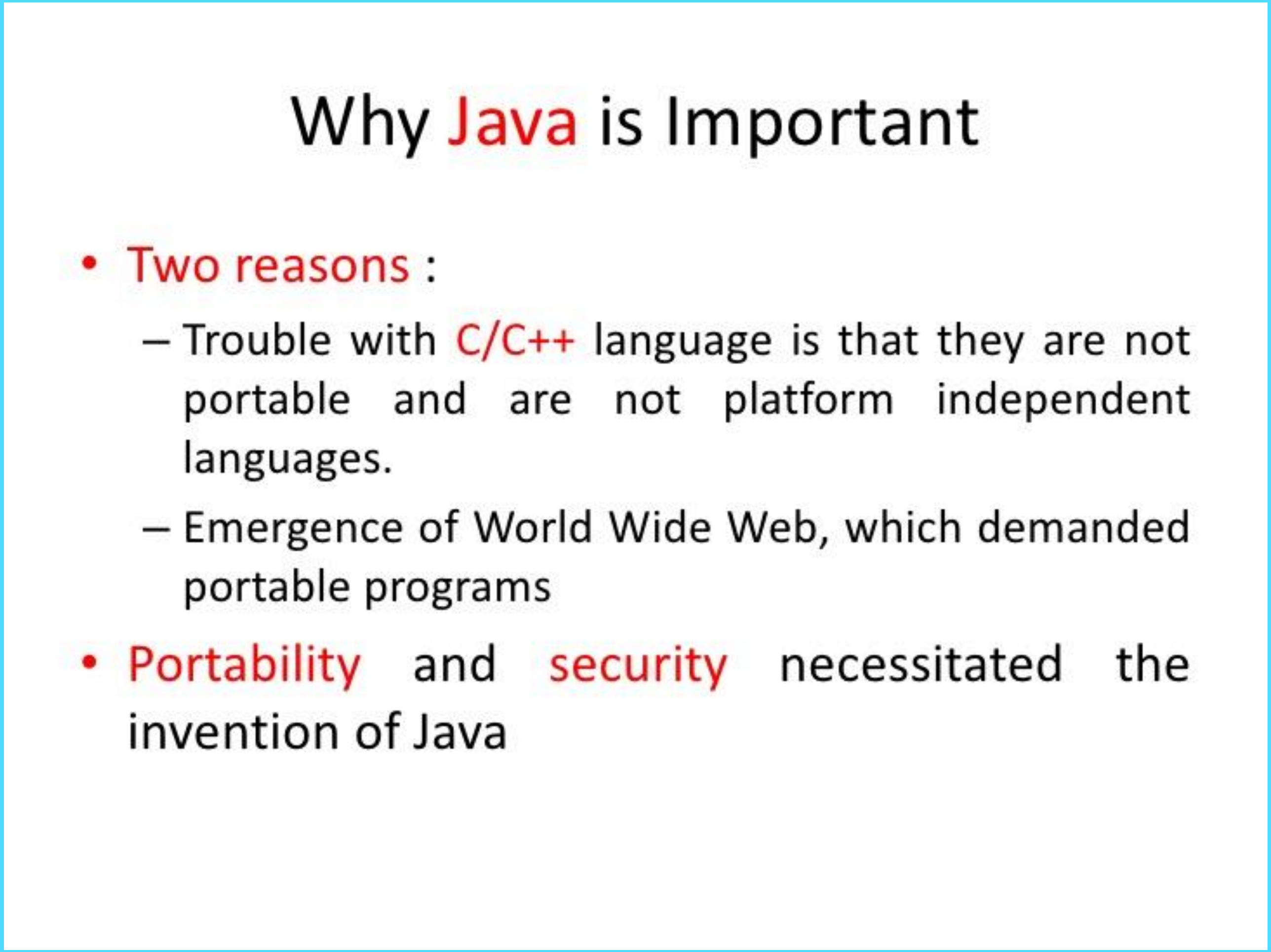
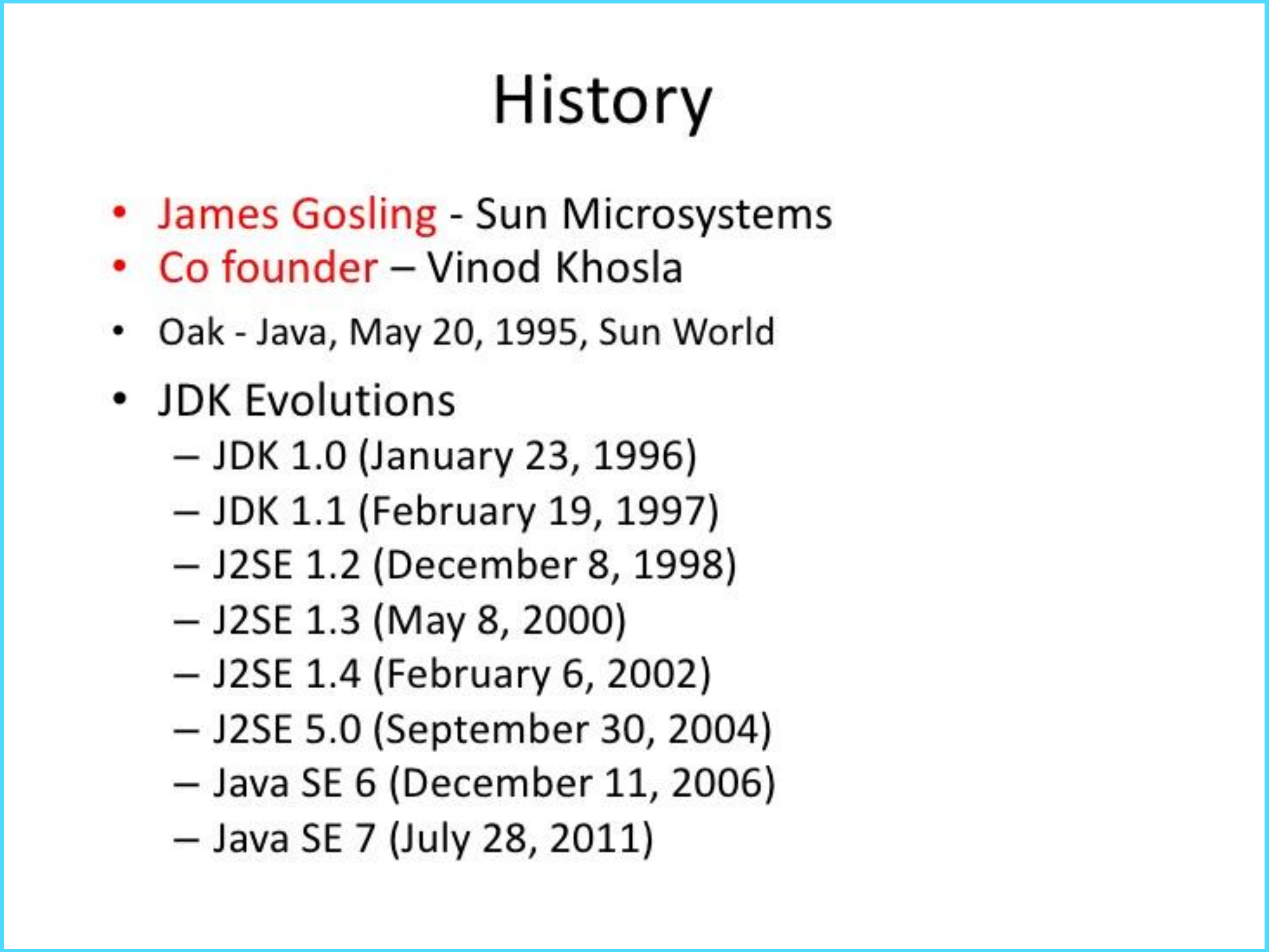


Why Java is Important

- Two reasons :
  - Trouble with C/C++ language is that they are not portable and are not platform independent languages.
  - Emergence of World Wide Web, which demanded portable programs
- Portability and security necessitated the invention of Java

Core Java Tutorial

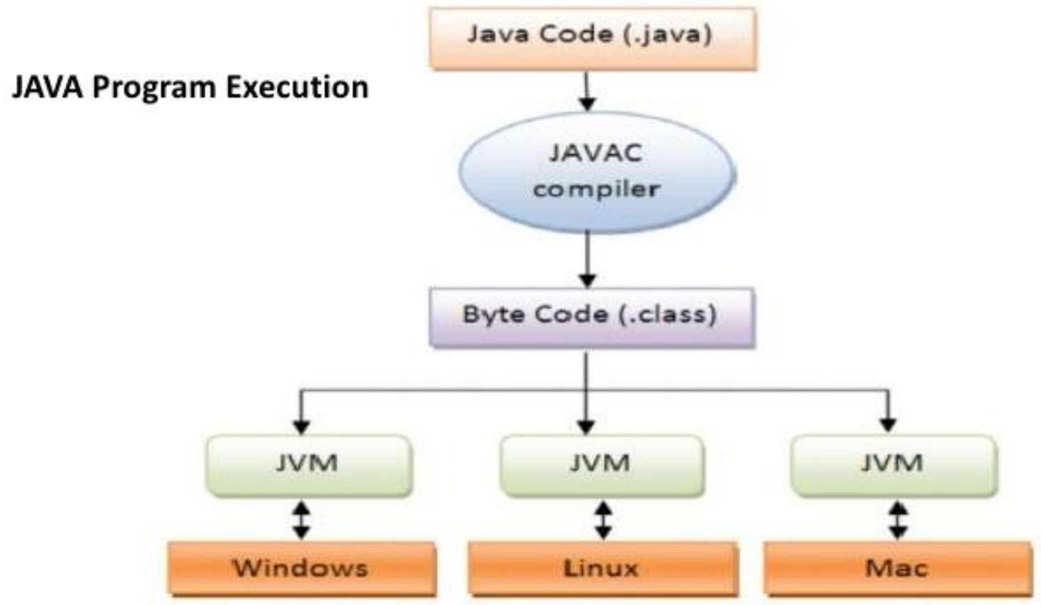


History

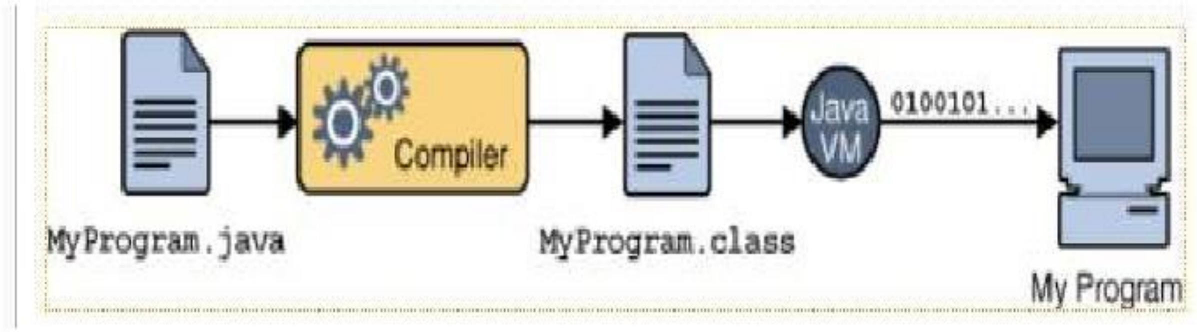
- James Gosling - Sun Microsystems
- Co founder – Vinod Khosla
- Oak - Java, May 20, 1995, Sun World
- JDK Evolutions
  - JDK 1.0 (January 23, 1996)
  - JDK 1.1 (February 19, 1997)
  - J2SE 1.2 (December 8, 1998)
  - J2SE 1.3 (May 8, 2000)
  - J2SE 1.4 (February 6, 2002)
  - J2SE 5.0 (September 30, 2004)
  - Java SE 6 (December 11, 2006)
  - Java SE 7 (July 28, 2011)

Core Java Tutorial

Java is architecture-neutral



WORA(Write Once Run Anywhere)



Core Java Tutorial

- What is Java?

Java is a **programming language** and a **platform**.

Platform Any hardware or software environment in which a program runs, known as a platform. Since Java has its own Runtime Environment (JRE) and API, it is called platform.

Where it is used?

- Desktop Applications such as acrobat reader, media player, antivirus etc.
- Web Applications such as irctc.co.in, javatpoint.com etc.
- Enterprise Applications such as banking applications.
- Mobile
- Embedded System
- Smart Card
- Robotics
- Games etc.

- Types of Java Applications

- **Standalone Application**
- **Web Application**
- **Enterprise Application**
- **Mobile Application**

Features of Java

- Simple
- Object-Oriented
- Platform Independent
- secured
- Robust
- Architecture Neutral
- Portable
- High Performance
- Distributed
- Multi-threaded

- Hello Java Example

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

save this file as Simple.java

To compile: javac Simple.java

To execute: java Simple

Output: Hello Java

- Understanding first java program

- **class** is used to declare a class in java.

- **public** is an access modifier which represents visibility, it means it is visible to all.

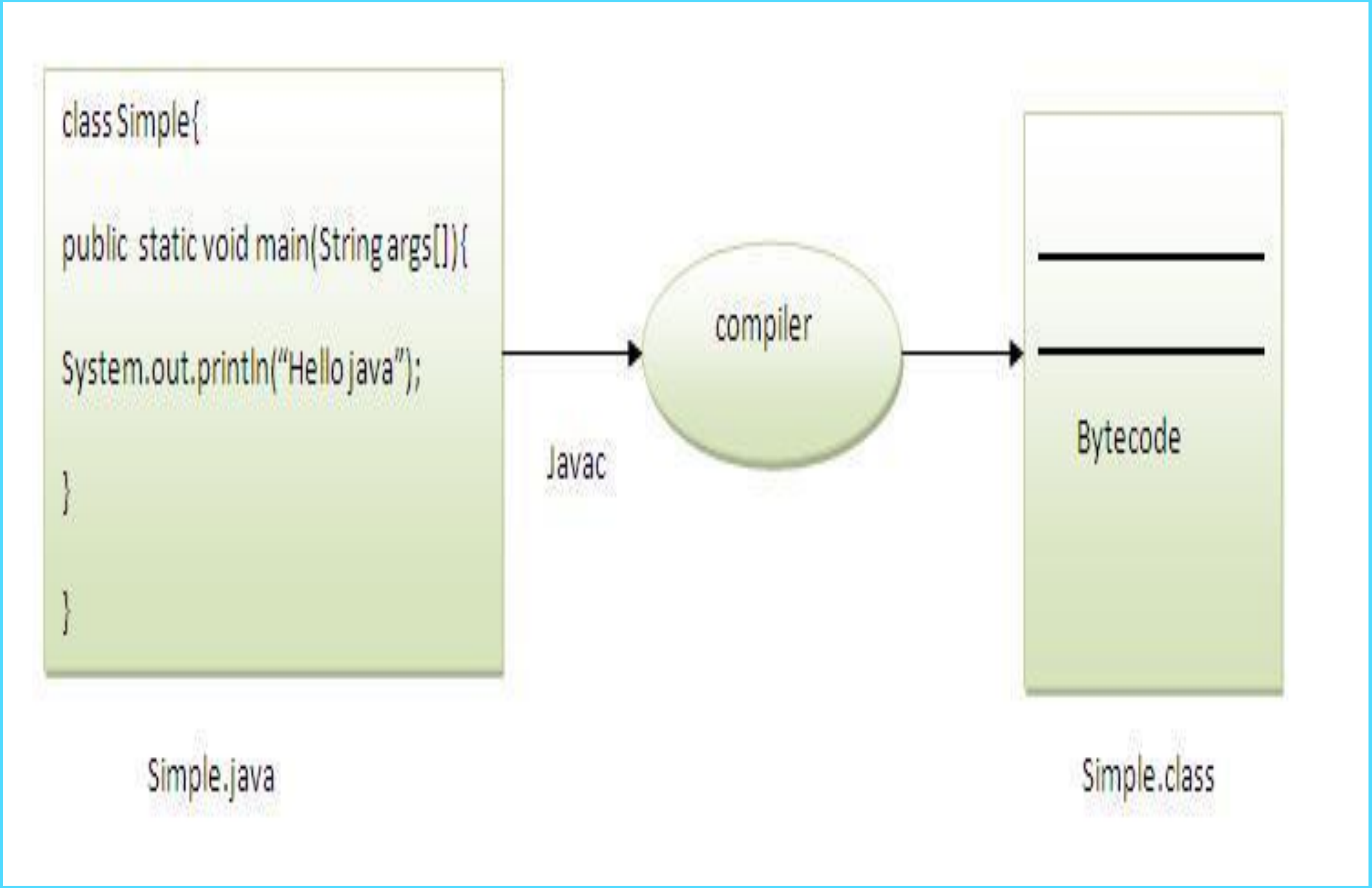
- **Static** is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create object to invoke the main method. So it saves memory.

- **void** is the return type of the method, it means it doesn't return any value.

- **main** represents start up of the program.

- **String[] args** is used for command line argument.

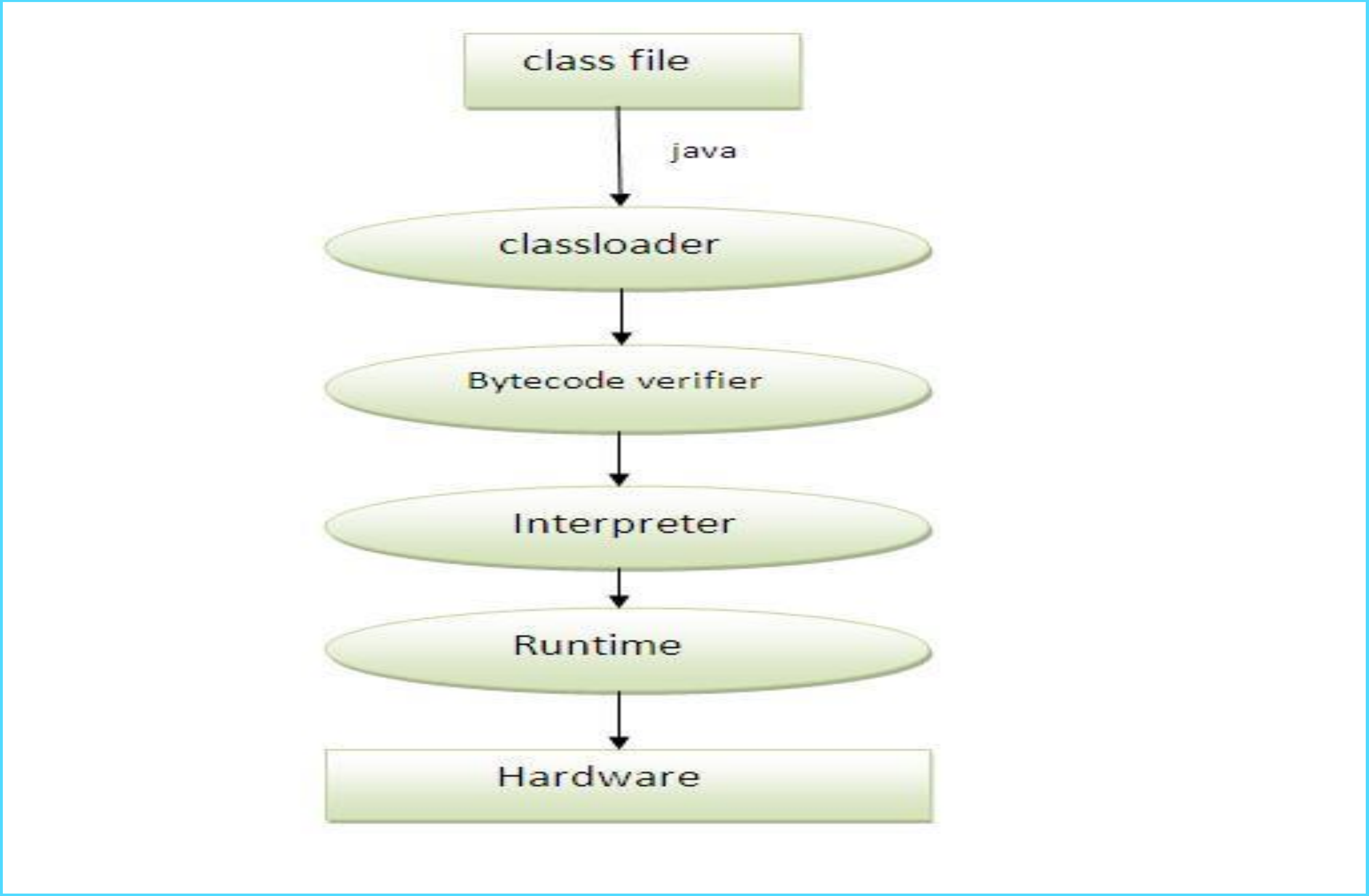
- **System.out.println()** is used print statement.



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

- What happens at compile time?

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


class file
java
classloader
Bytecode verifier
Interpreter
Runtime
Hardware

- What happens at runtime?

At runtime, following steps are performed:

- **Class loader:** is the subsystem of JVM that is used to load class files.
- **Byte code Verifier:** checks the code fragments for illegal code that can violate access right to objects.
- **Interpreter:** read byte code stream then execute the instructions.

- How to set path of JDK in Windows:

There are two ways to set path of JDK:

- temporary
- permanent

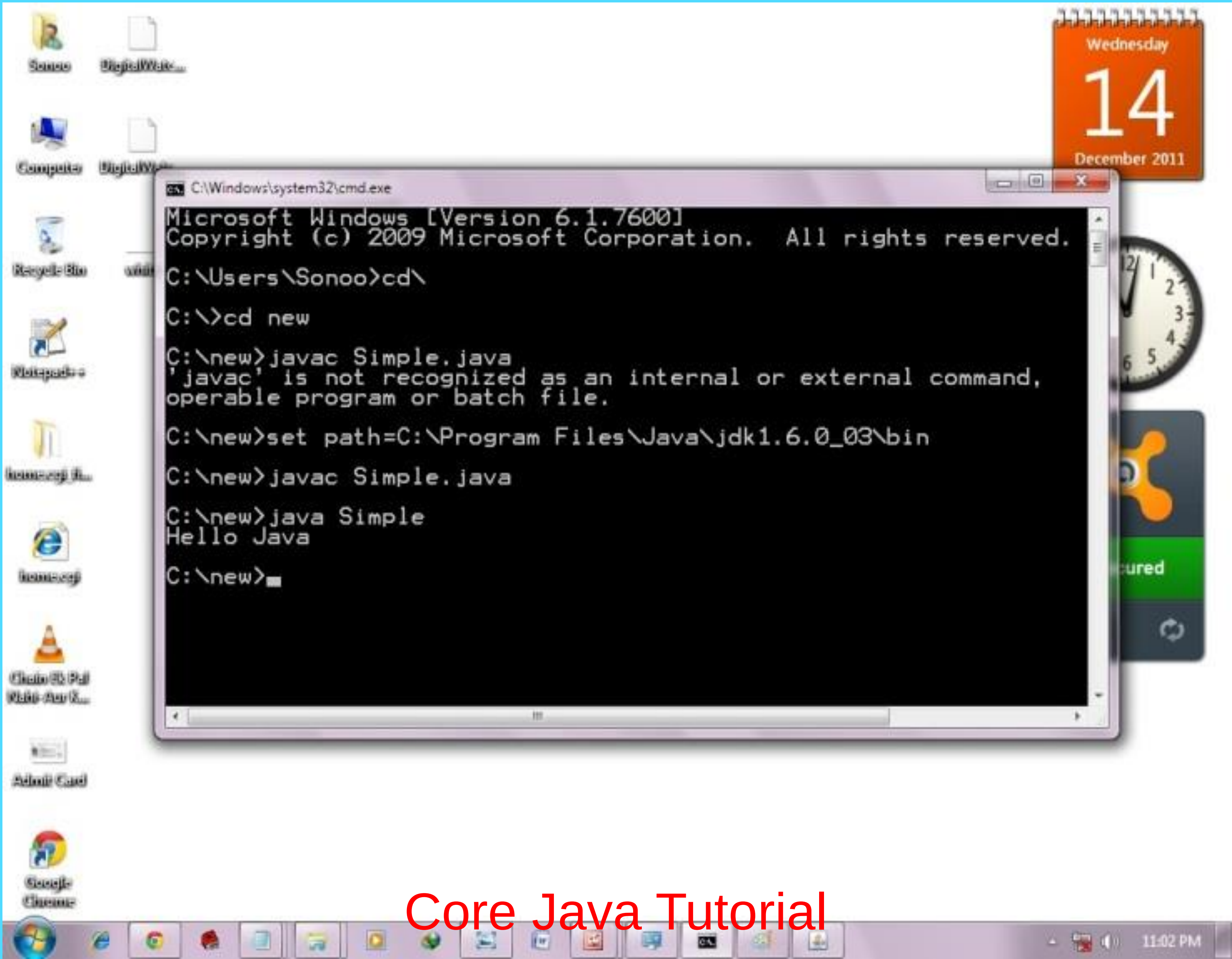
- Setting temporary Path of JDK in Windows:

For setting the temporary path of JDK, you need to follow these steps:

- Open command prompt
- copy the path of bin folder
- write in command prompt:
- set path=copied path

For Example:

set path=C:\Program Files\Java\jdk1.6.0_23\bin



```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Sonoo>cd\

C:\>cd new

C:\new>javac Simple.java
'javac' is not recognized as an internal or external command,
operable program or batch file.

C:\new>set path=C:\Program Files\Java\jdk1.6.0_03\bin

C:\new>javac Simple.java

C:\new>java Simple
Hello Java

C:\new>
```

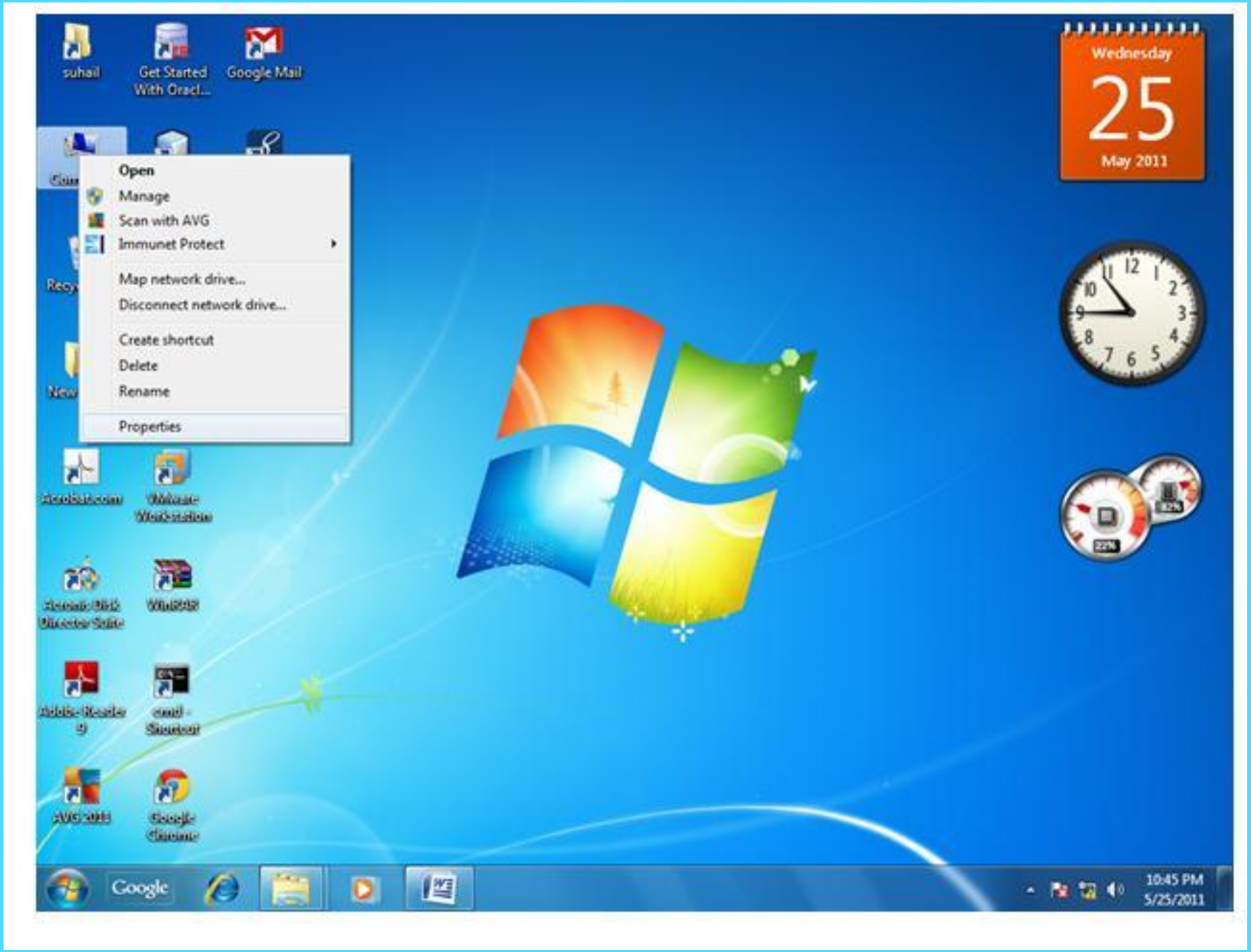
Core Java Tutorial

- ## Setting Permanent Path of JDK in Windows:

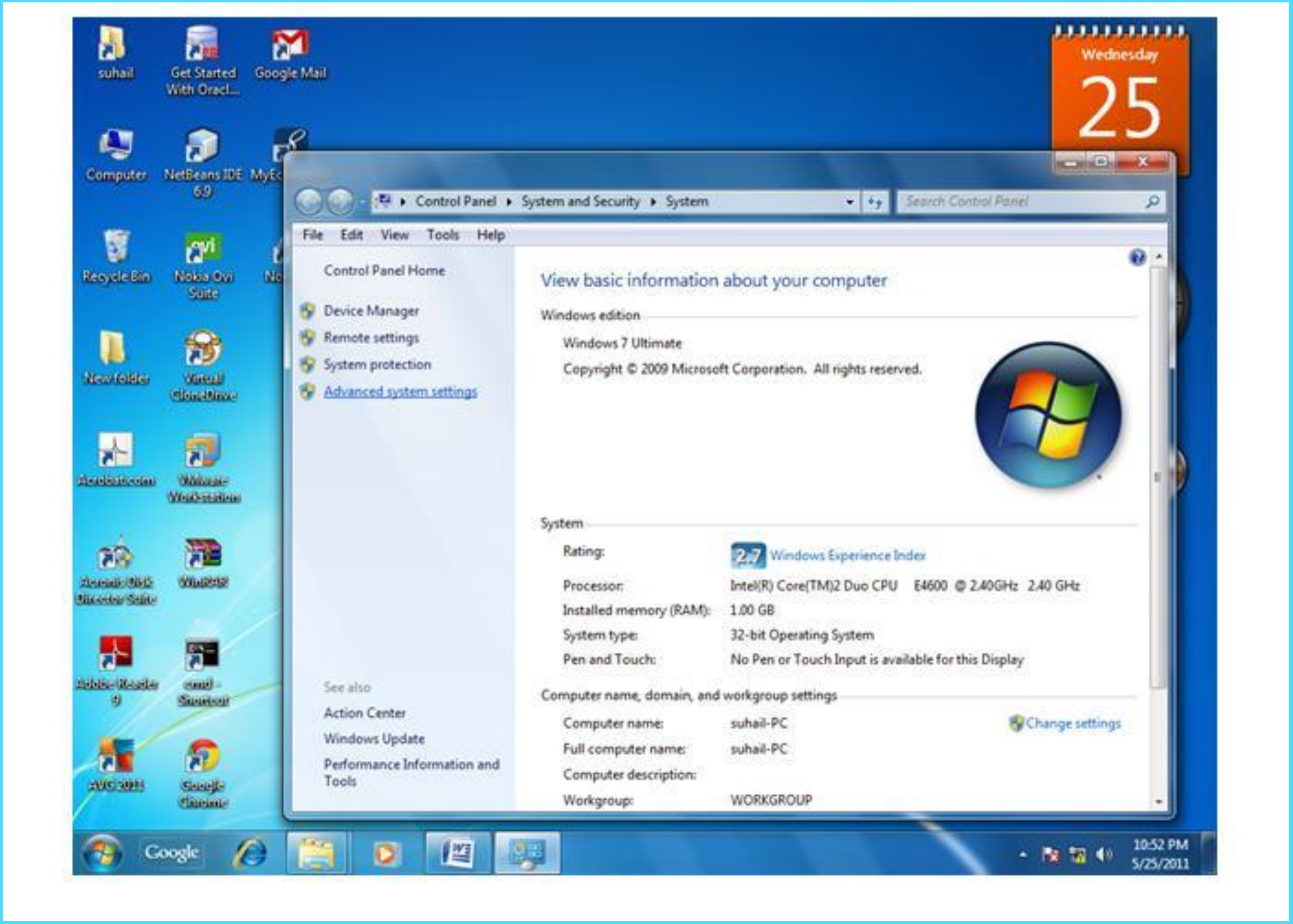
For setting the permanent path of JDK, you need to follow these steps:

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

• Go to MyComputer properties



- **click on advanced tab**



View basic information about your computer

Windows edition

Windows 7 Ultimate

Copyright © 2009 Microsoft Corporation. All rights reserved.

