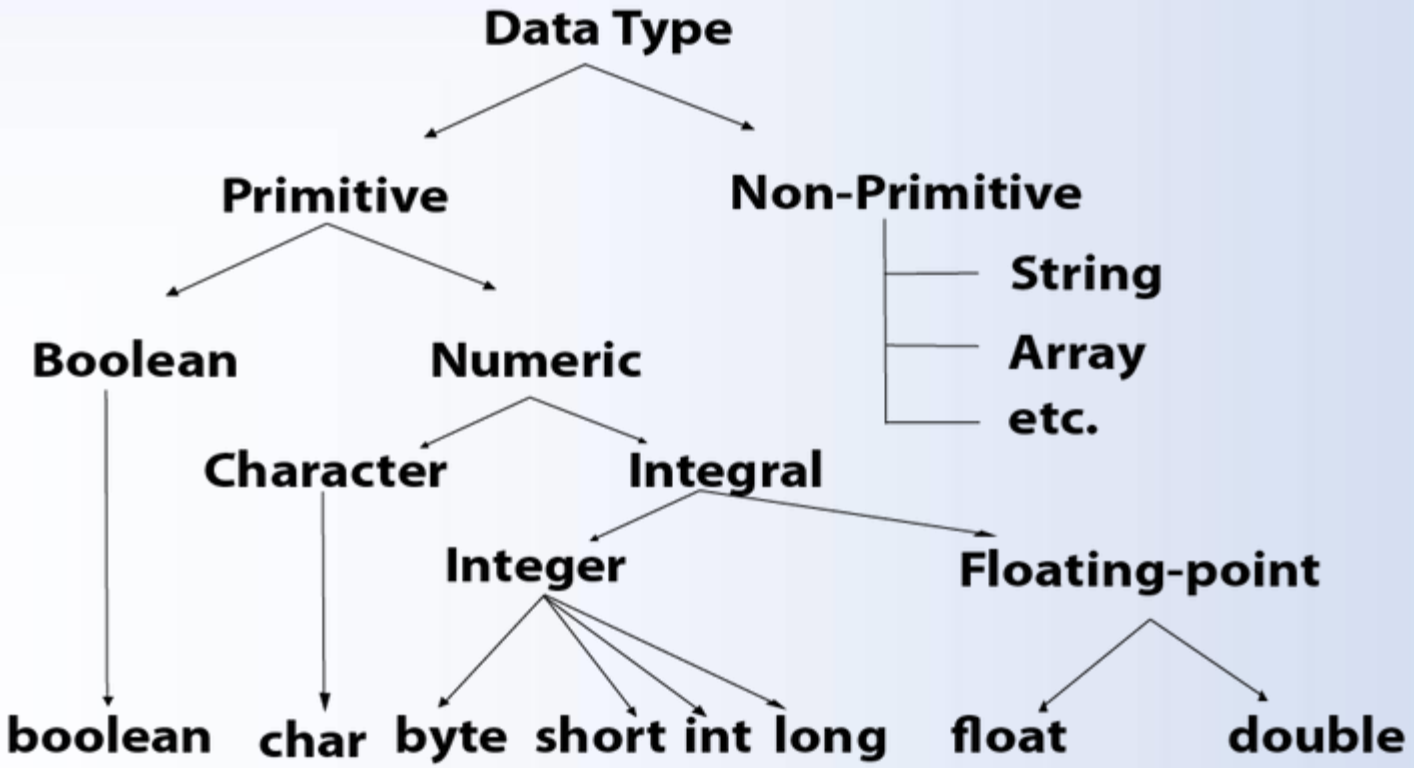
Primitive data types:
The primitive data types include boolean, char, byte, short, int, long, float and double.

Non-primitive data types: The non-primitive data types include Classes, Interfaces, and Arrays.

Primitive data types:

There are 8 types of primitive data types:

1. boolean
2. byte
3. char
4. short
5. int
6. long
7. float
8. double



An Overview of Java – Data Types and Variables

VISHNU
UNIVERSAL LEARNING

Primitive data types:

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

An Overview of Java – Variables

VISHNU
UNIVERSAL LEARNING

Variables in Java:

- A variable is a container which holds the value while the Java program is executed.
- A variable is assigned with a data type.
- Variable is a name of memory location.
- There are three types of variables in java:
  - local,
  - instance and
  - static.

An Overview of Java – Variables

VISHNU
UNIVERSAL LEARNING

1) Local Variable

A variable declared inside the body of the method is called local variable.
You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.
A local variable cannot be defined with "static" keyword.

2) Instance Variable

A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static.
It is called instance variable because its value is instance specific and is not shared among instances.

3) Static variable

A variable which is declared as static is called static variable.
It cannot be local. You can create a single copy of static variable and share among all the instances of the class.
Memory allocation for static variable happens only once when the class is loaded in the memory.

An Overview of Java – Variables

Example to understand the types of variables in java

```
class A {
int data=50; //instance variable
static int m=100; //static variable
void method() {
      int n=90; //local variable
      }
}//end of class
```

Next Session

Operators and Arrays

An Overview of Java – first java program

What is the difference between #include and import statement?

#include directive makes the compiler go to the C/C++ standard library and copy the code from the header files into the program. As a result, the program size increases, thus wasting memory and processor's time.

import statement makes the JVM go to the Java standard library, execute the code there, and substitute the result into the program. Here, no code is copied and hence no waste of memory or processor's time.

So, import is an efficient mechanism than #include.

An Overview of Java – first java program

After importing the classes into the program, the next step is to write a class, Since Java is purely an object-oriented programming language, we cannot write a Java program without having at least one class or object.

So, it is mandatory that every Java program should have at least one class in it,

How to write a class?

We should use class keyword for this purpose and then write the class name.

```
class First {
Statements;
}
```

An Overview of Java – first java program

- A class code starts with a { and ends with a }.
- We know that a class or an object contains variables and methods (functions), So we can create any number of variables and methods inside the class.
- This is our first program, so we will create only one method, i.e. compulsory, in it - main() method.
- Why should we write main {} method?
- Because, if main () method is not Written in a Java program, JVM will not execute it.
- main (} is the starting point for JVM to start execution of a Java program

An Overview of Java – first java program

public static void main (String args[])

Next to main () method's name, we have written String args [].

Before discussing the main () method, let us first see how a method works, A method, generally, performs two functions.

- It can accept some data from outside
- It can also return some result

main () method also accepts some data from us, For example, it accepts a group of strings, which is also called a string type array, This array is String args [] , which is written along with the main () method as:

public static void main(String args[])

An Overview of Java – first java program

public static void main (String args[])

- Here args [] is the array name and it is of String type.
- This means that it can store a group of strings, Remember, this array can also store a group of numbers but in the form of strings only.

- The values passed to main () method are called arguments.

- These arguments are stored into args [] array, so the name args [] is generally used for it.

An Overview of Java – first java program

A method is executed only when it is called, But, how to call a method? Methods are called using 2 steps in Java

1. Create an object to the class to which the method belongs. The syntax of creating the object is:

```
Classname objectname = new Classname();
```

2. Then the method should be called using the objectname .methodname () .

Since main () method exists in the class First; to call main () method, we should first of all create an object to First class, something like this:

```
First obj = new First();
```

Then call the main () method as:

```
obj.main();
```

An Overview of Java – first java program

So, if we write the statement
First obj = new First();
Inside the main method, then the JVM will execute this and create the object

```
class First {
public static void main(String args[]) {
First obj = new First(); //object is created hereafter JVM Executes this
}
}
```

An Overview of Java – first java program

By looking at the code, we can understand that an object could be created only after calling the main () method.
But for calling the main() method, first of all we require an object.
Now, how is it possible to create an object before calling the main () method?
So, we should call the main() method without creating an object. Such methods are called static methods and should be declared as static.

Static methods are the methods, which can be called and executed without creating the objects.
Since we want to call main () method without using an object, we should declare main () method as static. Then, how is the main () method called and executed? The answer is by using the classname.methodname().
JVM calls main () method using its class name as First.main () at the time of running the program

An Overview of Java – first java program

- JVM is a program written by JavaSoft people (Java development team) and main () is the method written by us.
- Since, main () method should be available to the JVM, it should be declared as public.
- If we don't declare main () method as public, then. it doesn't make itself available to JVM and JVM cannot execute it
- So, the main () method should always be written as shown here:

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

An Overview of Java – first java program

- What happens if String args[] is not written in main method?

- When main() method is written without String args[] as:

```
public static void main ()
```

- The code will compile but JVM cannot run the code because it cannot recognize the main () method as the method from where it should start execution of the Java program.
- Remember JVM always looks for main () method with string type array as parameter.

An Overview of Java – first java program

- So, finally java program will look like this

```
class First {
pubic static void main (String args[]) {
Statements;
}
}
```

Our aim of writing this program is just to display a string "Welcome to Java".
In Java, print () method is used to display something on the monitor. So, we can write it as:
print (“Welcome to java”);

An Overview of Java – first java program

- But this is not the correct way of calling a method in Java. A method should be called by using objectname.methodname ().
- So, to call print () method, we should create an object to the class to which print () method belongs.
- print () method belongs to PrintStream class. So, we should call print () method by creating an object to PrintStream class as:

PrinteStream obj.print("Welcome to java");

But as it is not possible to create the object to PrintStream class directly, an alternative is given to us, i.e. System.out.

Here, System is the class name and out is a static variable in System class. out is called afield in System class. When we call this field, a PrintStream class object will be created internally. So, we can call the print () method as shown below:

System.out.print("Welcome to java");

An Overview of Java – first java program

System.out.print(“Welcome to java”);

System.out gives the PrintStream class object. This object, by default, represents the standard output device, i.e. the monitor.

So, the string "Welcome to Java" will be sent to the monitor.

Now, let us see the final version of our Java program:

```
class First {
pubic static void main (String args[]) {
System.out.println(“Welcome to java”);
}
}
```

Output:

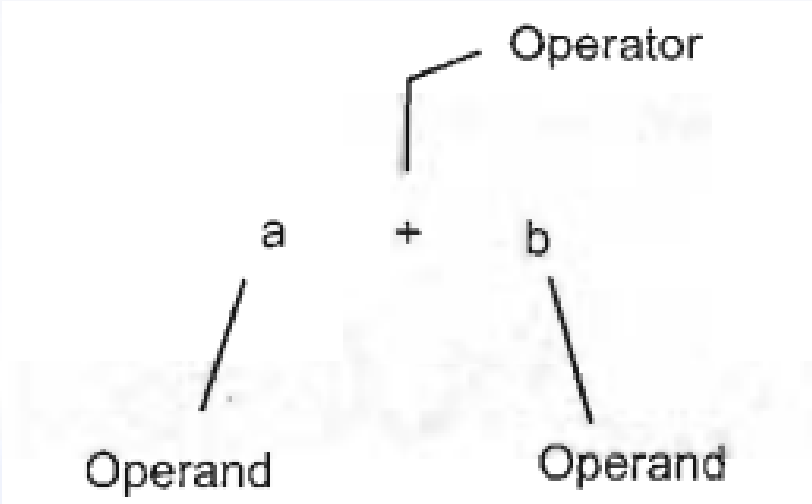
```
C:\> javac First.java
C:\>java First
Welcome to java
```

An Overview of Java – Operators

Operators in Java

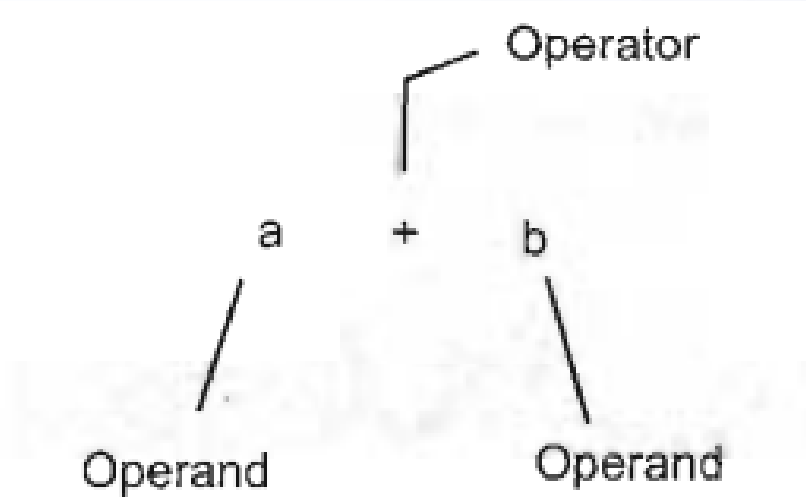
Operators

- An operator is a symbol that performs an operation.
- An operator acts on some variables, caped operands to get the desired result



Operators

- An operator is a symbol that performs an operation.
- An operator acts on some variables, caped operands to get the desired result
- If an operator acts on a single variable, it is called unary operator; if it acts on two variables, it is called binary operator; and if it acts on three variables, then it is called ternary operator.



Operators

- Arithmetic Operators
  - Unary Operator
  - Unary Minus Operator (-)
  - Increment Operator (++)
  - Decrement Operator (--)
  - Assignment Operator
- Relational Operators
- Logical Operators
- Boolean Operators
- Bitwise Operators
- Ternary or Conditional Operator
- Member Operator
- Instanceof Operator
- New Operator
- Cast Operator
- Priority of Operators

Operators

- Arithmetic operators
- These operators are used to perform fundamental arithmetic operations like addition, subtraction, etc.
- There are 5 arithmetic operators in Java. Since these operators act on two operands at a time, these are called binary operators.
- Table 5.1 displays the functioning of these operators. Here, we are assuming the value of a as 13 and b as 5.

- Addition operator (+) is also used to join two strings, as shown in the following code snippet:
  - String s1= "wel";
  - String s2= "come";
  - String s3= s1+s2; //here, '+' is joining s1 and s2.
- Now, we get welcome in s3.
- In this case, + is called String concatenation operator

Operator	Meaning	Example	Result
+	Addition operator	a + b	18
-	Subtraction operator	a – b	8
*	Multiplication operator	a * b	65
/	Division operator	a / b	2.6
%	Modulus operator (This gives the remainder of division)	a % b	3

Operators

- Unary operators
- As the name indicates, unary operators act on only one operand.
- There are 3 kinds of unary operators:
  - Unary minus operator (-)
  - Increment operator (++)
  - Decrement operator (--)

- Unary Minus Operator (-) This operator is used to negate a given value.
- Negation means converting a negative value into positive and vice versa, for example:
- Int x =5;
- System.out.println(x); // will display -5;
- System.out.println(-x); // will display 5;
- In this code snippet, the unary minus (-) operator is used on variable x to negate its value. The value of x is 5 in the beginning. It became -5 when unary minus is applied on it.

Operators

- Increment Operator (++)

- This operator increases the value of a variable by 1, for example:
- int x = 1;
- ++x; will make x = 2
- x++ now x =3;
- Here, the value of the variable x is incremented by 1 when ++ operator is used before or after it. Both are valid expressions in Java. Let us take an example to understand it better:
- x = x+1;
- In this statement, if x is 3, then x+1value will be 4.
- This value is stored again in the left hand side variable x.

- So, the value of x now becomes 4. The same thing is done by ++ operator also.
- Writing ++ before a variable is called pre incrementation and writing ++ after a variable is called post incrementation.
- In pre incrementation; incrementation is done first and any other operation is done next.
- In post incrementation, all the other operations are done first and incrementation is done only at the end.
- To understand the difference between pre and post incrementations, practice with a program

Operators

- Decrement Operator (--)

- This operator is used to decrement the value of a variable by 1

- int x = 1;
- --x; will make x = 0
- X-- now x =-1;
- This means, the value of x is decremented every time we use operator on it.
- This is same as writing x= x-1.
- Writing -- before a variable is called pre-decrementation and writing -- after a variable is called post-decrementation.
- Like the incrementation operator, here also the same rules apply.
- Predecrementation is done immediately then and there itself and Post-decrementation is done after all the other operations are carried out.

Operators

- Assignment operator (=)
- This operator is used to store some value into a variable.
- It is used in 3 ways:
  - It is used to store a value into a variable, for example int x = 5;
  - It is used to store the value of a variable into another variable, for example:
  - Int x = y;
  - It is used to store the value of an expression into a variable, for example:
  - Int x = y+z-4;

Operators

- Relational operators
- These operators are used for the purpose of comparing. For example, to know which one is bigger or whether two quantities are equal or not.
- Relational operators are of 6 types:
  - > greater than operator
  - >= greater than or equal to
  - < less than operator
  - <= less than or equal to s
  - == equal to operator
  - ! = not equal to operator

Operators

- Logical operators
- Logical operators are used to construct compound conditions. A compound condition is a combination of several simple conditions.
- Logical operators are of three types:

    - && and operator
    - || or operator
    - ! not operator

- If (a == 1 || b == 1 || c == 1) System.out.println("Yes");
- Here, there are 3 conditions: a==1, b==1,and c==1, which are combined || (or operator). In this case, if either of the a or b or c value becomes equal to 1, Yes will be displayed.
- If (x >y && y<z) System.out.println(" Hello");
- In the preceding statement, there are 2 conditions: x>y and y<z. Since they are combined by using && (and operator); if both the conditions are true, then only Hello is displayed.
- If(!(str1.equals(str2)) System.out.println("not equal");
- We are assuming that str1 and str2 are two string objects, which are being compared. See the ! (not operator) and the equals () methods in the earlier condition telling that if str1 is not equal to str2, then only Not equal will be displayed.

Operators

- boolean operators
- These operators act on boolean variables and produce boolean type result.
- The following 3 are boolean operators:
    - & boolean and operator
    - I boolean or operator
    - ! boolean not operator

- Boolean & operator returns true if both the variables are true.
- Boolean I operator returns true if any one of the variables is true.
- Boolean ! operator converts true to false and vice versa.
    - boolean' a, b; //declare two boolean type variables
    - a= true; //store boolean value true into a
    - b= false; //store boolean value false into b

Operators

- Bitwise operators
- There are 7 types of bitwise operators. Let us now discuss them one by one
- Bitwise Complement Operator (~)
- This operator gives the complement form of a given number.
- This operator symbol is ~, which is pronounced as tilde.
- Complement form of a positive number can be obtained by changing 0's as 1's and vice versa
- Example:
- If int x = 10. find the ~x value

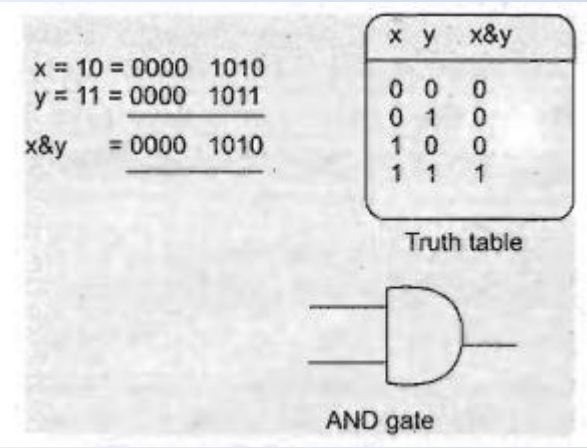
- X = 10 = 0000 1010
- By changing 0's as 1's and vice versa, we get 1111 0101.
- This is nothing but -11 (in decimal). So, ~x = -11.

Operators

- Bitwise and Operator (&)
- This operator performs and operation on the individual bits of the numbers.
- The symbol for this operator is &, which is called ampersand.
- To understand the and operation, see the truth table given in Figure

- Truth table is a table that gives relationship between the inputs and the output. From the table, we can conclude that by multiplying the input bits, we can get the output bit.
- The AND gate circuit present in the computer chip will perform the and operation.

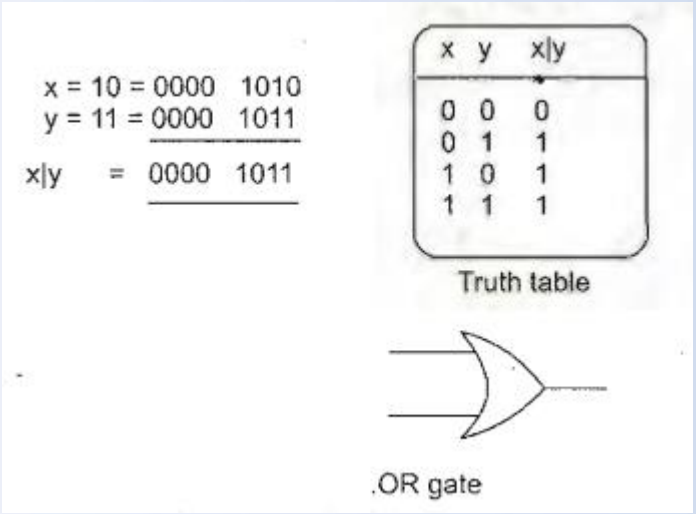
- Example:
- If int x = 10 , y =11 . Find the value of x&y
- X = 10 = 0000 1010
- Y = 11 = 0000 1011
- From the truth table, by multiplying the bits,
- we can get x&y =0000 1010. this is nothing but 10 (in decimal).



Operators

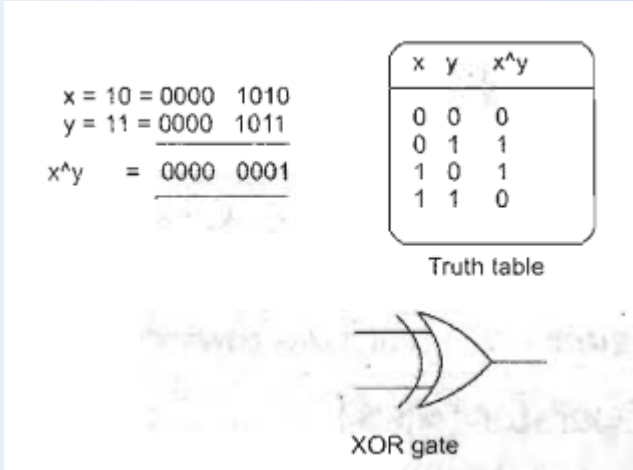
- Bitwise or Operator (|)
- This operator performs or operation on the bits of the numbers.
- The symbol is |, which is called pipe symbol.
- To understand this operation, see the truth table given in Figure. From the table, we can conclude that by adding the input bits, we can get the output bit.
- The OR gate circuit, which is present in the computer chip will perform the or operation:

- Example:
- If int x = 10 , y =11 . Find the value of x|y
- X = 10 = 0000 1010
- Y = 11 = 0000 1011
- From the truth table, by adding the bits,
- we can get x|y =0000 1011. this is nothing but 11 (in decimal).



Operators

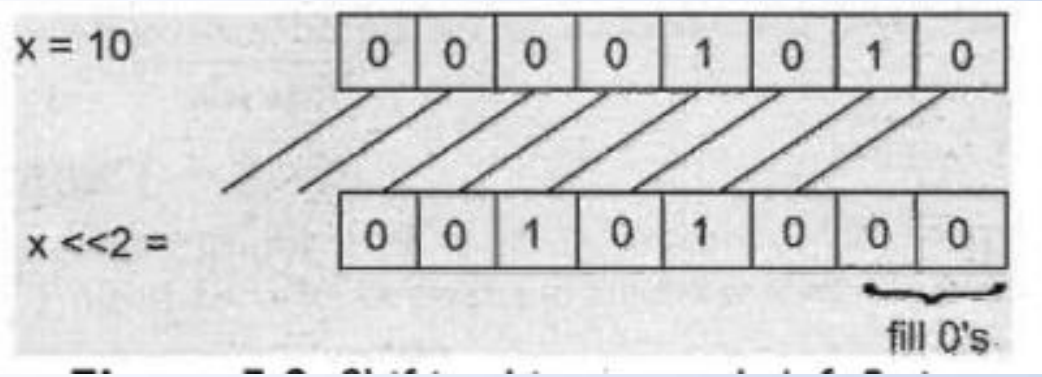
- Bitwise xor Operator (^)
- This operator performs exclusive or (xor) operation on the bits of the numbers.
- The symbol is ^, which is called cap, carat, or circumflex symbol.
- To understand the xor operation, see the truth table given in Figure.
- From the table, we can conclude that when we have odd number of 1's in the input bits, we can get the output bit as 1. The XOR gate circuit of the computer chip will perform this operation.
- Example:
- If int x = 10 , y =11 . Find the value of x^y
- X = 10 = 0000 1010
- Y = 11 = 0000 1011
- From the truth table, when odd number of 1's are there,
- we can get a 1 in the output.
- Thus, x^y = 0000 0001 is nothing but 1 (in decimal).



Operators

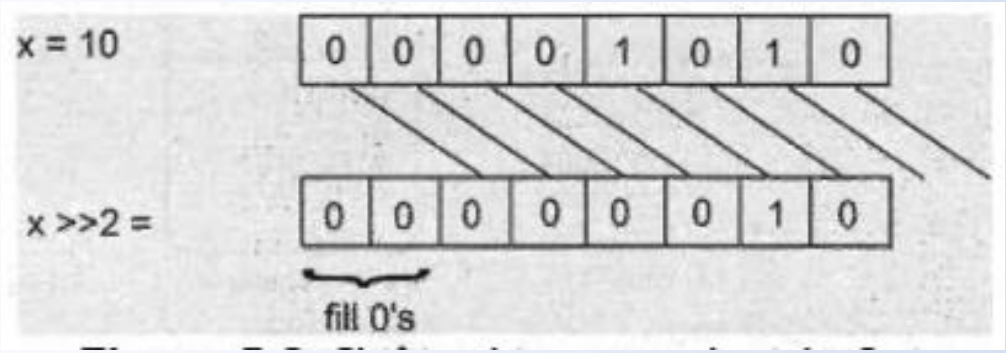
- Bitwise leftshift Operator (<<)
- This operator shifts the bits of the number towards left a specified number of positions.
- The symbol for this operator is <<, read as double less than.
- If we write x<<n, the meaning is to shift the bits of x towards left n positions

- Example:
- If int x = 10 , calculate x value if we write x<<2
- Shifting the value of x towards left 2 positions will make the leftmost 2 bits to be lost.
- The value of x is 10 = 0000 1010.
- Now x<<2 will be 0010 1000 = 40 (in decimal).
- The procedure to do this is explained in Figure



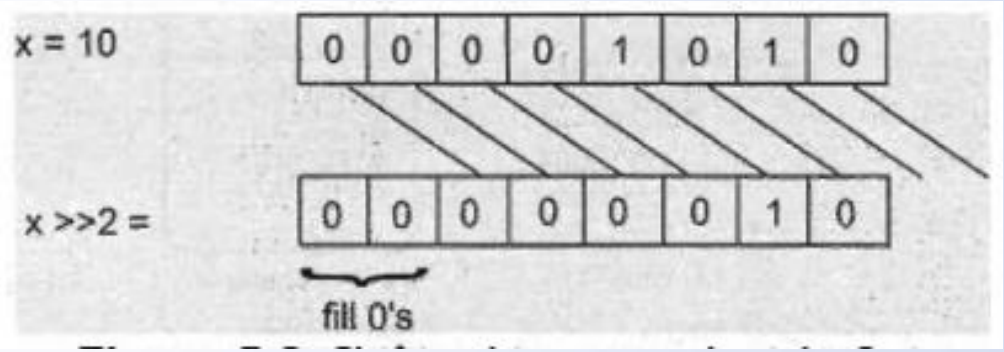
Operators

- Bitwise rightshift Operator (>>)
- This operator shifts the bits of the number towards right a specified number of positions. The symbol for this operator is >>, read as double greater than.
- If we write x>>n, the meaning is to shift the bits of x towards right n positions.
- >> shifts the bits towards right and also preserves the sign bit, which is the leftmost bit. Sign bit represents the sign of the number. Sign bit 0 represents a positive number and 1 represents a negative number. So, after performing >> operation on a positive number, we get a positive value in the result also.
- If right shifting is done on a negative number, again we get a negative value only.
- Example:
- If int x = 10 , then calculate x>>2 value
- Shifting the value of x towards right 2 positions will make the rightmost 2 bits to be lost.
- x value is 10 0000 1010.
- Now x>>2 will be: 0000 0010 = 2



Operators

- Bitwise Zero Fill Right Shift Operator (>>>)
- This operator also shifts the bits of the number towards right a specified number of positions. But, it stores 0 in the sign bit.
- The symbol for this operator is >>>, read as triple greater than. Since, it always fills 0 in the sign bit, it is called zero fill right shift operator.
- If we apply >>> on a positive number; it gives same output as that of >>. But in case of negative numbers, the output will be positive, since the sign bit is replaced by a 0
- Example:
- If int x = 10 , then calculate x>>2 value
- Shifting the value of x towards right 2 positions will make the rightmost 2 bits to be lost.
- x value is 10 0000 1010.
- Now x>>2 will be: 0000 0010 = 2



Operators

- Ternary or Conditional Operator (?:)
- This operator is called ternary because it acts on 3 variables.
- The other name for this operator is conditional operator, since it represents a conditional statement.
- Two symbols are used for this operator? and:
- Its syntax is variable = expression ? expression2 expression3;
- This means that first of all, expression is evaluated. If it is true, then expression2 value is stored into the variable. If expression is false, then expression3 value is stored into the variable. It means:

  if(expression1 is true) variable = expression2; else variable = expression3;
- Now, let us put the following condition

  max =(a>b) ? a : b;
- Here, (a>b) is evaluated first. If it is true; then the value of a is stored into the variable max, else the value of b is stored into max. This means:

  If(a>b) max =a;

  else max =b;

Operators

- member Operator (.)
- Member operator is also called dot operator since its symbol is a . (dot or period).
- This operator tells about member of a package or a class.
- It is used in three ways:
- We know a package contains classes.
- We can use . operator to refer to the class of a package.
- Syntax:

  packagename.classname;
- Example:

  java.io.BufferredReader;
- We know that each class contains variables or methods.
- To refer to the variables of a class, we can use this operator.
- Examples:

  classname.variablename;

  Or

  objectname.variablename;

  classname.methodname;

  objectname.methodname;

Operators

- instanceof Operator
- This operator is used to test if an object belongs to a class or not.
- Note that the word instance means object.
- This operator can also be used to check if an object belongs to an interface or not.

- Syntax:

  boolean variable = object instanceof class;

  boolean variable = object instanceof interface;
- Example:

  boolean x = emp instanceof Employee;

- Here, we are testing if emp is an object of Employee class or not.
- If emp is an object of Employee class, then true will be returned into x, otherwise x will contain false

Operators

- new Operator
- new operator is often used to create objects to classes.
- We know that objects are created on heap memory by JVM, dynamically (at runtime).

  Syntax:

  Classname obj = new Classname():

  Example:

  Employee e = new Employee();

Operators

- Cast Operator
- Cast operator is used to convert one datatype into another datatype.
- This operator can be used by writing datatype inside simple braces

```
double x = 10.54;
int y = x; // error because data types of x and y are different
```

- To store x value into y, we have to first convert the datatype of x into the datatype of y.
- It means double datatype should be converted into int type by writing int inside the simple braces as: (int).
- This is called cast operator

```
int y = (int) x; // here, x data type is converted into int and then
                    stored into y
```

Here, (int) is called the cast operator. Cast operator is generally used before a variable or before a method.

Priority of Operators

- When several operators are used in a statement, it is important to know which operator will execute first and which will come next.
- To determine that, certain rules of operator precedence are followed:

  - First, the contents inside the braces: () and [] will be executed.
  - Next, ++ and --.
  - Next, *, /, and % will execute.
  - + and - will come next.
  - Relational operators are executed next.
  - Boolean and bitwise operators
  - Logical operators will come afterwards.
  - Then ternary operator.
  - Assignment operators are executed at the last.

Arrays in Java

Array:

- An array represents a group of elements of same data type.
- It can store a group of elements.
- So, we can store a group of int values or a group of float values or a group of strings in the array.
- But we can not store some int values and some float values in the array

- In C, C++, by default, arrays are created on static memory unless - pointers are used to create them.
- In Java, arrays are created on dynamic memory, ie., allotted at runtime by JVM.

Arrays in Java

VISHNU
UNIVERSAL LEARNING

Types Arrays:

- Arrays are generally categorized into two parts as described here:
  - Single dimensional arrays (or 1D arrays)
  - Multi dimensional arrays (or 2D, 3D, ... arrays)

Arrays in Java

Single dimensional arrays (or 1D arrays) :

- A one dimensional (1D) or single dimensional array represents a row or a column of elements.
- For example, the marks obtained by a student in 5 different subjects can be represented by a 1D array, because these marks can be written as a row or as a column.

- Creating Single Dimensional Array:
- There are some ways of creating a single dimensional array as mentioned here:
- We can declare a one dimensional array and directly store elements at the time of its declaration, as:

```
int marks[] = { 50, 60, 75, 55, 62};
```

Arrays in Java

- Now JVM creates 5 blocks of memory as there are 5 elements being stored into the array.
- These blocks of memory can be individually referred to as
- marks[0], marks[1], marks[2], ...marks[4]..Here, 0,1,2,.. .4 is called 'index' of the array.
- It refers to the element position in the array.
- A one dimensional array will have only one index.
- In general, any element of the array can be shown by writing marks[i], where i = 0,1,2, .. .4.

Arrays in Java

Another way of creating a one dimensional array is by declaring the array first and then allotting memory for it by using new operator.

```
int marks[];
marks = new int[5]; // allot memory for storing 5 elements
```

These two statements can also be written by combining them into a single statement, as:

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

- Here, we should understand that JVM allots memory for storing 5 integer elements into the array
- But there are no actual elements stored in the array so far.

Arrays in Java

- To store the elements into the array, we can use statements like these in the program:

```
marks[0] = 50;
marks[1] = 60;
marks[2] = 75;
marks[3] = 55;
marks[4] = 62;
```

Or we can pass the values from the keyboard using a loop as;

```
BufferedReader br = new BufferedReader(new InputStreamReader (System.in));
int n = new Integer.parseInt(br.readLine());

for(int i=0; i < 5; i++){
marks[i] = Integer.parseInt(br.readLine());
}
```

Arrays in Java

Let us examine some more examples for 1D array:

- float salary[] = {5670.55f, 12000f, 4500.75f, 3000.50f, 9050f};
- float salary[ ] = new float[50];
- char ch [ ] = {'a' , ' b' , ' c' , ' d' , ' e' , ' f' };
- char ch [ ] = new char [6] ;
- String names[ ] = {"Raju", "Vijay", "Gopal", "Kiran"};
- String names[ ]= new String[10];

Arrays in Java

VISHNU
UNIVERSAL LEARNING

Task:

Write a program which performs sorting of group of integer values using bubble sort technique.

Arrays in Java

VISHNU
UNIVERSAL LEARNING

Multi-Dimensional Arrays (2D, 3D,... arrays);

- Multi Dimensional arrays represent 2D, 3D,..arrays which are combinations of several earlier types of arrays.
- For example, a two dimensional array is a combination of two or more (1D) one dimensional arrays.
- Similarly, a three dimensional array is a combination of two or more (2D) two dimensional arrays.

Arrays in Java

Let us understand the two dimensional arrays now:

- A two dimensional array represents several rows and columns of data.
- For example, the marks obtained by a group of students in five different subjects can be represented by a 2D array.

- If we write the marks of three students as:
- 50,60,55,67,70
- 62,65,70,70,81
- 72,66,77,80,69

- The preceding marks represent a 2D array since it got 3 rows (no. of students) and in each row 5 columns (no. of subjects).
- There are totally 3X5 = 15 elements.
- We can take the first row itself as a 1D array.
- Similarly the second row is a 1D array and the third row is another 1D array.
- So the preceding 2D array contains three 1D arrays within it.

Arrays in Java

Creating a (2D) Two Dimensional Array

- There are some ways of creating (20) two dimensional array as mentioned here:

- We can declare a two dimensional array and directly store elements at the time of its declaration, as:
- int marks[][] = { { 50, 60, 55, 67, 70},
- {62, 65, 70, 70, 81},
  {72, 66, 77, 80, 69}};

- Here, 'int' represents integer type elements stored into the array, and the array name is 'marks'.
- To represent a two dimensional array, we should use two pairs of square braces [ ][ ] after the array name.
- Each row of elements should be written inside the curly braces { and }.
- The rows and the elements in each row should be separated by commas.