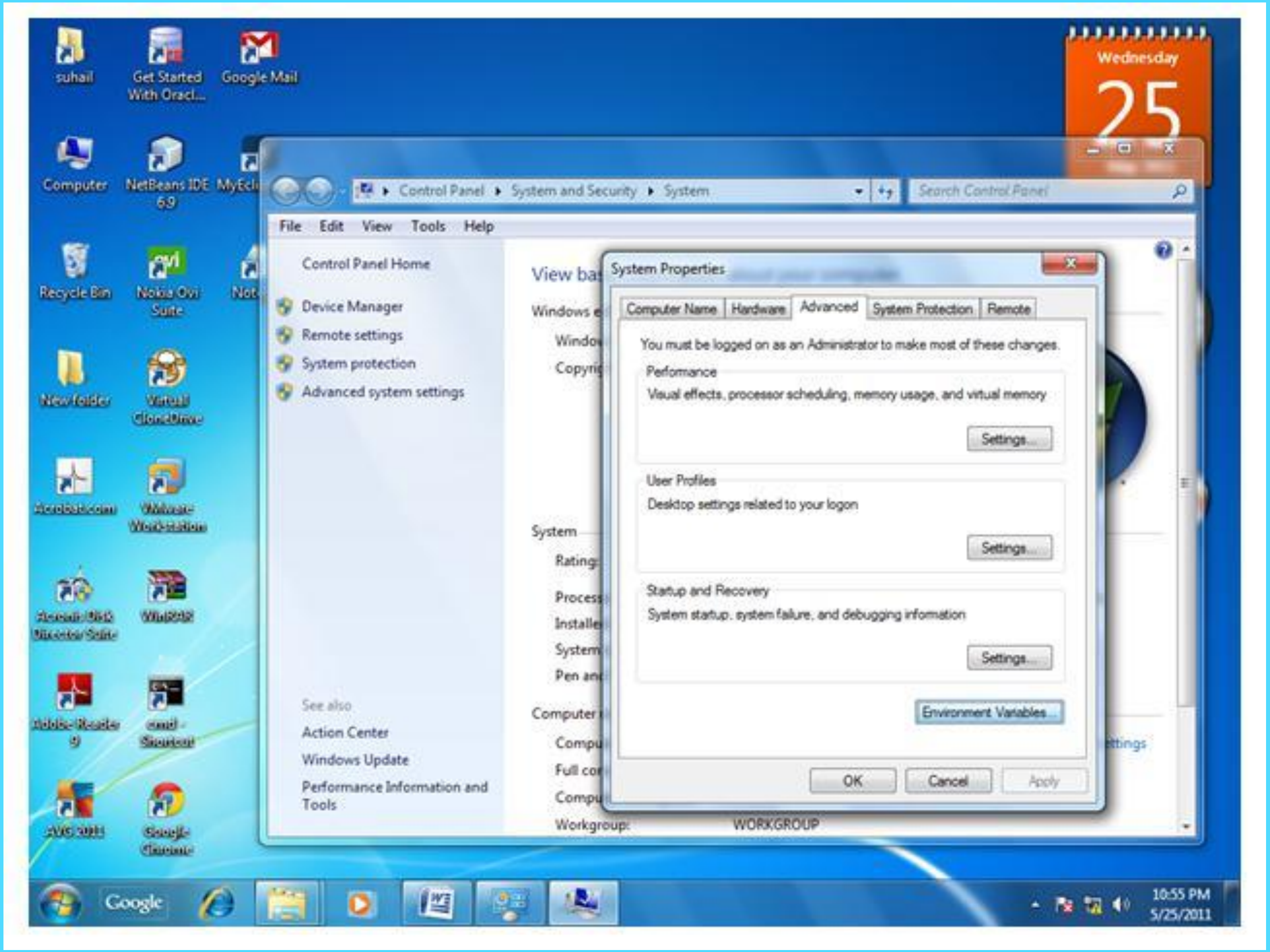
System

Rating:	2.7 Windows Experience Index
Processor:	Intel(R) Core(TM)2 Duo CPU E4600 @ 2.40GHz 2.40 GHz
Installed memory (RAM):	1.00 GB
System type:	32-bit Operating System
Pen and Touch:	No Pen or Touch Input is available for this Display

Computer name, domain, and workgroup settings

Computer name:	suhail-PC
Full computer name:	suhail-PC
Computer description:	
Workgroup:	WORKGROUP

- **click on environment variables**


System Properties

You must be logged on as an Administrator to make most of these changes.

Performance

Visual effects, processor scheduling, memory usage, and virtual memory

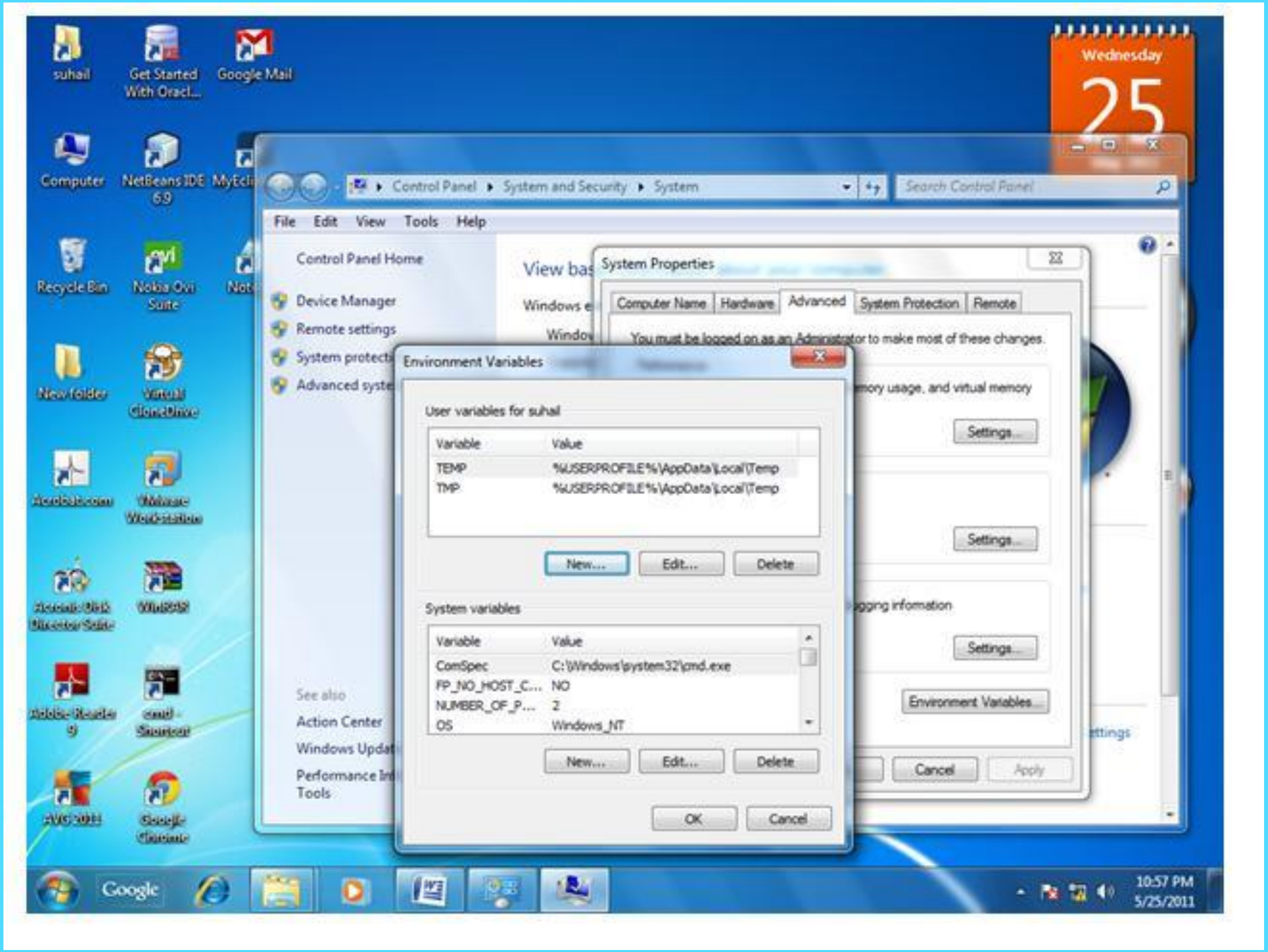
User Profiles

Desktop settings related to your logon

Startup and Recovery

System startup, system failure, and debugging information

- **click on new tab of user variables**



You must be logged on as an Administrator to make most of these changes.

User variables for suhail

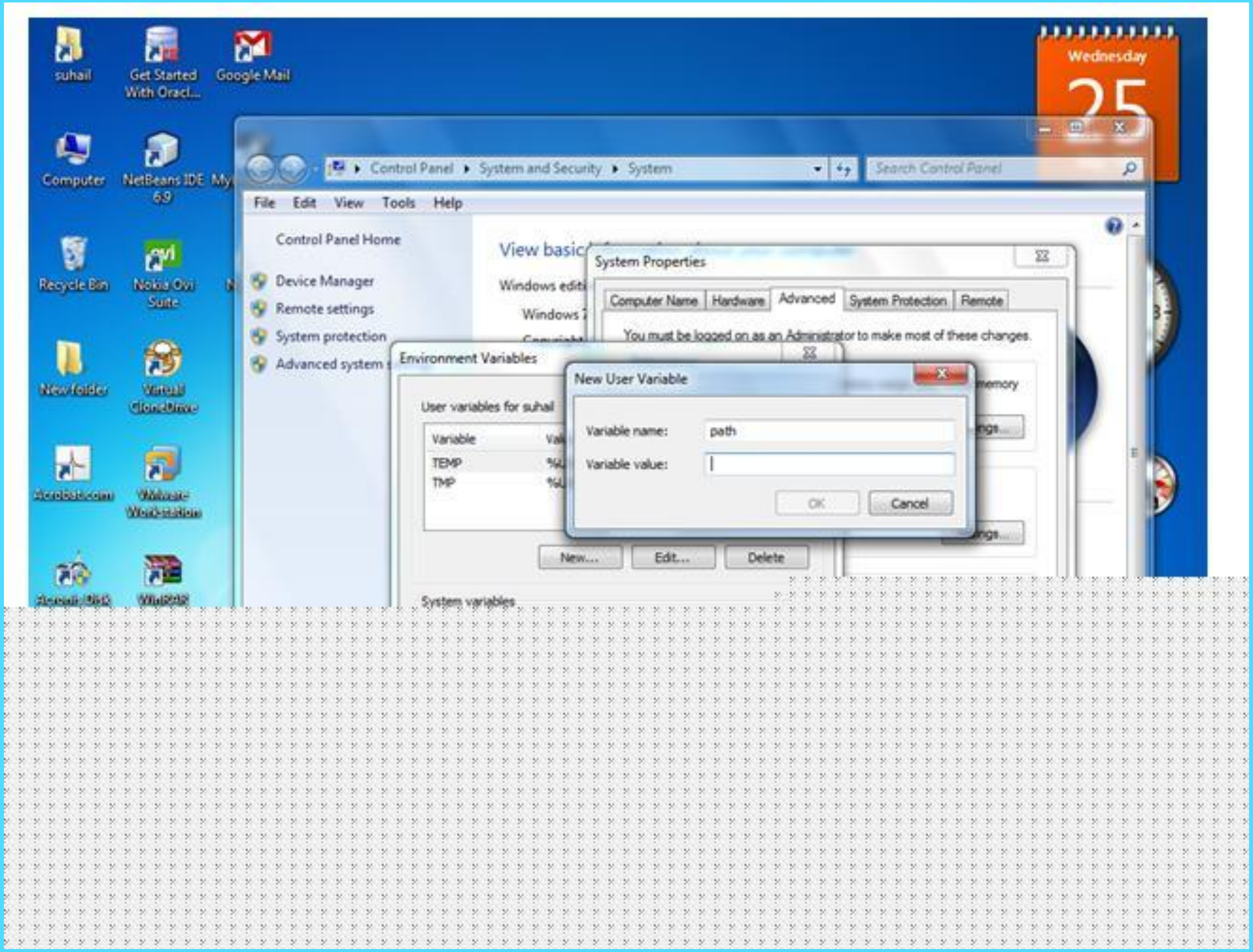
Variable	Value
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

System variables

Variable	Value
ComSpec	C:\Windows\system32\cmd.exe
FP_NO_HOST_C...	NO
NUMBER_OF_P...	2
OS	Windows_NT

Core Java Tutorial

- **write path in variable name**


New User Variable

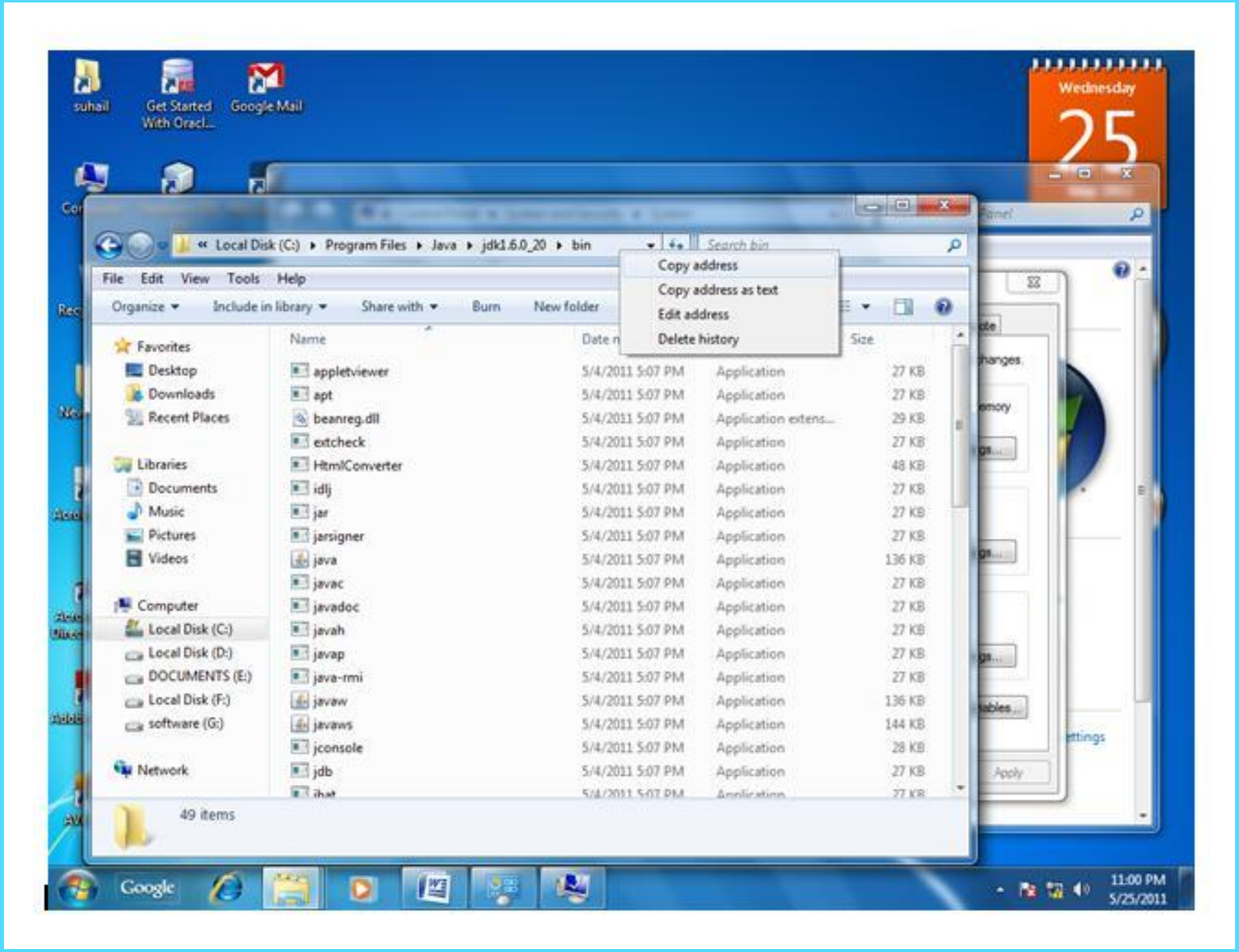
Variable name: path

Variable value:

Control Panel ▸ System and Security ▸ System
File Edit View Tools Help
Control Panel Home
Device Manager
Remote settings
System protection
Advanced system
System Properties
Computer Name Hardware Advanced System Protection Remote
You must be logged on as an Administrator to make most of these changes.
Environment Variables
User variables for suhail
Variable
TEMP
TMP
New... Edit... Delete
System variables
OK Cancel

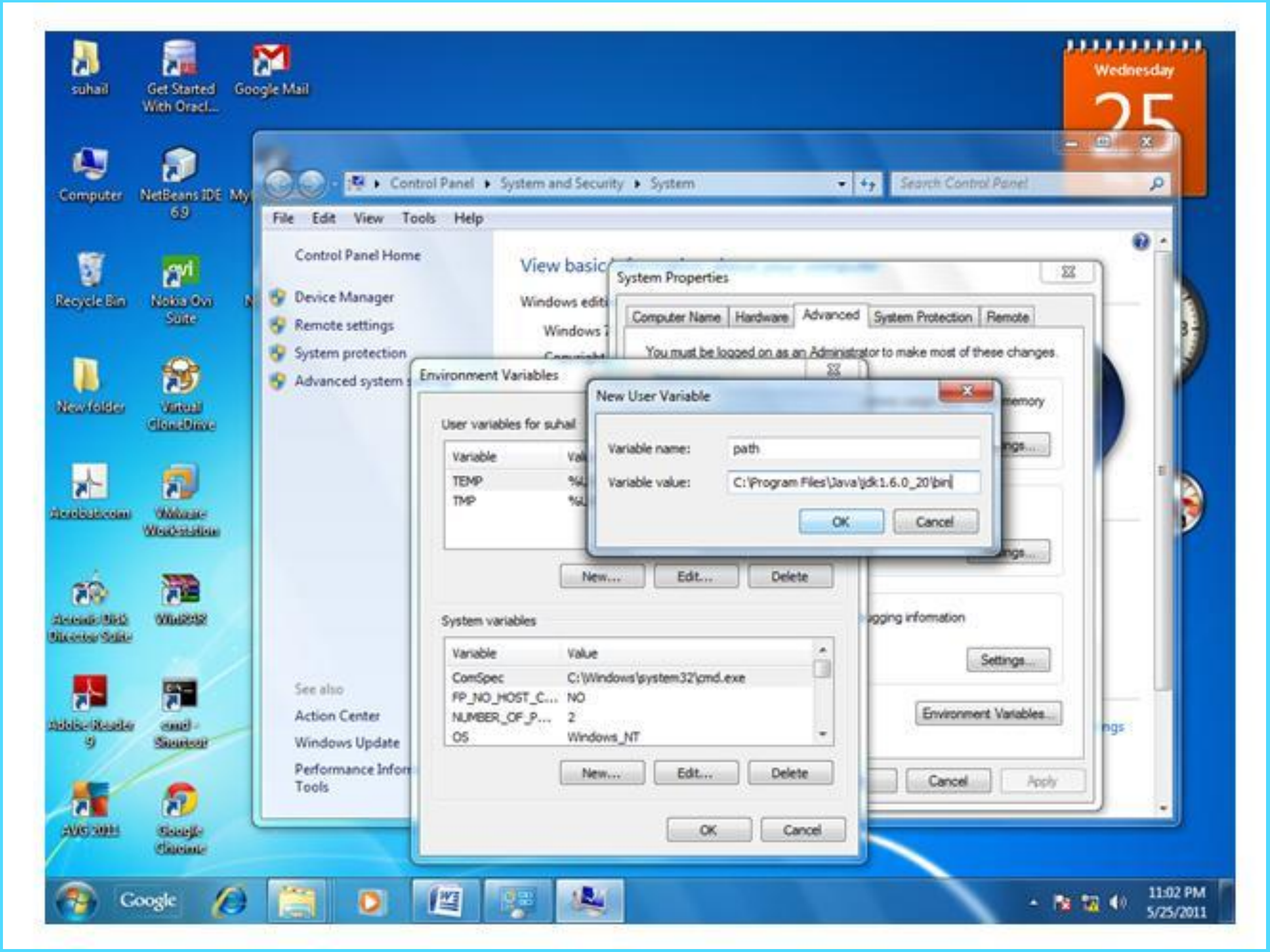
Core Java Tutorial

• Copy the path of bin folder



Name	Date m		Size
appletviewer	5/4/2011 5:07 PM	Application	27 KB
apt	5/4/2011 5:07 PM	Application	27 KB
beanreg.dll	5/4/2011 5:07 PM	Application extens...	29 KB
extcheck	5/4/2011 5:07 PM	Application	27 KB
HtmlConverter	5/4/2011 5:07 PM	Application	48 KB
idlj	5/4/2011 5:07 PM	Application	27 KB
jar	5/4/2011 5:07 PM	Application	27 KB
jarsigner	5/4/2011 5:07 PM	Application	27 KB
java	5/4/2011 5:07 PM	Application	136 KB
javac	5/4/2011 5:07 PM	Application	27 KB
javadoc	5/4/2011 5:07 PM	Application	27 KB
javah	5/4/2011 5:07 PM	Application	27 KB
javap	5/4/2011 5:07 PM	Application	27 KB
java-rmi	5/4/2011 5:07 PM	Application	27 KB
javaw	5/4/2011 5:07 PM	Application	136 KB
javaws	5/4/2011 5:07 PM	Application	144 KB
jconsole	5/4/2011 5:07 PM	Application	28 KB
jdb	5/4/2011 5:07 PM	Application	27 KB
jhat	5/4/2011 5:07 PM	Application	27 KB

- **paste path of bin folder in variable value**


New User Variable

Variable name: path

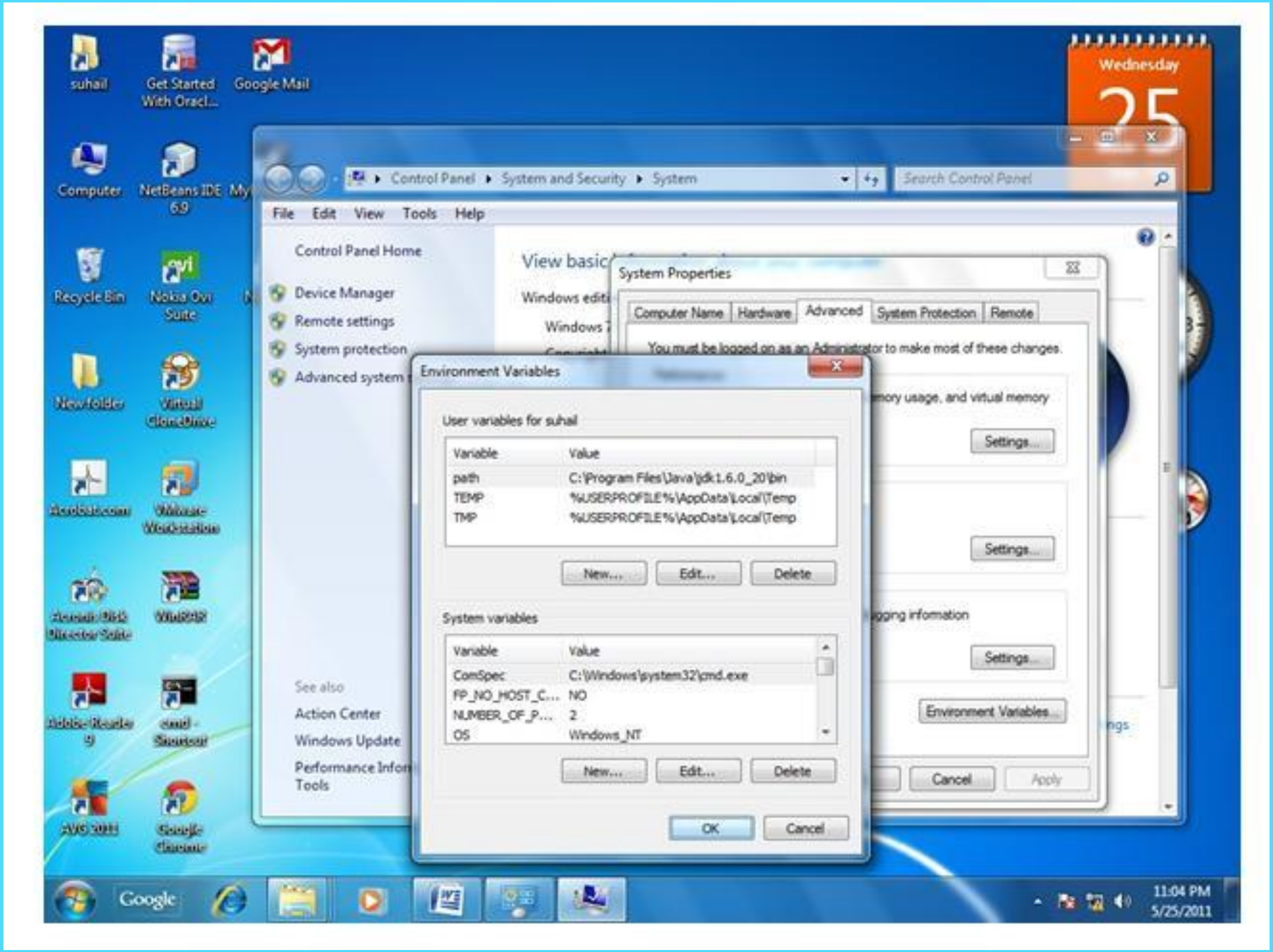
Variable value: C:\Program Files\Java\jdk1.6.0_20\bin

System variables

Variable	Value
ComSpec	C:\Windows\system32\cmd.exe
FP_NO_HOST_C...	NO
NUMBER_OF_P...	2
OS	Windows_NT

Core Java Tutorial

- **click on ok button**



Environment Variables

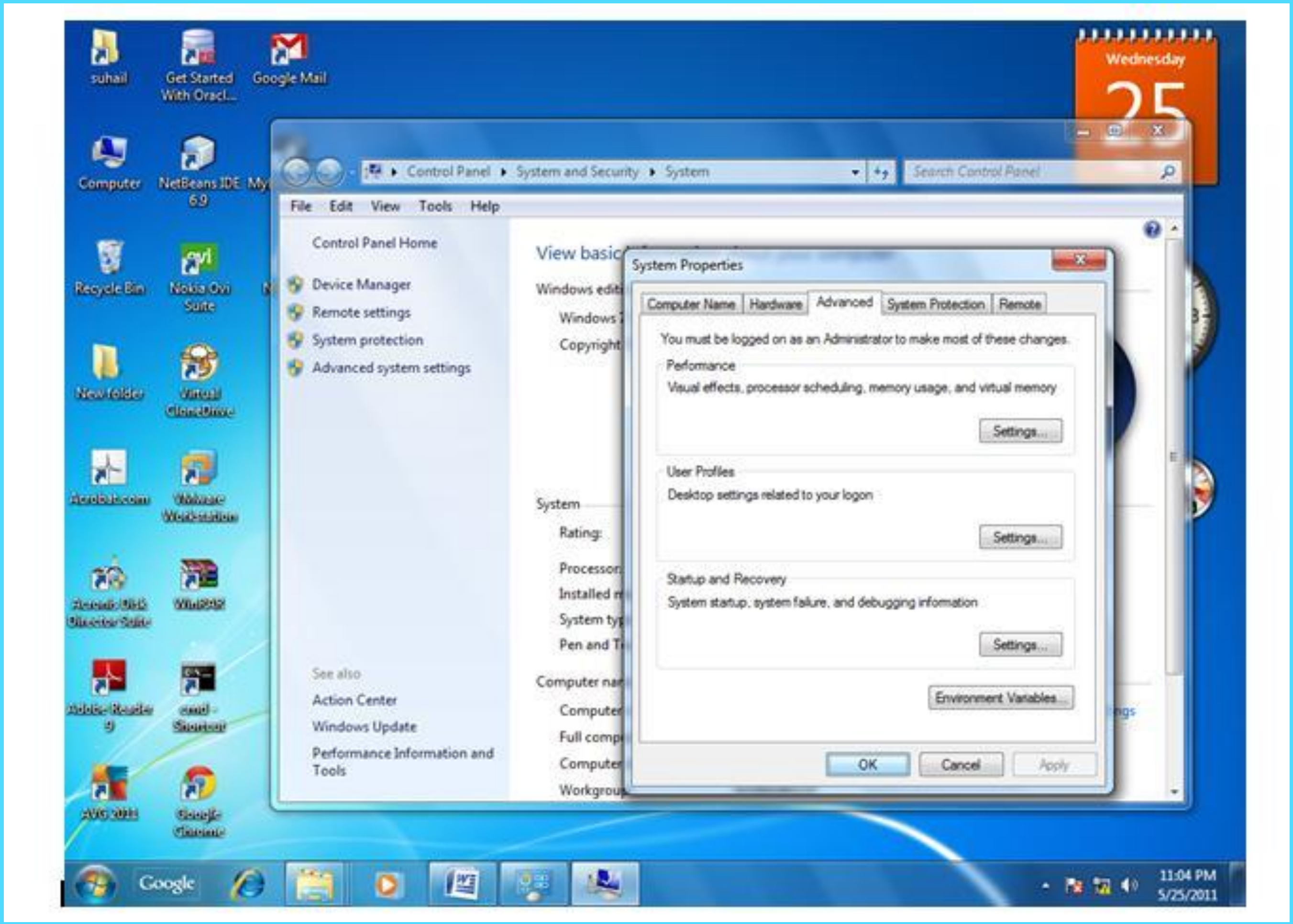
User variables for suhail

Variable	Value
path	C:\Program Files\Java\jdk1.6.0_20\bin
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

System variables

Variable	Value
ComSpec	C:\Windows\system32\cmd.exe
FP_NO_HOST_C...	NO
NUMBER_OF_P...	2
OS	Windows_NT

• **click on ok button**



You must be logged on as an Administrator to make most of these changes.

Performance
Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles
Desktop settings related to your logon

Startup and Recovery
System startup, system failure, and debugging information

Core Java Tutorial

- Difference between JDK,JRE and JVM

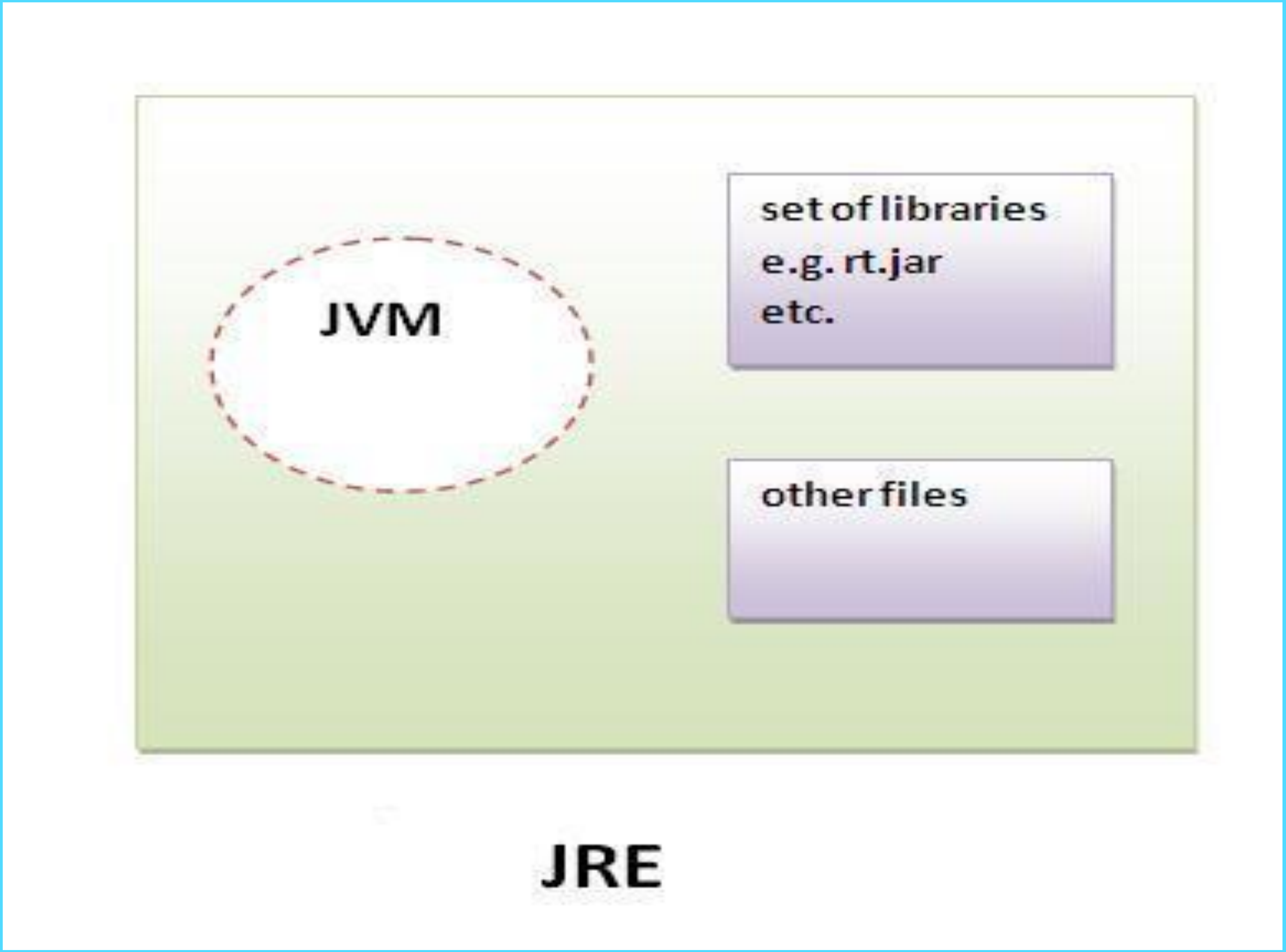
- JVM

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java byte code can be executed. JVMs are available for many hardware and software platforms (i.e.JVM is platform dependent).The JVM performs four main tasks:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

- JRE

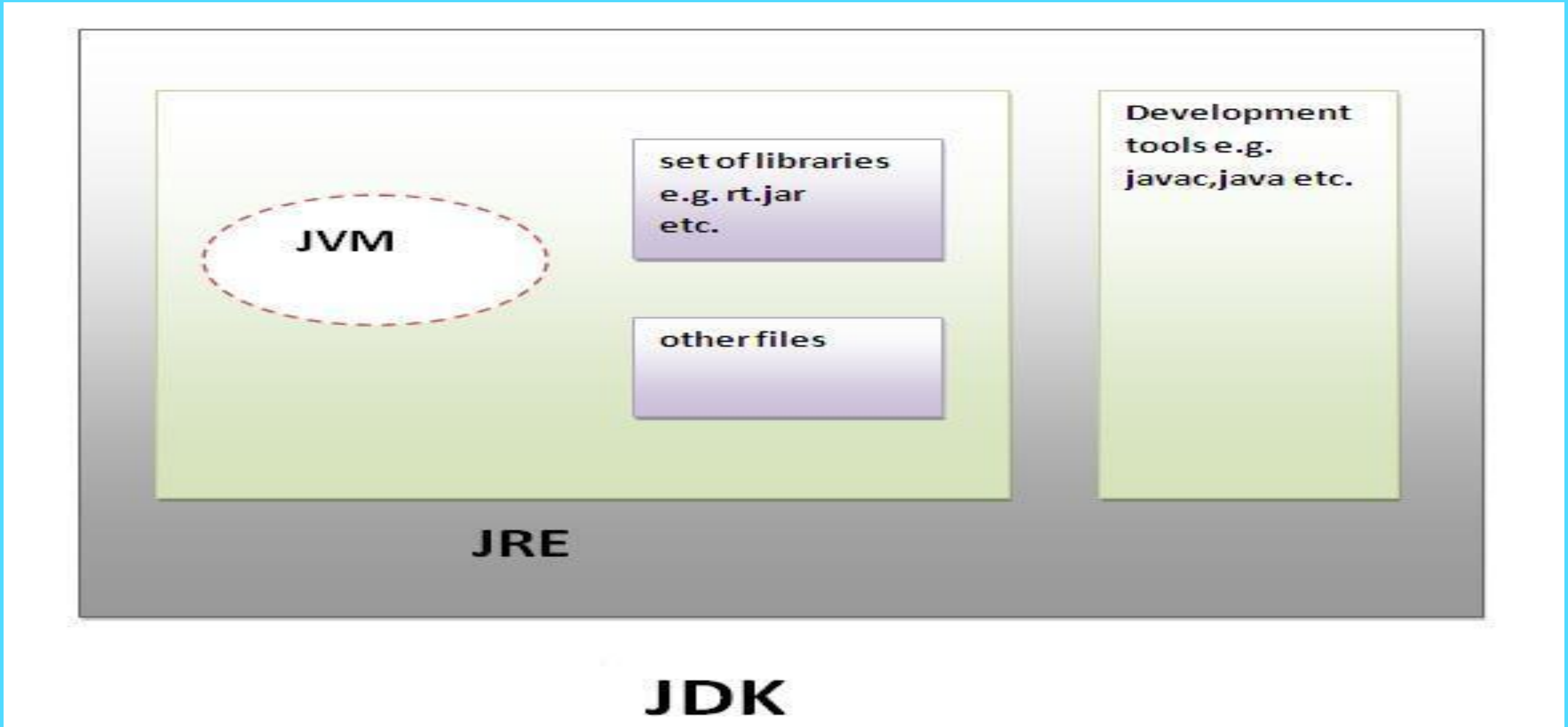
JRE is an acronym for Java Runtime Environment. It is used to provide runtime environment. It is the implementation of JVM. It physically exists. It contains set of libraries + other files that JVM uses at run time. Implementation of JVMs are also actively released by other companies besides Sun Micro Systems.