Arrays in Java

There are three rows and five columns in each row
so the JVM creates 3X5= 15 blocks of memory as there are 15 elements being stored into the array.
These blocks of memory can be individually referred to as mentioned here:

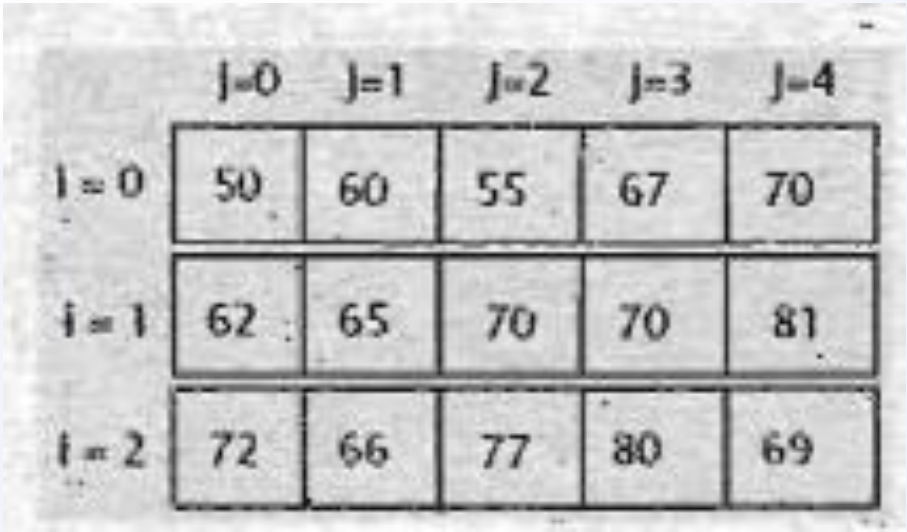
marks[0][0]	marks[0][1]	marks[0][2]	marks[0][3]	marks[0][4]
marks[1][0]	marks[1]1]	marks[1][2]	marks[1][3]	marks[1][4]
marks[2][0]	marks[2][1]	marks[2][2]	marks[2][3]	marks[2][4]

- By observing the preceding elements, we can understand the rows which are starting from 0 to 2 the columns are starting from 0 to 4.
- So any element can be referred in general as marks[i][j], where i represents row position and j represents column position.
- Thus, a two dimensional array two indexes: i and j.
- Similarly we can expect three indexes in case of a three dimensional array.

Arrays in Java

There are three rows and five columns in each row
so the JVM creates 3X5= 15 blocks of memory as there are 15 elements being stored into the array.
These blocks of memory can be individually referred to as mentioned here:

marks[0][0]	marks[0][1]	marks[0][2]	marks[0][3]	marks[0][4]
marks[1][0]	marks[1]1]	marks[1][2]	marks[1][3]	marks[1][4]
marks[2][0]	marks[2][1]	marks[2][2]	marks[2][3]	marks[2][4]



	j=0	j=1	j=2	j=3	j=4
i = 0	50	60	55	67	70
i = 1	62	65	70	70	81
i = 2	72	66	77	80	69

Arrays in Java

Another way of creating a two dimensional array is by declaring the array first and then allotting memory for it by using new operator.

int marks [] [] = new int [3] [4];

Here, JVM allots memory for storing 15 integer elements into the array.
But there are no actual elements stored in the array so far.
We can store these elements by accepting them from the keyboard or from within the program also.

Let us take some more examples for 2D arrays:

- float x[] [] = {{1.1f, 1.2f, 1.3f, 1.4f}, {2.1f, 2.2f, 2.3f, 2.4f}, {3.1f, 3.2f! 3.3f, 3.4f}};
- double d[][] = {{20.2, -5.5.g}, {15.5, 30.331}};
- byte b [] [] = new byte[20][50];
- String str [] [] = new String[10][20];

Arrays in Java

Alternative Way of Writing Two Dimensional Arrays .
while writing a two dimensional array, we can write the two pairs of square braces before or after e array name, as:

```
String str [] [] = new String[10][2];
String [] [] str  = new String[10][2];
```

Arrays in Java

Task1:

Write a program to take a 2D array and display its elements in the form of a matrix.
To display the elements of 2D array, we use two for loops, the outer for loop represents the rows and me inner one represents the columns in each loop.

Task2:

Write a program which accepts elements of a matrix and displaying its transpose.

Arrays in Java

3 Dimensional Array:

- We can consider a three dimensional array as a combination of several two dimensional arrays.
- This concept is useful when we want to handle group of elements belonging to another group.
- For example, a college has 3 departments: IT, CSE and ECE.
- We want to represent the marks, obtained by the students of each department in 3 different subjects.

```
int arr[ ][ ][ ] = {{{50,51,52}, {60,61,62}},
                    {{70,71,72}, {80,81,82}},
                    {{65,66,67}, {75,76,77}}};
```

Next Session

VISHNU
UNIVERSAL LEARNING

Control Statements

Control Statements in Java

A Java statement is the smallest unit that is complete instruction in itself.
Statements in Java generally contain expressions and end with a semi-colon.
The two most- commonly used statements in any programming language are as follows:

- Sequential statements: These are the statements which are executed one by one.
- Control statements: These are the statements that are executed randomly and repeatedly.

- What are control statements?

- Control statements are the statements which alter the flow of execution and provide better control to the programmer on the flow of execution. They are useful to write better and complex programs.

Control Statements in Java

The following control statements are available In Java:

- if ...else statement
- do...while loop
- while loop
- for loop
- for-each loop
- switch statement
- break statement
- continue statement

Control Statements in Java

if ...else statement in java

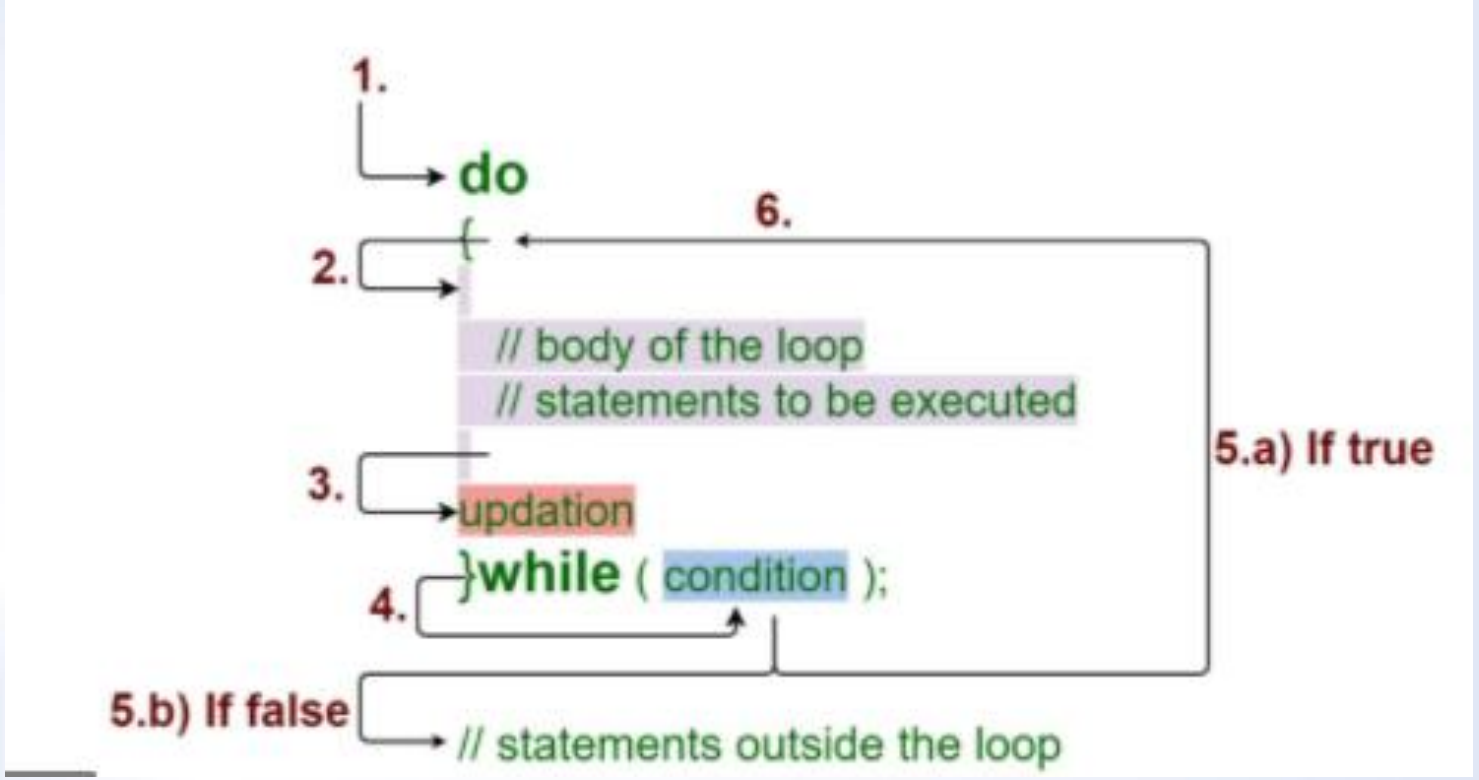
Syntax :

```
if (condition)
{
   statement1;
}
else
{
   statement2;
}
```

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

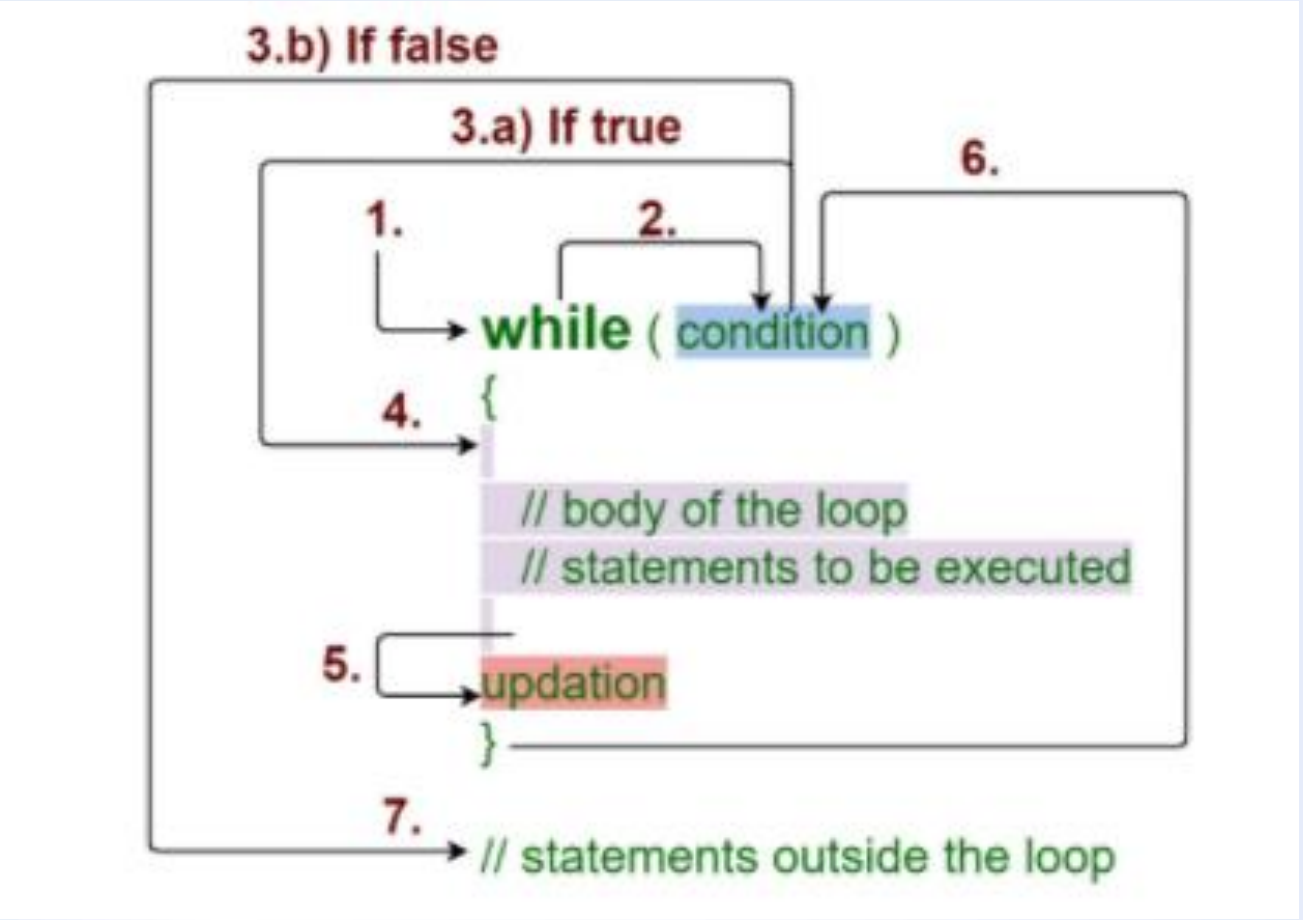
Control Statements in Java

- do...while loop



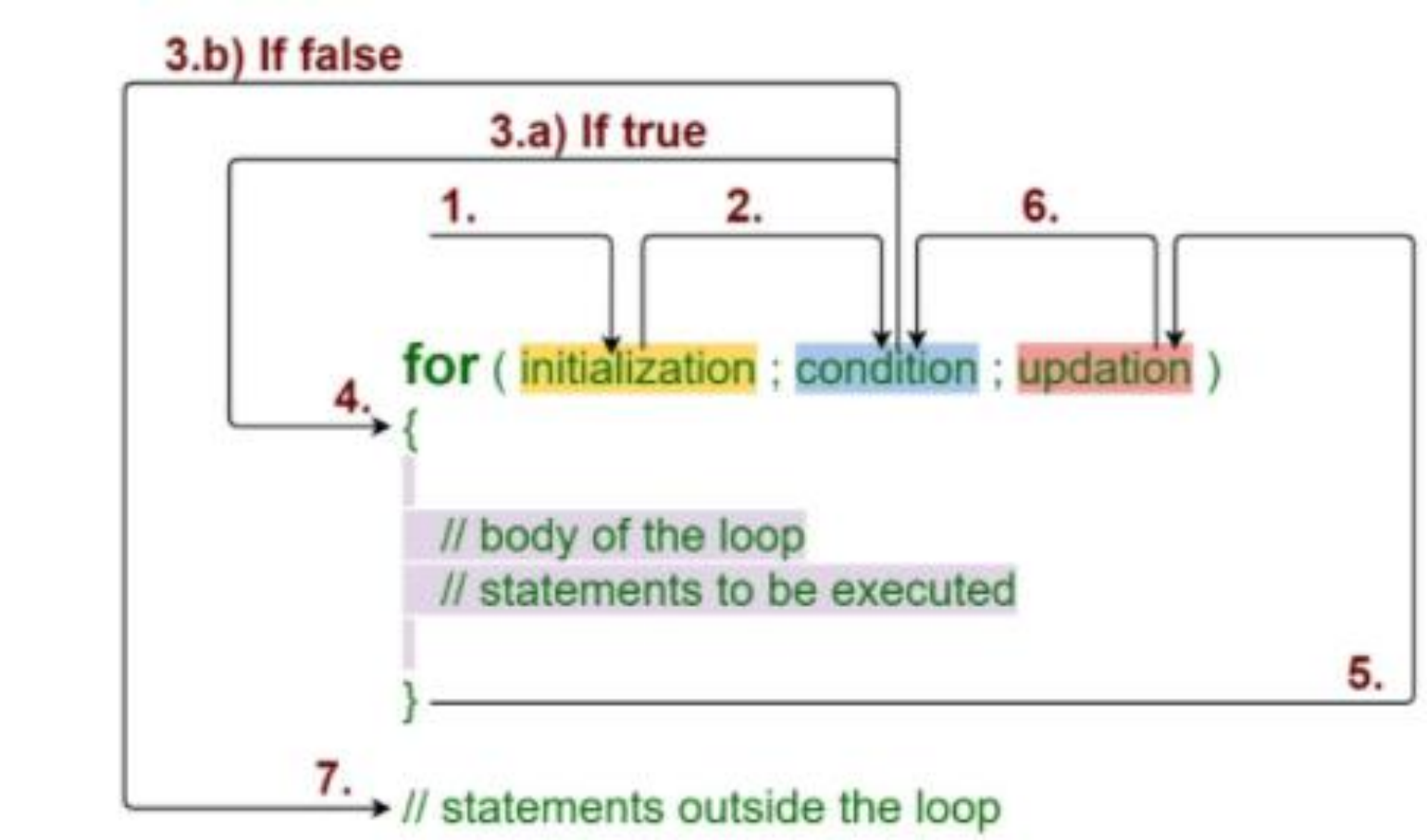
Control Statements in Java

- while loop



Control Statements in Java

- for loop

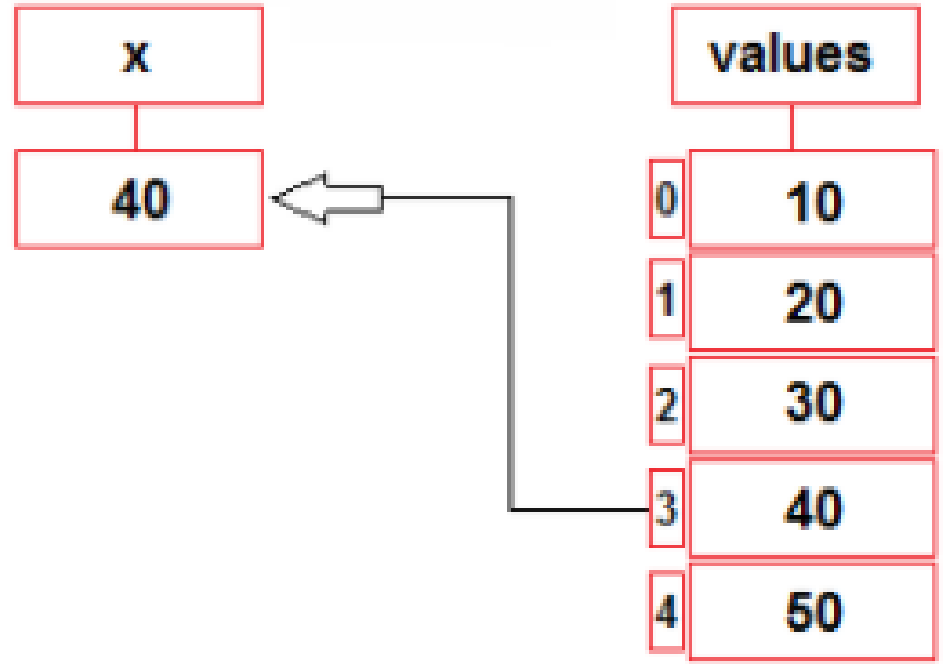


Control Statements in Java

- for-each loop

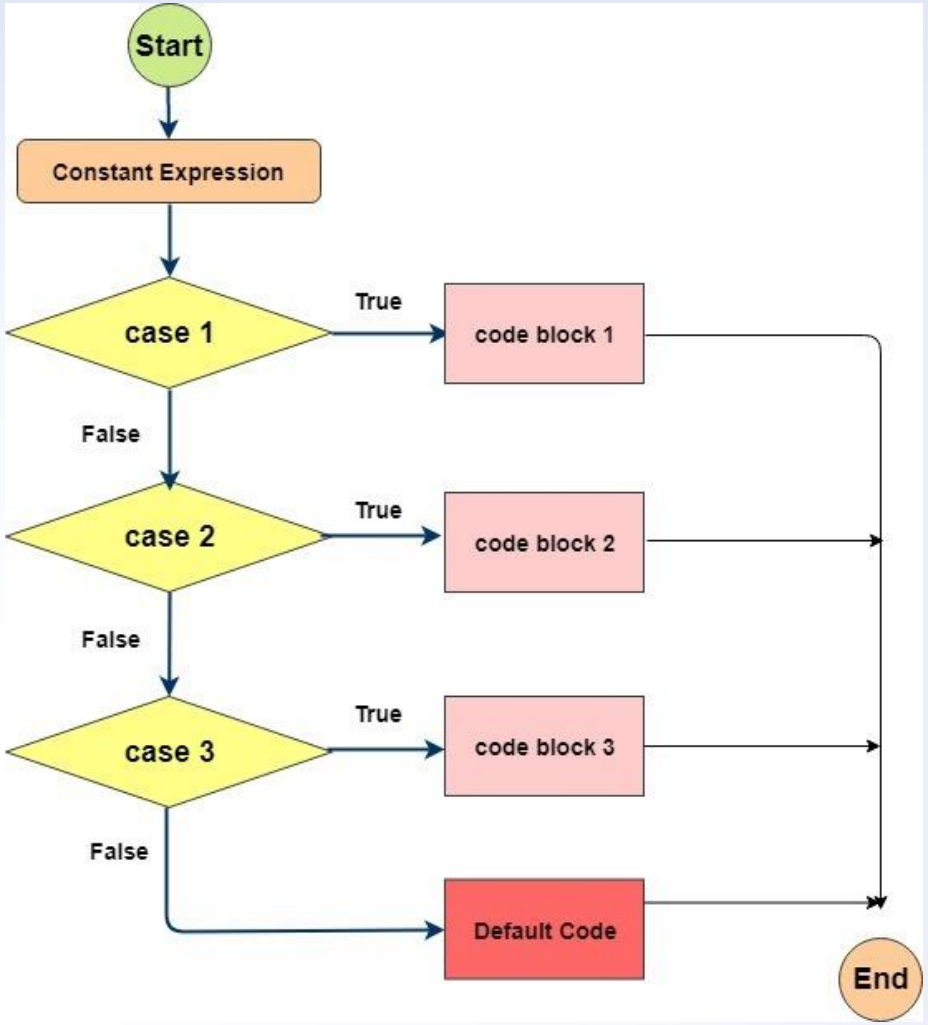
```
for(int x : values)
{

}
```



Control Statements in Java

- switch statement



Control Statements in Java

- break statement

```
for (init; condition; update) {
    // code
    if (condition to break) {
        break;
    }
    // code
}
```

```
while (condition) {
    // code
    if (condition to break) {
        break;
    }
    // code
}
```

Control Statements in Java

- continue statement

```
while (testExpression) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

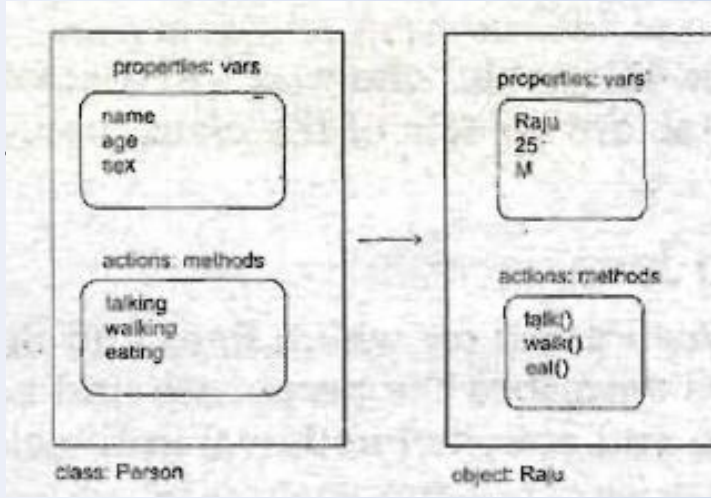
```
do {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
} while (testExpression);
```

```
for (init; testExpression; update) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

Introducing Classes

Class / Object:

- A class is model for creating objects and does not exist physically.
- An object is any thing that exists physically
- Both the class and objects contain variables and methods.
- An object cannot exist without a class, But class can exist without any object.
- We can think that a class is a model and if it physically forms, then becomes an object. So a object is called 'instance' (the thing physically happens) of a class



- **Object** is an instance of a **class**.
- **Class** is a blueprint or template from which **objects** are created.
- **Object** is a real world entity such as pen, laptop, mobile, bed, keyboard, mouse, chair etc.
- **Class** is a group of similar **objects**.

Introducing Classes

What is the purpose of a class?:

- A **class** is used in object-oriented programming to describe one or more objects.
- It serves as a template for creating, or instantiating, specific objects within a program.
- While each object is created from a single **class**, one **class** can be used to instantiate multiple objects.

Introducing Classes

VISHNU
UNIVERSAL LEARNING

Introduction classes and Objects:

- We know a class is a model for creating objects.
- This means, the properties and actions of the objects are written in the class.
- Properties are represented by variables and actions of the objects are represented by methods.
- So a class contains variables and methods.
- The same variables and methods are also available in the objects because they are created from the class.
- These variables are also called 'instance variables' because they are created inside the Object (instance)

Introducing Classes

Introduction classes and Objects:

If we take Person class, we can write code In the class that specifies the properties and actions performed by any person.
For example, a person has, properties like name, age, etc.
Similarly a person can perform actions like talking, walking, etc.
So, the class Person contains these properties and actions as shown here:

```
class Person {
public static void main (String args[]){
        String name;
        int age = 24;
        void talk (){
        System.out.println("Hello I am : " + name + " My age is :" + age);
        System.out.println("Person is talking");
}
}
```

Introducing Classes

Introduction classes and Objects:

- Writing a class like this is not sufficient. It should be used.
- To use a class, we should create an object to the class.
- Object creation represents allotting memory necessary to store the actual data of the variables, i.e., Wakeel and 44.

To create an object, the following syntax is used:

Classname objectname = new Classname();

Introducing Classes

VISHNU
UNIVERSAL LEARNING

Introduction classes and Objects:

To create an object to Person class, we can write:

Person Wakeel = new Person();

Here, 'new' is an operator that creates the object to Person class, hence the right hand side part of the statement is responsible for creating the object, What about the left side statement, which is:

Person Wakeel;

Here, Person is the class name and Wakeel is the object name. Wakeel is actually a variable of Person class.

This variable stores the reference number of the object returned by JVM, after creating the object.

If Wakeel is a variable, then what is 'Person'? It is, Of course the class name, but class is also a data type. So we can say that Wakeel is a variable of Person class type.

Introducing Classes

Introduction classes and Objects:

Object Creation:

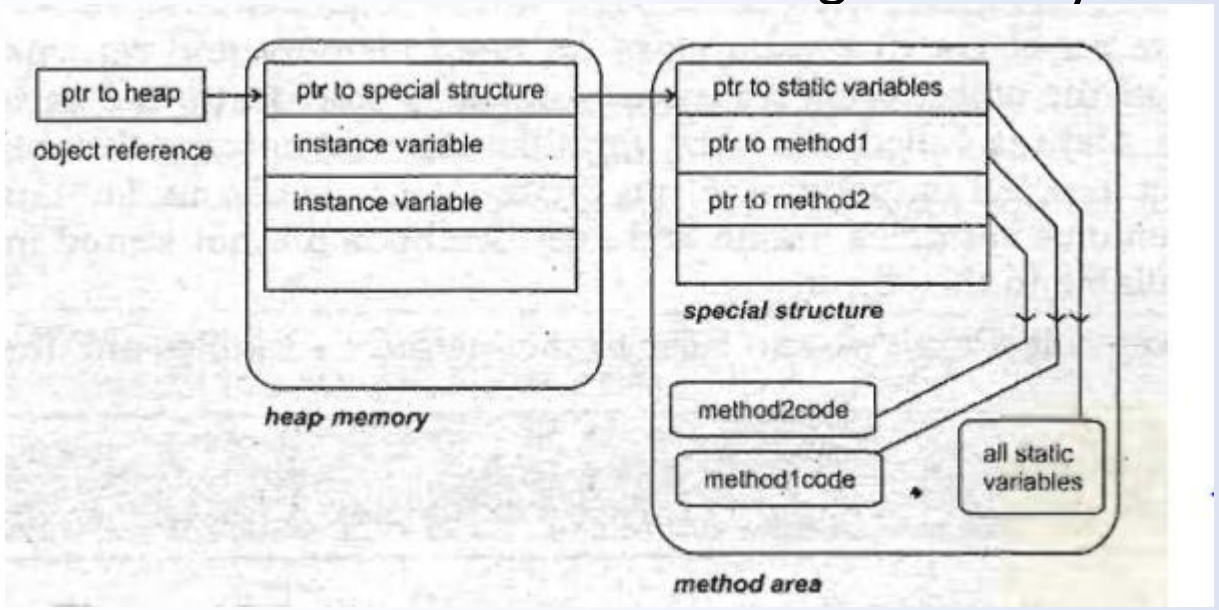
- We know that the class code along with method code is stored in 'method area' of the JVM.'
- When an abject is created, the memory is allocated on 'heap'.
- After creation of an object, JVM produces a 'unique reference number for the object from the memory address of the object.
- This reference number is also called hash code number.
- To know the hashcode number (or reference) of an object, we can use hashCode () method of Object class, as shown here:
- Employee e1 = new Employee(); //e1 is reference of Employee object.
- System.out.println (e1. hashCode ()); //displays hash code stored in e1

Introducing Classes

Introduction classes and Objects:

The object reference, (hash code) internally represents heap memory where instance variables arc stored. There would be a pointer (memory address) from heap memory to a special structure locate in method area. In method area a table is available which contains pointers to static variables and methods.

This is, how the instance variables and methods are organized by JVM internally at run time.



Introducing Classes

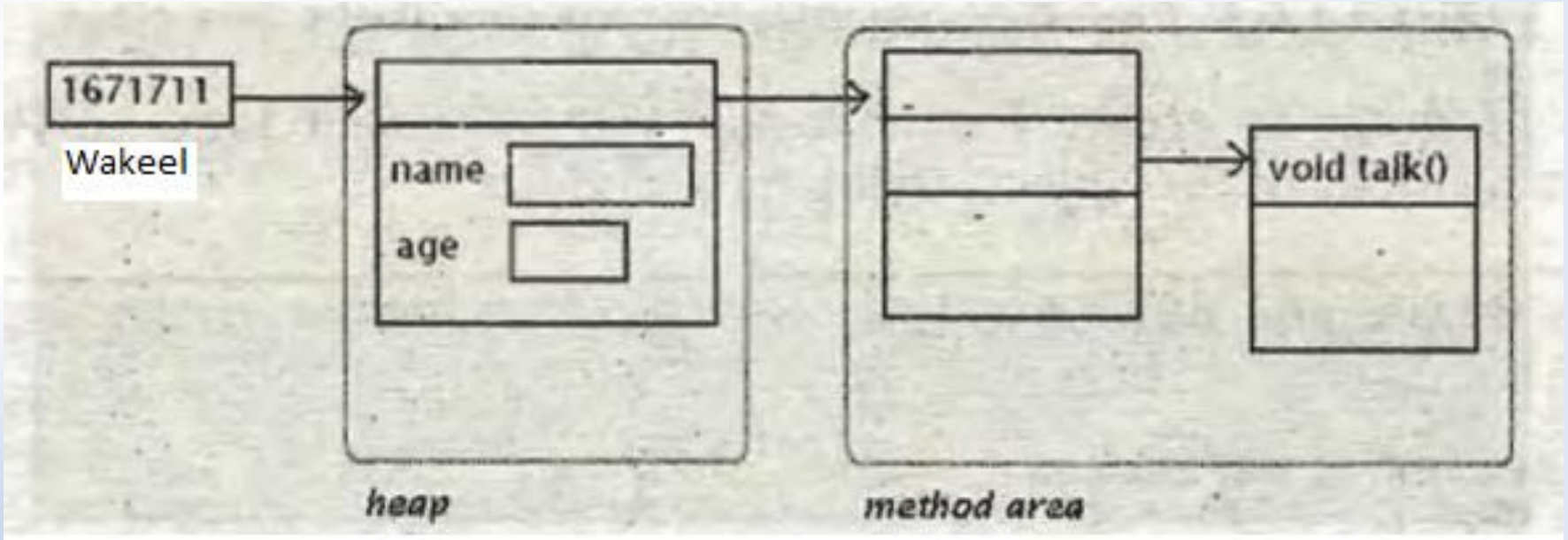
Exercise :

Write a program to create Person class and an object to Person class.

Let us display the hash code number of the object, using hashCode () .

Introducing Classes

Exercise :
Write a program to create Person class and an object to Person class.
Let us display the hash code number of the object, using hashCode () .



Introducing Classes

Default values of the instance variables:

Data type	Default value
byte	0
short	0
int	0
long	0
float	0.0
double	0.0
char	a space
String	null
any class type	null
boolean	false

```
HashCode =366712642
Name (String) =: null
age (int) =: 0
height (byte) =: 0
Account (long) =: 0
Phone Number(double) =: 0.0
Gender (char) =:
Boolean value is =: false
```

Access Modifiers

Access Modifiers:

- The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class.

- We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

Access Specifiers

There are four types of Java access modifiers:

Private: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
They are accessible only within the class by the methods of that class
Default: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. **If you do not specify any access level, it will be the default.**
Protected: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
Public: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Access Specifiers

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

- Can you declare a class as 'private'?

- No, if we declare a class as private, then it is not available to Java compiler and hence a compile time error occurs.
- But, inner classes can be declared as private.

Method in Java

- Method in Java

- a **method** is a way to perform some task.

- **Method in Java** is a collection of instructions that performs a specific task. It provides the reusability of code.

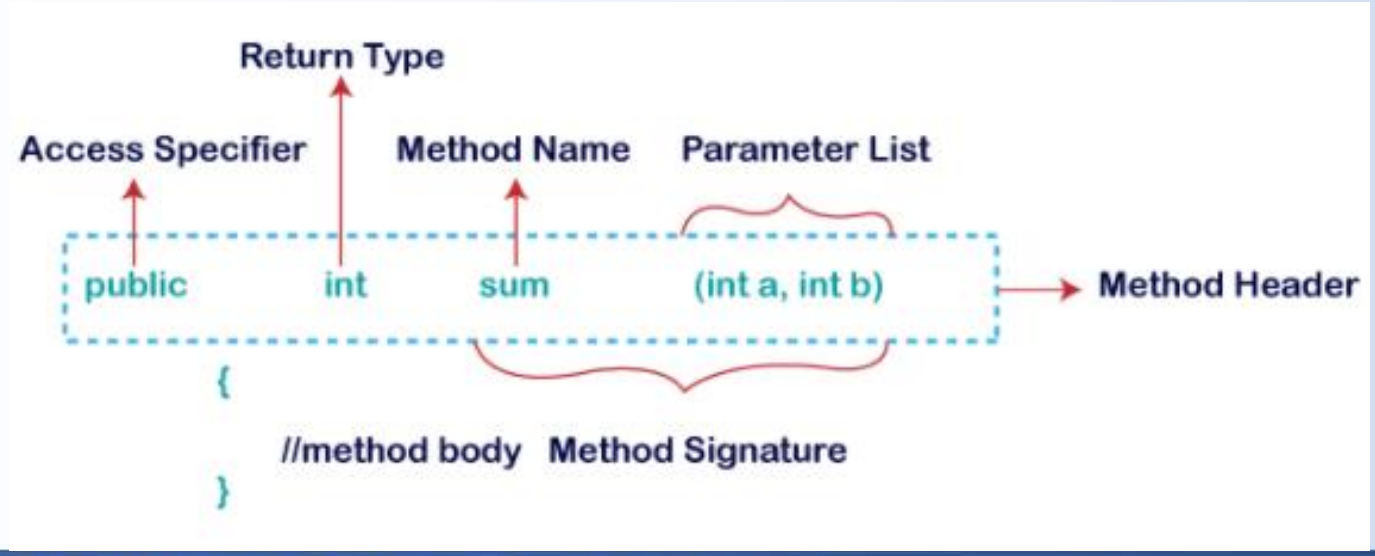
- We can also easily modify code using **methods**.

Method in Java

- Method declaration

- The method declaration provides information about method attributes, such as visibility, return-type, name, and arguments. It has six components that are known as **method header.**

```
public int sum (int a, int b) {
        Statements;
}
```



Method in Java

- **Method Signature**

Method Signature: Every method has a method signature. It is a part of the method declaration. It includes the **method name** and **parameter list**.