Core Java Tutorial

- JDK

JDK is an acronym for Java Development Kit. It physically exists. It contains JRE + development tools


JVM
set of libraries e.g. rt.jar etc.
other files
Development tools e.g. javac,java etc.
JRE
JDK

- JVM (Java Virtual Machine)

- Internal Architecture of JVM

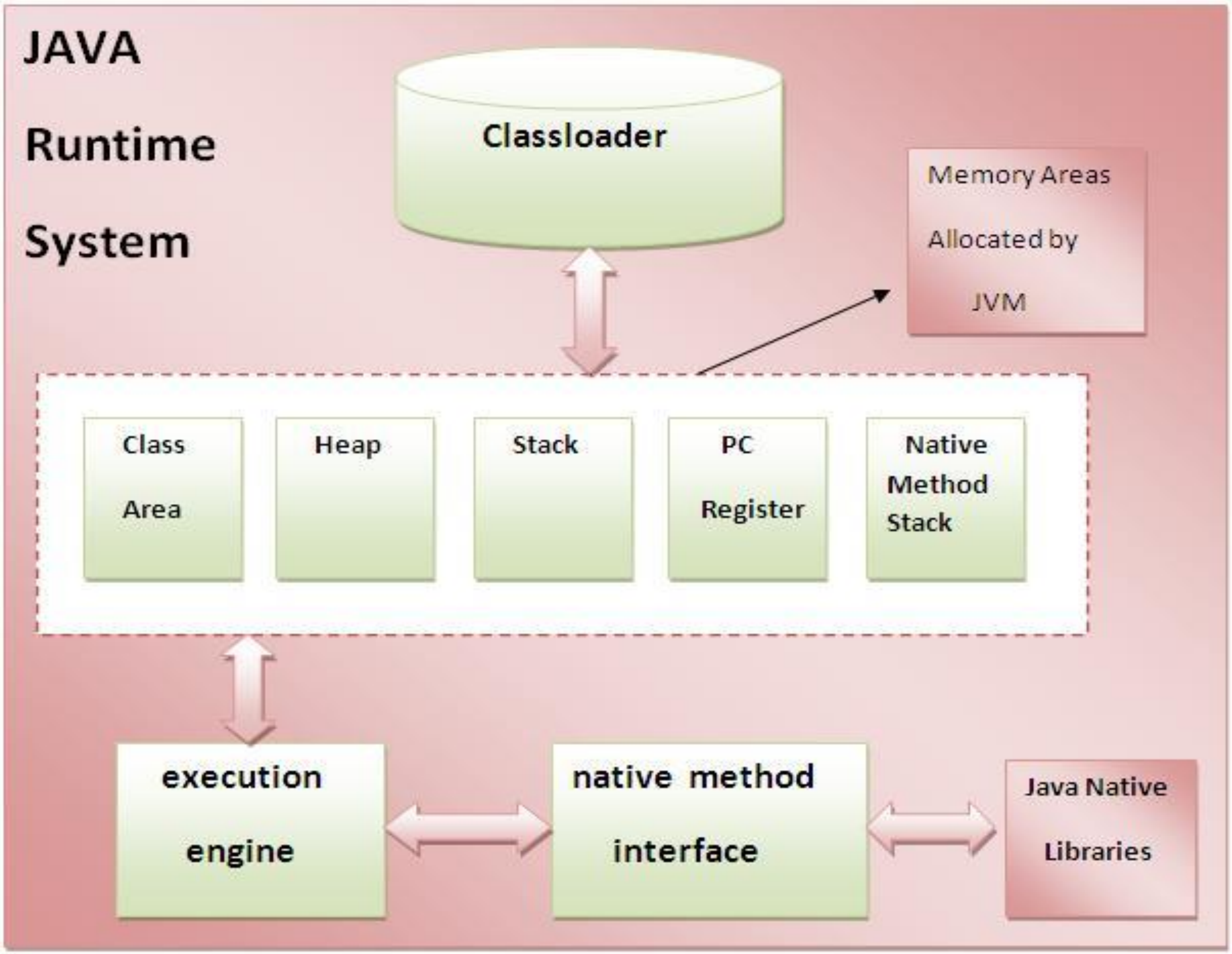
- Class loader
- Class Area
- Heap Area
- Stack Area
- Program Counter Register
- Native Method Stack
- Execution Engine

- JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java can be executed. JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).The JVM performs four main tasks:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment
- JVM provides definitions for
- Memory area
- Class file format
- Register set
- Garbage-collected heap

- Internal Architecture of JVM

Let's understand the internal architecture of JVM. It contains class loader, memory area, execution engine etc.



Core Java Tutorial

- Class loader:

Class loader is a subsystem of JVM that is used to load class files.

- Class(Method) Area:

Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

- Heap:

It is the runtime data area in which objects are allocated.

- Stack:

Java Stack stores frames. It holds local variables and partial results, and plays a part in method invocation and return. Each thread has a private JVM stack, created at the same time as thread. A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

- Program Counter Register:

PC (program counter) register. It contains the address of the Java virtual machine instruction currently being executed.

- Native Method Stack:

It contains all the native methods used in the application.

- Execution Engine:

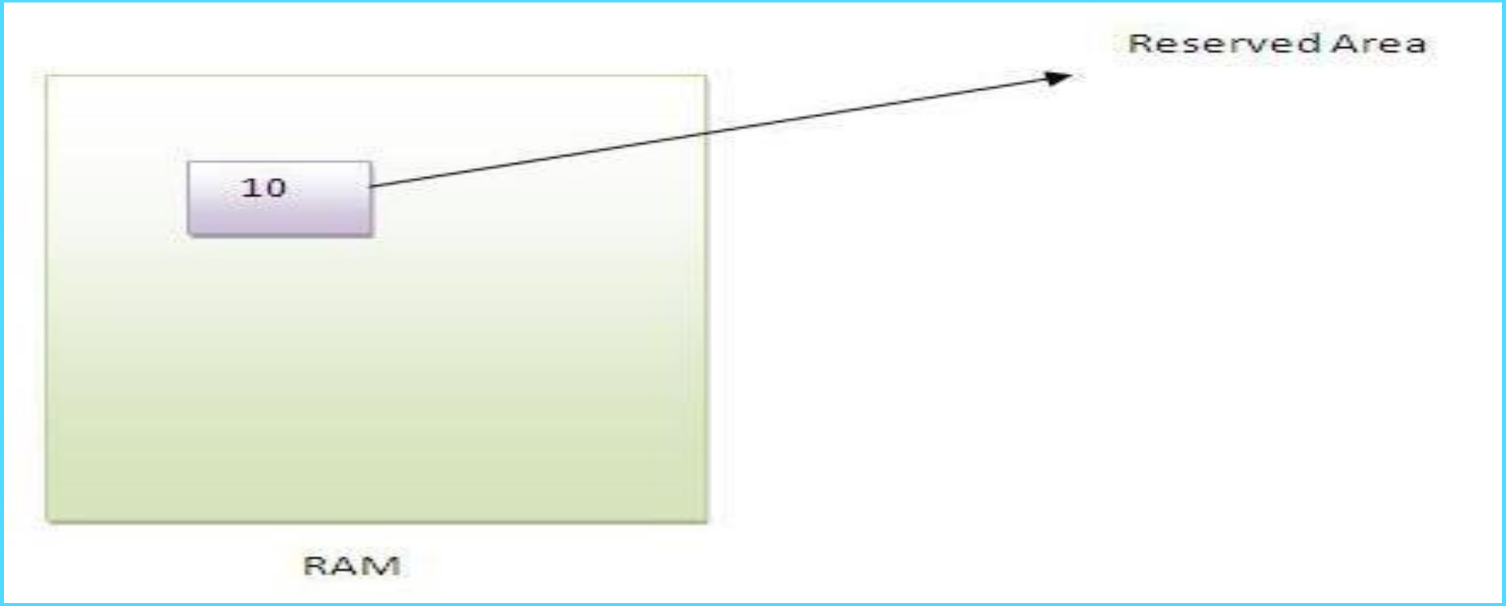
It contains:

- **A virtual processor**
- **Interpreter:** Read bytecode stream then execute the instructions.
- **Just-In-Time(JIT) compiler:** It is used to improve the performance

- Variable and Datat Type in Java

- Variable

Variable is name of reserved area allocated in memory.

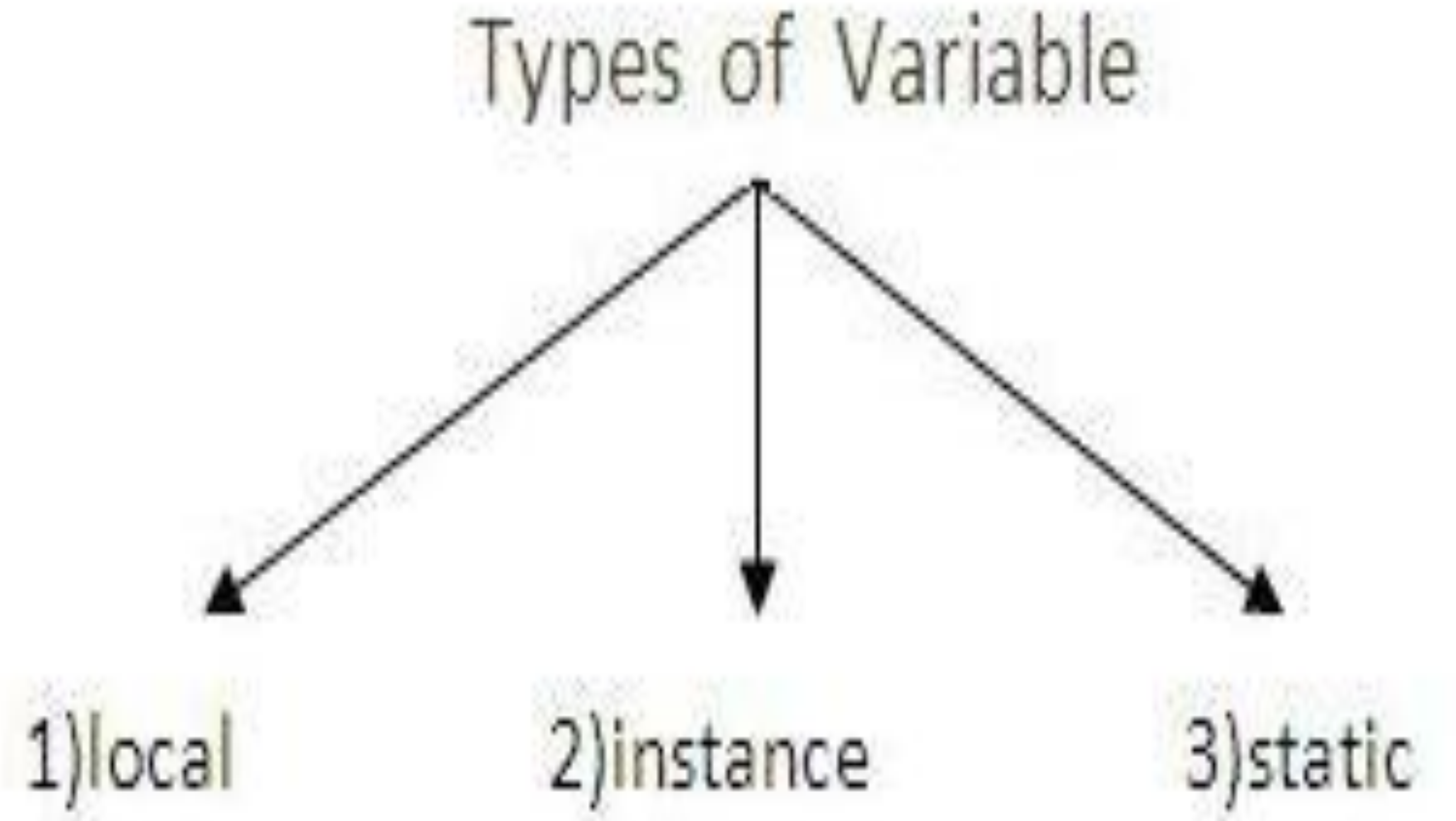


int data=50;//Here data is variable

- Types of Variable

There are three types of variables in

- javalocal variable
- instance variable
- static variable


Types of Variable
1)local
2)instance
3)static

- **Local Variable**
  A variable that is declared inside the method is called local variable.

- **Instance Variable**
  A variable that is declared inside the class but outside the method is called instance variable . It is not declared as static.

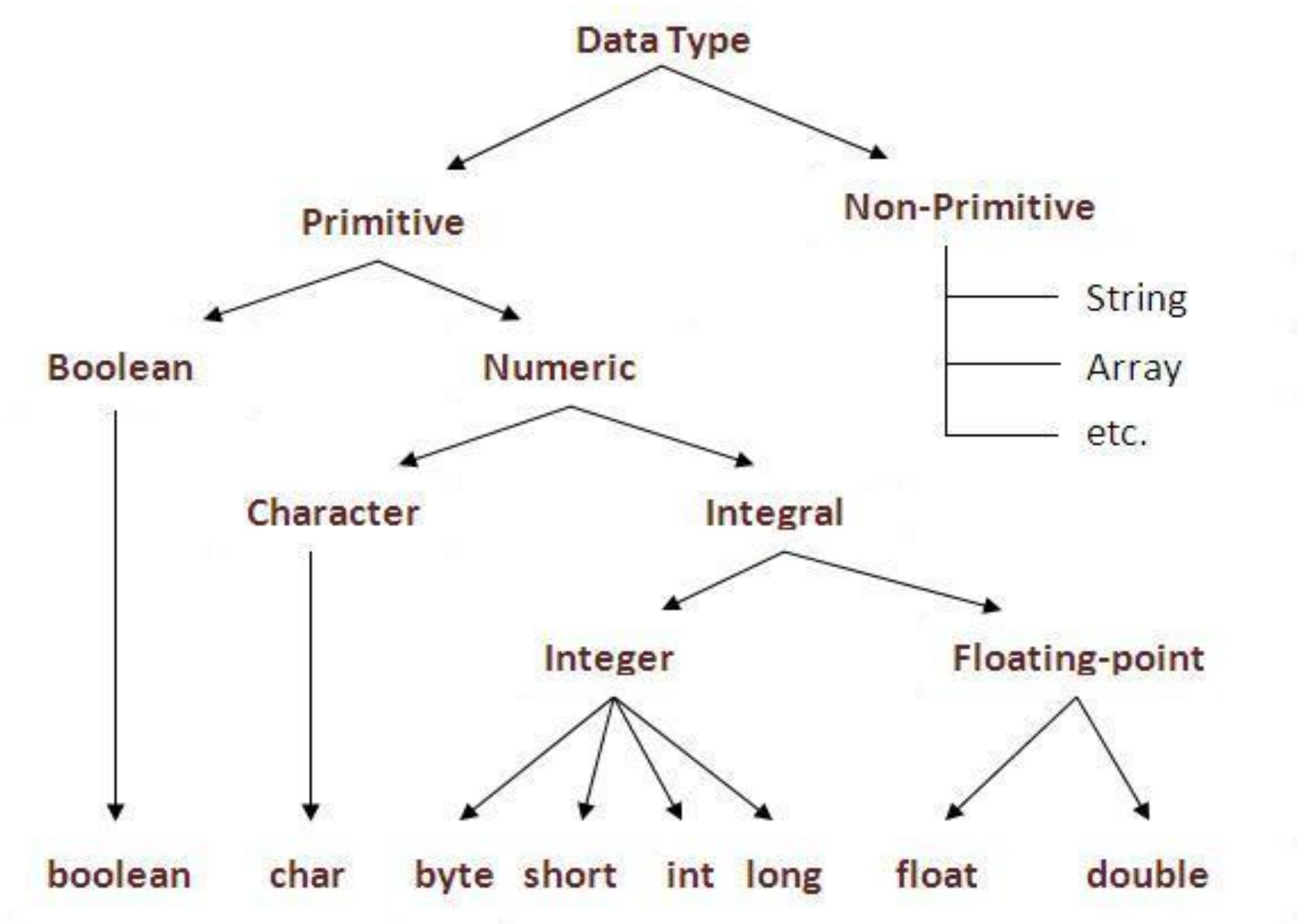
- **Static variable**
  A variable that is declared as static is called static variable. It cannot be local.
  We will have detailed learning of these variables in next chapters.

- Data Types in Java

In java, there are two types of data types

- primitive data types

- non-primitive data types


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

Data Type	Default Value	Default Size
Boolean	False	1 bit
Char	‘\u000’	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

- Unicode System

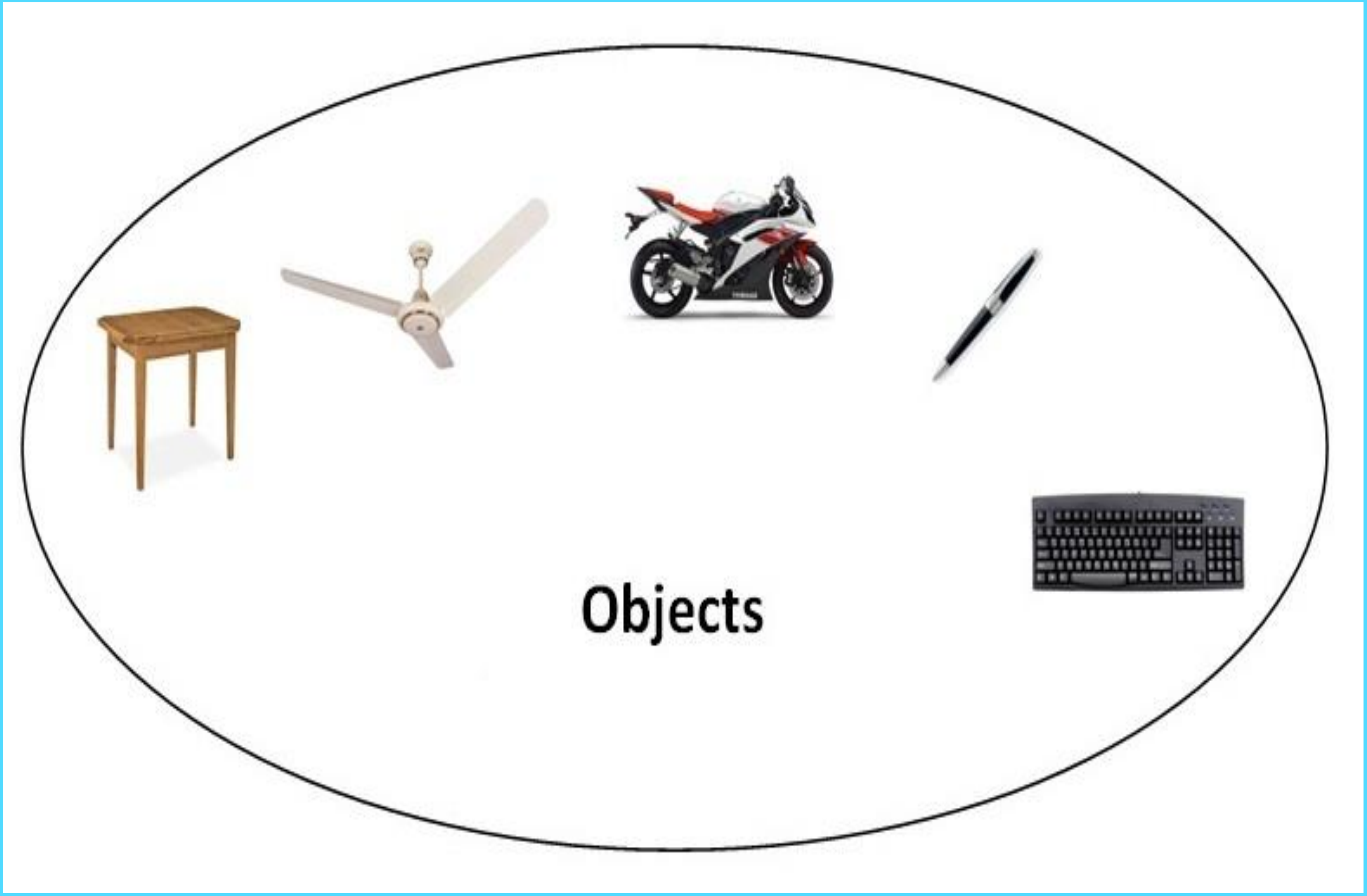
Unicode is a universal international standard character encoding that is capable of representing most of the world's written languages.

In unicode, character holds 2 byte, so java also uses 2 byte for characters.

lowest value: \u0000
highest value: \uFFFF

- **Java OOPs Concepts**

- **Object Oriented Programming System**
- **Object** means a real word entity such as pen, chair, table etc.
- **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:


Objects

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

- **Object**

  Any entity that has state and behaviour is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

- **Class**

Collection of objects is called class. It is a logical entity.

- **Inheritance**

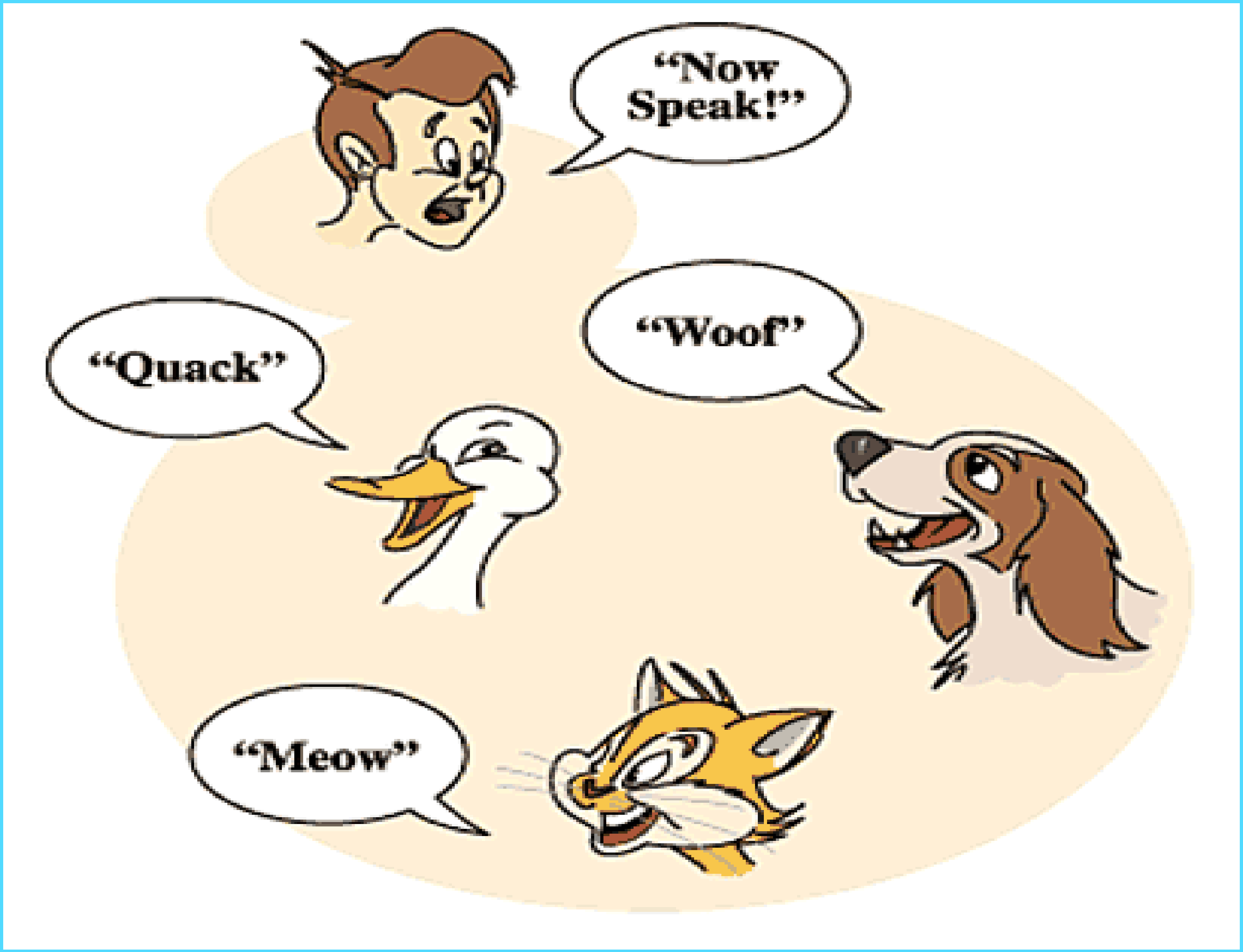
When one object acquires all the properties and behaviours of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

• Polymorphism

When **one task is performed by different ways** i.e. known as polymorphism. For example: to draw something e.g. shape or rectangle etc.

In java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something e.g. cat speaks meaw, dog barks woof etc.


"Now Speak!"
"Quack"
"Woof"
"Meow"

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing.

In java, we use abstract class and interface to achieve abstraction.

- **Encapsulation**

Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.

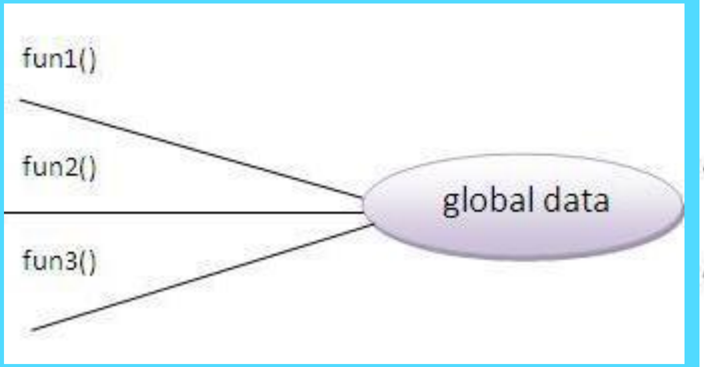
A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

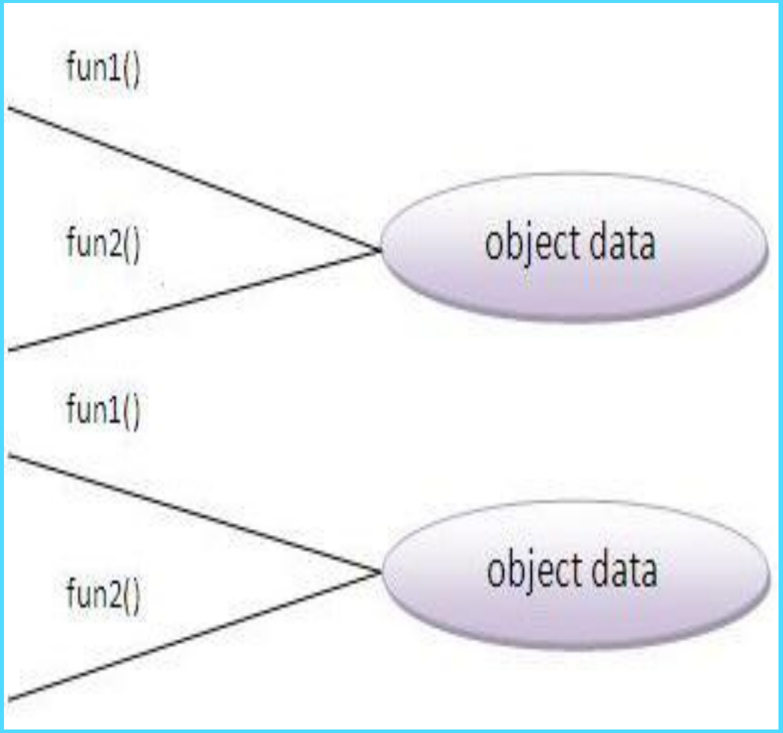
- **Advantage of OOPs over Procedure-oriented programming language**

- OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grows.

- OOPs provides data hiding whereas in Procedure-oriented programming language a global data can be accessed from anywhere.

- OOPs provides ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.


fun1()
fun2()
fun3()
global data


fun1()
fun2()
object data
fun1()
fun2()
object data

Object and Class in Java

Object:

A runtime entity that has state and behaviour is known as an object. For example: chair, table, pen etc. It can be tangible or intangible (physical or logical).

- **An object has three characteristics:**

- **state:** represents the data of an object.

- **behaviour:** represents the behaviour of an object.

- **identity:** Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user, but is used internally by the JVM to identify each object uniquely.

For Example: Pen is an object. Its name is Reynolds, color is white etc. known as its state. It is used to write, so writing is its behaviour. **Object is an instance of a class.** Class is a template or blueprint from which objects are created. So object is the instance(result) of a class.

- Class

A class is a group of objects that have common property. It is a template or blueprint from which objects are created. A class in java can contain **data member**

- **method**
- **constructor**
- **block**

- **Syntax to declare a class:**

```
class <class_name>{
data member;
method;
}
```

- **Instance variable**

A variable that is created inside the class but outside the method, is known as instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when object(instance) is created.

- **Method**

In java, a method is like function i.e. used to expose behaviour of an object.

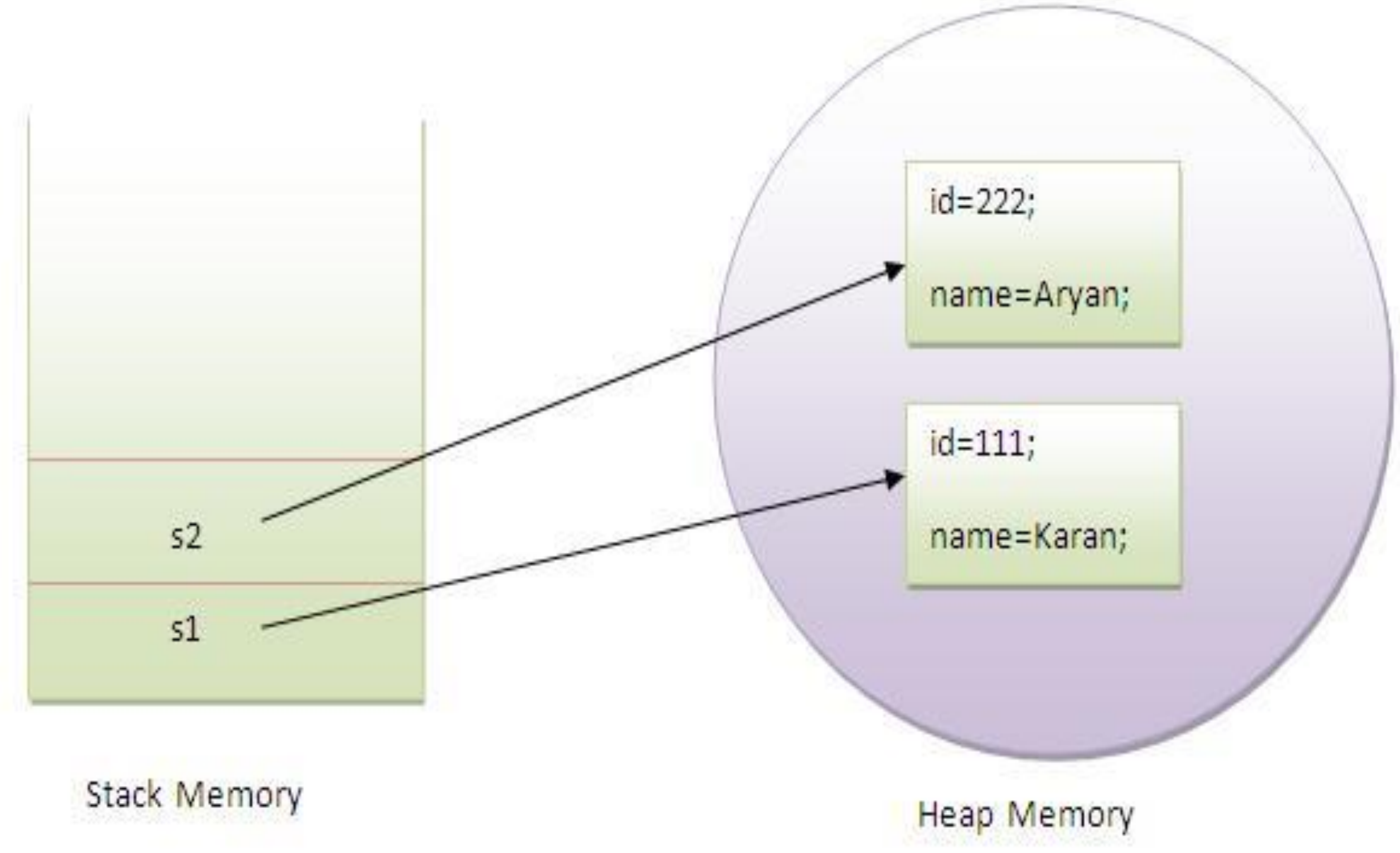
Advantages of Method

- Code Reusability
- Code Optimization

- **new keyword**

  The **new** keyword is used to allocate memory at runtime.

```
class Student{
 int rollno;
 String name;

 void insertRecord(int r, String n){
  rollno=r;
  name=n;
 }

 void displayInformation(){
System.out.println (rollno+" "+name);
}
public static void main(String args[]){
  Student s1=new Student();
  Student s2=new Student();

  s1.insertRecord(111,"Karan");
  s2.insertRecord(222,"Aryan");

  s1.displayInformation();
  s2.displayInformation();

 }
}
```


id=222;
name=Aryan;
id=111;
name=Karan;
s2
s1
Stack Memory
Heap Memory

- **What are the different ways to create an object in Java?**

There are many ways to create an object in java. They are:

- By new keyword
- By newInstance() method
- By clone() method
- By factory method etc.