Access Specifier: Access specifier or modifier is the access type of the method. It specifies the visibility of the method. Java provides **four** types of access specifier : public, private, protected and default

Return Type: Return type is a data type that the method returns. It may have a primitive data type, object, collection, void, etc. If the method does not return anything, we use void keyword.

Method Name: It is a unique name that is used to define the name of a method. It must be corresponding to the functionality of the method.

Parameter List: It is the list of parameters separated by a comma and enclosed in the pair of parentheses. It contains the data type and variable name. If the method has no parameter, left the parentheses blank.

Method Body: It is a part of the method declaration. It contains all the actions to be performed. It is enclosed within the pair of curly braces.

Method in Java

- Naming a Method

Single-word method name: sum(), volume()

Multi-word method name: areaOfCircle(), stringComparision()

- Types of Method

There are two types of methods in Java:

- Predefined Method
- User-defined Method

Method in Java

Predefined Method

- In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods.
- It is also known as the **standard library method** or **built-in method**.
- We can directly use these methods just by calling them in the program at any point.
- Some pre-defined methods are **length(), equals(), compareTo(), sqrt(),** etc.

```
public class Demo   {
public static void main(String[] args)   {
System.out.print("The maximum number is: " + Math.max(9,7));
}
}
```

Method in Java

User-defined Method

The method written by the user or programmer is known as **a user-defined** method.

These methods are modified according to the requirement.

How to Create a User-defined Method

```
//user defined method
public static void findEvenOdd(int num)
{
    if(num%2==0)
    System.out.println(num+" is even");
    else
    System.out.println(num+" is odd");
}
```

Method in Java

How to Call or Invoke a User-defined Method

Once we have defined a method, it should be called. The calling of a method in a program is simple.

When we call or invoke a user-defined method, the program control transfer to the called method.

```
import java.util.Scanner.*;
public class EvenOdd  {
public static void main (String args[])  {
    Scanner scan=new Scanner(System.in);
    System.out.print("Enter the number: ");
    int num=scan.nextInt();
    findEvenOdd(num);
}
```

Method in Java

How to Call or Invoke a User-defined Method

```
public class Addition  {
public static void main(String[] args)  {
    int a = 19;
    int b = 5;
    //method calling
    int c = add(a, b);  //a and b are actual parameters
    System.out.println("The sum of a and b is= " + c);
}
//user defined method
public static int add(int n1, int n2)  { //n1 and n2 are formal parameters
    int s;
    s=n1+n2;
    return s; //returning the sum
}
}
```

Method in Java

VISHNU
UNIVERSAL LEARNING

Static Method

- A method that has static keyword is known as static method.
- In other words, a method that belongs to a class rather than an instance of a class is known as a static method.
- The main advantage of a static method is that we can call it without creating an object.
- It can access static data members and also change the value of it.
- It is used to create an instance method.
- It is invoked by using the class name.

Method in Java

Static Method

Example of static method

```
public class Display  {
public static void main(String[] args)   {
        show();
}
static void show()   {
        System.out.println("It is an example of static method.");
}
}
```

Method in Java

Instance Method

- The method of the class is known as an **instance method**.
- It is a **non-static** method defined in the class.
- Before calling or invoking the instance method, it is necessary to create an object of its class.

Method in Java

Instance Method

```
public class InstanceMethodExample  {
public static void main(String [] args)  {
      InstanceMethodExample obj = new InstanceMethodExample();
      System.out.println("The sum is: "+obj.add(12, 13));
}
int s;
public int add(int a, int b)  {
      s = a+b;
      return s;
}
}
```

Method in Java

There are two types of instance method:

- **Accessor Method**
- **Mutator Method**

Accessor Method: The method(s) that reads the instance variable(s) is known as the accessor method.

We can easily identify it because the method is prefixed with the word **get**.

It is also known as **getters**.

It returns the value of the private field.

It is used to get the value of the private field.

Example

```
public int getId()    {
      return Id;
}
```

Method in Java

There are two types of instance method:

- **Accessor Method**
- **Mutator Method**

Mutator Method: The method(s) read the instance variable(s) and also modify the values.

We can easily identify it because the method is prefixed with the word **set**.

It is also known as **setters** or **modifiers**.

It does not return anything.

It accepts a parameter of the same data type that depends on the field.

It is used to set the value of the private field.

Example

```
public void setRoll(int roll)
{
        this.roll = roll;
}
```

Method in Java

```
public class Student   {
        private int roll;
        private String name;
public int getRoll()   { //accessor method
            return roll;
}
public void setRoll(int roll) { //mutator method
            this.roll = roll;
}
public String getName()   {
            return name;
}
public void setName(String name)   {
            this.name = name;
}
public void display()  {
            System.out.println("Roll no.: "+roll);
            System.out.println("Student name: "+name);
}
}
```

Method in Java

Abstract Method

- The method that does not has method body is known as abstract method.
- In other words, without an implementation is known as abstract method.
- It always declares in the **abstract class**.
- It means the class itself must be abstract if it has abstract method.
- To create an abstract method, we use the keyword **abstract**.

```
abstract void method_name();
```

Method in Java

There are two types of instance method:

- **Accessor Method**
- **Mutator Method**

Accessor Method: The method(s) that reads the instance variable(s) is known as the accessor method.

We can easily identify it because the method is prefixed with the word **get**.

It is also known as **getters**.

It returns the value of the private field.

It is used to get the value of the private field.

Method in Java

There are two types of instance method:

- **Accessor Method**
- **Mutator Method**

Mutator Method: The method(s) read the instance variable(s) and also modify the values.

We can easily identify it because the method is prefixed with the word **set**.

It is also known as **setters** or **modifiers**.

It does not return anything.

It accepts a parameter of the same data type that depends on the field.

It is used to set the value of the private field.

Method in Java

```
public class Person {
  private String name; // private = restricted access
  // Getter
  public String getName() {
    return name;
  }
  // Setter
  public void setName(String newName) {
    this.name = newName;
  }
}

public class Main {
  public static void main(String[] args) {
    Person p = new Person();
    p.setName(“Wakeel"); // Set the value of the name variable to "John"
    System.out.println(p.getName());
  }
}
```

Method in Java

Abstract Method

- The method that does not has method body is known as abstract method.
- In other words, without an implementation is known as abstract method.
- It always declares in the **abstract class**.
- It means the class itself must be abstract if it has abstract method.
- To create an abstract method, we use the keyword **abstract**.

```
abstract void method_name();
```

Constructors in Java

Constructor in Java

- The third possibility of initialization is using constructors.
- A constructor is similar to a method is used to initialize the instance variables.
- The sole purpose of a constructor is to initialize instance variables.
- A constructor has the following characteristics
  - The constructor's name and class name should be same. And the constructor's name should end with a pair of simple braces.
  - Example

```
Person (){

}
```

Constructors in Java

Constructor in Java

- constructor may have or may not have parameters:
- Parameters are variables to receive data from outside into the constructor.
- If a constructor does not have any parameters, it is called default constructor.
- If a constructor has 1 or more parameters, it is called 'parameterized constructor;
- For example, we can write a default constructor: as: -

```
Person (){

}
```

Constructors in Java

Constructor in Java

- parameterized constructor with two parameters, as:

```
Person (String s, int i){

}
```

- A constructor does not return any value, not even 'void'. Recollect, if a method does not return any value, we 'write 'void' before the method name.
- That means, the method is returning void which means 'nothing'.
- But in case of a constructor, we should not even write 'void' before the constructor.

Constructors in Java

Constructor in Java

- constructor is automatically called and executed at the time of creating an object.
- While creating an object, if nothing is passed to the object, the default constructor is called and executed.
- If some values are passed to the object, then the parameterized constructor is called.
- For example, if we create the object as:

```
Person raju = new Person (); // here default constructor is called
Person raju= new Person ("Raju", 22); // here parameterzed constructor will receive "Raju" and 22.
```

Constructors in Java

Constructor in Java

- constructor is called and executed only once per object.
- This means when we create an object, the constructor is called.
- When we create second object, again the constructor is called second-time:

Constructors in Java

VISHNU
UNIVERSAL LEARNING

Constructor in Java

- When is a constructor called, before or after creating the object?

- A constructor is called concurrently when the object creation is going on.
- JVM first allocates memory for the object and then executes the constructor to initialize the instance variable.
- By the time, object creation is completed, the constructor execution is also completed.

Constructors in Java

VISHNU
UNIVERSAL LEARNING

Constructor in Java

Example:

```
class Person {
        private String name;
        private int age;
        Person() {
                name = "Wakeel";
                age = 55;
        }

        void argue(){
                System.out.println("Hello I am : " + name);
                System.out.println("My Age is : " + age);
        }
}
```

Constructors in Java

```
class ConstructorDemo{
public static void main (String args[]){
      Person Pawan = new Person();
      Pawan.argue();
      Person Nanda = new Person();
      Nanda.argue();
}
}
```

Constructors in Java

Parameterized Constructor

```
class Person {
        private String name;
        private int age;
        Person() {
                name = "Wakeel";
                age = 55;
        }
        Person(String s, int a){
                name = s;
                age = a;
        }
        void argue(){
                System.out.println("Hello I am : " + name);
                System.out.println("My Age is : " + age);
        }
}
```

Constructors in Java

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

Person Pawan = new Person();
Pawan.argue();
Person Nanda = new Person("Nanda", 60);
Nanda.argue();
}
}
```

Constructors in Java

Constructor Overloading:

- Please observe the above Program, where we have written two constructors.
- Both the constructors have same name, but there is a difference in the parameters. This is called Constructor overloading.

- What is constructor overloading?
- Writing two or more constructors with the same name but with difference in the parameters is called constructor overloading.
- Such constructors are useful to perform different tasks

Constructors in Java

Difference Between Default and Parameterized Constructor

Default constructor	Parameterized constructor
Default constructor is useful to initialize all objects with same data.	Parameterized constructor is useful to initialize each object with different data.
Default constructor does not have any parameters.	Parameterized constructor will have 1 or more parameters.
When data is not passed at the time of creating an object, default constructor is called.	When data is passed at the time of creating an object, parameterized constructor is called.

Constructors in Java

Difference Between Constructors and Methods

Constructors	Methods
A constructor is used to initialize the instance variables of a class.	A method is used for any general purpose processing or calculations.
A constructor's name and class name should be same.	A method's name and class name can be same or different.
A constructor is called at the time of creating the object.	A method can be called after creating the object.
A constructor is called only once per object.	A method can be called several times on the object.
A constructor is called and executed automatically.	A method is executed only when we call it.

String Handling in Java

- In C/C++ languages, a string represents an array of characters, where the last character will '\0' (called null character).
- This last character being '\0' represents the end of the string.
- But this is not valid In ava.
- In java, a String is an object of String class.
- It is not character array.
- In java, we got character array also, but strings are given a separate treatment because of their extensive use on Internet.
- A separate class is available in java with the name 'String' in java.lang (language) package with all necessary methods to work with strings.

String Handling in Java

- Declaration

```
//Declaration type 1
char c[] = {'w','a','k','e','e','l'};
String s = new String(c);

//Declaration type 2
String s1 = "Wakeel Saab";

//Declaration type 3
String s2 = new String("Wakeel Bhai");
```

String Handling in Java

- String class methods
  - String concat(String s)
  - int length ()
  - char charAt (int i)
  - int compareTo (String s)
  - int ·compareToIgnoreCase (String s)
  - boolean equals (String· s)
  - boolean equalsIgnoreCase (String s)
  - boolean startsWith (String s)
  - boolean endsWith (String s)
  - int indexOf(String s)
  - String replace (char cl, char c2)
  - String substring (int il, int i2)
  - String toLowerCase()

String Handling in Java

- String class methods
  - String toUpperCase ()
  - String trim ()
  - oid getChars (int i1, int i2, char arr [ ], int i3)
  - String ( ] split (delimiter)

String Handling in Java

- String Comparison

String s1 = "Hello";
String s2 = **new String("Hello");**

```
if (s1 ==  s2)
System.out.println("Strings are equal");
else
System.out.println("Not Equal");
```

String Handling in Java

- String Comparison

String s1 = "Hello";
String s2 = **new String("Hello");**

```
if(s1.equals(s2))
System.out.println("s1 and s2 are equal");
else
System.out.println("s1 and s2 are not EQUAL");
```

- String Comparison

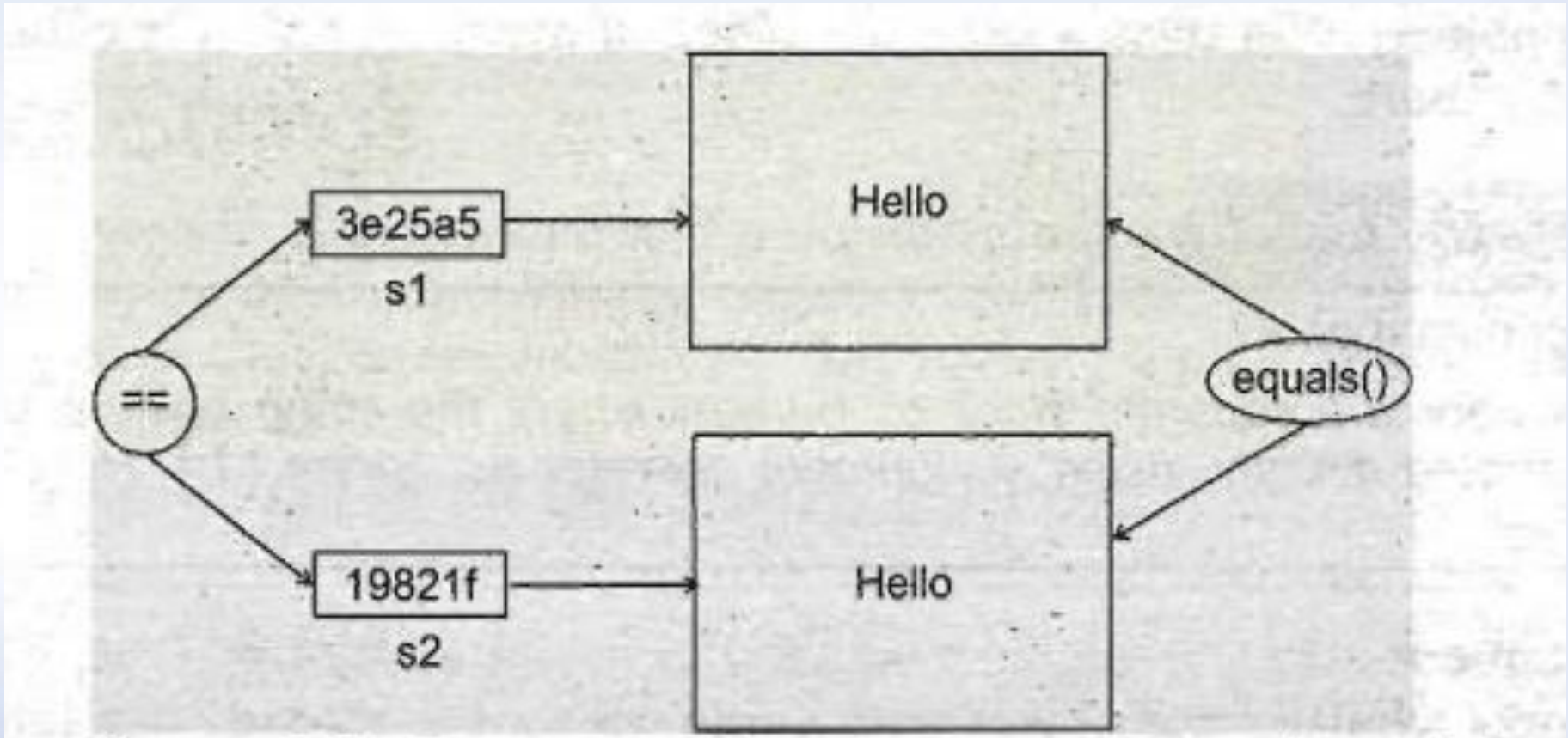
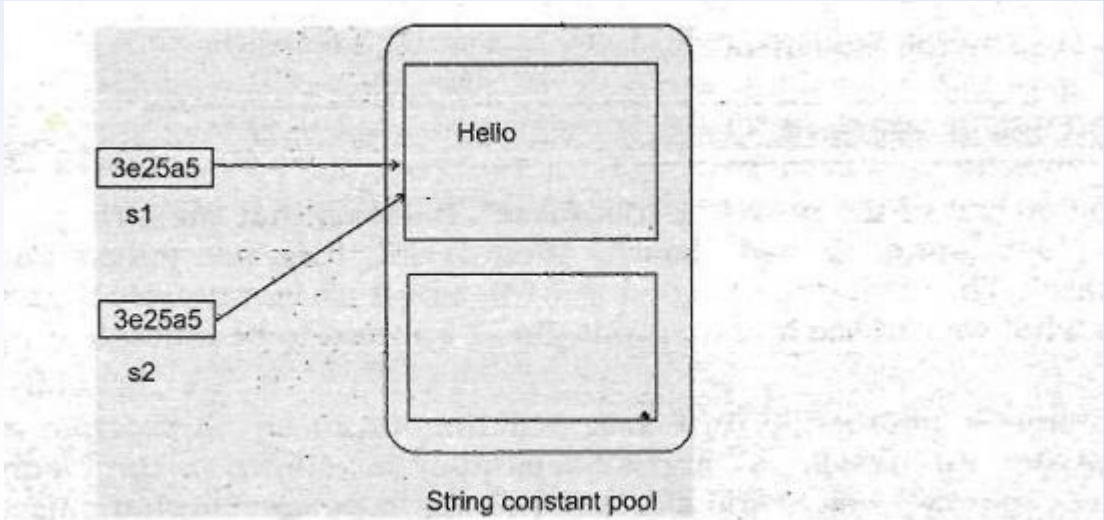


Figure 9.1 Comparing the strings using == and equals()

String Handling in Java

- String Comparison