- Annonymous object

Annonymous simply means nameless. An object that have no reference is known as annonymous object. If you have to use an object only once, annonymous object is a good approach.

- Method Overloading in Java

If a class have multiple methods by same name but different parameters, it is known as **Method Overloading**.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behaviour of the method because its name differs. So, we perform method overloading to figure out the program quickly.

- **Advantage of method overloading**

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

Different ways to overload the method

- There are two ways to overload the method in java
- By changing number of arguments
- By changing the data type

Note: ***In java, Method Overloading is not possible by changing the return type of the method.***

- Constructor in Java

- **Constructor** is a **special type of method** that is used to initialize the object.

- Constructor is **invoked at the time of object creation**. It constructs the values i.e. provides data for the object that is why it is known as constructor.

- **Rules for creating constructor**

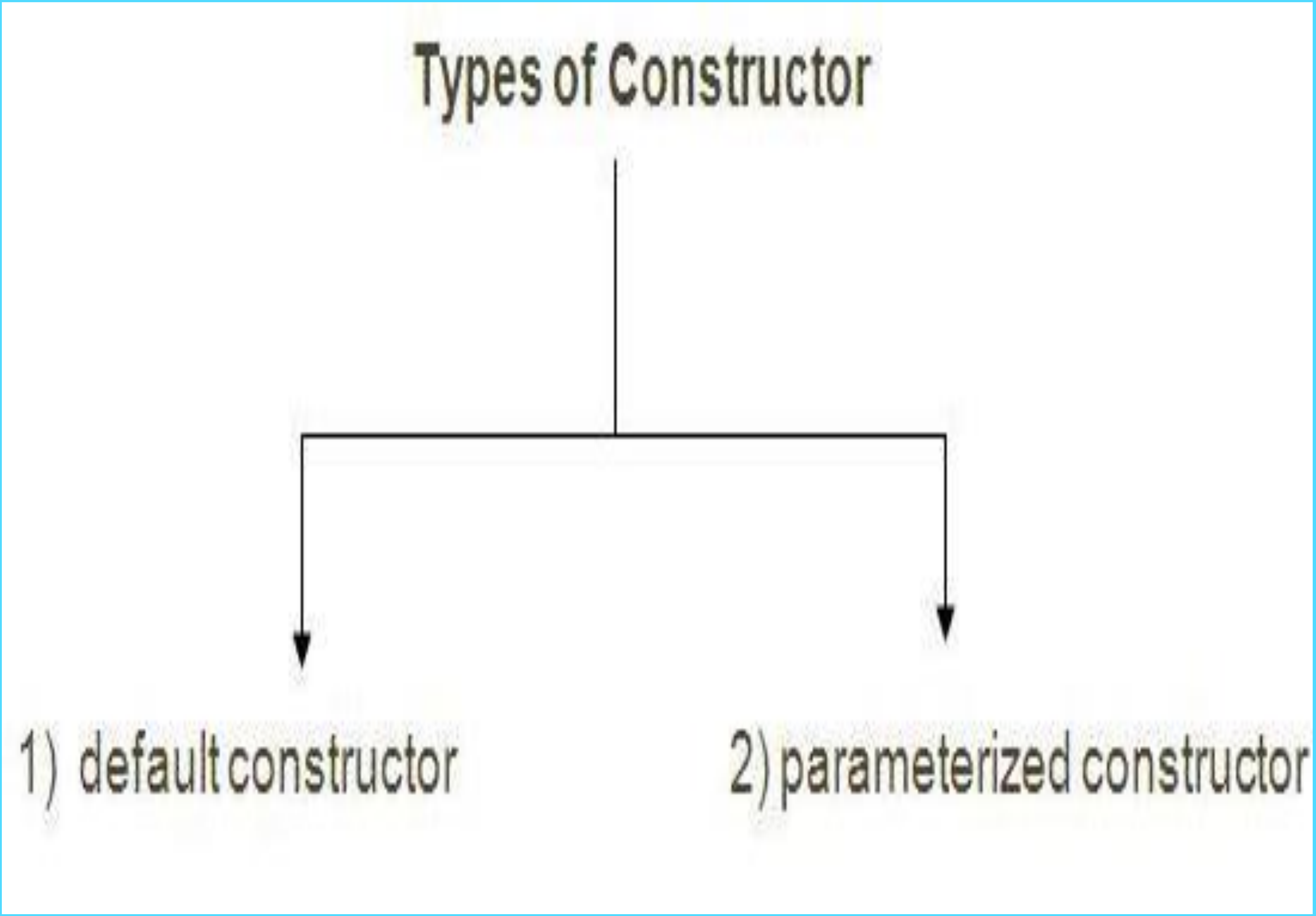
There are basically two rules defined for the constructor.

- Constructor name must be same as its class name
- Constructor must have no explicit return type

- **Types of constructors**

There are two types of constructors:

- default constructor (no-arg constructor)
- parameterized constructor


Types of Constructor
1) default constructor
2) parameterized constructor

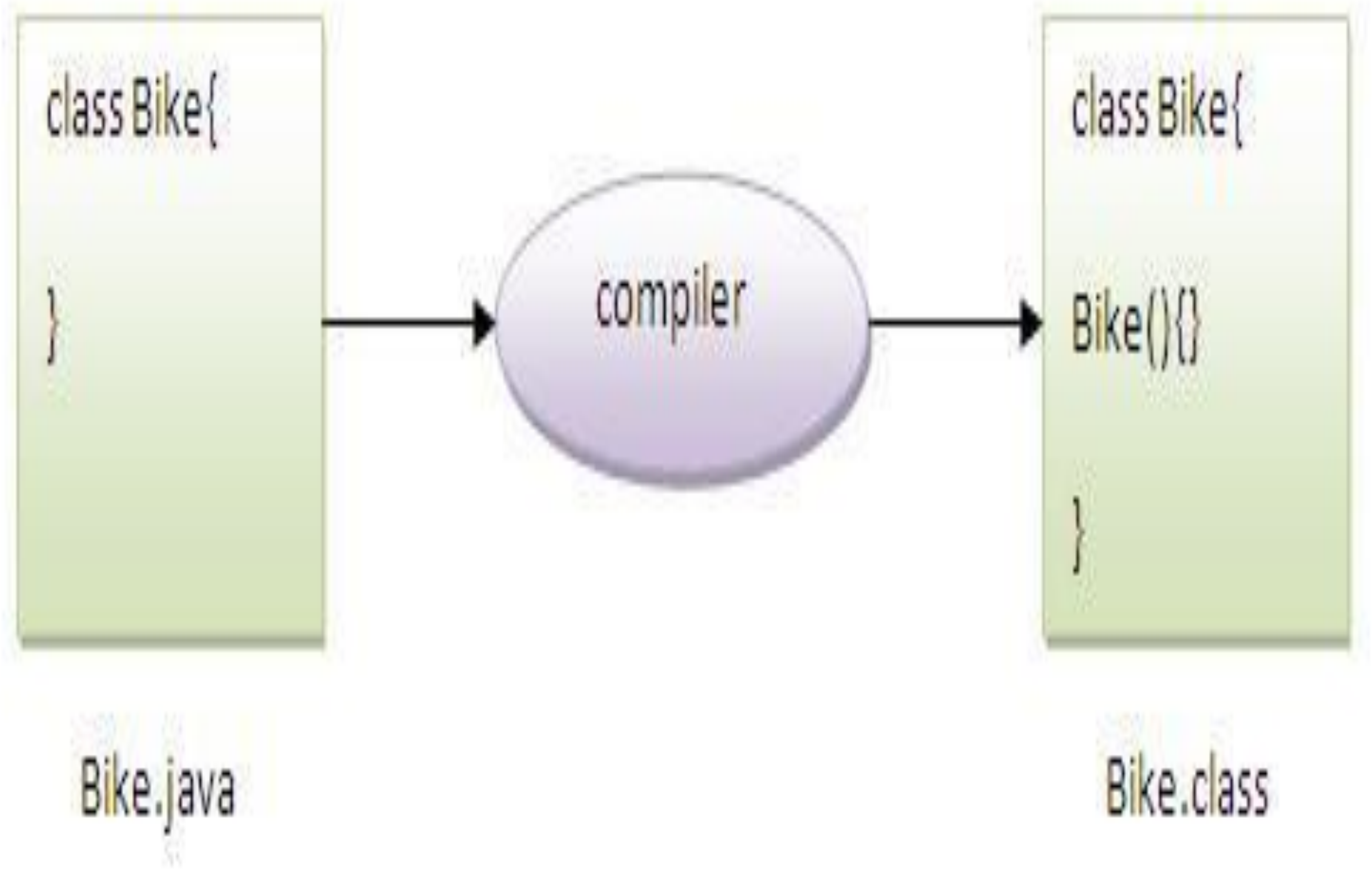
- default Constructor

A constructor that have no parameter is known as default constructor.

Syntax of default constructor:

<class_name>(){}

Rule:* *If there is no constructor in a class, compiler automatically creates a default constructor.


class Bike{
}
compiler
class Bike{
Bike(){}
}
Bike.java
Bike.class

- Parameterized constructor

A constructor that have parameters is known as parameterized constructor.

- **Why use parameterized constructor?**

Parameterized constructor is used to provide different values to the distinct objects.

- Constructor Overloading

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

Constructor	Method
Constructor is used to initialize the state of an object.	Method is used to expose behaviour of an object.
Constructor must not have return type.	Method must have return type.
Constructor is invoked implicitly.	Method is invoked explicitly.
The java compiler provides a default constructor if you don't have any constructor.	Method is not provided by compiler in any case.
Constructor name must be same as the class name.	Method name may or may not be same as class name.

• **static** keyword

The **static keyword** is used in java mainly for memory management. We may apply static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than instance of the class.

The static can be:

- variable (also known as class variable)
- method (also known as class method)
- block
- nested class

- static variable

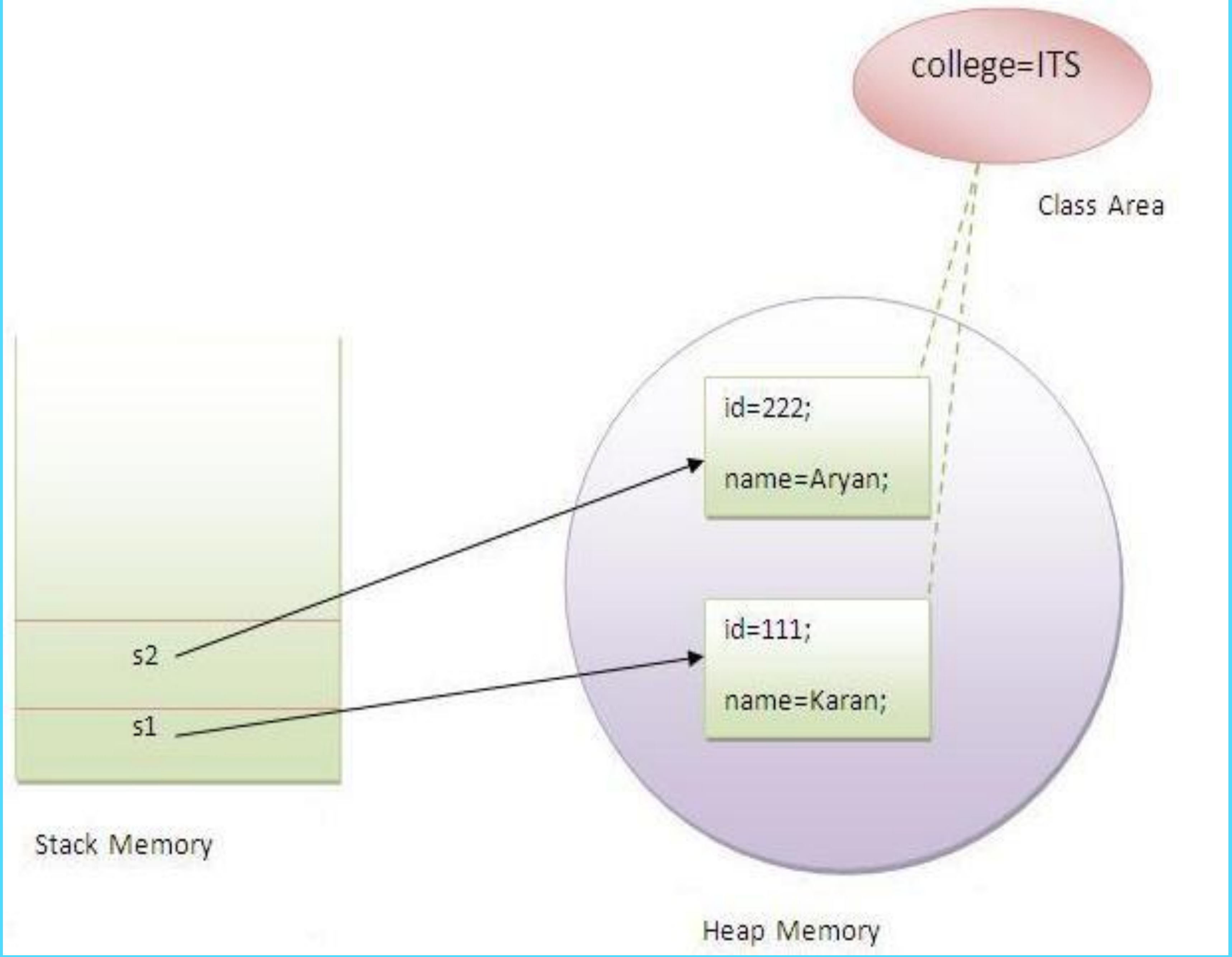
If you declare any variable as static, it is known static variable. The static variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees, college name of students etc.

The static variable gets memory only once in class area at the time of class loading.

- **Advantage of static variable**

  It makes your program **memory efficient** (i.e it saves memory). **Suppose there are 500 students in my college, now all instance data members will get memory each time when object is created. All student have its unique rollno and name so instance data member is good. Here, college refers to the common property of all objects. If we make it static, this field will get memory only once.**

- ***static property is shared to all objects.***


college=ITS
Class Area
id=222;
name=Aryan;
id=111;
name=Karan;
s2
s1
Stack Memory
Heap Memory

• static method

If you apply static keyword with any method, it is known as static method. A static method belongs to the class rather than object of a class.

A static method can be invoked without the need for creating an instance of a class.

static method can access static data member and can change the value of it.

- **Restrictions for static method**

There are two main restrictions for the static method.

They are:

- The static method can not use non static data member or call non-static method directly.
- **this** and **super** cannot be used in static context.

- static block
- Is used to initialize the static data member.
- It is executed before main method at the time of class loading.

- this keyword

In java, **this** is a **reference variable** that refers to the current object.

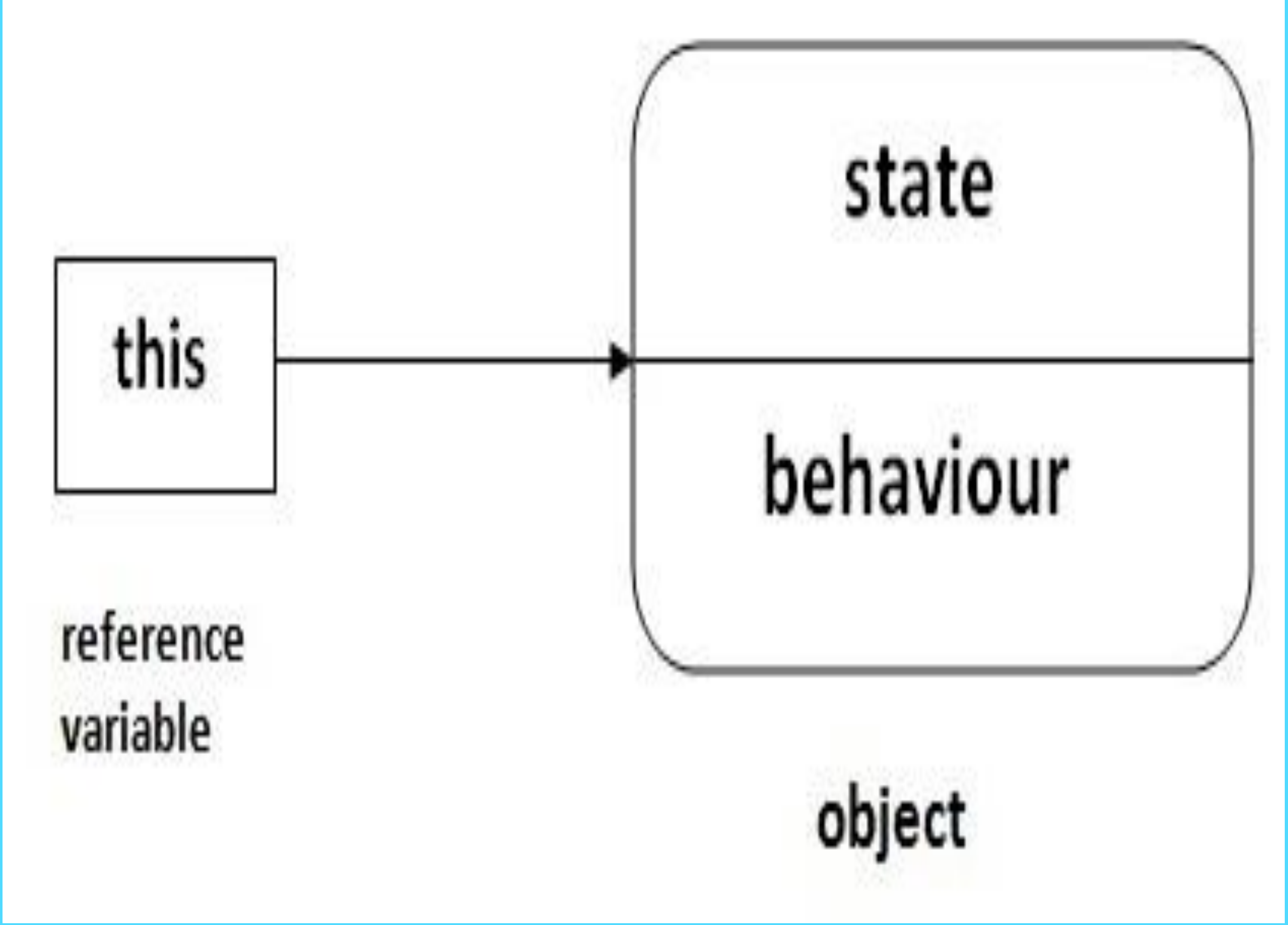
- Usage of this keyword

- this keyword can be used to refer current class instance variable.
- this() can be used to invoke current class constructor.
- this keyword can be used to invoke current class method (implicitly)

- this can be passed as an argument in the method call.

- this can be passed as argument in the constructor call.

- this keyword can also be used to return the current class instance.


this
reference
variable
state
behaviour
object

- Inheritance in Java

Inheritance is a mechanism in which one object acquires all the properties and behaviours of parent object.

The idea behind inheritance is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you reuse (or inherit) methods and fields, and you add new methods and fields to adapt your new class to new situations.

Inheritance represents the **IS-A relationship**.

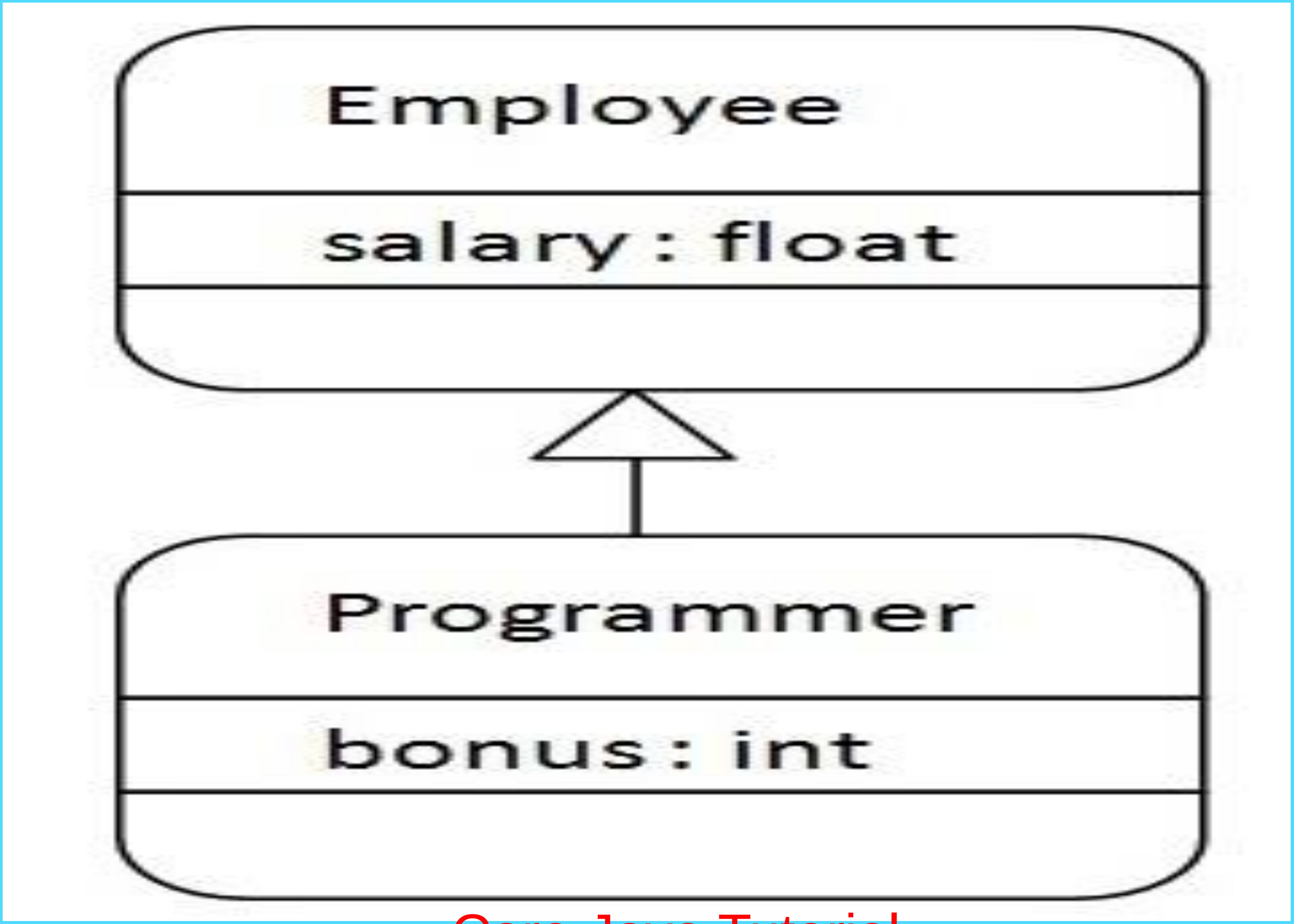
- **Why use Inheritance?**

For Method Overriding (So Runtime Polymorphism).
For Code Reusability.

- **Syntax of Inheritance**

```
 class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

The keyword extends indicates that you are making a new class that derives from an existing class. In the terminology of Java, a class that is inherited is called a super class. The new class is called a subclass.

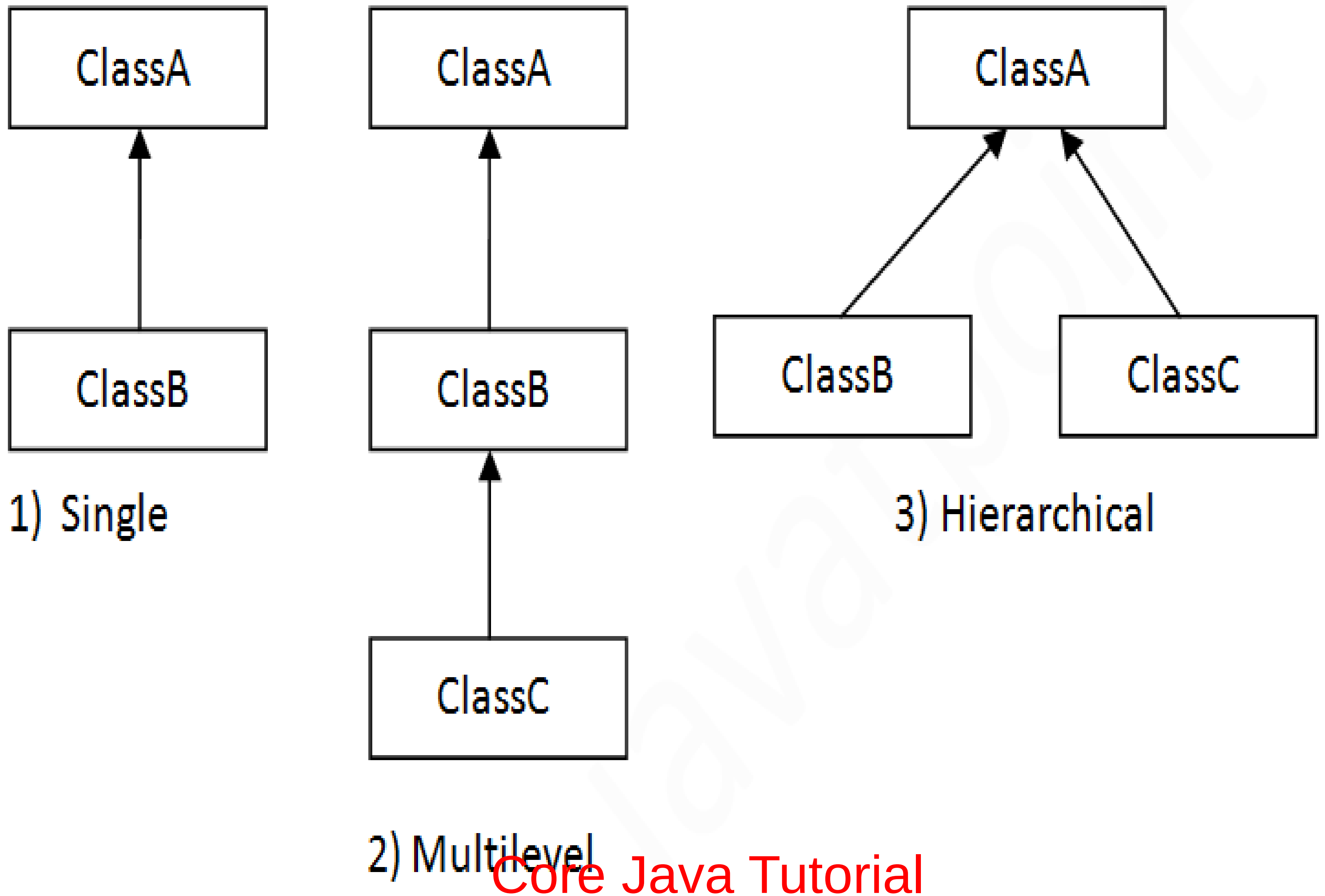

Employee
salary : float
Programmer
bonus : int

- Types of Inheritance

On the basis of class, there can be three types of inheritance:

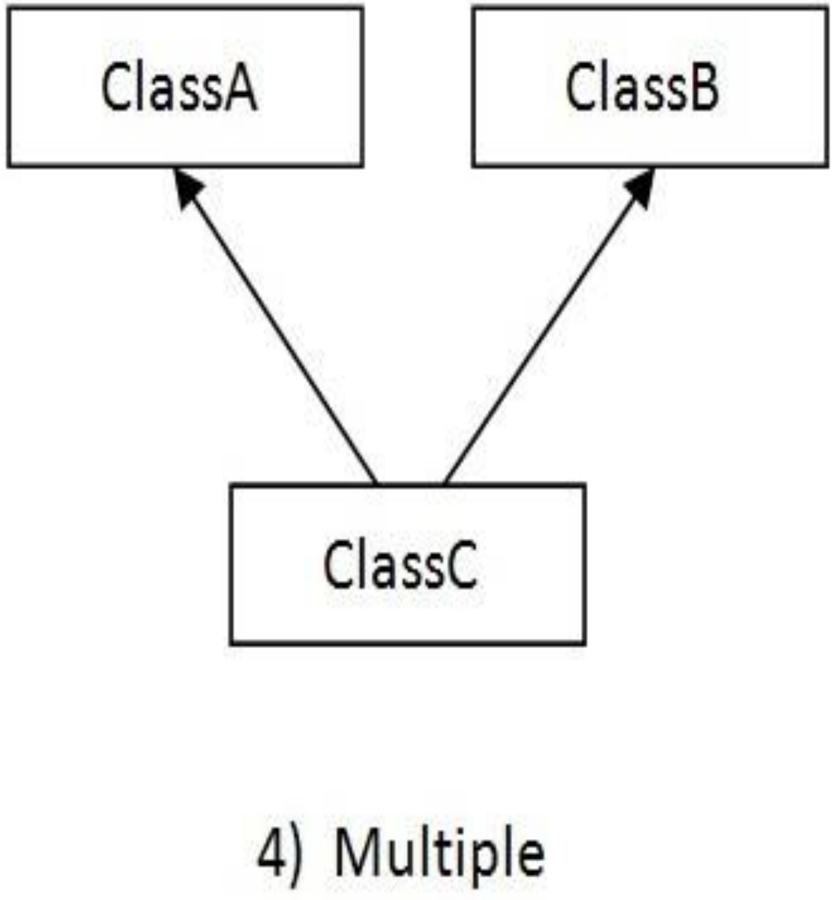
- **single**
- **multilevel**
- **hierarchical.**

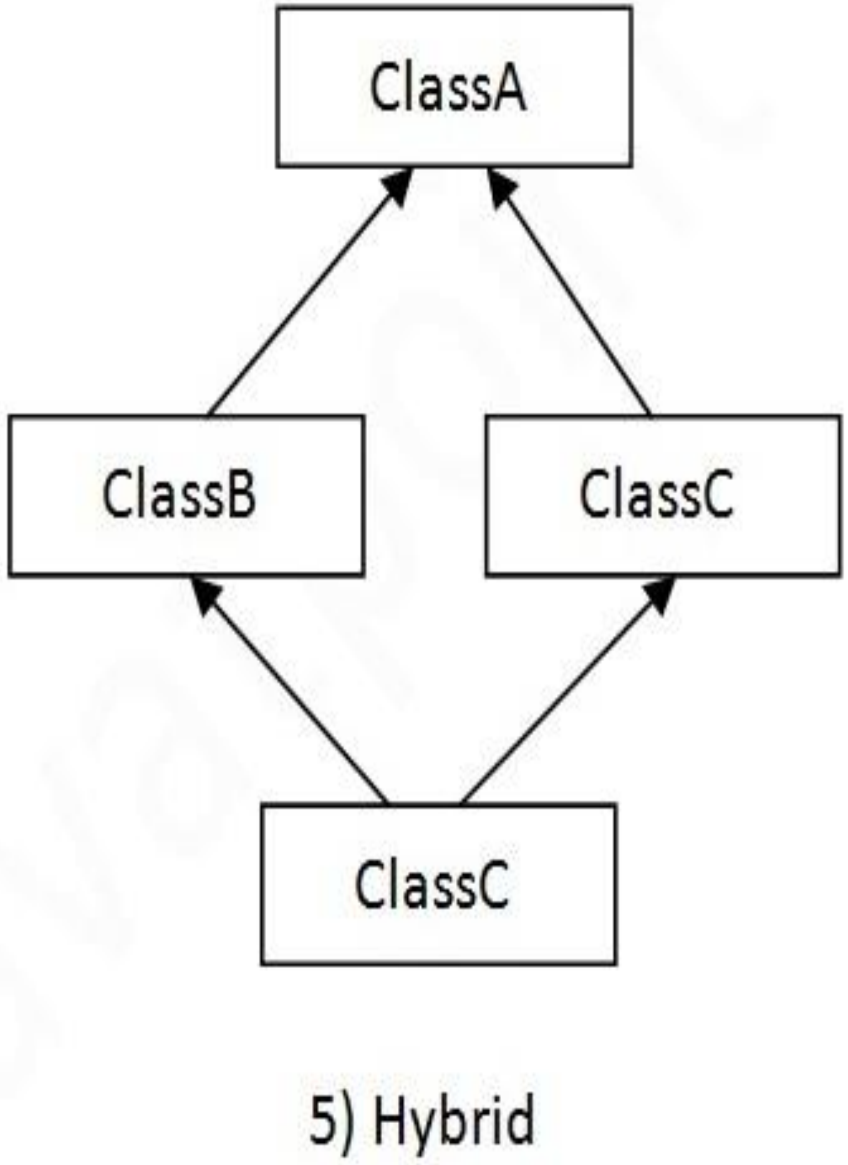
Multiple and Hybrid is supported through interface only.


ClassA
ClassB
1) Single
ClassA
ClassB
ClassC
2) Multilevel
ClassA
ClassB
ClassC
3) Hierarchical

- ***Multiple inheritance is not supported in java in case of class.***

- **When a class extends multiple classes i.e. known as multiple inheritance.**


ClassA
ClassB
ClassC
4) Multiple


ClassA
ClassB
ClassC
ClassC
5) Hybrid

- ## Aggregation in Java

If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.

Consider a situation, Employee object contains many informations such as id, name, email Id etc. It contains one more object named address, which contains its own informations such as city, state, country, zip code etc. as given below.