```
String ss1 = "Wakeel";
String ss2 = "Wakeel";
if (ss1 == ss2)
System.out.println("Equal");
else
System.out.println("not Equal");
```



String comparison when the strings are created using assignment

String Handling in Java

- What is a string constant pool?

String constant pool is a separate block of memory where the string objects are held by JVM.
If a string object is created directly, using assignment operator as:

String s1= “Wakeel”; then it is stored in string constant pool.

String Handling in Java

- Immutability of Strings

- We can divide objects broadly as, mutable and immutable objects.

- Mutable objects are those objects whose contents can be modified.
- Immutable objects are those objects, once created can not be modified.

- And String class objects are immutable.

String Handling in Java

- Immutability of Strings

- Let us take a program to understand whether the String objects are immutable or not

```
String s1 = “data";
String s2 = “base";
s1 = s1 + s2;
System.out.println(s1);
```

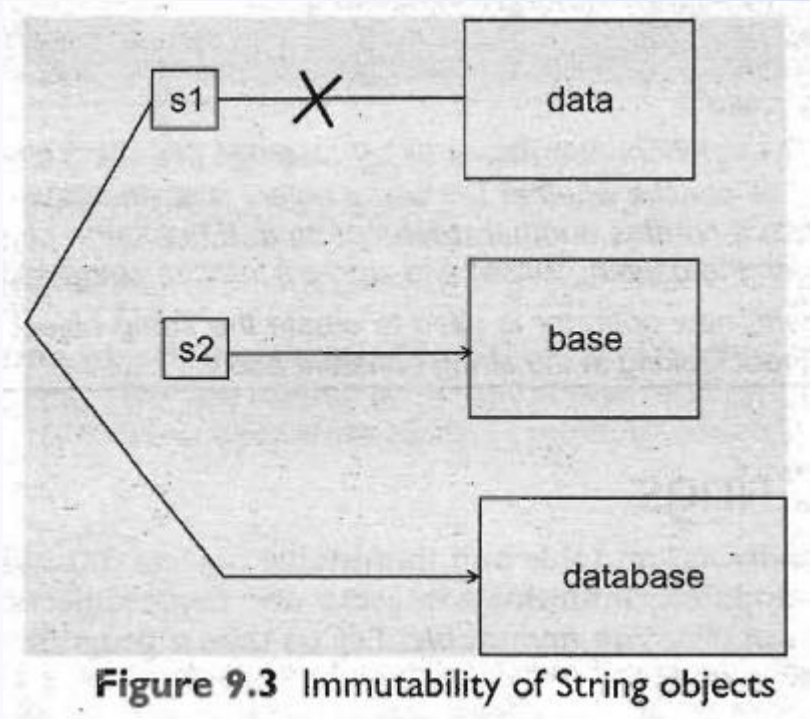


Figure 9.3 Immutability of String objects

String Handling in Java

- Conclusion

- Strings are very important since they are most often used on a network while passing data from one system to another system.
- The data thus passed will be generally in the form of strings.
- Java's String class comes with necessary methods to work with strings.
- Since String class objects are immutable, they can be shared between different applications.

Next

VISHNU
UNIVERSAL LEARNING

Inheritance

String Handling in Java

- String Comparison

String s1 = "Hello";
String s2 = **new String("Hello");**

```
if (s1 ==  s2)
System.out.println("Strings are equal");
else
System.out.println("Not Equal");
```

String Handling in Java

- String Comparison

String s1 = "Hello";
String s2 = **new String("Hello");**

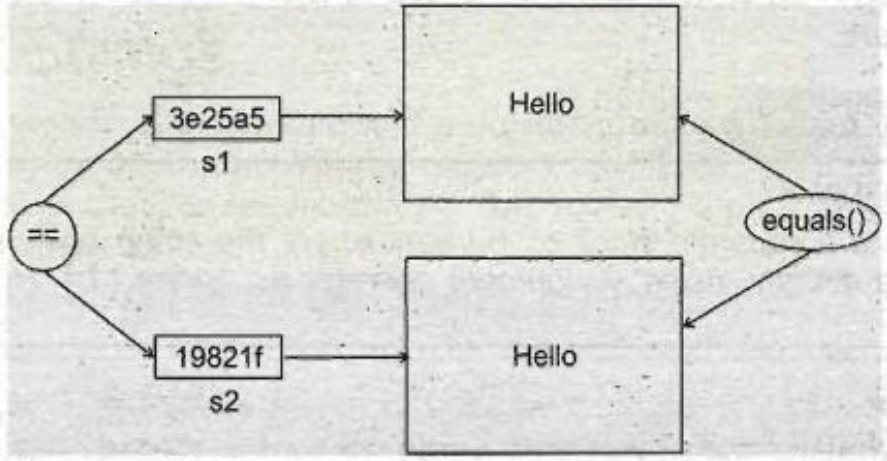


Figure 9.1 Comparing the strings using == and equals()

```
if(s1.equals(s2))
System.out.println("s1 and s2 are equal");
else
System.out.println("s1 and s2 are not EQUAL");
```

String Handling in Java

- String Comparison

In general both equals() and "==" operator in Java are used to compare objects to check equality but here are some of the differences between the two:

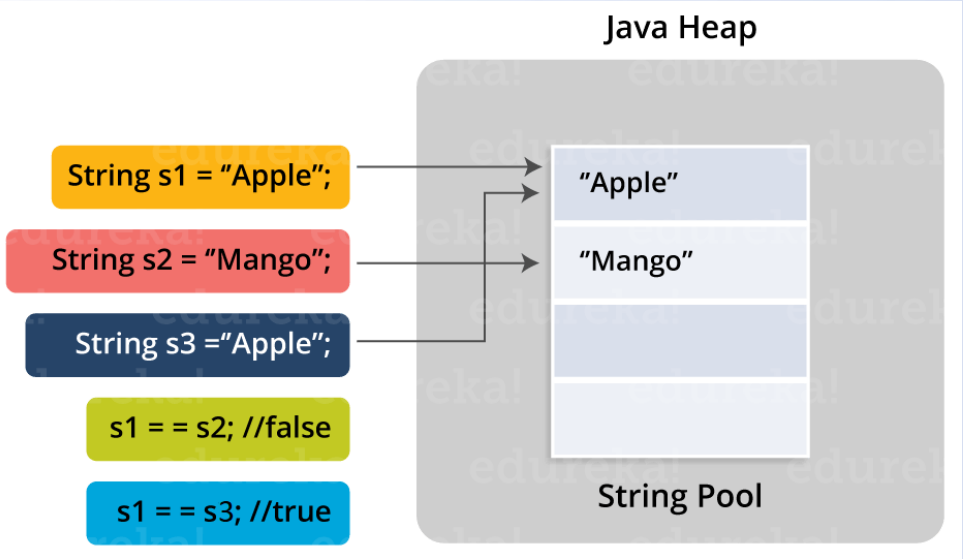
- Main difference between equals() method and == operator is that one is method and other is operator.
- We can use == operators for reference comparison (**address comparison**) and equals() method for **content comparison**.
- In simple words, == checks if both objects point to the same memory location whereas equals() evaluates to the comparison of values in the objects.

String Handling in Java

- What is a string constant pool?

String constant pool is a separate block of memory where the string objects are held by JVM.
If a string object is created directly, using assignment operator as:

String s1= “Wakeel”; then it is stored in string constant pool.



String Handling in Java

- Immutability of Strings

- We can divide objects broadly as, mutable and immutable objects.

- Mutable objects are those objects whose contents can be modified.
- Immutable objects are those objects, once created can not be modified.

- And String class objects are immutable.

String Handling in Java

- Immutability of Strings

- Let us take a program to understand whether the String objects are immutable or not

```
String s1 = “data";
String s2 = “base";
s1 = s1 + s2;
System.out.println(s1);
```

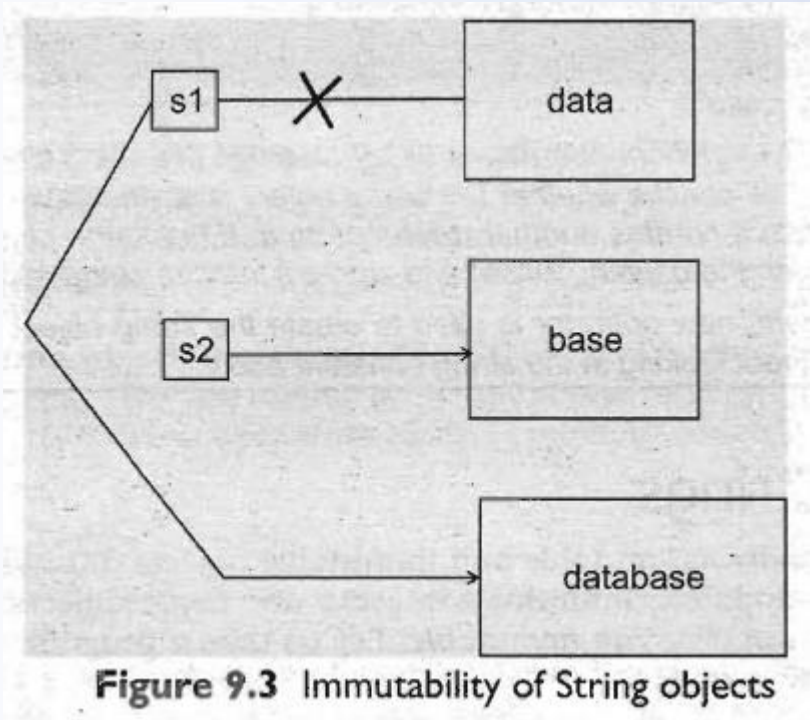


Figure 9.3 Immutability of String objects

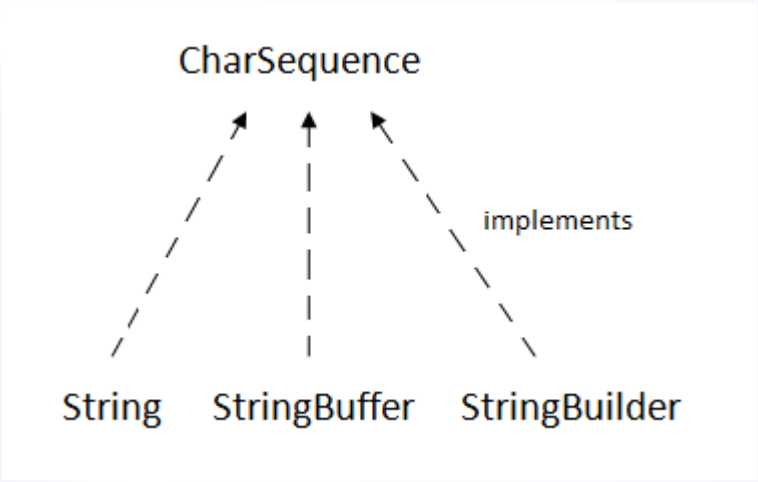
String Handling in Java

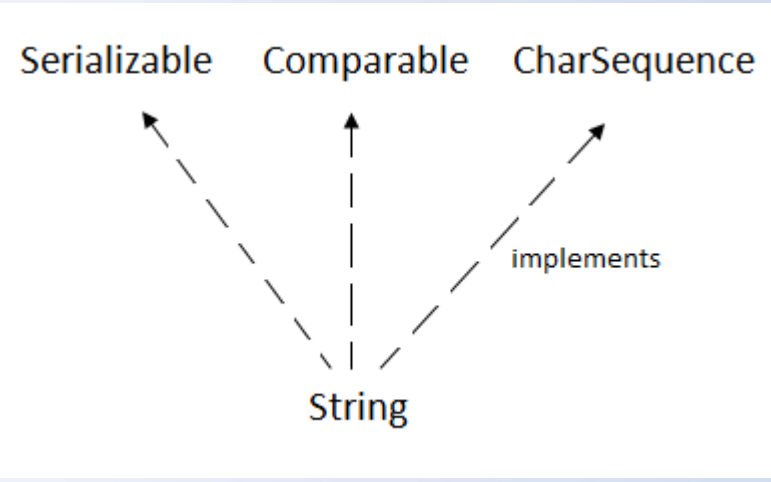
- Conclusion

- Strings are very important since they are most often used on a network while passing data from one system to another system.
- The data thus passed will be generally in the form of strings.
- Java's String class comes with necessary methods to work with strings.
- Since String class objects are immutable, they can be shared between different applications.

String Handling in Java

- StringBuffer and StringBuilder Classes
- Since String class objects are immutable, they can not be changed.
- To overcome this; we got another class 'StringBuffer' which represents string in such a way that their data can be modified.
- It means StringBuffer class objects are mutable. And there methods provided in this class which directly manipulate the data inside the object.





String Handling in Java

- StringBuffer and StringBuilder Classes

- What is the difference between String and StringBuffer classes?

- String class objects are immutable and hence their contents cannot be modified. StringBuffer class objects are mutable; so they can be modified. Moreover the methods that directly manipulate data of the object are not available in String class. Such methods are available in StringBuffer class.

- Are there any other classes whose objects are Immutabie?
- Yes, classes like Character, Byte, Integer, Float, Double, Long... called 'wrapper classes' are created as 'immutable'.
- char, int, byte, float – Character, Byte, Interger, Float
- Classes like Class, BigInteger, BigDecimal are also immutable.

this keyword in Java

Usage of java this keyword

Here is given the 6 usage of java this keyword.

- this can be used to refer current class instance variable.
- this can be used to invoke current class method (implicitly)
- this() can be used to invoke current class constructor.
- this can be passed as an argument in the method call.
- this can be passed as argument in the constructor call.
- this can be used to return the current class instance from the method.

this keyword in Java

```
class Account{
int a;
int b;
public void setData(int a , int b){
    this. a=a;
    this. b=b;
}
public static void main(string args[]){
Account obj = new Account();


}
```

use keyword "This" to differentiate instance variable from local variable

```
public void setData(int c , int d){
    this.a=c;
    this.b=d;
public static void main(string args[]){
Account object1 = new Account();
object1.setData(2,3);
Account object2 = new Account();
object2.setData(4,3);
}
}
```

use keyword "this" infront of instance variable

Next

VISHNU
UNIVERSAL LEARNING

Inheritance

String Handling in Java

- Inheritance
- What is inheritance?
- Deriving new classes from existing classes such that the new classes acquire all the features of existing classes is called inheritance

String Handling in Java

- Inheritance

```
class A {
        protected int a;
        protected int b;
        public void method1(){
        }
}
class B extends A {
        private int c;
        public void method2(){
        }
}
```

String Handling in Java

- Inheritance

Example Program
Teacher.java, Student. Java, Use.java

String Handling in Java

- Inheritance – Super Keyword

Example Program
Super1.java

String Handling in Java

- Inheritance – Constructors

  Example Program
  TestDefaultConsDemo.java
  ParameterizedConstructorDemo.java

this keyword in Java

```
class Account{
int a;
int b;
public void setData(int a , int b){
    this. a=a;
    this. b=b;
}
public static void main(string args[]){
Account obj = new Account();