```
class Employee{

int id;
String name;
Address address;  //Address is a class


}
```

- **Why use Aggregation?**
- For Code Reusability

- When use Aggregation?

Code reuse is also best achieved by aggregation when there is no is-a relationship.

Inheritance should be used only if the relationship is-a is maintained throughout the lifetime of the objects involved; otherwise, aggregation is the best choice.

- Method Overriding in Java

Having the same method in the subclass as declared in the parent class is known as **method overriding**.

In other words, If subclass provides the specific implementation of the method i.e. already provided by its parent class, it is known as Method Overriding.

- **Advantage of Method Overriding**

Method Overriding is used to provide specific implementation of a method that is already provided by its super class.

- Method Overriding is used for Runtime Polymorphism

- **Rules for Method Overriding:**

- method must have same name as in the parent class
- method must have same parameter as in the parent class.
- must be inheritance (IS-A) relationship.

- **Can we override static method?**

No, static method cannot be overridden. It can be proved by runtime polymorphism.

- **Why we cannot override static method?**

because static method is bound with class whereas instance method is bound with object. Static belongs to class area and instance belongs to heap area.

Method Overloading	Method Overriding
1) Method overloading is used to increase the readability of the program.	Method overriding is used to provide the specific implementation of the method that is already provided by its super class.
2) method overlaoding is performed within a class.	Method overriding occurs in two classes that have IS-A relationship.
3) In case of method overloading parameter must be different.	In case of method overriding parameter must be same.

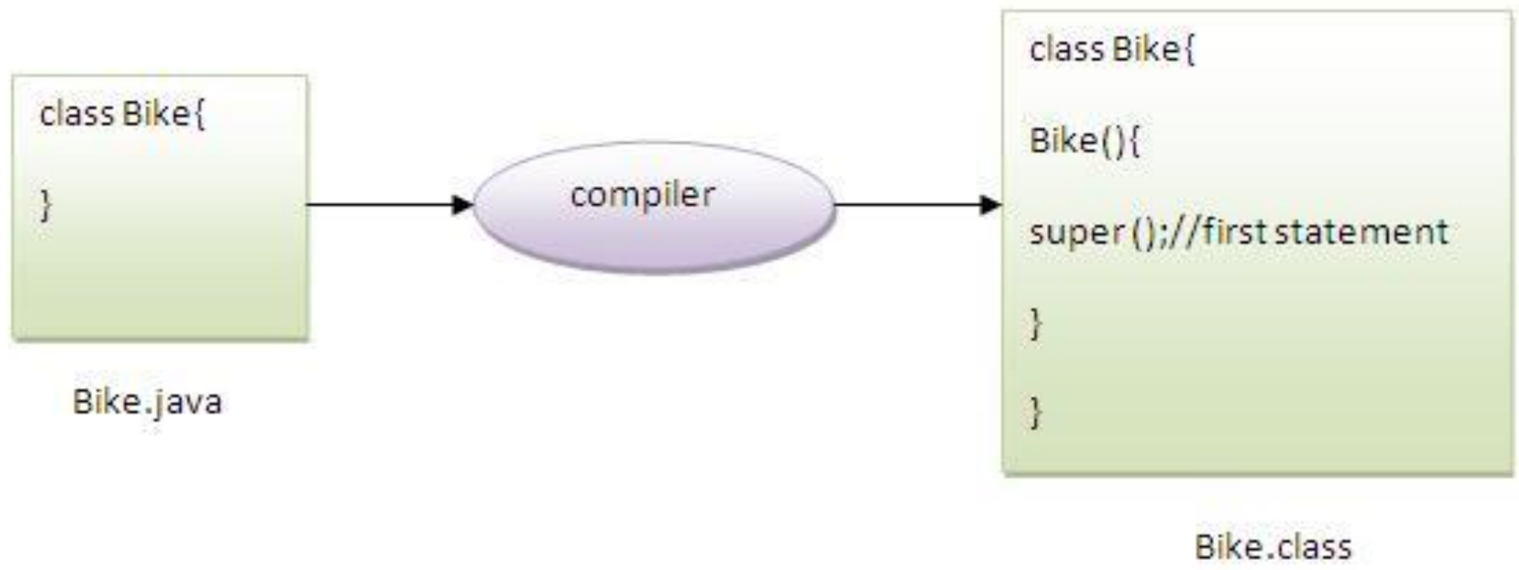
super keyword

- The **super** is a reference variable that is used to refer immediate parent class object.

- Whenever you create the instance of subclass, an instance of parent class is created implicitly i.e. referred by super reference variable.

- **Usage of super Keyword**
- super is used to refer immediate parent class instance variable.
- super() is used to invoke immediate parent class constructor.
- super is used to invoke immediate parent class method.

super() is added in each class constructor automatically by compiler.



As we know well that default constructor is provided by compiler automatically but it also adds super() for the first statement .If you are creating your own constructor and you don't have either this() or super() as the first statement, compiler will provide super() as the first statement of the constructor.

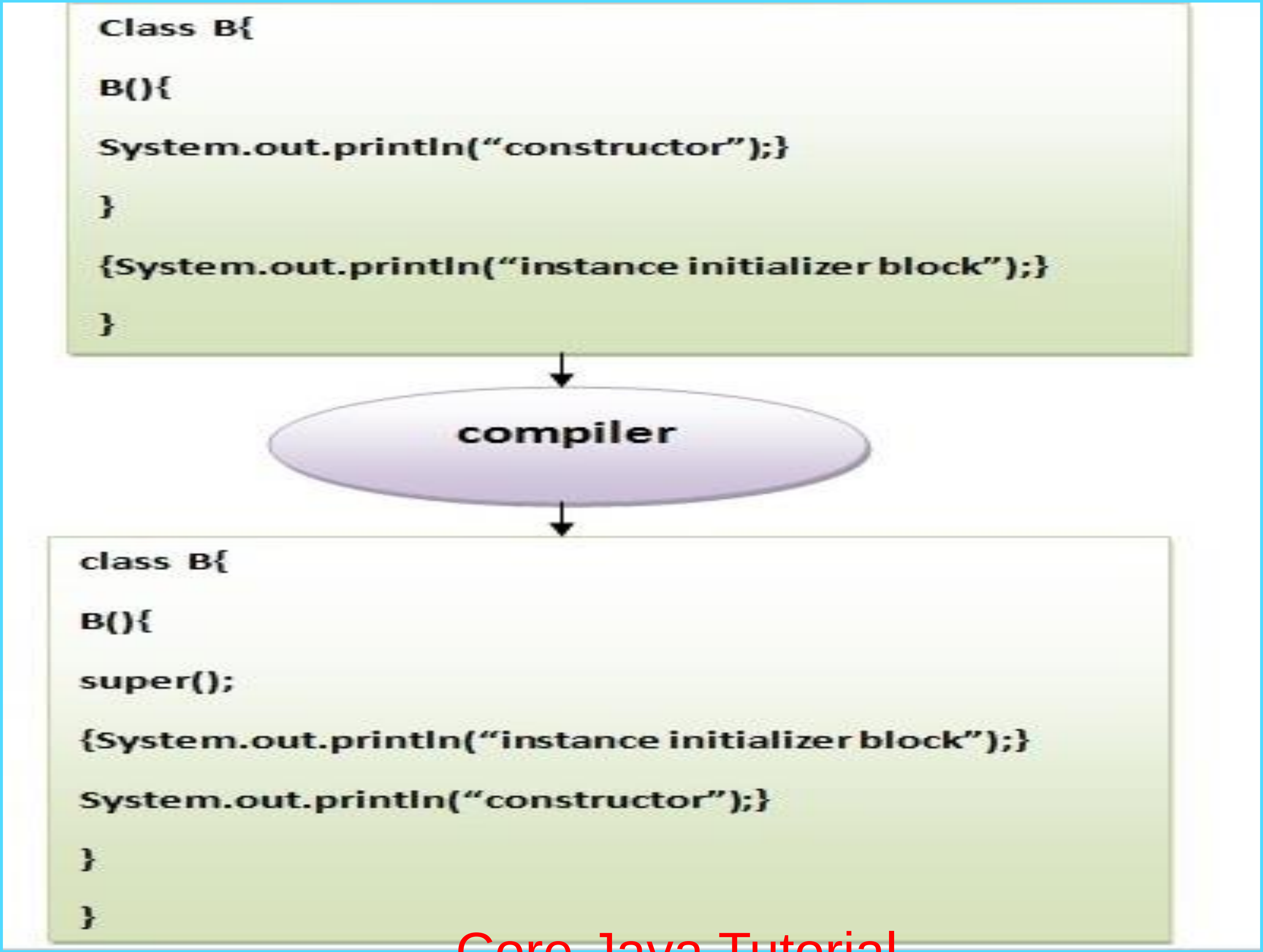
- ## Instance initializer block:

Instance Initializer block is used to initialize the instance data member. It run each time when object of the class is created. The initialization of the instance variable can be directly but there can be performed extra operations while initilizing the instance variable in the instance initializer block.

- Why use instance initializer block?

Suppose I have to perform some operations while assigning value to instance data member e.g. a for loop to fill a complex array or error handling etc.

Note: The java compiler copies the code of instance initializer block in every constructor.



```
Class B{
B(){
System.out.println("constructor");}
}
{System.out.println("instance initializer block");}
}
```

```
class B{
B(){
super();
{System.out.println("instance initializer block");}
System.out.println("constructor");}
}
}
```

- Rules for instance initializer block :

- There are mainly three rules for the instance initializer block. They are as follows:
- The instance initializer block is created when instance of the class is created.
- The instance initializer block is invoked after the parent class constructor is invoked (i.e. after super() constructor call).
- The instance initializer block comes in the order in which they appear.

Final Keyword in Java

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

- variable
- method
- class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.

- final variable

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

- final method

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

- final class

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

- **Is final method inherited?**
- Yes, final method is inherited but you cannot override it.

• Blank Final Variable

A final variable that is not initalized 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.

• 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.

- Abstract class in Java

- Abstraction

  Abstraction is a process of hiding the implementation details and showing only functionality to the user.

  Another way, it shows only important things to the user and hides the internal details for example sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery.

Abstraction lets you focus *on what the object does instead of how it does it.*

- **Ways to achieve Abstaction**

There are two ways to achieve abstraction in java

- Abstract class (0 to 100%)
- Interface (100%)

- Abstract class

A class that is declared as abstract is known as **abstract class**. It needs to be extended and its method implemented. It cannot be instantiated.

- **Syntax to declare the abstract class**

abstract class <class_name>{}

- **abstract method**

A method that is declared as abstract and does not have implementation is known as abstract method.

- **Syntax to define the abstract method**

abstract return_type <method_name>();//no brac es{}

- **Abstract class having constructor, data member, methods etc.**

- ***Note: An abstract class can have data member, abstract method, method body, constructor and even main() method.***

- ***Rule: If there is any abstract method in a class, that class must be abstract.***

- ***Rule: If you are extending any abstact class that have abstract method, you must either provide the implementation of the method or make this class abstract.***

- Interface

- An **interface** is a blueprint of a class. It has static constants and abstract methods.

The interface is **a mechanism to achieve fully abstraction** in java. There can be only abstract methods in the interface. It is used to achieve fully abstraction and multiple inheritance in Java.

- Interface also **represents IS-A relationship**.

  It cannot be instantiated just like abstract class.

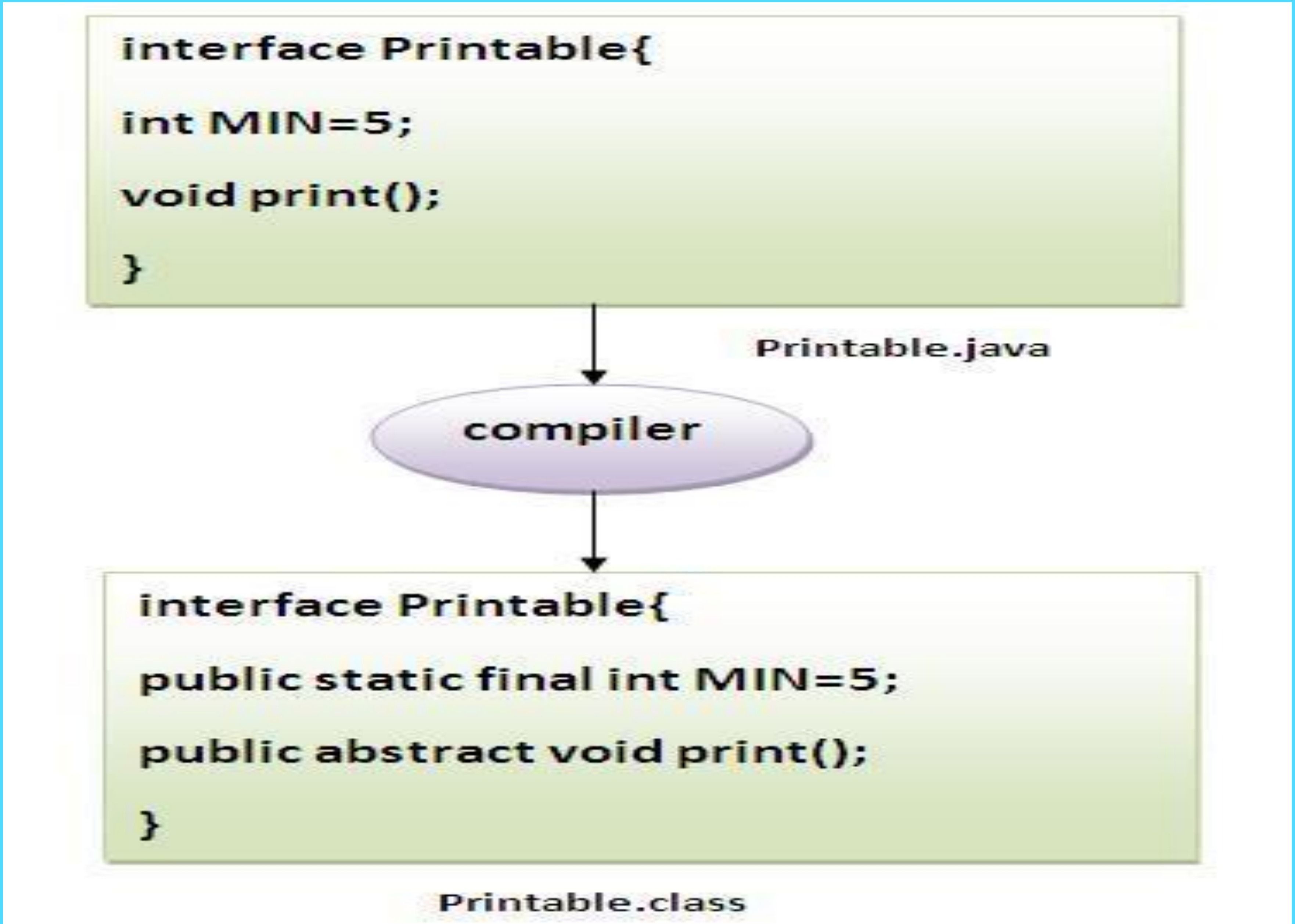
- Why use Interface?

  There are mainly three reasons to use interface. They are given below.

- It is used to achieve fully abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

- ***The java compiler adds public and abstract keywords before the interface method and public, static and final keywords before data members.***

In other words, Interface fields are public, static and final by default, and methods are public and abstract.

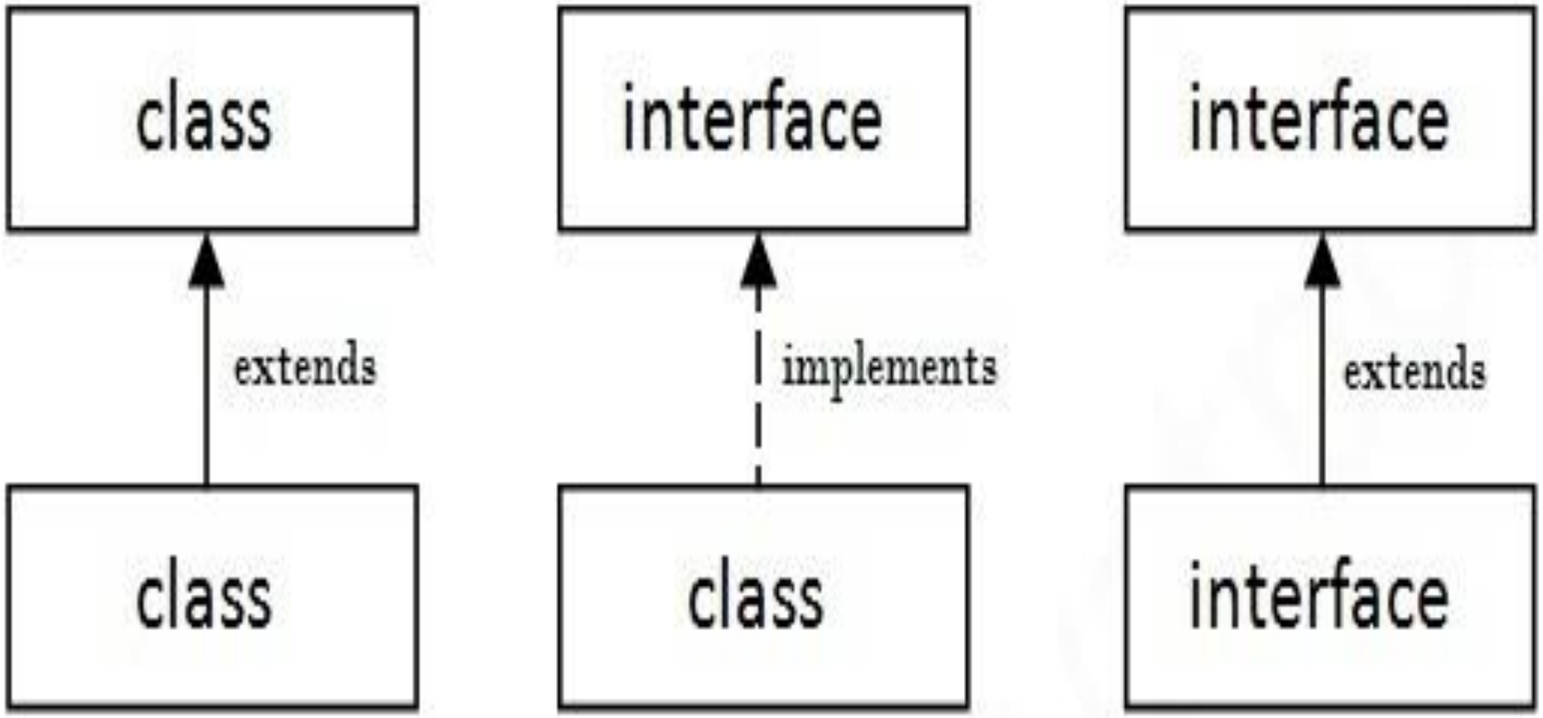


```
interface Printable{
int MIN=5;
void print();
}
```

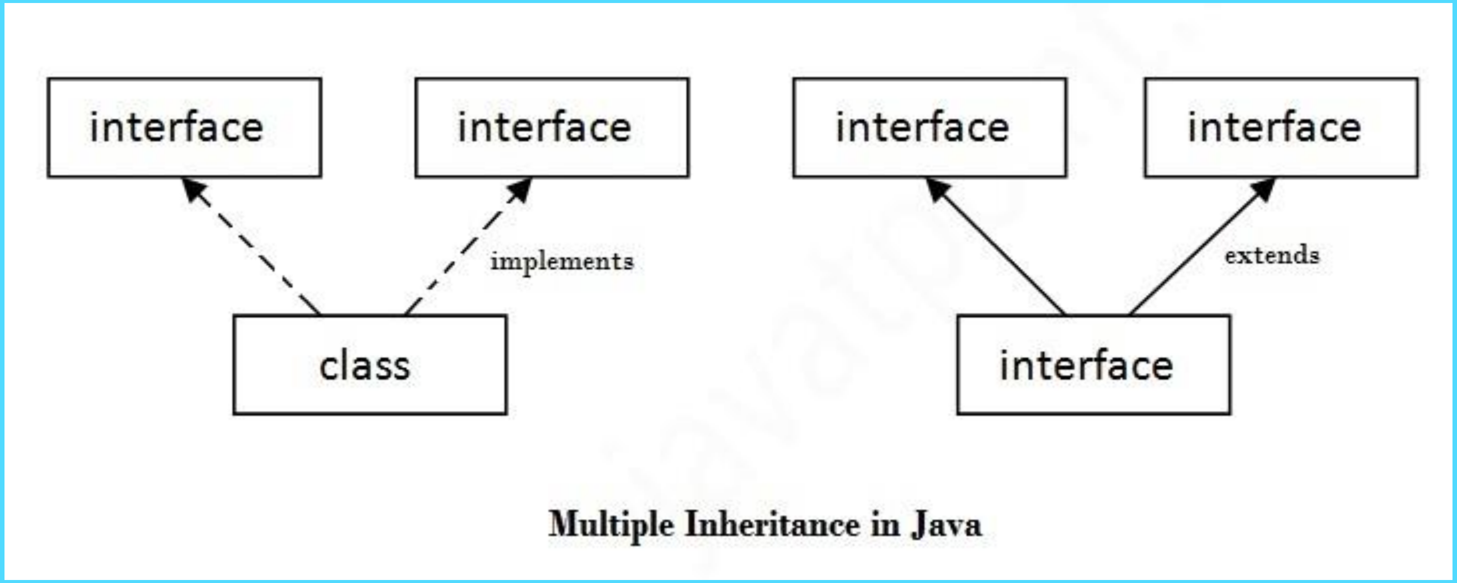
```
interface Printable{
public static final int MIN=5;
public abstract void print();
}
```

- **Understanding relationship between classes and interfaces**

As shown in the figure given below, a class extends another class, an interface extends another interface but a **class implements an interface**.


class
class
extends
interface
class
implements
interface
interface
extends

- Multiple inheritance in Java by interface
- **If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.**



- ***Note: A class implements interface but One interface extends another interface .***

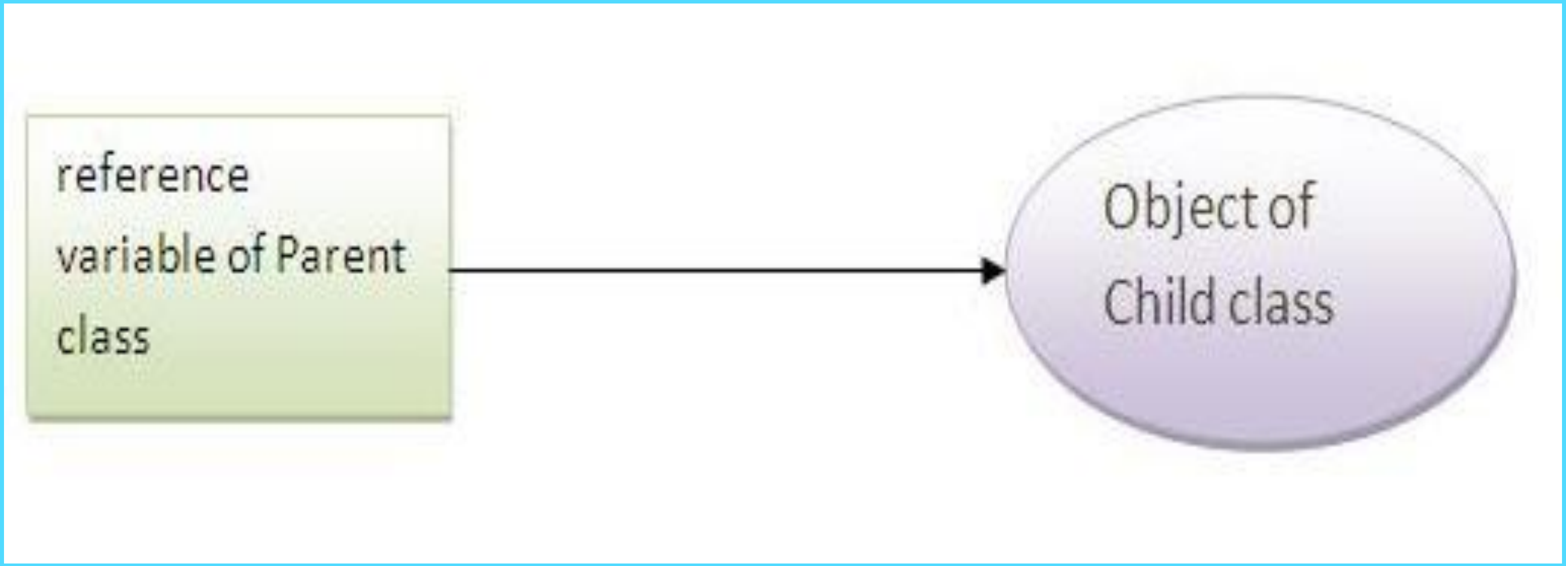
- Runtime Polymorphism

- **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 super class. 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 up casting before Runtime Polymorphism.
- **Up casting**

When reference variable of Parent class refers to the object of Child class, it is known as up casting. For example:



```
class A{}
class B extends A{}
A a=new B();//up casting
```

- ***Rule: Runtime polymorphism can't be achieved by data members.***

- 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).

Understanding Type

Let's understand the type of instance.

- **variables have a type**
- **Each variable has a type, it may be primitive and non-primitive.**

int data=30;

Here data variable is a type of int.

- **References have a type**

```
class Dog{
 public static void main(String args[]){
  Dog d1;  //Here d1 is a type of Dog
 }
}
```

- **Objects have a type**

An object is an instance of particular java class, but it is also an instance of its super class.

```
class Animal{}
class Dog extends Animal{
public static void main(String args[]){
Dog d1=new Dog();
}
}
```

Here d1 is an instance of Dog class, but it is also an instance of Animal.

- **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.

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

- instanceof operator

The instanceof operator is used to test whether the object is an instance of the specified type (class or subclass or interface).

The instanceof operator is also known as type comparison operator because it compares the instance with type. It returns either true or false. If we apply the instanceof operator with any variable that have null value, it returns false.

• Package

A **package** is a group of similar types of classes, interfaces and sub-packages.

Package can be categorized in two form, built-in package and user-defined package. There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

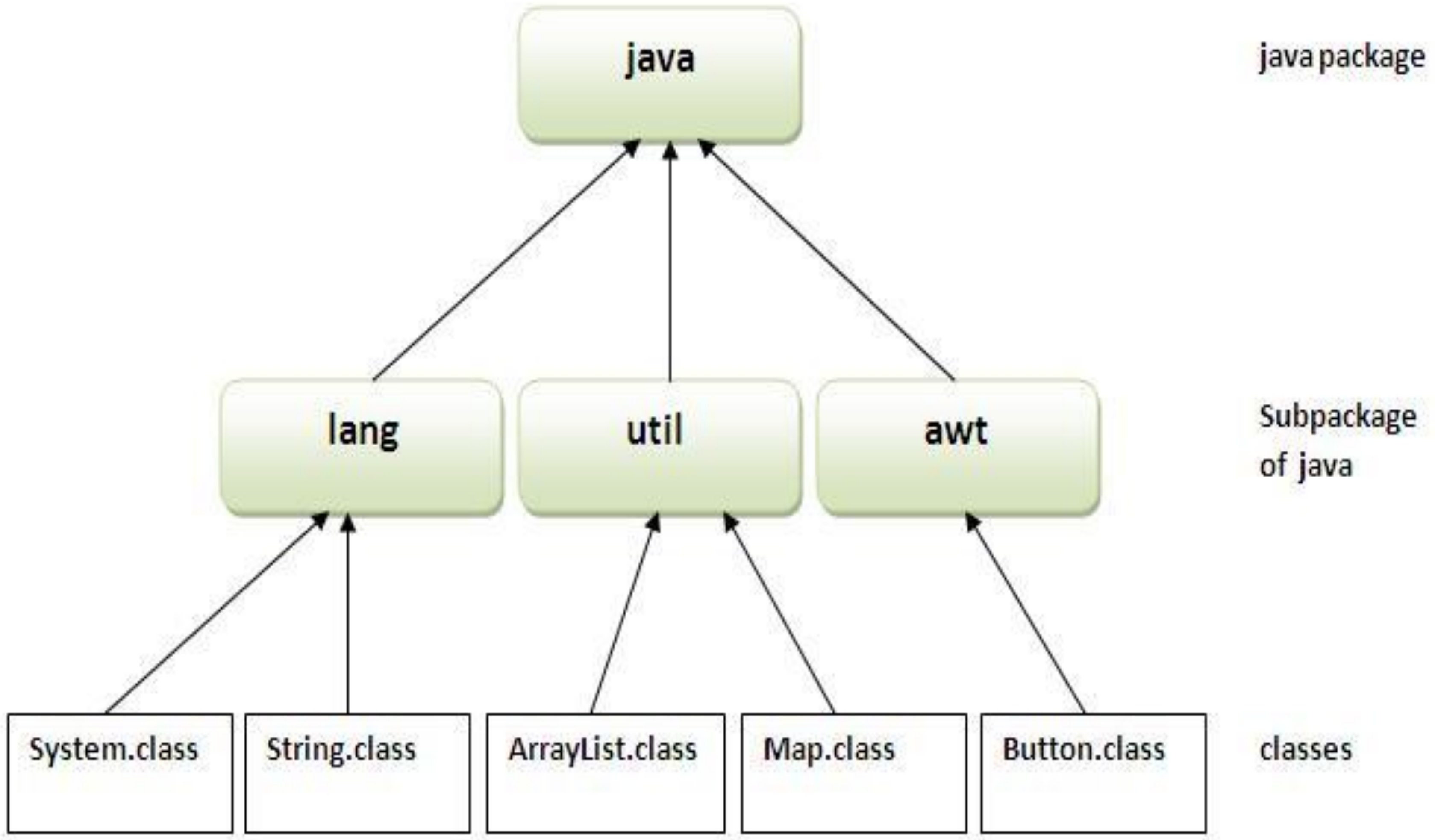
Here, we will have the detailed learning of creating and using user-defined packages.

- **Advantage of Package**

- Package is used to categorize the classes and interfaces so that they can be easily maintained.

- Package provides access protection.

- Package removes naming collision.


java
java package
lang
util
awt
Subpackage of java
System.class
String.class
ArrayList.class
Map.class
Button.class
classes

- **How to run the Package (if not using IDE) through Note Pad**

You need to use fully qualified name e.g. mypack. Simple etc to run the class.

- **To Compile:** javac -d . Simple.java
- **To Run:** java mypack.Simple

- Type the following 2 codes in c:\jdk1.6.0_24\bin and save as a. java & b.java respectively.

<u>a.Java</u>

```
package pack;
public class a
{
public void msg()
{
  System.out.println(“Hello”);
}
}
```

b.Java