}
```

use keyword "This" to differentiate instance variable from local variable

```
public void setData(int c , int d){
    this.a=c;
    this.b=d;
public static void main(string args[]){
Account object1 = new Account();
object1.setData(2,3);
Account object2 = new Account();
object2.setData(4,3);
}
}
```

use keyword "this" infront of instance variable

Next

VISHNU
UNIVERSAL LEARNING

Inheritance

String Handling in Java

- Inheritance
- What is inheritance?
- Deriving new classes from existing classes such that the new classes acquire all the features of existing classes is called inheritance

String Handling in Java

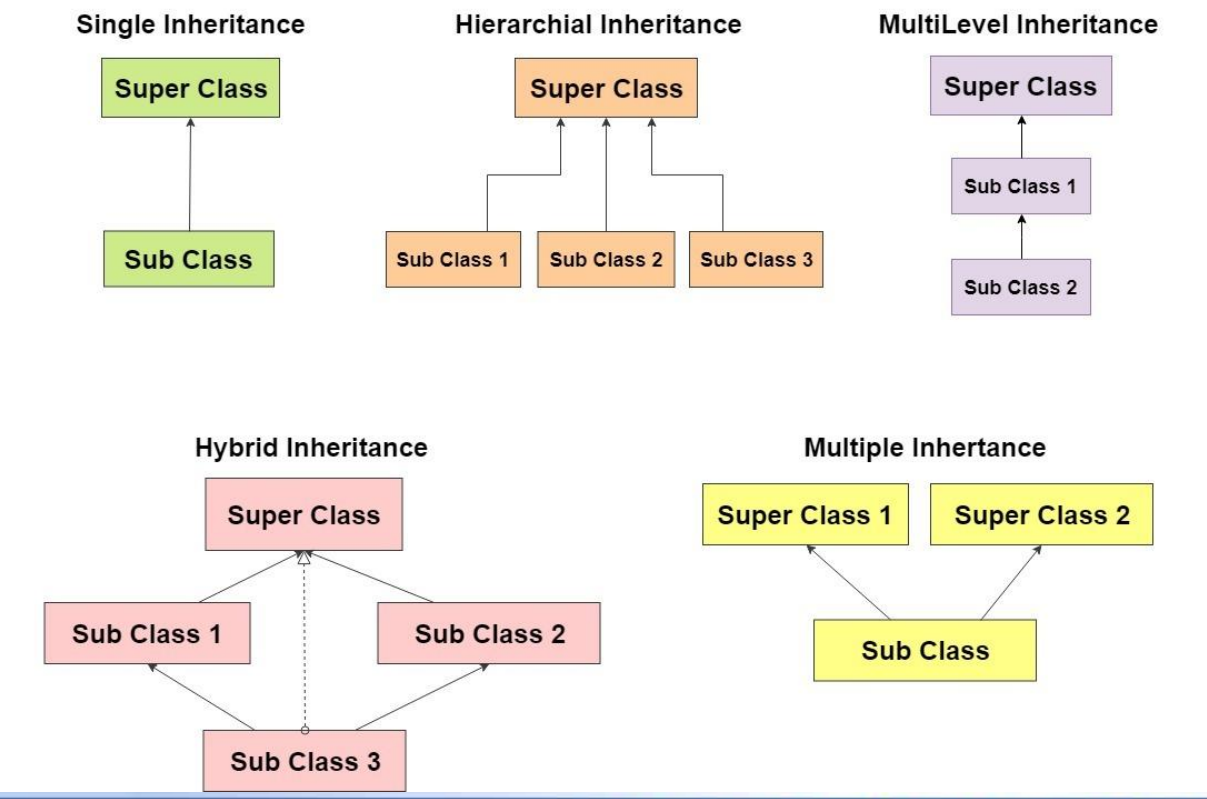
- Inheritance

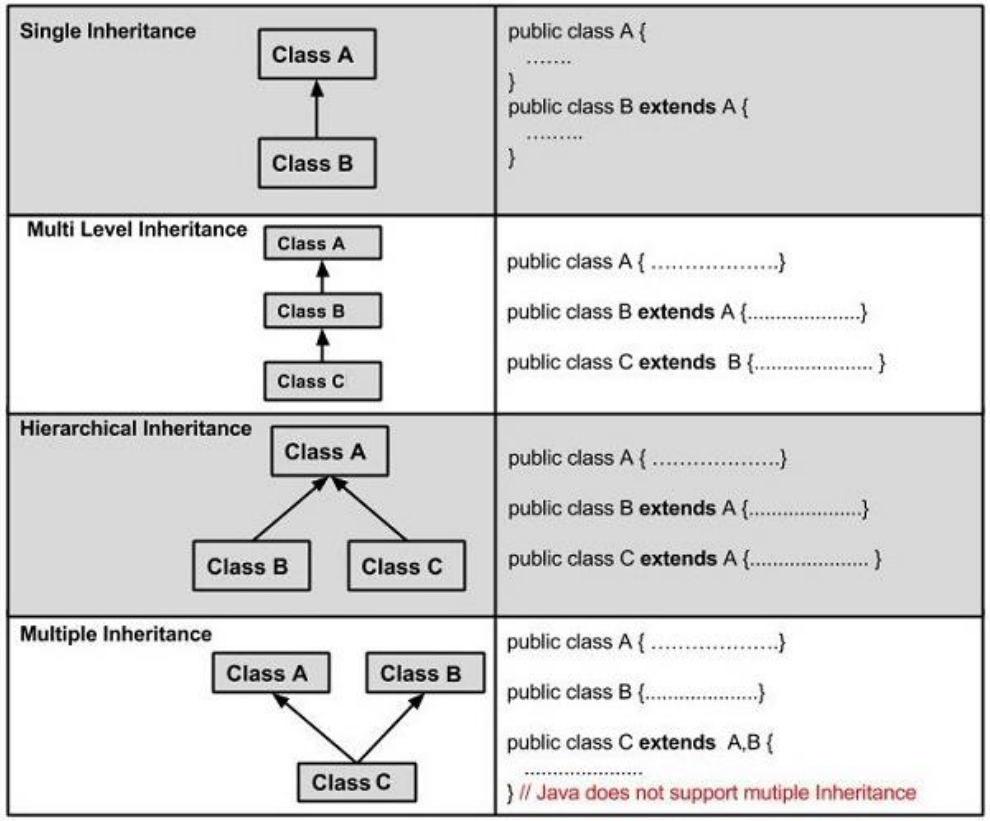
```
class A {
        protected int a;
        protected int b;
        public void method1(){
        }
}
class B extends A {
        private int c;
        public void method2(){
        }
}
```

String Handling in Java

- Inheritance

Example Program
Teacher.java, Student. Java, Use.java





- **Access** Control and **Inheritance**

  - Methods declared public in a superclass also must be public in all subclasses.
  - Methods declared protected in a superclass must either be protected or public in subclasses; they cannot be private.
  - Methods declared private are not **inherited** at all, so there is no **rule** for them.

  - Example
  - ProtectedDemo.java

String Handling in Java

- Inheritance – Super Keyword

- The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.
- Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword

- super can be used to refer immediate parent class instance variable.
- super can be used to invoke immediate parent class method.
- super() can be used to invoke immediate parent class constructor.

Example Program
Super1.java

String Handling in Java

- Inheritance – Constructors

- A subclass **inherits** all the members (fields, methods, and nested classes) from its superclass.
- **Constructors** are not members, so they are not **inherited** by subclasses, but the **constructor** of the superclass can be invoked from the subclass.
- If you don't declare a **constructor** of any type, a default is added

Example Program
TestDefaultConsDemo.java
ParameterizedConstructorDemo.java

String Handling in Java

- Inheritance – final keyword

The **final keyword** in java is used to restrict the user.
The java final keyword can be used in many context.

Final can be:

- variable
- method
- class

String Handling in Java

- Inheritance – final keyword

Java final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

```
class Bike9{
 final int speedlimit=90;//final variable
 void run(){
  speedlimit=400;
 }
 public static void main(String args[]){
 Bike9 obj=new  Bike9();
 obj.run();
 }
}
```

String Handling in Java

- Inheritance – final keyword

Java final method

If you make any method as final, you cannot override it.

```
class Bike{
  final void run(){System.out.println("running");}
}

class Honda extends Bike{
   void run(){System.out.println("running safely with 100kmph");}

   public static void main(String args[]){
   Honda honda= new Honda();
   honda.run();
   }
}
```

Output:Compile Time Error

String Handling in Java

- Inheritance – final keyword

Java final class

If you make any class as final, you cannot extend it.

```
final class Bike{}

class Honda1 extends Bike{
  void run(){System.out.println("running safely with 100kmph");}

  public static void main(String args[]){
  Honda1 honda= new Honda1();
  honda.run();
  }
}
```

Output:Compile Time Error

String Handling in Java

- Inheritance – final keyword

Is final method inherited?

Yes, final method is inherited but you cannot override it. For Example:

```
class Bike{
  final void run(){System.out.println("running...");}
}
class Honda2 extends Bike{
   public static void main(String args[]){
    new Honda2().run();
   }
}
```

Output:Compile Running

String Handling in Java

- Inheritance – final keyword

What is blank or uninitialized final variable?

- A final variable that is not initialized at the time of declaration is known as blank final variable.
- If you want to create a variable that is initialized at the time of creating object and once initialized may not be changed, it is useful.
- For example PAN CARD number of an employee.
- It can be initialized only in constructor.

```
class Student{
int id;
String name;
final String PAN_CARD_NUMBER;
...
}
```

String Handling in Java

- Inheritance – final keyword

Can we initialize blank final variable?

- Yes, but only in constructor. For example:

```
class Bike10{
  final int speedlimit;//blank final variable

  Bike10(){
  speedlimit=70;
  System.out.println(speedlimit);
  }

  public static void main(String args[]){
    new Bike10();
 }
}
```

String Handling in Java

- Inheritance – final keyword

static blank final variable

- A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block.

```
class A{
  static final int data;//static blank final variable
  static{ data=50;}
  public static void main(String args[]){
    System.out.println(A.data);
 }
}
```

String Handling in Java

- Inheritance – final keyword

What is final parameter?

- If you declare any parameter as final, you cannot change the value of it.

```
class Bike11{
  int cube(final int n){
   n=n+2;//can't be changed as n is final
   n*n*n;
  }
  public static void main(String args[]){
    Bike11 b=new Bike11();
    b.cube(5);
 }
}
```

String Handling in Java

- Inheritance – final keyword

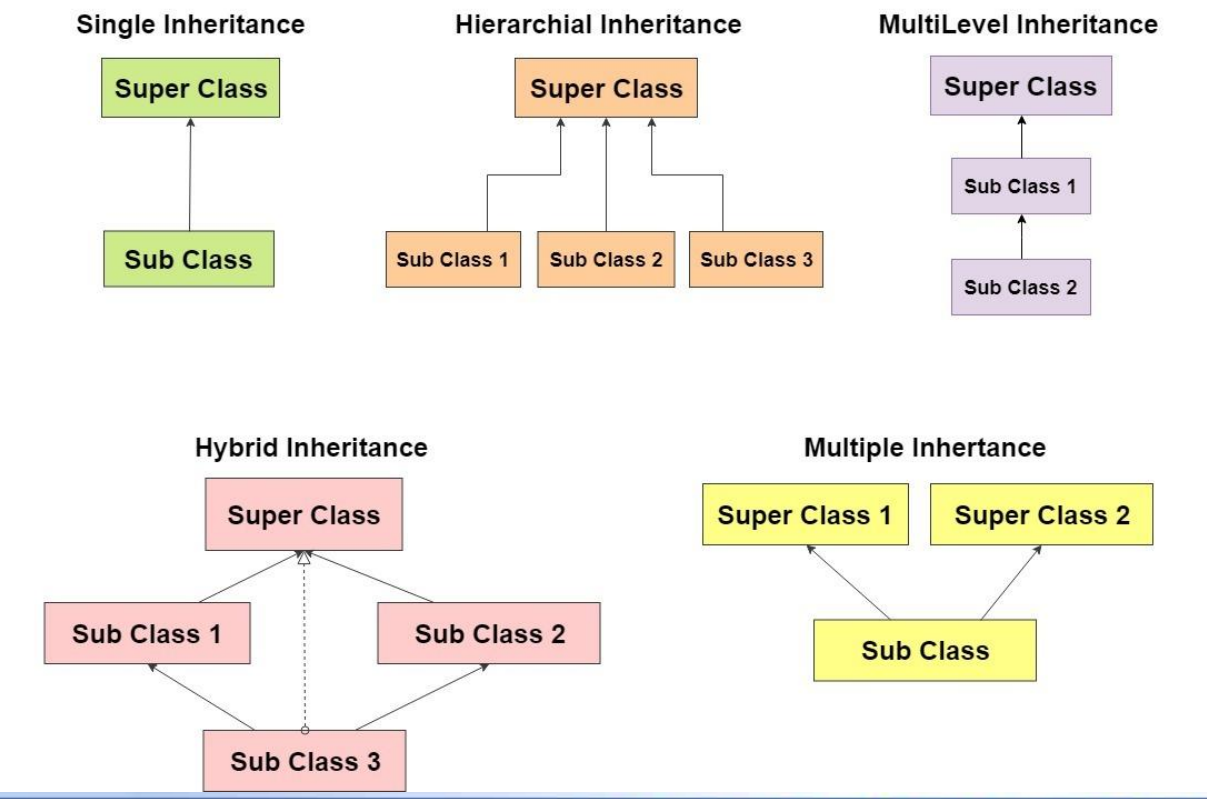
Can we declare a constructor final?

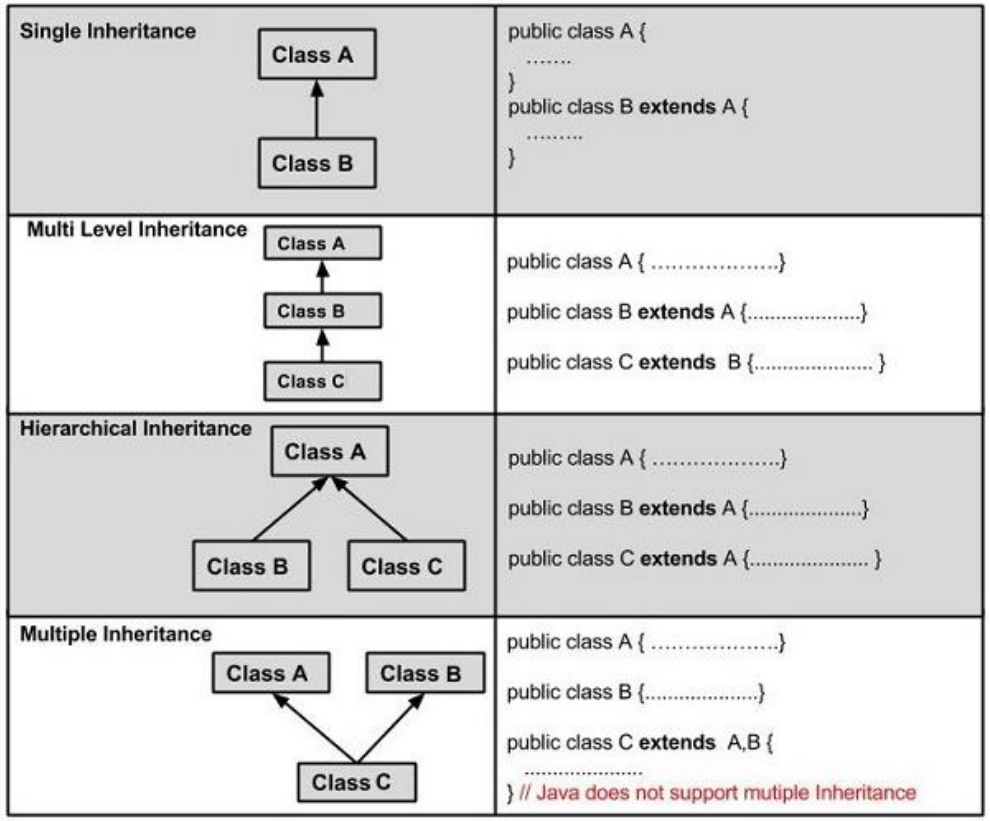
- No, because constructor is never inherited.

String Handling in Java

- Inheritance

Example Program
Teacher.java, Student. Java, Use.java





String Handling in Java

- Inheritance – Constructors

- A subclass **inherits** all the members (fields, methods, and nested classes) from its superclass.
- **Constructors** are not members, so they are not **inherited** by subclasses, but the **constructor** of the superclass can be invoked from the subclass.
- If you don't declare a **constructor** of any type, a default is added

Example Program
TestDefaultConsDemo.java
ParameterizedConstructorDemo.java

String Handling in Java

- Inheritance – final keyword

The **final keyword** in java is used to restrict the user.
The java final keyword can be used in many context.

Final can be:

- Final variable
- Final method
- Final class

String Handling in Java

- Inheritance – final keyword

Java final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

```
class Bike9{
 final int speedlimit=90;//final variable
 void run(){
  speedlimit=400;
 }
 public static void main(String args[]){
 Bike9 obj=new  Bike9();
 obj.run();
 }
}
```

String Handling in Java

- Inheritance – final keyword

Java final method

If you make any method as final, you cannot override it.

```
class Bike{
  final void run(){System.out.println("running");}
}

class Honda extends Bike{
   void run(){System.out.println("running safely with 100kmph");}

   public static void main(String args[]){
   Honda honda= new Honda();
   honda.run();
   }
}
```

Output:Compile Time Error

String Handling in Java

- Inheritance – final keyword

Java final class

If you make any class as final, you cannot extend it.

```
final class Bike{}

class Honda1 extends Bike{
  void run(){System.out.println("running safely with 100kmph");}

  public static void main(String args[]){
  Honda1 honda= new Honda1();
  honda.run();
  }
}
```

Output:Compile Time Error

String Handling in Java

- Inheritance – final keyword

Is final method inherited?

Yes, final method is inherited but you cannot override it. For Example:

```
class Bike{
  final void run(){System.out.println("running...");}
}
class Honda2 extends Bike{
   public static void main(String args[]){
    new Honda2().run();
   }
}
```

Output:Compile Running

String Handling in Java

- Inheritance – final keyword

What is blank or uninitialized final variable?

- A final variable that is not initialized at the time of declaration is known as blank final variable.
- If you want to create a variable that is initialized at the time of creating object and once initialized may not be changed, it is useful.
- For example PAN CARD number of an employee.
- It can be initialized only in constructor.

```
class Student{
int id;
String name;
final String PAN_CARD_NUMBER;
...
}
```

String Handling in Java

- Inheritance – final keyword

Can we initialize blank final variable?

- Yes, but only in constructor. For example:

```
class Bike10{
  final int speedlimit;//blank final variable

  Bike10(){
  speedlimit=70;
  System.out.println(speedlimit);
  }

  public static void main(String args[]){
    new Bike10();
 }
}
```

String Handling in Java

- Inheritance – final keyword

static blank final variable

- A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block.

```
class A{
  static final int data;//static blank final variable
  static{ data=50;}
  public static void main(String args[]){
    System.out.println(A.data);
 }
}
```

String Handling in Java

- Inheritance – final keyword

What is final parameter?

- If you declare any parameter as final, you cannot change the value of it.

```
class Bike11{
  int cube(final int n){
   n=n+2;//can't be changed as n is final
   n*n*n;
  }
  public static void main(String args[]){
    Bike11 b=new Bike11();
    b.cube(5);
 }
}
```

String Handling in Java

- Inheritance – final keyword

Can we declare a constructor final?

- No, because constructor is never inherited.

- **This, final, static, super.**

Polymorphism in Java

- Polymorphism-
  - ad hoc polymorphism,
  - pure polymorphism,
  - method overriding

Polymorphism in Java

- Polymorphism-
  - Polymorphism is derived from 2 Greek words: poly and morphs.
  - The word "poly" means many and "morphs" means forms.

  - So polymorphism means many forms.

Polymorphism in Java

- Polymorphism-
  - There are two types of polymorphism in Java:
    - compile-time polymorphism and
    - runtime polymorphism.

  - We can perform polymorphism in java by
    - method overloading and
    - method overriding.

Polymorphism in Java

Runtime Polymorphism in Java

Runtime polymorphism or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

In this process, an overridden method is called through the reference variable of a superclass.

The determination of the method to be called is based on the object being referred to by the reference variable.

Let's first understand the upcasting before Runtime Polymorphism.

Polymorphism in Java

```
class Bike{
  void run(){System.out.println("running");}
}
class Splendor extends Bike{
  void run(){System.out.println("running safely with 60km");}

  public static void main(String args[]){
    Bike b = new Splendor();//upcasting
    b.run();
  }
}
```

Polymorphism in Java

```
class Bank{
float getRateOfInterest(){return 0;}
}
class SBI extends Bank{
float getRateOfInterest(){return 8.4f;}
}
class ICICI extends Bank{
float getRateOfInterest(){return 7.3f;}
}
class AXIS extends Bank{
float getRateOfInterest(){return 9.7f;}
}
```

Polymorphism in Java

```
class TestPolymorphism{
public static void main(String args[]){
Bank b;
b=new SBI();
System.out.println("SBI Rate of Interest: "+b.getRateOfInterest());
b=new ICICI();
System.out.println("ICICI Rate of Interest: "+b.getRateOfInterest());
b=new AXIS();
System.out.println("AXIS Rate of Interest: "+b.getRateOfInterest());
}
}
```

Polymorphism in Java

```
class Shape{
void draw(){System.out.println("drawing...");}
}
class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle...");}
}
class Circle extends Shape{
void draw(){System.out.println("drawing circle...");}
}
class Triangle extends Shape{
void draw(){System.out.println("drawing triangle...");}
}
```

Polymorphism in Java

```
class TestPolymorphism2{
public static void main(String args[]){
Shape s;
s=new Rectangle();
s.draw();
s=new Circle();
s.draw();
s=new Triangle();
s.draw();
}
}
```

Polymorphism in Java

Static Binding and Dynamic Binding

Connecting a method call to the method body is known as binding.

There are two types of binding

- Static Binding (also known as Early Binding).
- Dynamic Binding (also known as Late Binding).

Polymorphism in Java

static binding

When type of the object is determined at compiled time(by the compiler), it is known as static binding.
If there is any private, final or static method in a class, there is static binding.

```
class Dog{
 private void eat(){System.out.println("dog is eating...");}

 public static void main(String args[]){
  Dog d1=new Dog();
  d1.eat();
 }
}
```

Polymorphism in Java

Dynamic binding

When type of the object is determined at run-time, it is known as dynamic binding.

```
class Animal{
 void eat(){System.out.println("animal is eating...");}
}
  class Dog extends Animal{
 void eat(){System.out.println("dog is eating...");}

   public static void main(String args[]){
  Animal a=new Dog();
  a.eat();
 }
}
```

object type cannot be determined by the compiler, because the instance of Dog is also an instance of Animal.So compiler doesn't know its type, only its base type.

Polymorphism in Java

- Polymorphism-
  - ad hoc polymorphism
  - pure polymorphism
  - method overriding

Polymorphism in Java

- Polymorphism-
  - Polymorphism is derived from 2 Greek words: poly and morphs.
  - The word "poly" means many and "morphs" means forms.

  - So polymorphism means many forms.

Polymorphism in Java

- Polymorphism- **Ad hoc polymorphism**
- The ad hoc polymorphism is a technique used to define the same method with different implementations and different arguments. In a java programming language, ad hoc polymorphism carried out with a method overloading concept.
- In ad hoc polymorphism the method binding happens at the time of compilation. Ad hoc polymorphism is also known as compile-time polymorphism.
- Every function call binded with the respective overloaded method based on the arguments.

The ad hoc polymorphism implemented within the class only.

Polymorphism in Java

- Polymorphism- **Pure polymorphism**

The pure polymorphism is a technique used to define the same method with the same arguments but different implementations.
In a java programming language, pure polymorphism carried out with a method overriding concept.
In pure polymorphism, the method binding happens at run time.

Pure polymorphism is also known as run-time polymorphism.

Every function call binding with the respective overridden method based on the object reference.

Polymorphism in Java

Runtime Polymorphism in Java

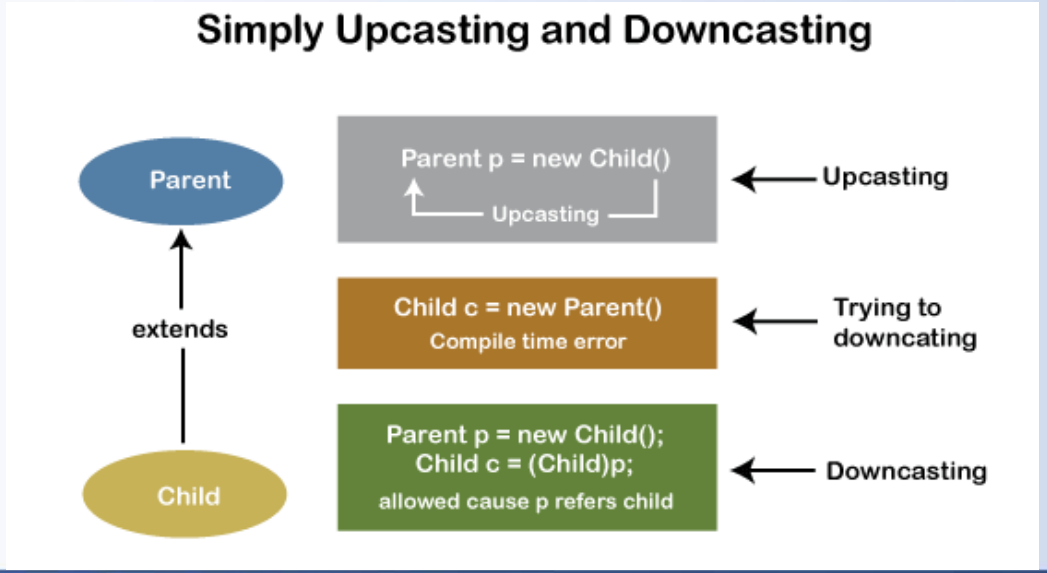
Runtime polymorphism or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

- In this process, an overridden method is called through the reference variable of a superclass.
- The determination of the method to be called is based on the object being referred to by the reference variable.
- Let's first understand the upcasting before Runtime Polymorphism.

Polymorphism in Java

```
class Bike{
      void run(){
     System.out.println("running");
     }
}
class Splendor extends Bike{
      void run(){
     System.out.println("running safely with 60km");
     }

  public static void main(String args[]){
        Bike b = new Splendor();//upcasting
        b.run();
      }
}
```



Polymorphism in Java

```
class Bank{
        float getRateOfInterest(){return 0;}
}
class SBI extends Bank{
        float getRateOfInterest(){return 8.4f;}
}
class ICICI extends Bank{
        float getRateOfInterest(){return 7.3f;}
}
class AXIS extends Bank{
        float getRateOfInterest(){return 9.7f;}
}
```

Polymorphism in Java

```
class TestPolymorphism{
public static void main(String args[]){
    Bank b;
    b=new SBI();
    System.out.println("SBI Rate of Interest: "+b.getRateOfInterest());
    b=new ICICI();
    System.out.println("ICICI Rate of Interest: "+b.getRateOfInterest());
    b=new AXIS();
    System.out.println("AXIS Rate of Interest: "+b.getRateOfInterest());
}
}
```

Polymorphism in Java

```
class Shape{
        void draw(){System.out.println("drawing...");}
}
class Rectangle extends Shape{
        void draw(){System.out.println("drawing rectangle...");}
}
class Circle extends Shape{
        void draw(){System.out.println("drawing circle...");}
}
class Triangle extends Shape{
        void draw(){System.out.println("drawing triangle...");}
}
```

Polymorphism in Java

```
class TestPolymorphism2{
public static void main(String args[]){
        Shape s;
    s=new Rectangle();
    s.draw();
    s=new Circle();
    s.draw();
    s=new Triangle();
    s.draw();
}
}
```

Polymorphism in Java

Static Binding and Dynamic Binding

Connecting a method call to the method body is known as binding.

There are two types of binding

- Static Binding (also known as Early Binding).
- Dynamic Binding (also known as Late Binding).

Polymorphism in Java

VISHNU
UNIVERSAL LEARNING

static binding

When type of the object is determined at compiled time(by the compiler), it is known as static binding.
If there is any private, final or static method in a class, there is static binding.

```
class Dog{
 private void eat(){System.out.println("dog is eating...");}

 public static void main(String args[]){
  Dog d1=new Dog();
  d1.eat();
 }
}
```

Polymorphism in Java

Dynamic binding
When type of the object is determined at run-time, it is known as dynamic binding.

```
class Animal{
 void eat(){System.out.println("animal is eating...");}
}
  class Dog extends Animal{
 void eat(){System.out.println("dog is eating...");}

   public static void main(String args[]){
  Animal a=new Dog();
  a.eat();
 }
}
```

object type cannot be determined by the compiler, because the instance of Dog is also an instance of Animal. So compiler doesn't know its type, only its base type.

Abstract Class in Java

- Class is model for creating objects.
- A Class consist of description of properties or variables and actions or methods of its objects.
- If an object has the properties and actions as mentioned in the class, then that object belongs to the class. The rule is that any thing is written in the class is applicable to all of its objects.
- If a method is written in the class, it is available as it is to all of the class objects.

Abstract Class in Java

- For example, take a class Myclass that contains a method calculate () that calculates square value of a given number.

- If we create three objects to this class, all the three objects get the copy of this method and hence, from any object, we can call and use this method. See Program 1.

Abstract Class in Java

- Write a program where Myclass's calculate () method is available to all the objects and hence every object can calculate the square value

```
class MyClass {
        void calculate(double x){
                System.out.println("Square =" + (x*x));
        }
}
class Common {
public static void main (String args[]){
    MyClass obj1 = new MyClass();
    MyClass obj2 = new MyClass();
    MyClass obj3 = new MyClass();

    obj1.calculate(3);
    obj2.calculate(4);
    obj3.calculate(5);
}
}
```

Abstract Class in Java

- Of course, in the preceding program, the requirement of all the objects is same, ie., to calculate square value.
- But, some times the requirement of the objects will be different and entirely dependent on the specific object only.
- For example, in the preceding program, if the first obj wants to calculate square value, the second object wants the square root value and the third obj wants cube value.
- In such a case, how to write the calculate () method in Myclass ? Since, calculate() method has to perform three different tasks depending on the object, we Can write the code to calculate square value in the body of calculate() method.
- On the other hand, if write three different methods like calculate_Square () , calculate_Sqrt () , calculate_Cube () in Myclass, then all the three methods are available to all the three objects which is not advisable. When each object wants one method, providing all the three does not reasonable.

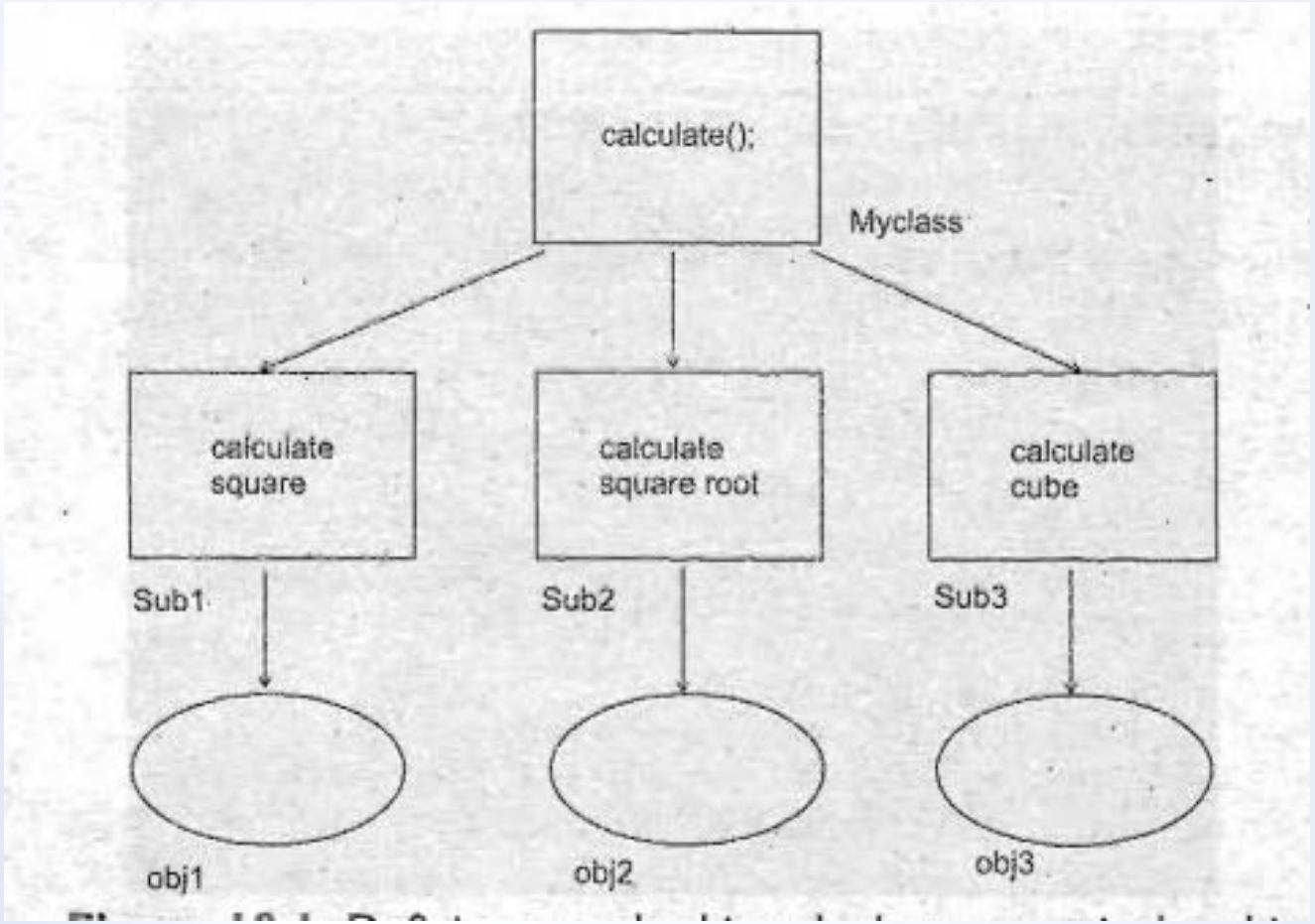
Abstract Class in Java

- To serve each object with the one and only required method, we can follow the steps:

- First, let us write a calculate () method in Myclass. This means every object wants calculate something.
- 2. If we write body for calculate () method, it is commonly available to all the objects. So lets not write body for calculate () method. Such a method is called abstract method. Since, write abstract method in Myclass, it is called abstract class.
- 3. Now derive a Subclass from Myclass, so that the calculate () method is available to sub class. Provide body for calculate () method in Sub1 such that it calculates square value Similarly, we create another sub class Sub2 where we write the calculate () method with to calculate square root value. We create the third sub class Sub3 where we write calculate () method to calculate cube value

Abstract Class in Java

- This hierarchy is shown in Figure. It is possible to create objects for the sub classes. Using these objects, the respective method can be called and used. Thus, every object will have its requirement fulfilled.



Abstract Class in Java

- Abstract Method and Abstract Class

- An Abstract method does not contain any body
- It contains only the method header. So we can say that it is incomplete method.

- An abstract class is a class that generally contains some methods. Both the abstract class and the abstract methods should be declared by using the word 'abstract'

- Since, abstract class contains incomplete methods, it is not possible to estimate the total memory required to create the object. So, JVM can not create objects to an abstract class.
- We should create sub classes and all the abstract methods should be implemented (body should be written) in the sub classes.
- Then, it is possible to create objects to-the sub classes since they are complete classes.

Abstract Class in Java

- Let us make a program where- the abstract class Myclass -has one a:bstract method which has got various implementations in sub classes.

```
abstract class MyClass {
        abstract void calculate(double x);
}
class sub1 extends MyClass{
        void calculate(double x){
                System.out.println("Square is : " + (x*x));
        }
}
class sub2 extends MyClass{
        void calculate(double x){
                System.out.println("Square root is : " +Math.sqrt(x));
        }
}
class sub3 extends MyClass{
        void calculate(double x){
                System.out.println("Cube is : " + (x*x*x));
        }
}
```

Abstract Class in Java

```
class Different {
public static void main (String args[]) {
        Sub1 obj1 = new Sub1();
        Sub2 obj2 = new Sub2();
        Sub3 obj3 = new Sub3();

        obj1.calculate(3);
        obj2.calculate(4);
        obj3.calculate(5);
}
}
```

Output:

Square is : 9.0

Square root is : 2.0

Cube is : 125.0

Object class in Java

Object class in Java

The **Object class** is the parent class of all the classes in java by default. In other words, it is the topmost class of java.

- The Object class is beneficial if you want to refer any object whose type you don't know. Notice that parent class reference variable can refer the child class object, know as upcasting.
- Let's take an example, there is getObject() method that returns an object but it can be of any type like Employee, Student etc, we can use Object class reference to refer that object.

- For example:

Object obj=getObject();//we don't know what object will be returned from this method

Object class in Java

Object class in Java

```
public class ObjectClassDemo {
public static void main(String[] args) {
    ObjectClassDemo obj = new ObjectClassDemo();

    System.out.println(obj.getClass());
    System.out.println(obj.hashCode());
}
}
```

Object class in Java

Object class methods

Method	Description
public final Class getClass()	returns the Class class object of this object. The Class class can further be used to get the metadata of this class.
public int hashCode()	returns the hashcode number for this object.
public boolean equals(Object obj)	compares the given object to this object.
protected Object clone() throws CloneNotSupportedException	creates and returns the exact copy (clone) of this object.
public String toString()	returns the string representation of this object.
public final void notify()	wakes up single thread, waiting on this object's monitor.
public final void notifyAll()	wakes up all the threads, waiting on this object's monitor.
public final void wait(long timeout)throws InterruptedException	causes the current thread to wait for the specified milliseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait(long timeout,int nanos)throws InterruptedException	causes the current thread to wait for the specified milliseconds and nanoseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait()throws InterruptedException	causes the current thread to wait, until another thread notifies (invokes notify() or notifyAll() method).
protected void finalize()throws Throwable	is invoked by the garbage collector before object is being garbage collected.

Forms of Inheritance

- The inheritance concept used for the number of purposes in the java programming language.

- One of the main purposes is substitutability.
- The substitutability means that when a child class acquires properties from its parent class, the object of the parent class may be substituted with the child class object.

- For example, if B is a child class of A, anywhere we expect an instance of A we can use an instance of B.

Forms of Inheritance

The substitutability can achieve using inheritance, whether using extends or implements keywords.

The following are the different forms of inheritance in java.

- Specialization
- Specification
- Construction
- Extension
- Limitation
- Combination

Forms of Inheritance

Specialization

It is the most ideal form of inheritance. The subclass is a special case of the parent class. It holds the principle of substitutability.

Specification

This is another commonly used form of inheritance. In this form of inheritance, the parent class just specifies which methods should be available to the child class but doesn't implement them. The java provides concepts like abstract and interfaces to support this form of inheritance. It holds the principle of substitutability.

Construction

This is another form of inheritance where the child class may change the behavior defined by the parent class (overriding). It does not hold the principle of substitutability.

Forms of Inheritance

Extension

This is another form of inheritance where the child class may add its new properties. It holds the principle of substitutability.

Limitation

This is another form of inheritance where the subclass restricts the inherited behavior. It does not hold the principle of substitutability.

Combination

This is another form of inheritance where the subclass inherits properties from multiple parent classes. Java does not support multiple inheritance type.