```
package mypack;
import pack.*;
class b
{
 public static void main(String args[])
{
    a obj=new a();
obj.msg();
}
}
```

- Compiling Steps
- C:\jdk1.6.0_24\bin> javac –d . a.java
- C:\jdk1.6.0_24\bin> javac –d . b.java
- C:\jdk1.6.0_24\bin> set CLASSPATH =.;C:\Jdk1.6.0_24\bin;
- C:\jdk1.6.0_24\bin>java mypack.b

Ouput : Hello

Note: We set the Class path to point to 2 places. .(dot) & C:\dir

Set CLASS PATH=.;c:\....................

- How to access package from another package?
- There are three ways to access the package from outside the package.
- import package.*;
- import package.classname;
- fully qualified name.

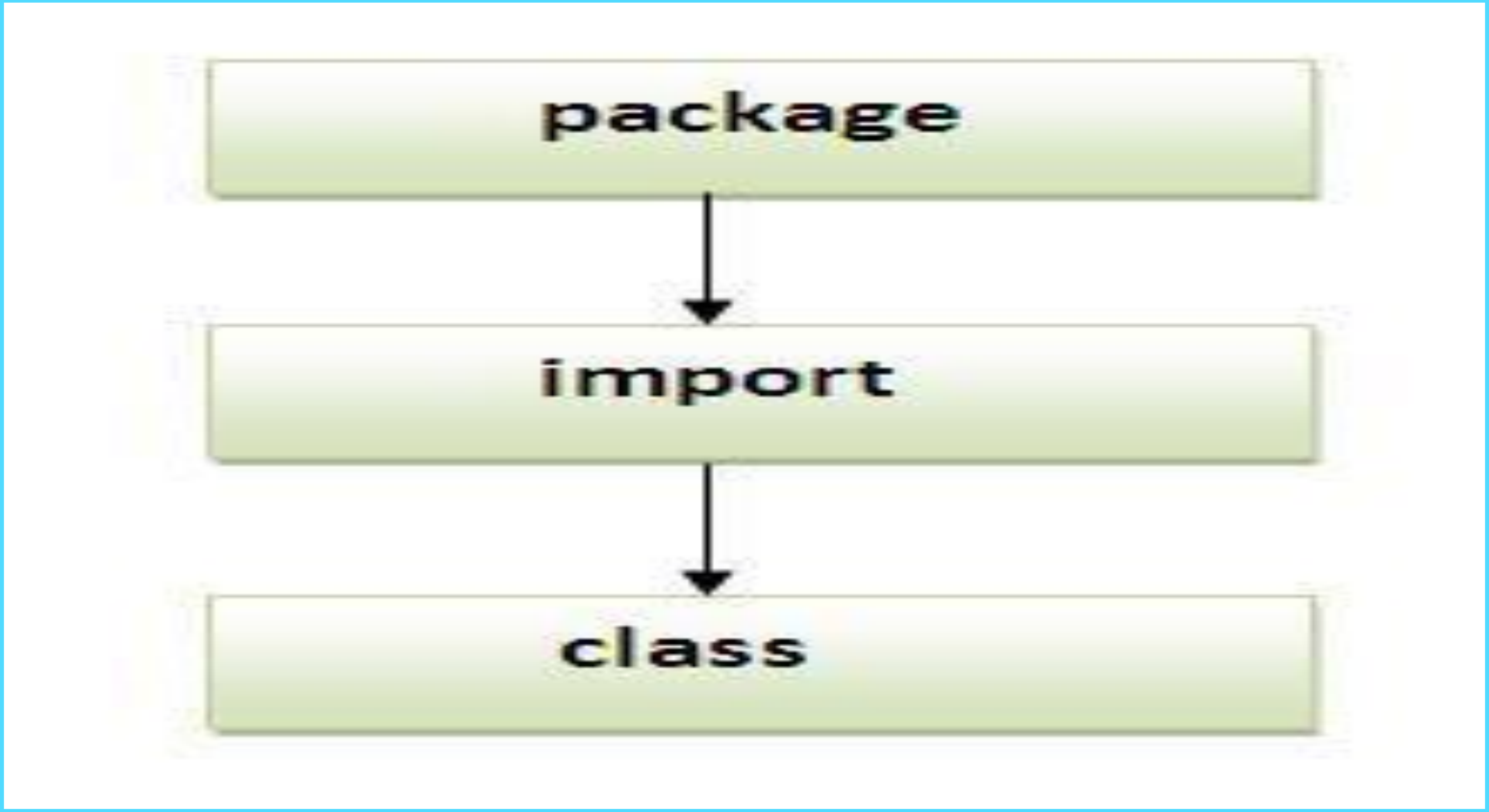
Using packagename.*

- If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.

- The import keyword is used to make the classes and interface of another package accessible to the current package.

- ***Note: If you import a package, subpackages will not be imported.***
- **If you import a package, all the classes and interface of that package will be imported excluding the classes and interfaces of the subpackages. Hence, you need to import the subpackage as well.**

Note: Sequence of the program must be package then import then class.



- Access Modifiers

- There are two types of modifiers in java: **access modifier** and **non-access modifier**. The access modifiers specifies accessibility (scope) of a data member, method, constructor or class.

- There are 4 types of access modifiers:

- private
- default
- protected
- public

There are many non-access modifiers such as static, abstract, synchronized, native, volatile, transient etc. Here, we will learn access modifiers.

- private

  The private access modifier is accessible only within class.

- ***Note: A class cannot be private or protected except nested class.***

- default
- If you don't use any modifier, it is treated as **default** by default. The default modifier is accessible only within package.

- protected
- **The protected access modifier is accessible within package and outside the package but through inheritance only.**
- **The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.**

- public

- The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

- Object class in Java

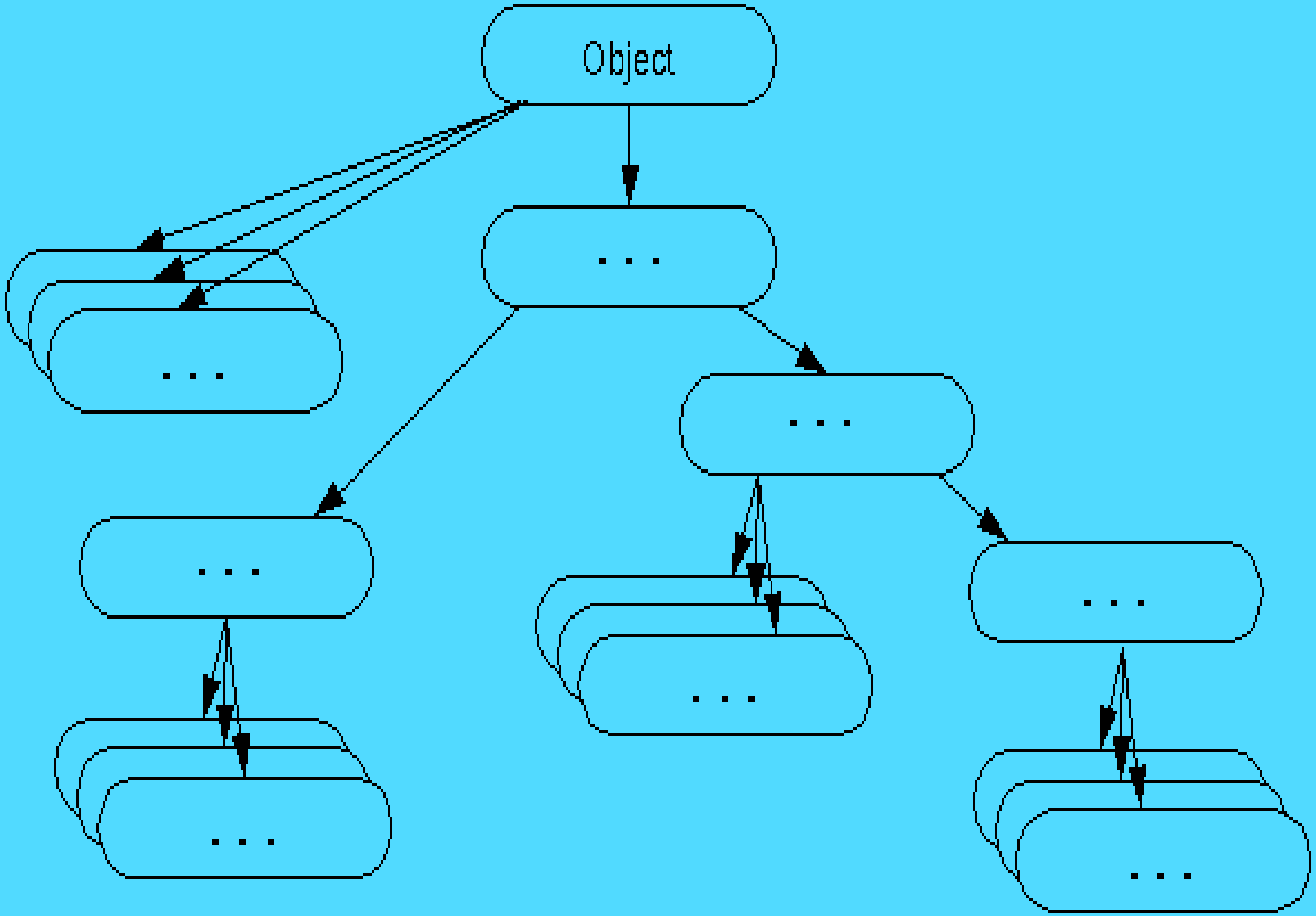
- The **Object class** is the parent class of all the classes in java bydefault. In other words, it is the topmost class of java.

- 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 up casting.

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 what object wo uld be returned from this method

The Object class provides some common behaviours to all the objects such as object can be compared, object can be cloned, object can be notified etc.


Object
. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .

- Methods of Object Class

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.

Method	Description
public final void wait(long timeout)throws InterruptedException	causes the current thread to wait for the specified miliseconds, 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 miliseconds 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.

- Object Cloning in Java
- The **object cloning** is a way to create exact copy of an object. For this purpose, clone() method of Object class is used to clone an object.

  The **java.lang.Cloneable interface** must be implemented by the class whose object clone we want to create. If we don't implement Cloneable interface, clone() method generates**CloneNotSupportedException**.

- The **clone() method** is defined in the Object class.
- Syntax of the clone() method is as follows:

- **protected**

  Object clone() **throws** CloneNotSupportedException

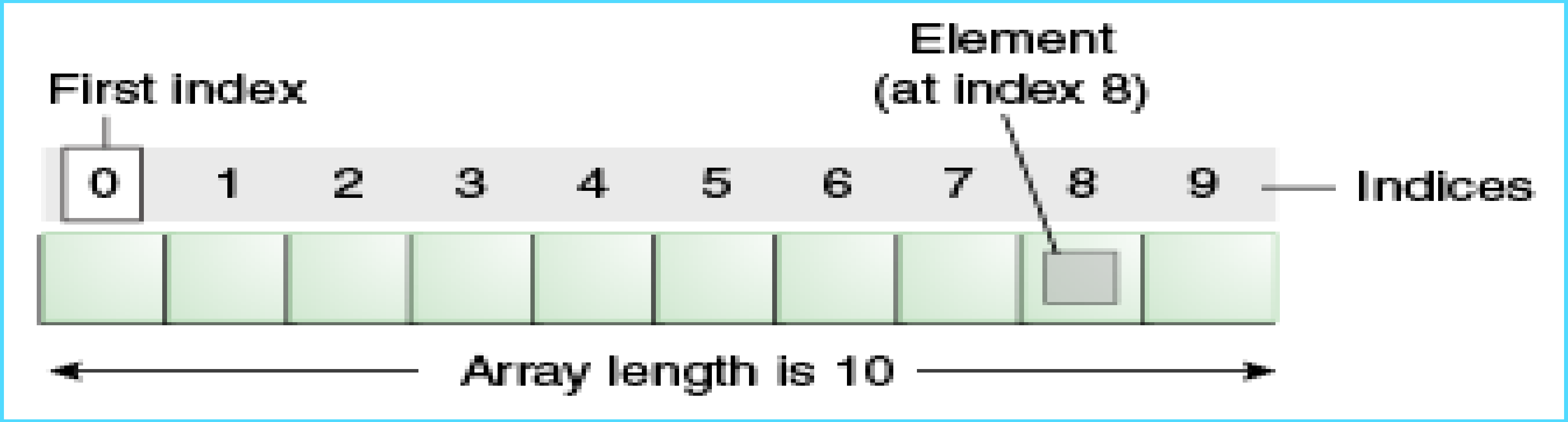
- **Why use clone() method ?**
- The **clone() method** saves the extra processing task for creating the exact copy of an object. If we perform it by using the new keyword, it will take a lot of processing to be performed that is why we use object cloning.
- **Advantage of Object cloning**
- Less processing task.

• Arrays in Java

Normally, array is a collection of similar type of elements that have contagious memory location.

In java, array is an object the contains elements of similar data type. It is a data structure where we store similar elements. We can store only fixed elements in an array.

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



- **Advantages of Array**
- **Code Optimization:** It makes the code optimized, we can retrive or sort the data easily.
- **Random access:** We can get any data located at any index position.

- **Disadvantage of Array**

- **Size Limit:** We can store only fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in java.
- Types of Array
- There are two types of array.
- Single Dimensional Array
- Multidimensional Array

- Single Dimensional Array
- **Syntax to Declare an Array in java**

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

- **Instantiation of an Array in java**

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

- Multidimensional array

In such case, data is stored in row and column based index (also known as matrix form).

- **Syntax to Declare Multidimensional Array in java**

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

- ## Call by Value and Call by Reference in Java

There is only call by value in java, not call by reference. If we call a method passing a value, it is known as call by value. The changes being done in the called method, is not affected in the calling method.

- Command Line Argument

The command-line argument is an argument passed at the time of running the java program. The argument can be received in the program and used as an input. So, it provides an convenient way to check out the behaviour of the program on different values. You can pass **N** numbers of arguments from the command prompt.

- String Handling

String Handling provides a lot of concepts that can be performed on a string such as concatenating string, comparing string, substring etc.In java, string is basically an immutable object. We will discuss about immutable string later. Let's first understand what is string and how we can create the string object. String

Generally string is a sequence of characters. But in java, string is an object. String class is used to create string object.

- How to create String object?

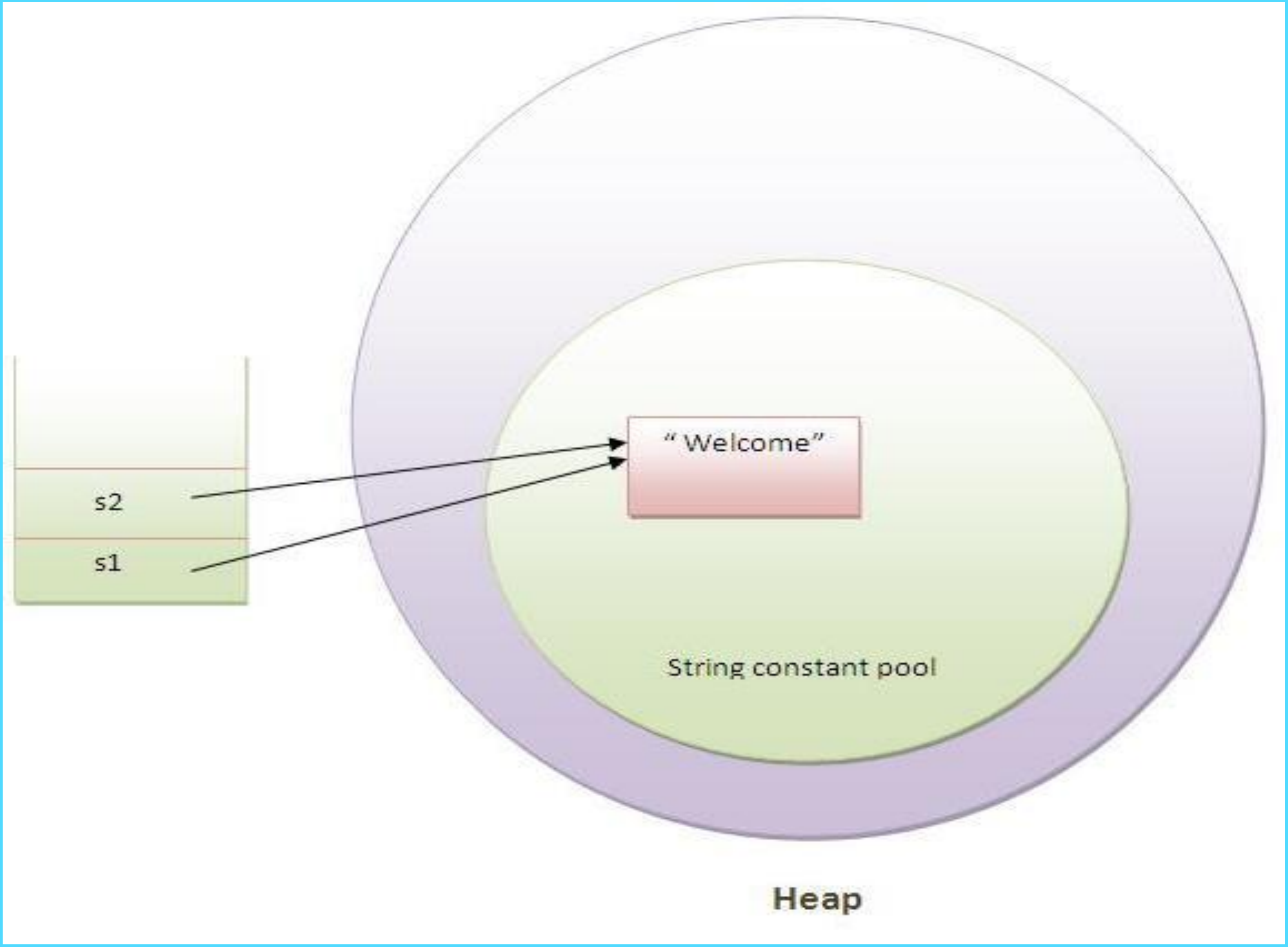
- There are two ways to create String object:
- By string literal
- By new keyword
- String literal:
-
- String literal is created by double quote.
  Example:
- String s="Hello";

Each time you create a string literal, the JVM checks the string constant pool first. If the string already exists in the pool, a reference to the pooled instance returns. If the string does not exist in the pool, a new String object instantiates, then is placed in the pool.

For example:

String s1="Welcome";

String s2="Welcome";//no new object will be cre ated


s2
s1
" Welcome"
String constant pool
Heap

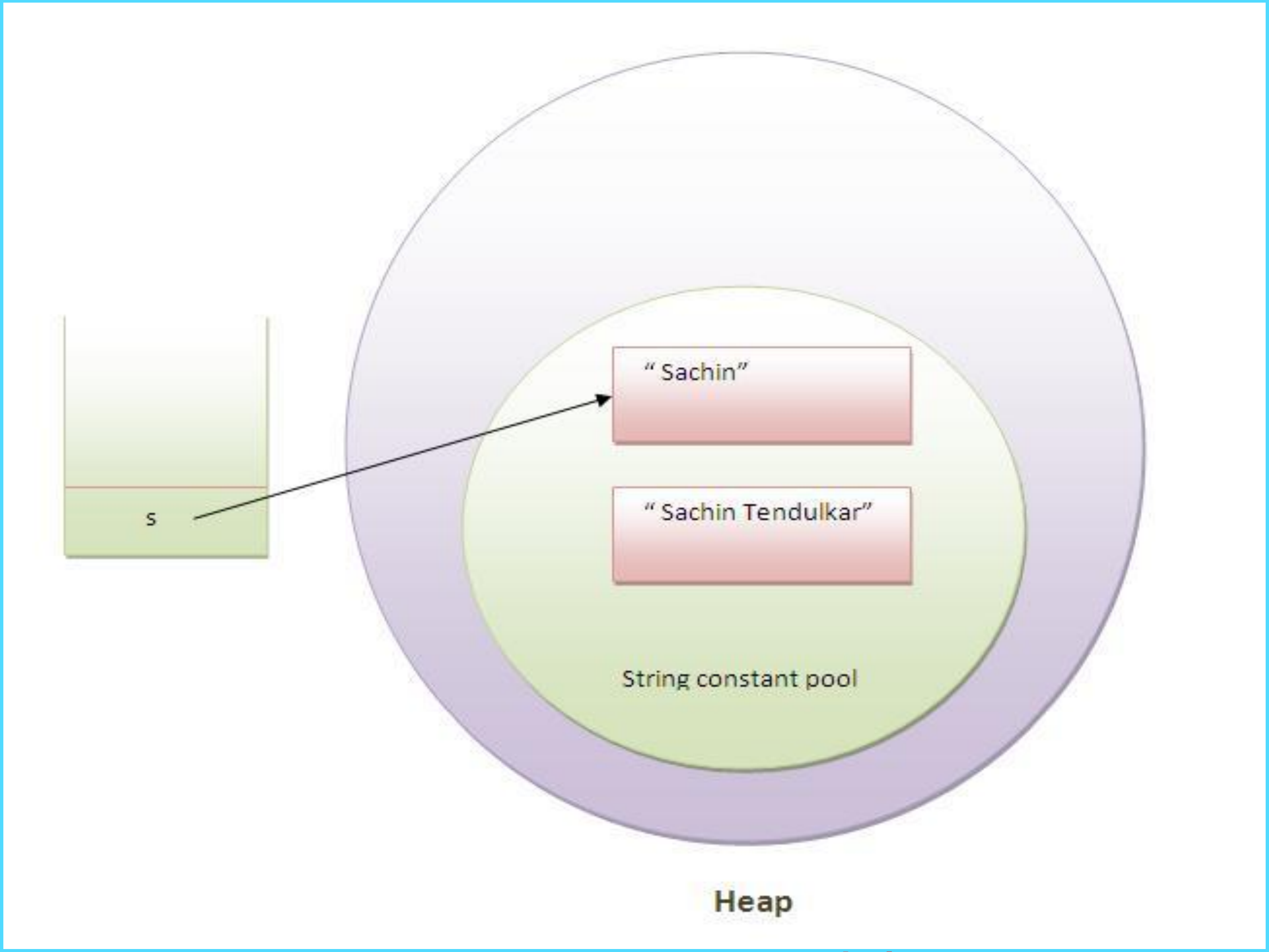
- By new keyword:

String s=**new** String("Welcome");//creates two objects and one reference variable

In such case, JVM will create a new String object in normal (nonpool) Heap memory and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in Heap(nonpool).

- Why string objects are immutable in java?

Because java uses the concept of string literal. Suppose there are 5 reference variables, all refers to one object "sachin". If one reference variable changes the value of the object, it will be affected to all the reference variables. That is why string objects are immutable in java.



s
" Sachin"
" Sachin Tendulkar"
String constant pool
Heap

- String comparison in Java

- We can compare two given on the basis of content and reference. It is used in authentication (equals() method), sorting (compareTo() method) etc.
- There are three ways to compare String objects:
- By equals() method
- By = = operator
- By compareTo() method

- **By equals() method:**
- equals() method compares the original content of the string.It compares values of string for equality.String class provides two methods:
- **public boolean equals(Object another){}** compares this string to the specified object.
- **public boolean equalsIgnoreCase(String another){}** compares this String to another String, ignoring case.

- **By == operator:**

  The = = operator compares references not values.

- **By compareTo() method:**

  compareTo() method compares values and returns an int which tells if the values compare less than, equal, or greater than.Suppose s1 and s2 are two string variables.If:**s1 == s2** :0

  s1 > s2 :positive value

  s1 < s2 :negative value

- String Concatenation in Java

Concatenating strings form a new string i.e. the combination of multiple strings.

- There are two ways to concat string objects:
- By + (string concatenation) operator
- By concat() method

Note: If either operand is a string, the resulting operation will be string concatenation. If both operands are numbers, the operator will perform an addition.

- By concat() method

concat() method concatenates the specified string to the end of current string.

- **Syntax:** public String concat(String another){}

- Substring in Java

A part of string is called **substring**. In other words, substring is a subset of another string.

In case of substring startIndex starts from 0 and endIndex starts from 1 or startIndex is inclusive and endIndex is exclusive.

You can get substring from the given String object by one of the two methods:

public String substring(int startIndex): This method returns new String object containing the substring of the given string from specified startIndex (inclusive).

public String substring(int startIndex,int endIndex): This method returns new String object containing the substring of the given string from specified startIndex to endIndex.

- In case of string: **startIndex:** starts from index 0(inclusive).
- **endIndex:** starts from index 1(exclusive).
- Methods of String class

Method	**Description**
1)public boolean equals(Object anObject)	Compares this string to the specified object.
2)public boolean equalsIgnoreCase(String another)	Compares this String to another String, ignoring case.
3)public String concat(String str)	Concatenates the specified string to the end of this string.

Method	Description
4)public int compareTo(String str)	Compares two strings and returns int
5)public int compareToIgnoreCase(String str)	Compares two strings, ignoring case differences.
6)public String substring(int beginIndex)	Returns a new string that is a substring of this string.
7)public String substring(int beginIndex,int endIndex)	Returns a new string that is a substring of this string.
8)public String toUpperCase()	Converts all of the characters in this String to upper case
9)public String toLowerCase()	Converts all of the characters in this String to lower case.
10)public String trim()	Returns a copy of the string, with leading and trailing whitespace omitted.

Method	Description
11)public boolean startsWith(String prefix)	Tests if this string starts with the specified prefix.
12)public boolean endsWith(String suffix)	Tests if this string ends with the specified suffix.
13)public char charAt(int index)	Returns the char value at the specified index.
14)public int length()	Returns the length of this string.
15)public String intern()	Returns a canonical representation for the string object.

- StringBuffer class:

The StringBuffer class is used to created mutable (modifiable) string. The StringBuffer class is same as String except it is mutable i.e. it can be changed.

Note: StringBuffer class is thread-safe i.e. multiple threads cannot access it simultaneously .So it is safe and will result in an order.

- Commonly used Constructors of StringBuffer class:
- **StringBuffer():** creates an empty string buffer with the initial capacity of 16.
- **StringBuffer(String str):** creates a string buffer with the specified string.
- **StringBuffer(int capacity):** creates an empty string buffer with the specified capacity as length.
- Commonly used methods of StringBuffer class:

- **public synchronized StringBuffer append(String s):**

  is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.

- **public synchronized StringBuffer insert(int offset, String s):** is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.

- **public synchronized StringBuffer replace(int startIndex, int endIndex, String str):** is used to replace the string from specified startIndex and endIndex.

- **public synchronized StringBuffer delete(int startIndex, int endIndex):** is used to delete the string from specified startIndex and endIndex.
- **public synchronized StringBuffer reverse():** is used to reverse the string.
- **public int capacity():** is used to return the current capacity.
- **public void ensureCapacity(int minimumCapacity):** is used to ensure the capacity at least equal to the given minimum.

- **public char charAt(int index):** is used to return the character at the specified position.
- **public int length():** is used to return the length of the string i.e. total number of characters.
- **public String substring(int beginIndex):** is used to return the substring from the specified beginIndex.
- **public String substring(int beginIndex, int endIndex):** is used to return the substring from the specified beginIndex and endIndex.

- What is mutable string?

A string that can be modified or changed is known as mutable string. StringBuffer and StringBuilder classes are used for creating mutable string.

- StringBuilder class:

- The StringBuilder class is used to create mutable (modifiable) string. The StringBuilder class is same as StringBuffer class except that it is non-synchronized. It is available since JDK1.5.Commonly used Constructors of StringBuilder class:

- **StringBuilder():** creates an empty string Builder with the initial capacity of 16.
- **StringBuilder(String str):** creates a string Builder with the specified string.

- **StringBuilder(int length):** creates an empty string Builder with the specified capacity as length.
- Commonly used methods of StringBuilder class:
- **public StringBuilder append(String s):** is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.

- **public StringBuilder insert(int offset, String s):** is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.

- **public StringBuilder replace(int startIndex, int endIndex, String str):** is used to replace the string from specified startIndex and endIndex.

- **public StringBuilder delete(int startIndex, int endIndex):** is used to delete the string from specified startIndex and endIndex.
- **public StringBuilder reverse():** is used to reverse the string.
- **public int capacity():** is used to return the current capacity.
- **public void ensureCapacity(int minimumCapacity):** is used to ensure the capacity at least equal to the given minimum.

- **public char charAt(int index):** is used to return the character at the specified position.
- **public int length():** is used to return the length of the string i.e. total number of characters.
- **public String substring(int beginIndex):** is used to return the substring from the specified beginIndex.
- **public String substring(int beginIndex, int endIndex):** is used to return the substring from the specified beginIndex and endIndex.

- Understanding toString() method

If you want to represent any object as a string, **toString() method** comes into existence.

The toString() method returns the string representation of the object.

If you print any object, java compiler internally invokes the toString() method on the object. So overriding the toString() method, returns the desired output, it can be the state of an object etc. depends on your implementation.

• Advantage of the toString() method

By overriding the toString() method of the Object class, we can return values of the object, so we don't need to write much code.

- Exception Handling in Java

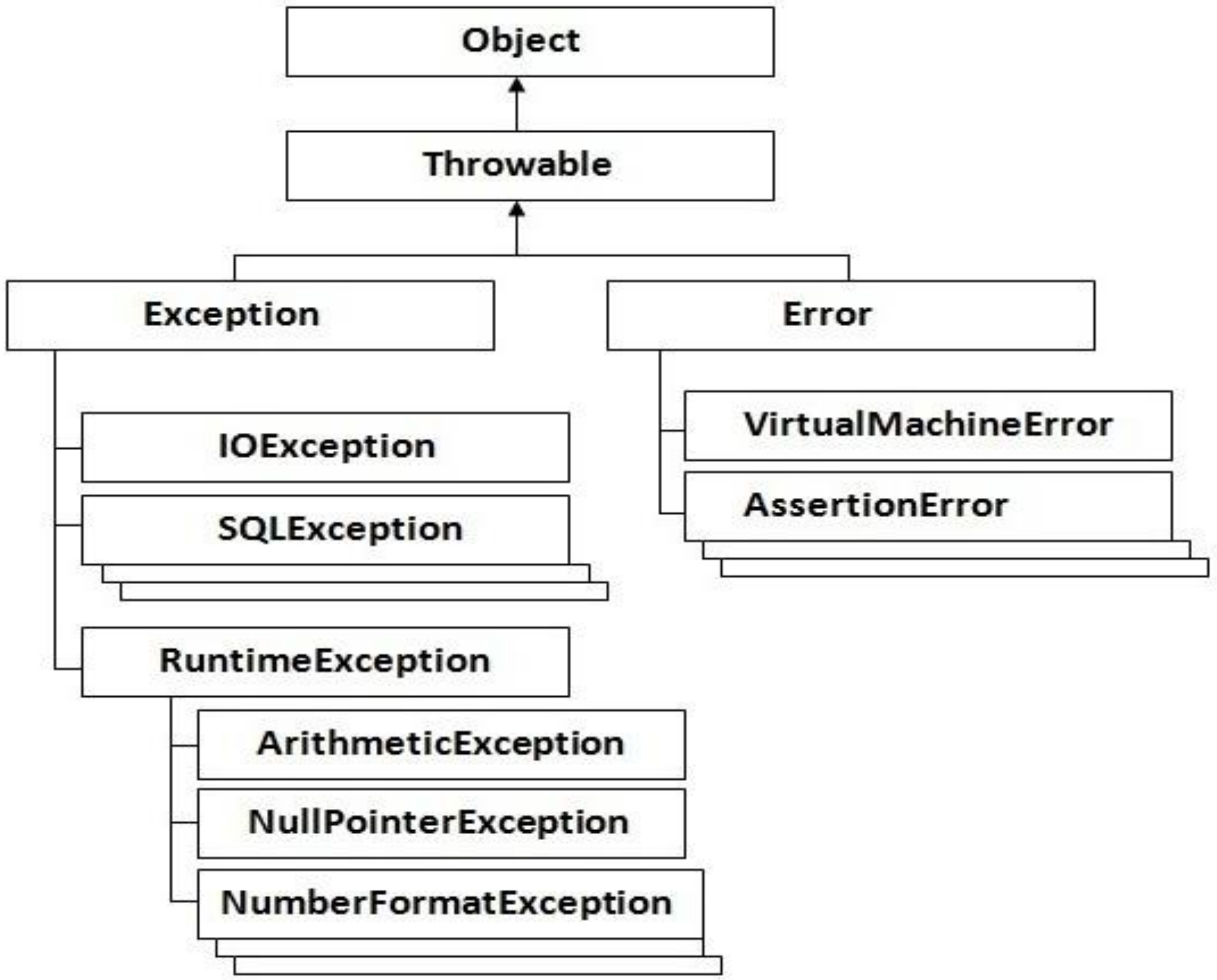
The exception handling is one of the powerful mechanism provided in java. It provides the mechanism to handle the runtime errors so that normal flow of the application can be maintained.

In this page, we will know about exception, its type and the difference between checked and unchecked exceptions.

- Exception

- **Dictionary Meaning:** Exception is an abnormal condition.
- In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

- Exception Handling
- Exception Handling is a mechanism to handle runtime errors.


Object
Throwable
Exception
Error
IOException
SQLException
RuntimeException
ArithmeticException
NullPointerException
NumberFormatException
VirtualMachineError
AssertionError

- Types of Exception:

- There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. The sun microsystem says there are three types of exceptions:

- Checked Exception
- Unchecked Exception
- Error

- **Checked Exception**

- The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException, SQLException etc. Checked exceptions are checked at compile-time.

- **Unchecked Exception**
- The classes that extend RuntimeException are known as unchecked exceptions

e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

- **Error**

- Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

- **Scenario where ArithmeticException occurs**

  If we divide any number by zero, there occurs an ArithmeticException.

  int a=50/0;//ArithmeticException

- **Scenario where NullPointerException occurs**

  we have null value in any variable, performing any operation by the variable occurs an NullPointerException.

  String s=**null**;

  System.out.println(s.length());//NullPointerException

- **Scenario where NumberFormatException occurs**

The wrong formatting of any value, may occur NumberFormatException. Suppose I have a string variable that have characters, converting this variable into digit will occur NumberFormatException.

String s="abc";

int i=Integer.parseInt(s);//NumberFormatException

- **Scenario where ArrayIndexOutOfBoundsException occurs**

If you are inserting any value in the wrong index, it would result ArrayIndexOutOfBoundsException as shown below:

int a[]=**new** **int**[5];

a[10]=50; //ArrayIndexOutOfBoundsException

- Use of try-catch block in Exception handling
- Five keywords used in Exception handling:
- try
- catch
- finally
- throw
- throws

- try block

Enclose the code that might throw an exception in try block. It must be used within the method and must be followed by either catch or finally block.**Syntax of try with catch block**

```
try{
  ...
}catch(Exception_class_Name reference){}
```

- **Syntax of try with finally block**

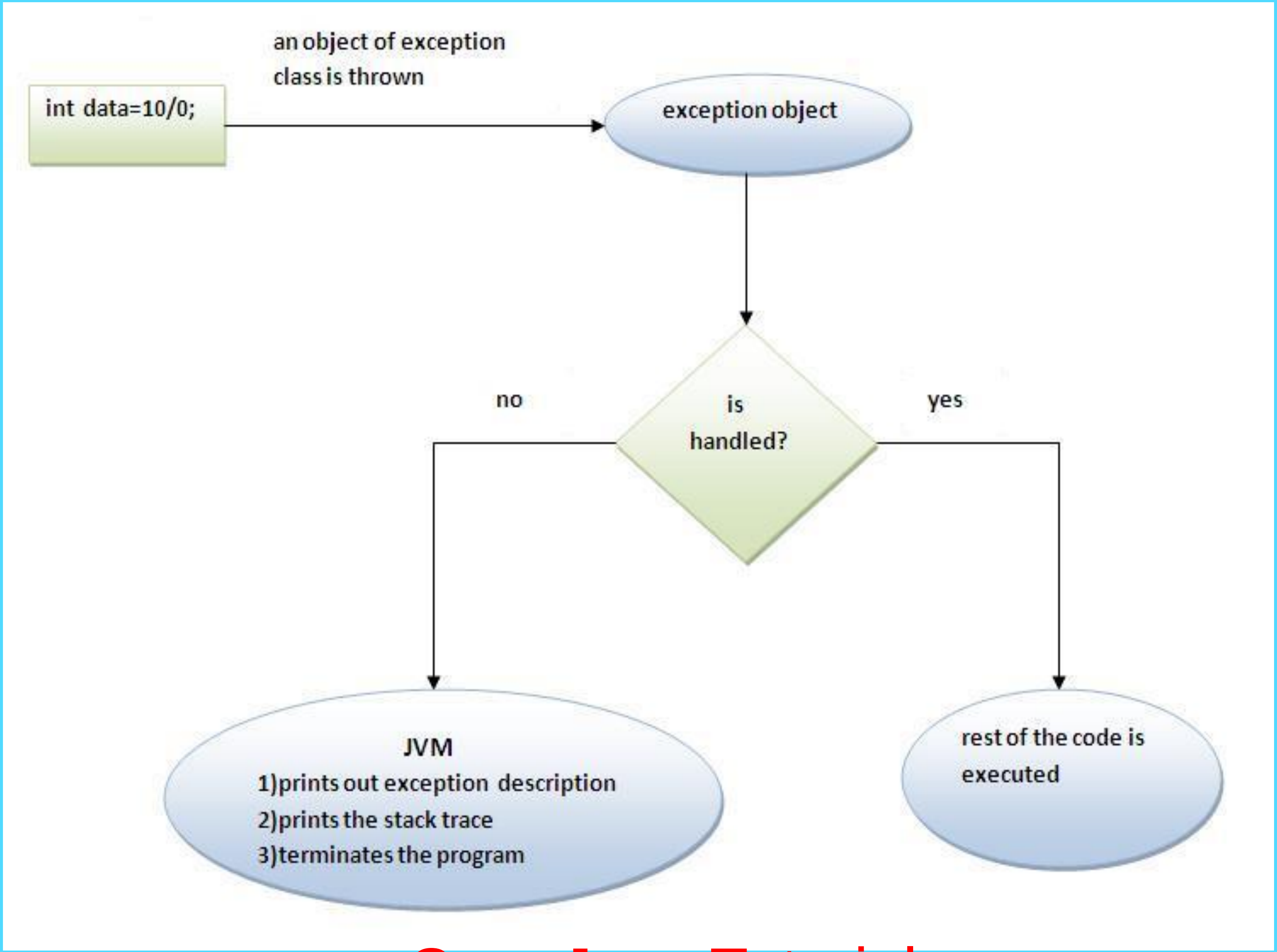
  try{

  ...

  }**finally**{}

- catch block

- Catch block is used to handle the Exception. It must be used after the try block.



an object of exception class is thrown
int data=10/0;
exception object
is handled?
no
yes
JVM
1)prints out exception description
2)prints the stack trace
3)terminates the program
rest of the code is executed

The JVM firstly checks whether the exception is handled or not. If exception is not handled, JVM provides a default exception handler that performs the following tasks:

- Prints out exception description.
- Prints the stack trace (Hierarchy of methods where the exception occurred).
- Causes the program to terminate.
- But if exception is handled by the application programmer, normal flow of the application is maintained i.e. rest of the code is executed.

- ***Rule:At a time only one Exception is occured and at a time only one catch block is executed.***

- ***Rule:All catch blocks must be ordered from most specific to most general i.e. catch for ArithmeticException must come before catch for Exception .***

If you have to perform different tasks at the occrence of different Exceptions, use multple catch block.

- Nested try block:

- try block within a try block is known as nested try block.

Syntax:

```
....
try
{
   statement 1;
   statement 2;
   try
   {
      statement 1;
      statement 2;
   }
   catch(Exception e)
   {
   }
}
catch(Exception e)
{
}
....
```

- finally block

The finally block is a block that is always executed. It is mainly used to perform some important tasks such as closing connection, stream etc.

- **Note:**Before terminating the program, JVM executes finally block(if any).
- **Note:** finally must be followed by try or catch block.

- **finally creates a block of code that will be executed after a try/catch block has**
- completed and before the code following the **try/catch block. The finally block will**
- execute whether or not an exception is thrown. If an exception is thrown, the **finally**
- block will execute even if no **catch statement matches the exception. Any time a method**
- is about to return to the caller from inside a **try/catch block, via an uncaught exception or**

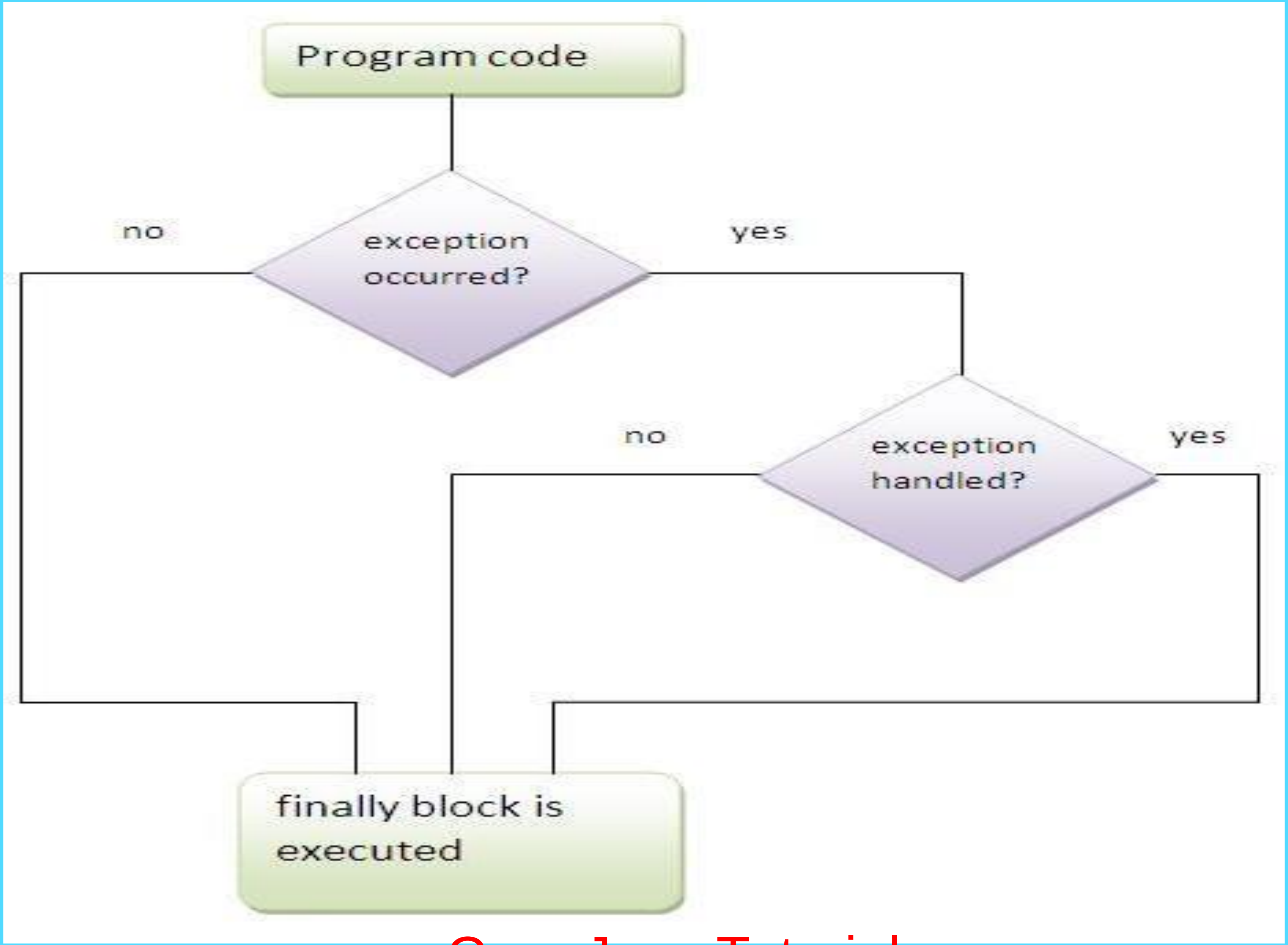
- an explicit return statement, the **finally clause is also executed just before the method**
- returns. This can be useful for closing file handles and freeing up any other resources that
- might have been allocated at the beginning of a method with the intent of disposing of them
- before returning. The **finally clause is optional.**

- **Note:**Before terminating the program, JVM executes finally block(if any).
- **Note:**finally must be followed by try or

  catch block.

- Why use finally block?

- Finally block can be used to put "cleanup" code such as closing a file, closing connection etc.
-


Program code
exception occurred?
no
yes
exception handled?
no
yes
finally block is executed

- ***For each try block there can be zero or more catch blocks, but only one finally block.***

Note: ***The finally block will not be executed if program exits(either by calling System.exit() or by causing a fatal error that causes the process to abort).***

• throw keyword

The throw keyword is used to explictily throw an exception.We can throw either checked or uncheked exception. The throw keyword is mainly used to throw custom exception. We will see custom exceptions later.

- Exception propagation:

  An exception is first thrown from the top of the stack and if it is not caught, it drops down the call stack to the previous method, If not caught there, the exception again drops down to the previous method, and so on until they are caught or until they reach the very bottom of the call stack. This is called exception propagation.

- ***Rule:* By default Unchecked Exceptions are forwarded in calling chain (propagated).**

- ***Rule:*** ***By default, Checked Exceptions are not forwarded in calling chain (propagated).***

Program which describes that checked exceptions are not propagated

- throws keyword:

The throws keyword is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing check up before the code being used.

- Syntax of throws keyword:

  void method_name() **throws** exception_class_name{

  ...

  }

- Advantage of throws keyword:

- Now Checked Exception can be propagated (forwarded in call stack).

- Rule: If you are calling a method that declares an exception, you must either caught or declare the exception.
- There are two cases:

- **Case1:**You caught the exception i.e. handle the exception using try/catch.

- **Case2:**You declare the exception i.e. specifying throws with the method.

Throw	Throws
1)throw is used to explicitly throw an exception.	throws is used to declare an exception.
2)checked exception can not be propagated without throws.	checked exception can be propagated with throws.
3)throw is followed by an instance.	throws is followed by class.
4)throw is used within the method.	throws is used with the method signature.
5)You cannot throw multiple exception	You can declare multiple exception e.g. public void method()throws IOException,SQLException.

- ExceptionHandling with MethodOverriding

There are many rules if we talk about methodoverriding with exception handling. The Rules are as follows:

- **If the superclass method does not declare an exception**
- If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but it can declare unchecked exception.

- **If the superclass method declares an exception**
- If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.
- If the superclass method does not declare an exception
- ***Rule:*** ***If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception.***

- ***Rule: If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but can declare unchecked exception.***

- If the superclass method declares an exception

- ***Rule:* If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.**
- Custom Exception :
- If you are creating your own Exception that is known as custom exception or user-defined exception.

- Nested classes (Inner classes)

- A class declared inside a class is known as nested class. We use nested classes to logically group classes in one place so that it can be more readable and maintainable code. Moreover, it can access all the members of outer class including private members.

- **Syntax of Nested class**

```
class Outer_class_Name{
 ...
 class Nested_class_Name{
  ...
 }
 ...
}
```

- Advantage of nested classes

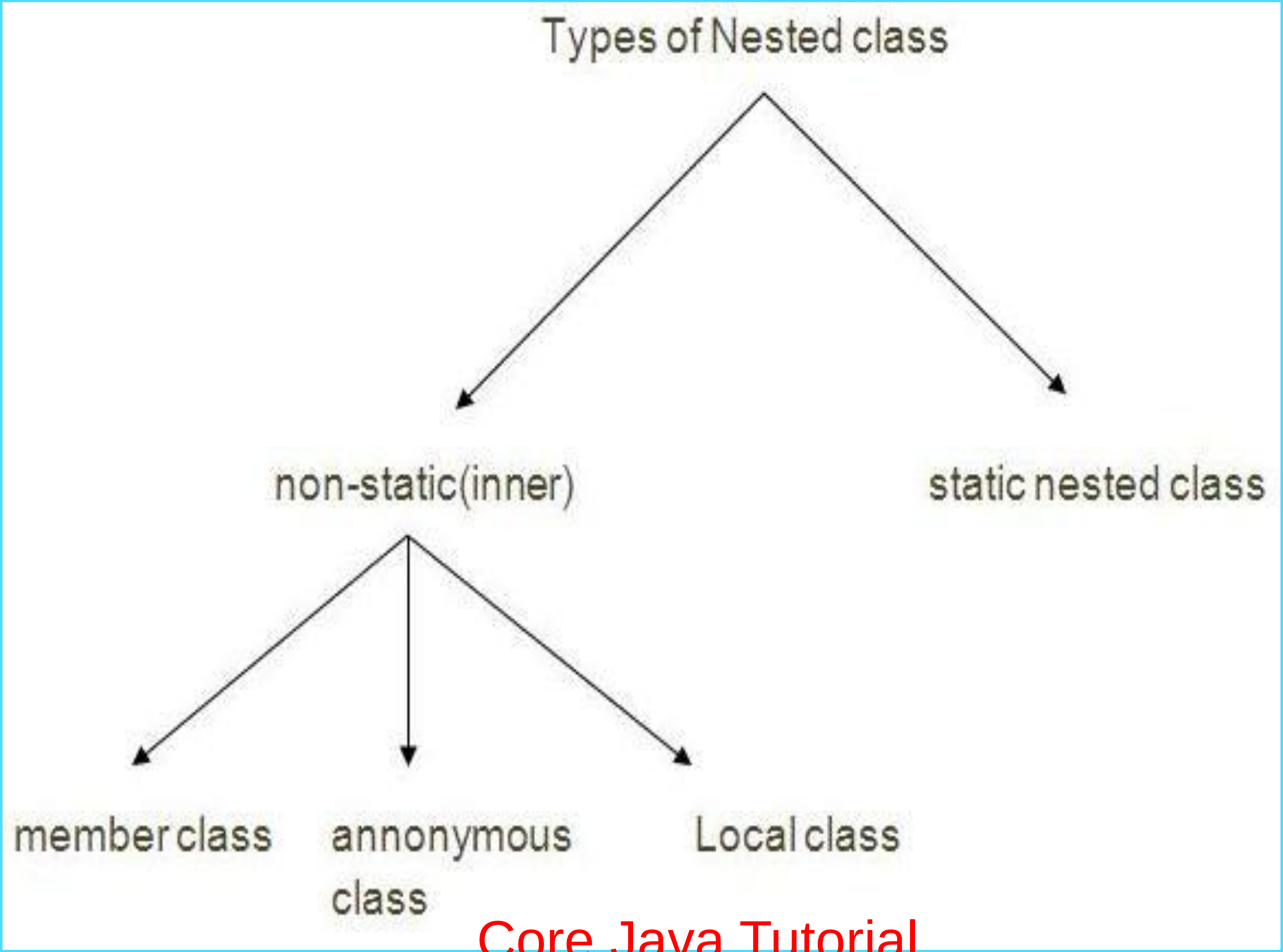
- There are basically three advantages of nested classes. They are
- Nested classes represent a special type of relationship that is it can access all the members (data members and methods) of outer class including private.
- Nested classes can lead to more readable and maintainable code because it logically group classes in one place only.
- Code Optimization as we need less code to write.

- What is the difference between nested class and inner class?
- Inner class is a part of nested class. Non-static nested classes are known as nested classes.

- Types of Nested class:
- There are two types of nested classes
- non-static
- static nested classes.

The non-static nested classes are also known as inner classes. Non-static nested class(inner class)

  - Member inner class
  - Annomynous inner class
  - Local inner class
- static nested class



Core Java Tutorial

- Member inner class

- A class that is declared inside a class but outside a method is known as member inner class.

- Invocation of Member Inner class

- From within the class
- From outside the class

- anonymous class
- **What does this mean?**
- An anonymous class is just what its name implies -- it has no name. It combines the class declaration and the creation of an instance of the class in one step.
- Probably most of the classes you have developed so far have been named classes. Each of these classes contained a full definition; you could instantiate objects from any of these classes.

- When defining an anonymous class, the definition and object instantiation is done all in one step. The class itself has no name (this is why it's *anonymous*) and this is the only way an object can be instantiated.
- Since anonymous classes have no name, objects can not be instantiated from outside the class in which the anonymous class is defined. In fact, an anonymous object can only be instantiated from within the same *scope* in which it is defined.

- **Why use an anonymous class?**
- Anonymous classes can be time-savers and reduce the number of .java files necessary to define an application. You may have a class that is only used in a specific situation such as a Comparator. This allows an "on the fly" creation of an object.
- You may find you prefer to use anonymous classes; many people use them extensively to implement listeners on GUIs.

- Annonymous inner class

- A class that have no name is known as annomymous inner class.

Annonymous class can be created by:

- Class (may be abstract class also).
- Interface

- A class is created but its name is decided by the compiler which extends the Person class and provides the implementation of the eat() method.
- An object of Annonymous class is created that is reffered by p reference variable of Person type. As you know well that Parent class reference variable can refer the object of Child class.

- Local inner class

- A class that is created inside a method is known as local inner class. If you want to invoke the methods of local inner class, you must instantiate this class inside the method.

- ***Rule: Local variable can't be private, public or protected.***

- ***Local inner class cannot be invoked from outside the method.***

- ***Local inner class cannot access non-final local variable.***

- static nested class

- A static class that is created inside a class is known as static nested class. It cannot access the non-static members.

- It can access static data members of outer class including private.

- static nested class cannot access non-static (instance) data member or method.

- Nested Interface

- An interface which is declared within another interface or class is known as nested interface. The nested interfaces are used to group related interfaces so that they can be easy to maintain. The nested interface must be referred by the outer interface or class. It can't be accessed directly. Points to remember for nested interfaces

- There are given some points that should be remembered by the java programmer.

- Nested interface must be public if it is declared inside the interface but it can have any access modifier if declared within the class.

- Nested interfaces are declared static implicitely.

- Syntax of nested interface which is declared within the interface

```
interface interface_name{
 ...
 interface nested_interface_name{
  ...
 }
}
```

- Syntax of nested interface which is declared within the class

```
class class_name{
 ...
 interface nested_interface_name{
  ...
 }
}
```

- Can we define a class inside the interface ?

- Yes, Ofcourse! If we define a class inside the interface, java compiler creates a static nested class. Let's see how can we define a class within the interface:

```
interface M{
  class A{}
}
```

- **Multithreading in Java**
- **Multithreading is a process of executing multiple threads simultaneously. Thread is basically a lightweight subprocess, a smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking. But we use multithreading than mulitprocessing because threads share a common memory area. They don't allocate separate memory area so save memory, and context-switching between the threads takes less time than processes. Multithreading is mostly used in games, animation etc.**
- **Multitasking**
- **Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved by two ways:**
- **Process-based Multitasking(Multiprocessing)**
- **Thread-based Multitasking(Multithreading)**

- Process-based Multitasking (Multiprocessing)
- Each process have its own address in memory i.e. each process allocates separate memory area.
- Process is heavyweight.
- Cost of communication between the process is high.
- Switching from one process to another require some time for saving and loading registers, memory maps, updating lists etc.

- Thread-based Multitasking (Multithreading)

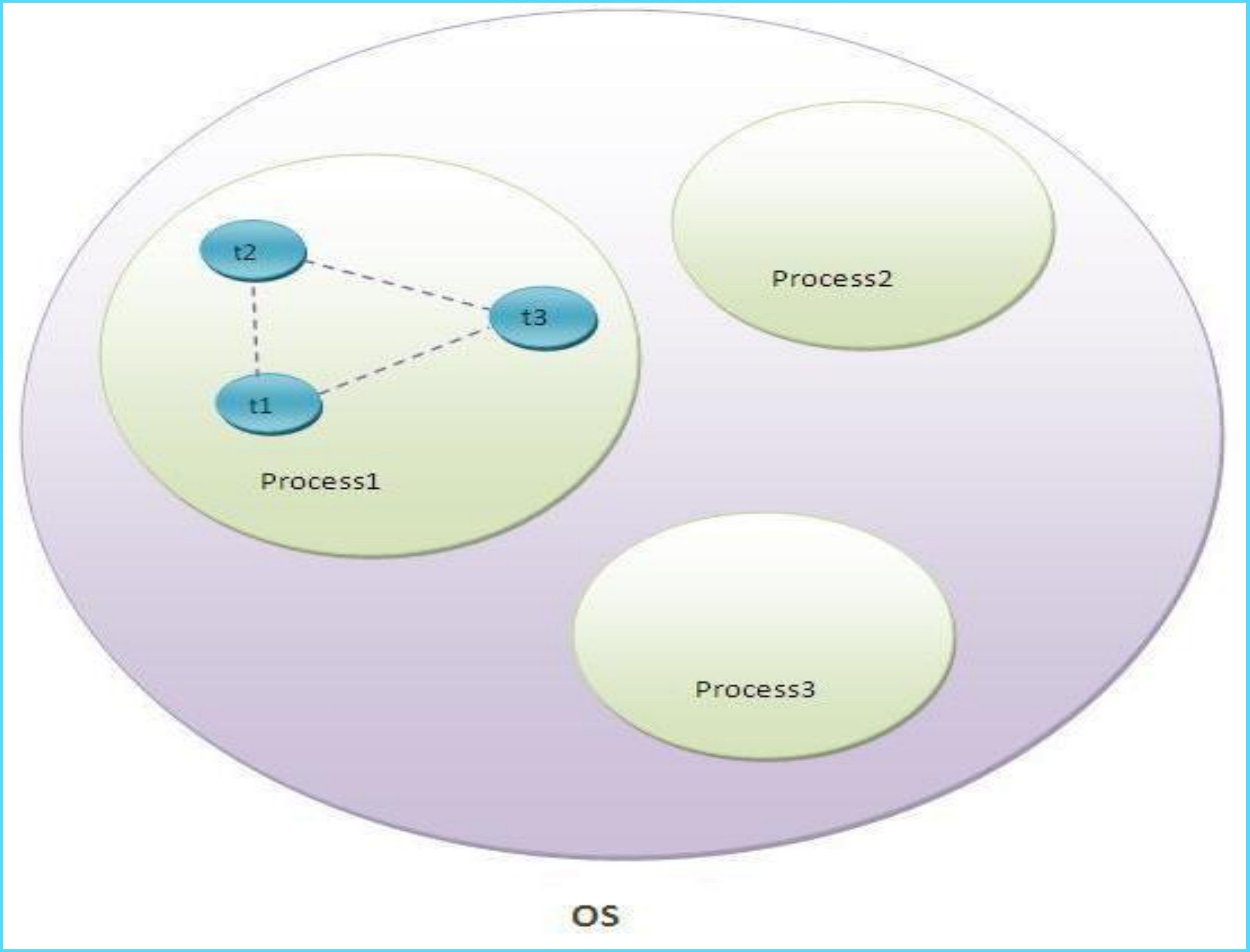
- Threads share the same address space.
- Thread is lightweight.
- Cost of communication between the thread is low.
- **Note:** At least one process is required for each thread.

- What is Thread?

- A thread is a lightweight sub process, a smallest unit of processing. It is a separate path of execution. It shares the memory area of process.

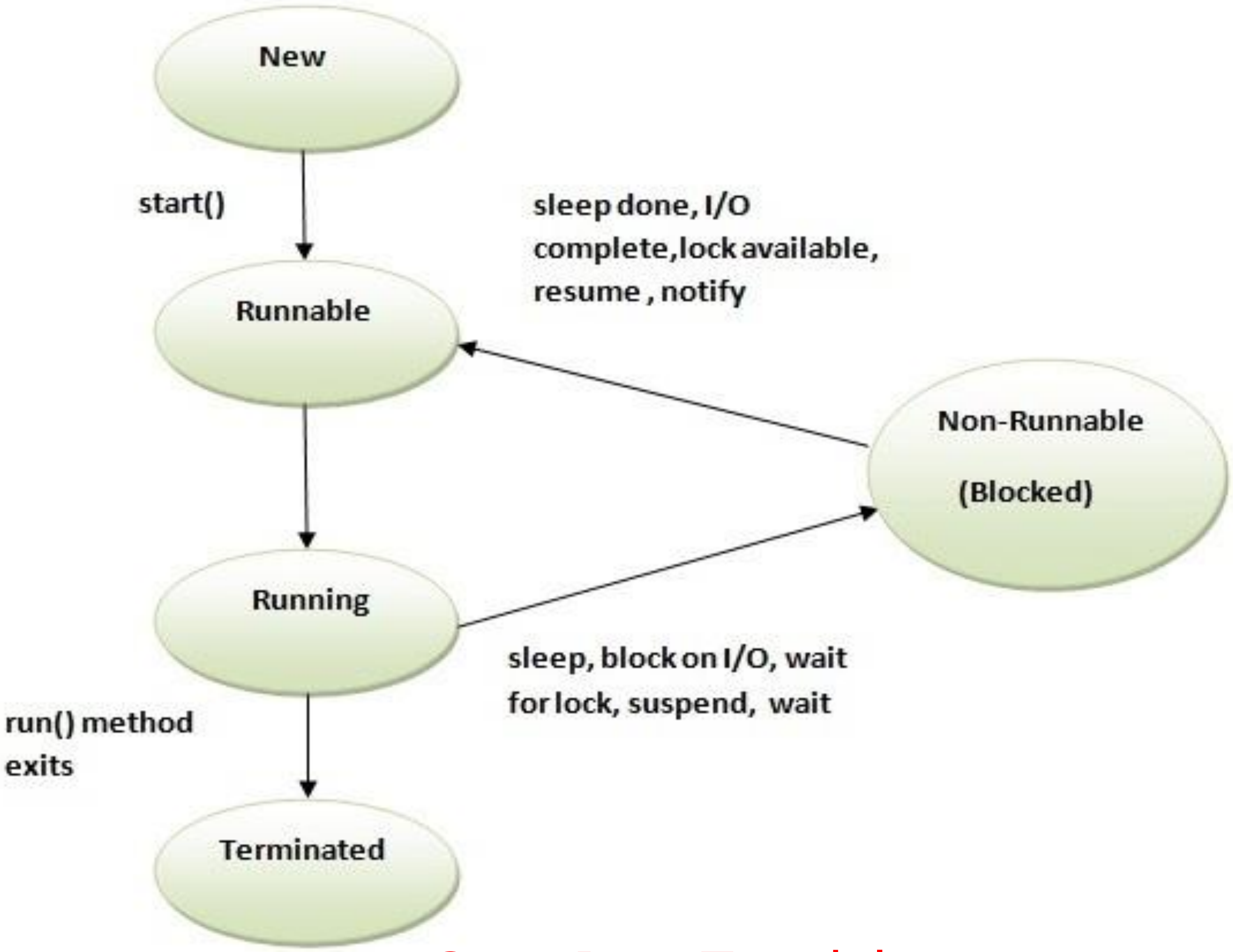
  As shown in the figure, thread is executed inside the process. There is context-switching between the threads. There can be multiple processes inside the OS and one process can have multiple threads.

  Note: At a time only one thread is executed.


t2
t3
t1
Process1
Process2
Process3
OS

- Life cycle of a Thread (Thread States)
- A thread can be in one of the five states in the thread. According to sun, there is only 4 states new, runnable, non-runnable and terminated. There is no running state. But for better understanding the threads, we are explaining it in the 5 states. The life cycle of the thread is controlled by JVM. The thread states are as follows:

- New
- Runnable
- Running
- Non-Runnable (Blocked)
- Terminated


New
start()
Runnable
sleep done, I/O complete, lock available, resume , notify
Non-Runnable (Blocked)
Running
sleep, block on I/O, wait for lock, suspend, wait
run() method exits
Terminated

- New

The thread is in new state if you create an instance of Thread class but before the invocation of start() method.

- Runnable

The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.

- Running

  The thread is in running state if the thread scheduler has selected it.

- Non-Runnable (Blocked)

  This is the state when the thread is still alive, but is currently not eligible to run.

- Terminated

  A thread is in terminated or dead state when its run() method exits.

- How to create thread:

- There are two ways to create a thread:

- By extending Thread class
- By implementing Runnable interface.

- Thread class:
- Thread class provide constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.

- Commonly used Constructors of Thread class:

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r,String name)

- Commonly used methods of Thread class:

- **public void run():** is used to perform action for a thread.
- **public void start():** starts the execution of the thread.JVM calls the run() method on the thread.
- **public void sleep(long miliseconds):** Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.

- **public void join():** waits for a thread to die.
- **public void join(long miliseconds):** waits for a thread to die for the specified miliseconds.
- **public int getPriority():** returns the priority of the thread.
- **public int setPriority(int priority):** changes the priority of the thread.
- **public String getName():** returns the name of the thread.

- **public void setName(String name):** changes the name of the thread.
- **public Thread currentThread():** returns the reference of currently executing thread.
- **public int getId():** returns the id of the thread.
- **public Thread.State getState():** returns the state of the thread.
- **public boolean isAlive():** tests if the thread is alive.

- **public void yield():** causes the currently executing thread object to temporarily pause and allow other threads to execute.
- **public void suspend():** is used to suspend the thread(depricated).
- **public void resume():** is used to resume the suspended thread(depricated).
- **public void stop():** is used to stop the thread(depricated).
- **public boolean isDaemon():** tests if the thread is a daemon thread.

- **public void setDaemon(boolean b):** marks the thread as daemon or user thread.
- **public void interrupt():** interrupts the thread.
- **public boolean isInterrupted():** tests if the thread has been interrupted.
- **public static boolean interrupted():** tests if the current thread has been interrupted.

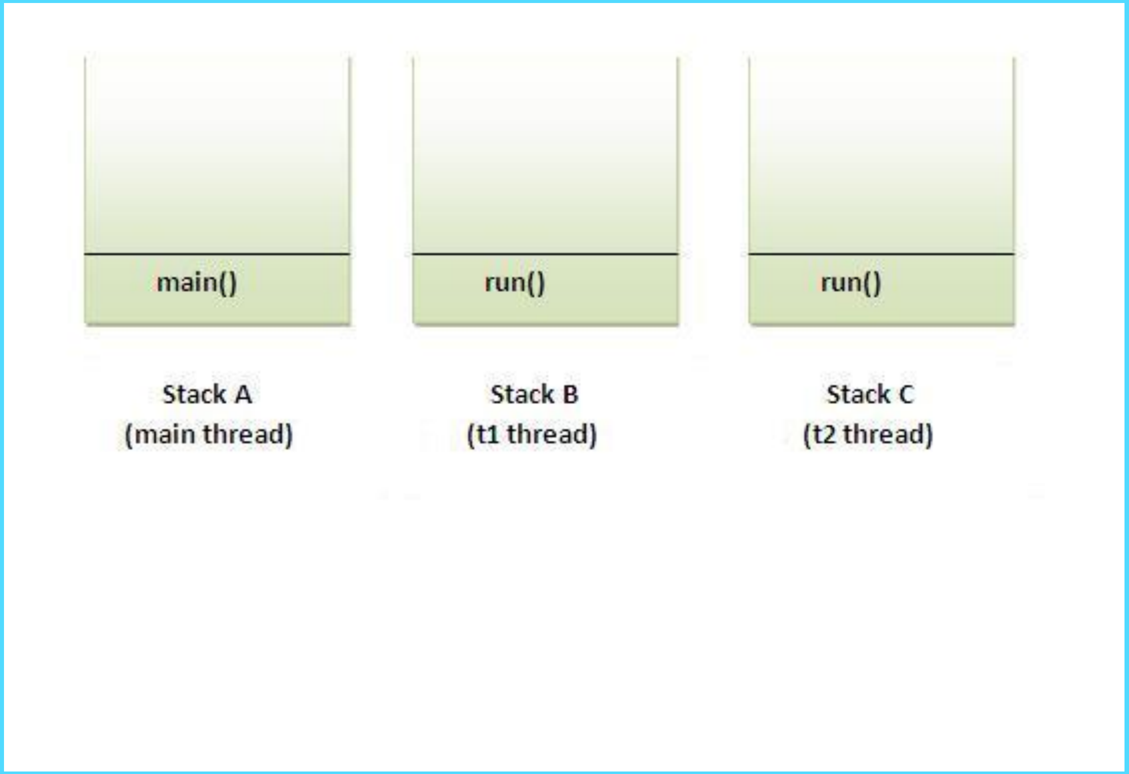
- Runnable interface:

- The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().
- **public void run():** is used to perform action for a thread.

- **Starting a thread:**

- **start() method** of Thread class is used to start a newly created thread. It performs following tasks:
- A new thread starts(with new callstack).
- The thread moves from New state to the Runnable state.
- When the thread gets a chance to execute, its target run() method will run.

- Multi Threading



- The Thread Schedular:

The thread scheduler is the part of the JVM that decides which thread should run.

There is no guarantee that which runnable thread will be chosen to run by the thread schedular.

Only one thread at a time can run in a single process.

The thread schedular mainly uses preemptive or time slicing scheduling to schedule the threads.

- Sleeping a thread (sleep() method):

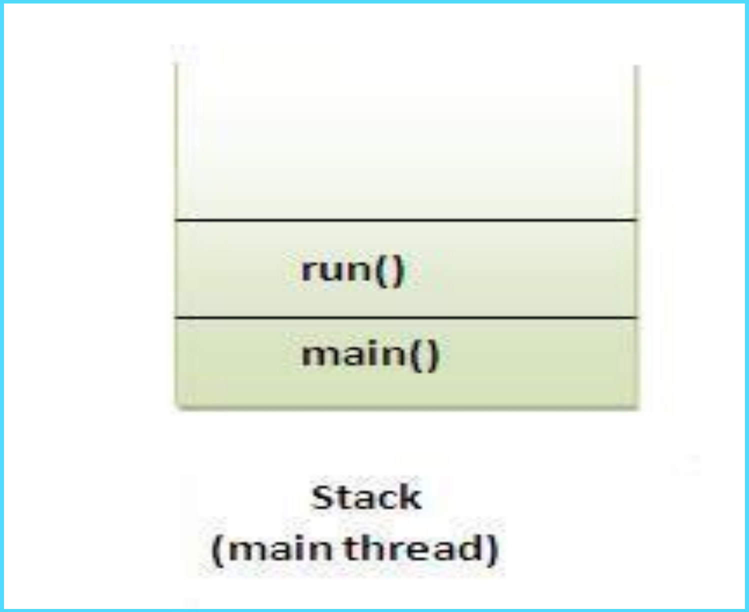
- The sleep() method of Thread class is used to sleep a thread for the specified time.

- Syntax of sleep() method:

Thread class provides two methods for sleeping a thread:

- public static void sleep(long miliseconds)throws InterruptedException
- public static void sleep(long miliseconds, int nanos)throws InterruptedException

- What if we call run() method directly instead start() method?
- Each thread starts in a separate call stack.
- Invoking the run() method from main thread, the run() method goes onto the current call stack rather than at the beginning of a new call stack.



- The join() method:

- The join() method waits for a thread to die. In other words, it causes the currently running threads to stop executing until the thread it joins with completes its task.

- Syntax:

- public void join()throws InterruptedException
- public void join(long miliseconds)throws InterruptedException

- The currentThread() method:

  The currentThread() method returns a reference to the currently executing thread object.
- Naming a thread:

  The Thread class provides methods to change and get the name of a thread.
- **public String getName():** is used to return the name of a thread.
- **public void setName(String name):** is used to change the name of a thread.

- The currentThread() method:
- The currentThread() method returns a reference to the currently executing thread object.
- Syntax of currentThread() method:
- **public static Thread currentThread():**

  returns the reference of currently running thread.

- ## Priority of a Thread (Thread Priority):

Each thread have a priority. Priorities are represented by a number between 1 and 10. In most cases, thread schedular schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specifification that which sheduling it chooses.3 constants defiend in Thread class:

- public static int MIN_PRIORITY
- public static int NORM_PRIORITY
- public static int MAX_PRIORITY

- Default priority of a thread is 5 (NORM_PRIORITY).
- The value of MIN_PRIORITY is 1
- and the value of MAX_PRIORITY is 10.

- Daemon Thread
- There are two types of threads user thread and daemon thread. The daemon thread is a service provider thread. It provides services to the user thread. Its life depends on the user threads i.e. when all the user threads dies, JVM termintates this thread automatically.
- Points to remember for Daemon Thread:
- It provides services to user threads for background supporting tasks. It has no role in life than to serve user threads.
- Its life depends on user threads.
- It is a low priority thread.

- Why JVM termintates the daemon thread if there is no user thread remaining?

- The sole purpose of the daemon thread is that it provides services to user thread for background supporting task. If there is no user thread, why should JVM keep running this thread. That is why JVM terminates the daemon thread if there is no user thread.

- Methods for Daemon thread:

- The java.lang.Thread class provides two methods related to daemon thread
- **public void setDaemon(boolean status):** is used to mark the current thread as daemon thread or user thread.

- **public boolean isDaemon():** is used to check that current is daemon.

N***ote: If you want to make a user thread as Daemon, it must not be started otherwise it will throw IllegalThreadStateException.***

- Thread Pooling in Java
- Thread pool represents a group of worker threads that are waiting for the job. Here, threads are executed whenever they get the job.
- In case of thread pool, a group of fixed size threads are created. A thread from the thread pool is pulled out and assigned a job by the service provider. After completion of the job, thread is contained in the thread pool again.
- **Advantages of Thread Pool**
- **Better performance** It saves time because there is no need to create new thread.
- **Where is it used?**
- It is used in Servlet and JSP where container creates a thread pool to process the request.

- Shutdown Hook
- The shutdown hook can be used to perform cleanup resource or save the state when JVM shuts down normally or abruptly. Performing clean resource means closing log file, sending some alerts or something else. So if you want to execute some code before JVM shuts down, use shutdown hook.
- When does the JVM shut down?
- The JVM shuts down when user presses ctrl+c on the command prompt
- System.exit(int) method is invoked
- user logoff
- user shutdown etc.

- **The addShutdownHook(Runnable r) method**
- The addShutdownHook() method of Runtime class is used to register the thread with the Virtual Machine.

- Syntax:
- **public void** addShutdownHook(Runnable r){}

The object of Runtime class can be obtained by calling the static factory method getRuntime().

For example:

Runtime r = Runtime.getRuntime();

- **Factory method**

The method that returns the instance of a class is known as factory method.

- ***Note: The shutdown sequence can be stopped by invoking the halt(int) method of Runtime class***

- Garbage Collection:

- In java, garbage means unreferenced objects. Garbage Collection is process of reclaiming the runtime unused memory automatically.
- Advantage of Garbage Collection:
- It makes java memory efficient because garbage collector removes the unreferenced objects from heap memory.
- It is automatically done by the garbage collector so we don't need to make extra efforts.

- How can an object be unreferenced?

There are many ways:

- By nulling the reference
- By assigning a reference to another
- By annonymous object etc.

- By nulling a reference:

```
Employee e=new Employee();
e=null;
```

- By assigning a reference to another:

```
Employee e1=new Employee();
 Employee e2=new Employee();



e1=e2;//now the first object reffered by e1 is avail
able for garbage collection
```

- By annonymous object:

```
new Employee();
```

- finalize() method:
- The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in System class as:

- **protected void** finalize(){}

- **Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).**

- gc() method:

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

- **public static void** gc(){}

Note: Garbage collection is performed by a daemon thread called Garbage Collector(GC). This thread calls the finalize() method before object is garbage collected.

- Synchronization

  Synchronization is the capability of control the access of multiple threads to any shared resource. Synchronization is better in case we want only one thread can access the shared resource at a time.

- Why use Synchronization?

  The synchronization is mainly used to

- To prevent thread interference.
- To prevent consistency problem.

- Types of Synchronization

There are two types of synchronization

- Process Synchronization
- Thread Synchronization

Here, we will discuss only thread synchronization.

- Thread Synchronization

There are two types of thread synchronization

- mutual exclusive
- and inter-thread communication.

- Mutual Exclusive
  - Synchronized method.
  - Synchronized block.
- Cooperation (Inter-thread communication)

- Mutual Exclusive

- Mutual Exclusive helps keep threads from interfering with one another while sharing data. This can be done by three ways in java:

- by synchronized method
- by synchronized block

Understanding the concept of Lock

Synchronization is built around an internal entity known as the lock or monitor.Every object has an lock associated with it. By convention, a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.From Java 5 the package java.util.concurrent.locks contains several lock implementations.

- Synchronized block
- Synchronized block can be used to perform synchronization on any specific resource of the method.

  Suppose you have 50 lines of code in your method, but you want to synchronize only 5 lines, you can use synchronized block.

  If you put all the codes of the method in the synchronized block, it will work same as the synchronized method.

- **Syntax to use synchronized block**

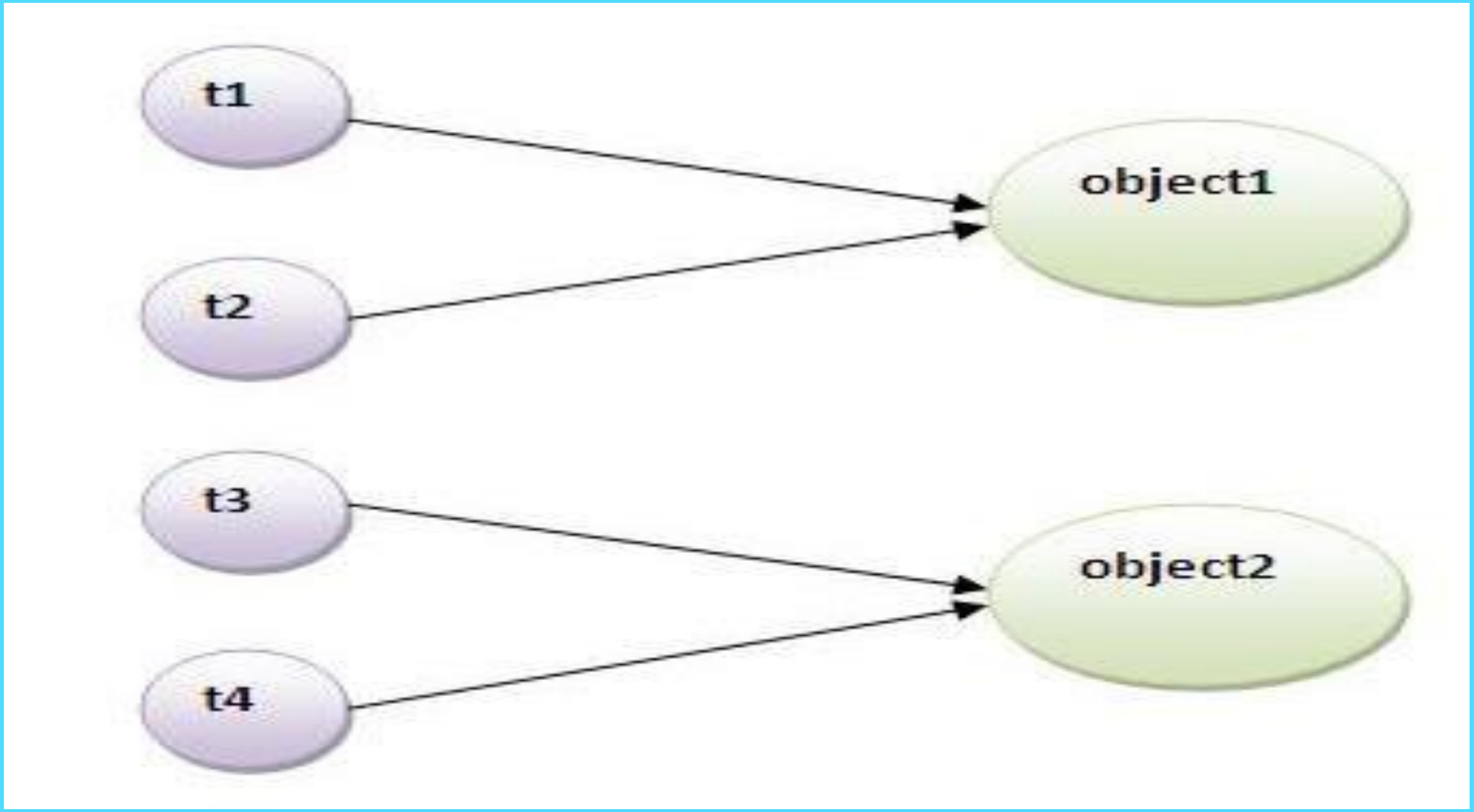
  synchronized (object reference expression) {

  //code block

  }

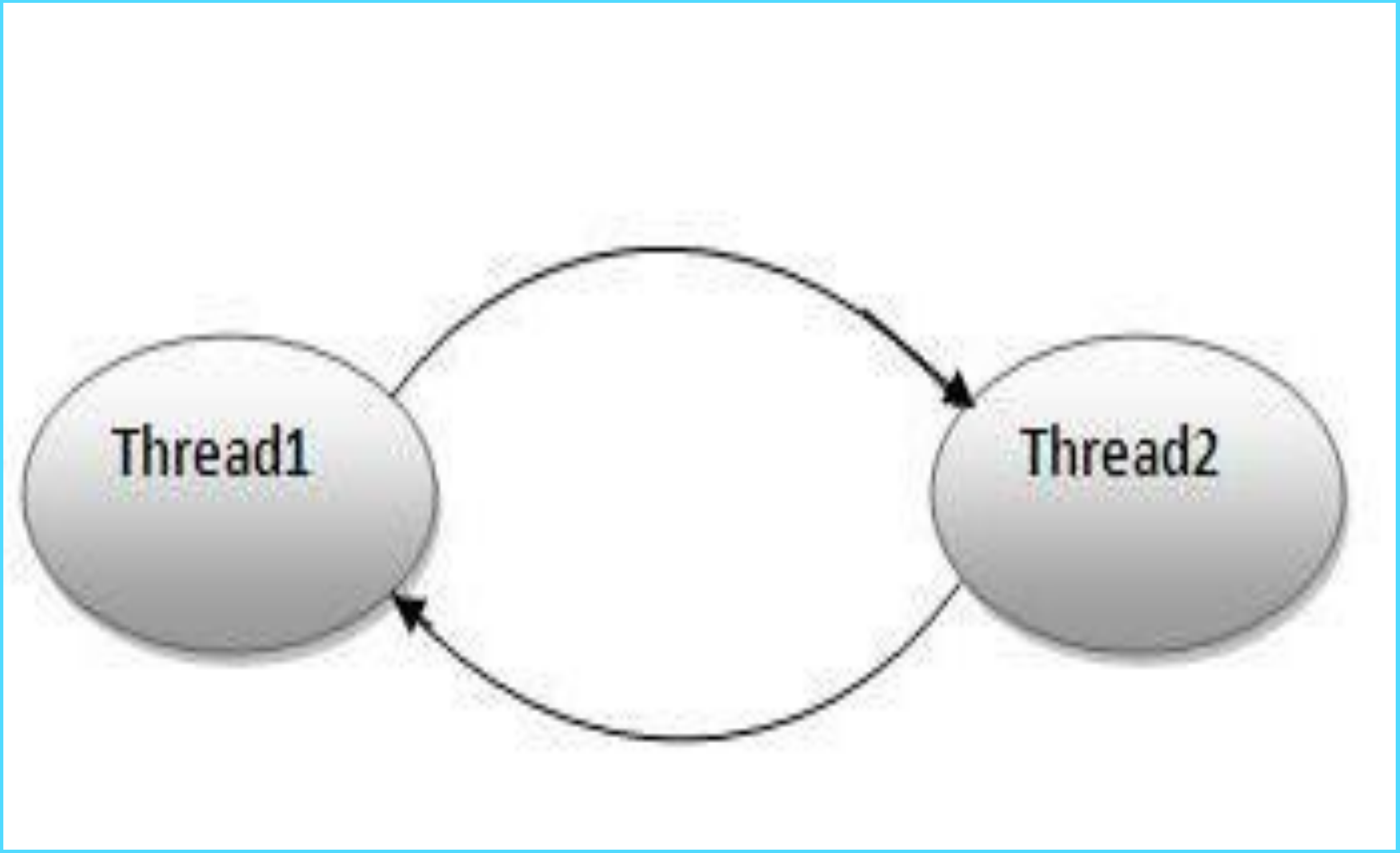
- Static synchronization

If you make any static method as synchronized, the lock will be on the class not on object.



Deadlock:

Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread. Since, both threads are waiting for each other to release the lock, the condition is called deadlock.


Thread1
Thread2

- Inter-thread communication (Cooperation):

Cooperation(Inter-thread communication) is all about making synchronized threads communicate with each other. Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed. It is implemented by following methods of Object class:

- wait()
- notify()
- notifyAll()

- wait() method:

  Causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed. The current thread must own this object's monitor.

- Syntax:

public final void wait()throws InterruptedException

- public final void wait(long timeout)throws InterruptedException

- notify() method:

  Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened.

  Syntax:

- public final void notify()
- notifyAll() method:

  Wakes up all threads that are waiting on this object's monitor.

- public final void notifyAll()

- Interrupting a Thread:

If any thread is in sleeping or waiting state (i.e. sleep() or wait() is invoked), calling the interrupt() method on the thread, breaks out the sleeping or waiting state throwing InterruptedException. If the thread is not in the sleeping or waiting state, calling the interrupt() method performs normal behaviour and doesn't interrupt the thread but sets the interrupt flag to true. Let's first see the methods provided by the Thread class for thread interruption.

- The 3 methods provided by the Thread class for interrupting a thread

- **public void interrupt()**
- **public static boolean interrupted()**
- **public boolean isInterrupted()**

Input and Output in Java

Input and Output (I/O) is used to process the input and produce the output based on the input. Java uses the concept of stream to make I/O operations fast. java.io package contains all the classes required for input and output operations.

Stream

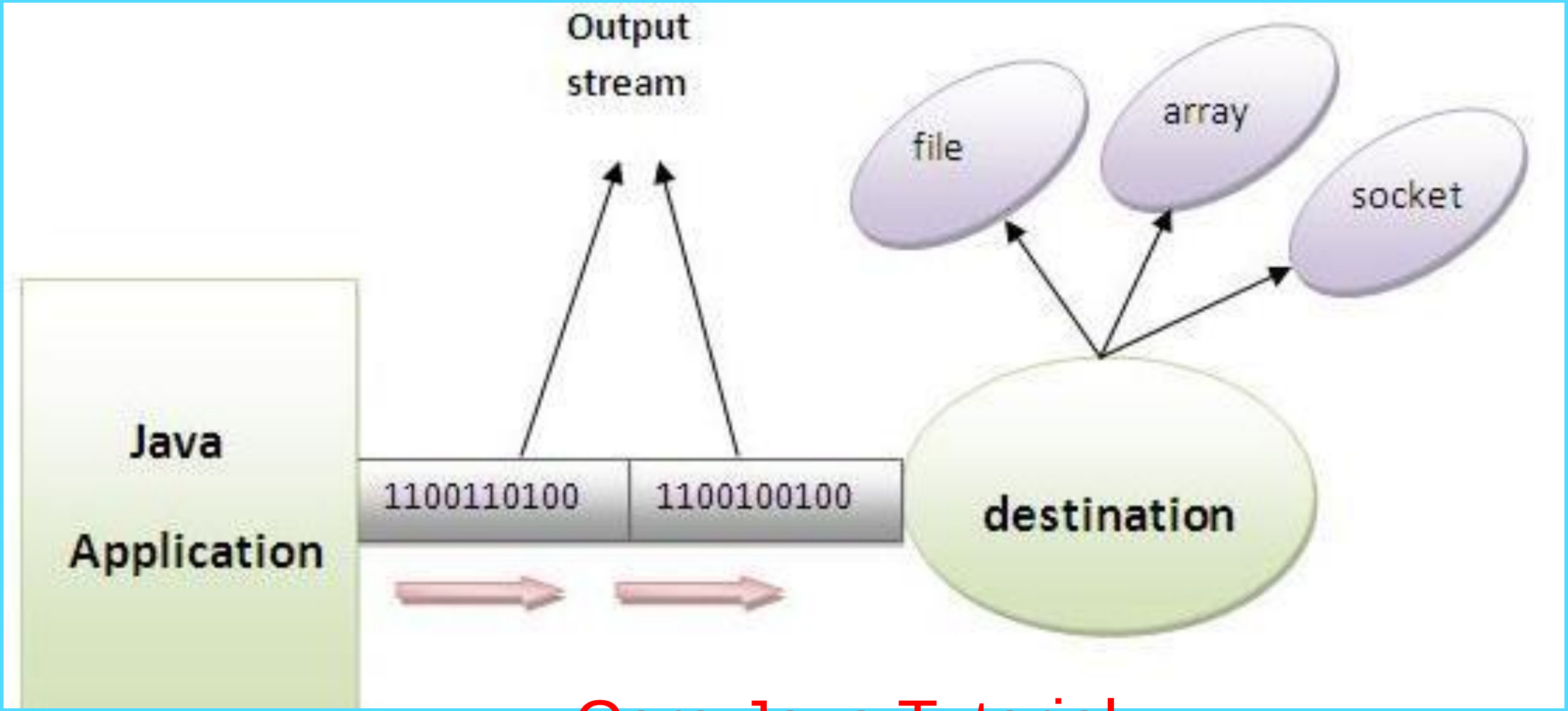
A stream is a sequence of data.In Java a stream is composed of bytes. It's called a stream because it's like a stream of water that continues to flow.

Three streams are created for us automatically

- **System.out:** standard output stream
- **System.in:** standard input stream
- **System.err:** standard error

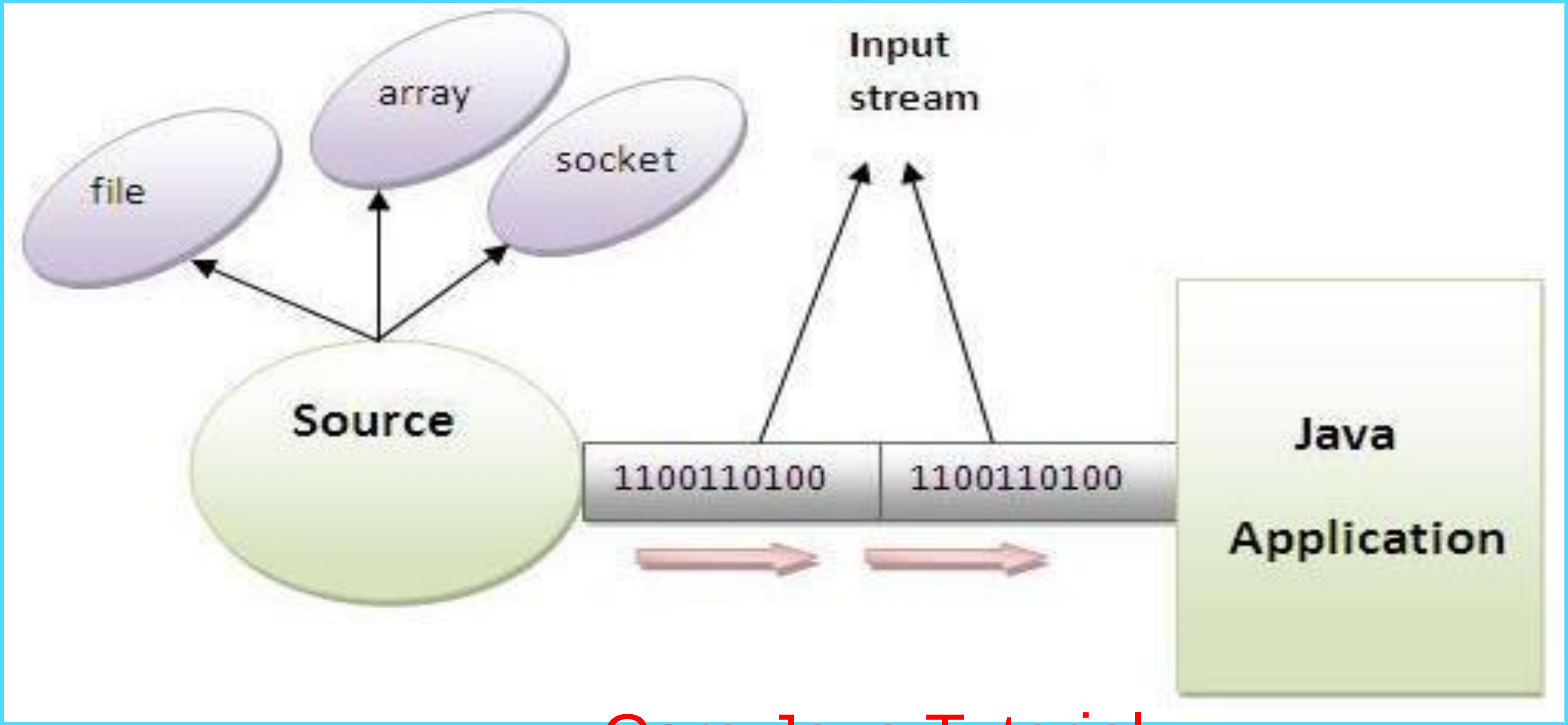
- OutputStream

Java application uses an output stream to write data to a destination, it may be a file, an array,peripheral device or socket.



- InputStream

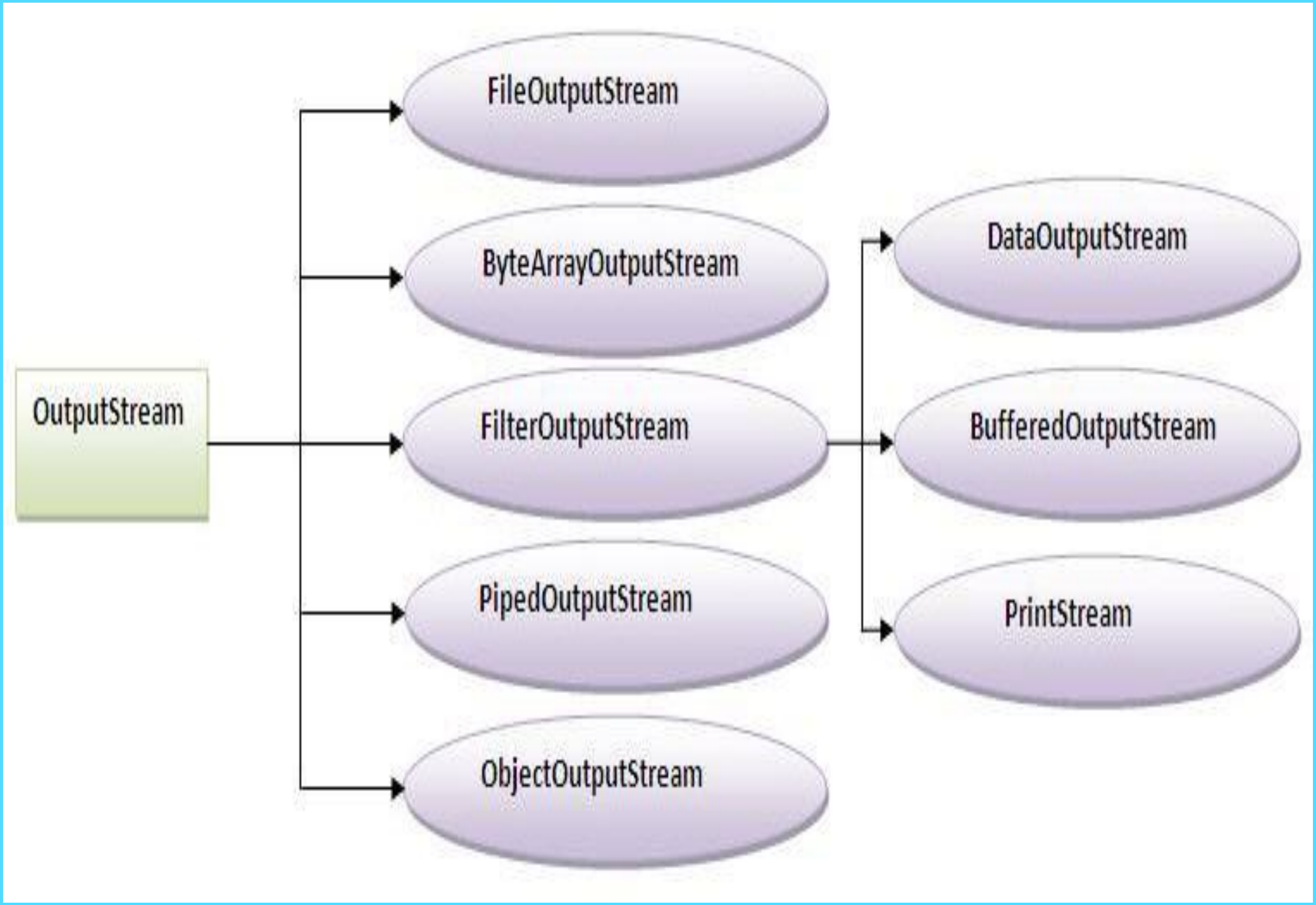
Java application uses an input stream to read data from a source, it may be a file,an array,peripheral device or socket.



- OutputStream class

OutputStream class ia an abstract class. It is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Method	Description
1) public void write(int)throws IOException:	is used to write a byte to the current output stream.
2) public void write(byte[])throws IOException:	is used to write an array of byte to the current output stream.
3) public void flush()throws IOException:	flushes the current output stream.
4) public void close()throws IOException:	is used to close the current output stream.

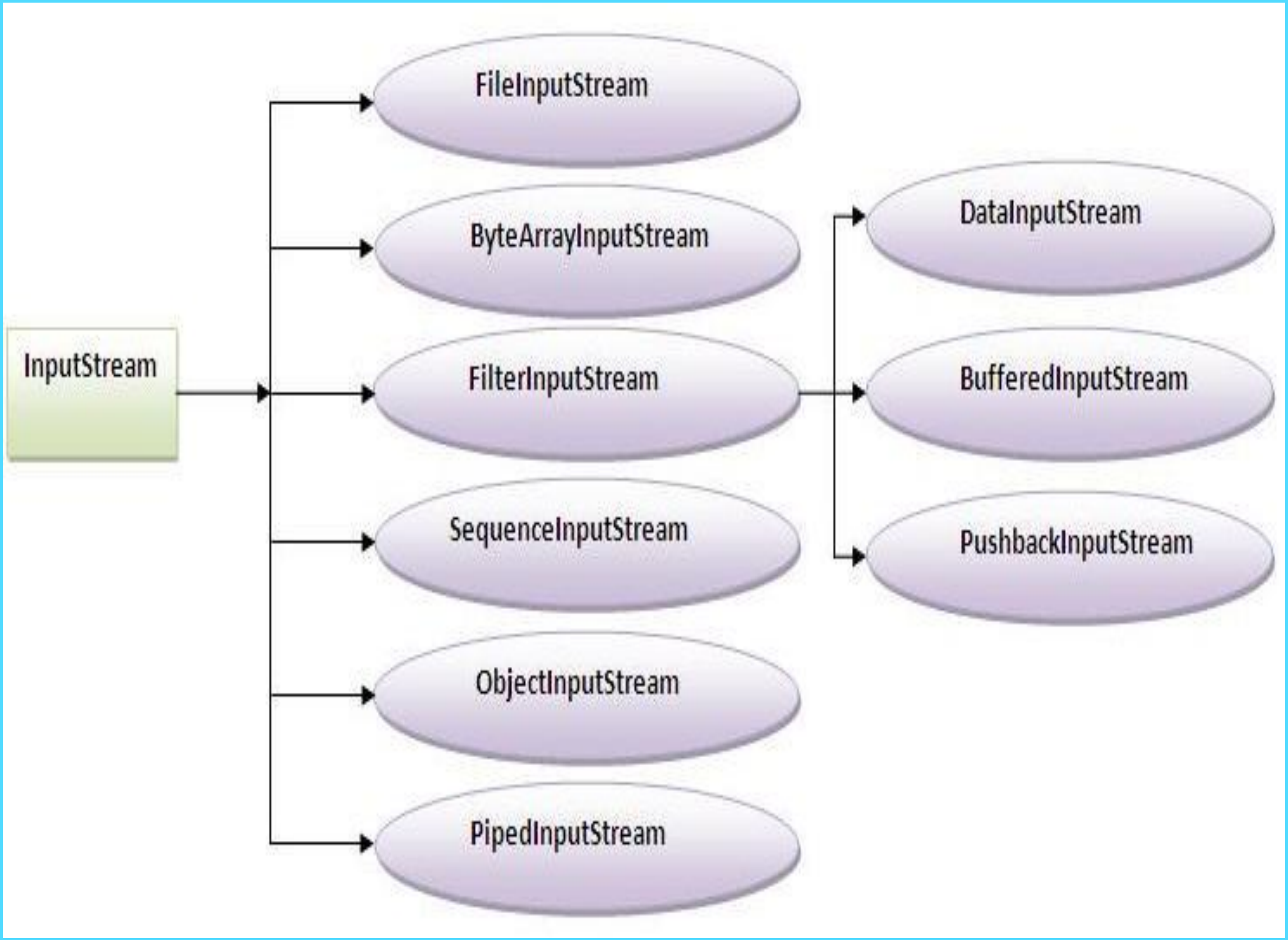

OutputStream
FileOutputStream
ByteArrayOutputStream
FilterOutputStream
PipedOutputStream
ObjectOutputStream
DataOutputStream
BufferedOutputStream
PrintStream

- **InputStream class**

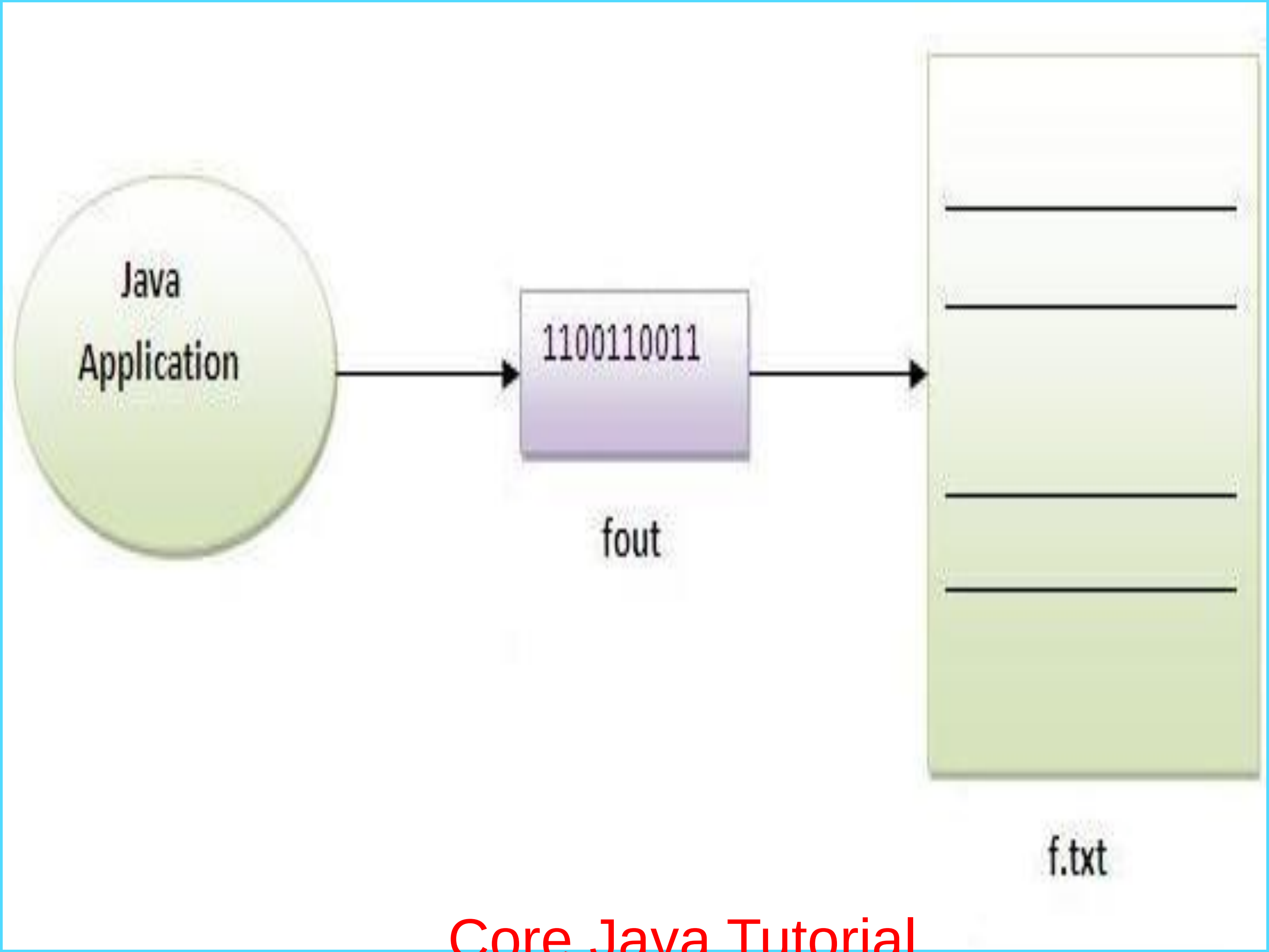
InputStream class ia an abstract class.It is the superclass of all classes representing an input stream of bytes.

- **Commonly used methods of InputStream class**

Method	Description
1) public abstract int read()throws IOException:	reads the next byte of data from the input stream.It returns -1 at the end of file.
2) public int available()throws IOException:	returns an estimate of the number of bytes that can be read from the current input stream.
3) public void close()throws IOException:	is used to close the current input stream.


InputStream
FileInputStream
ByteArrayInputStream
FilterInputStream
SequenceInputStream
ObjectInputStream
PipedInputStream
DataInputStream
BufferedInputStream
PushbackInputStream

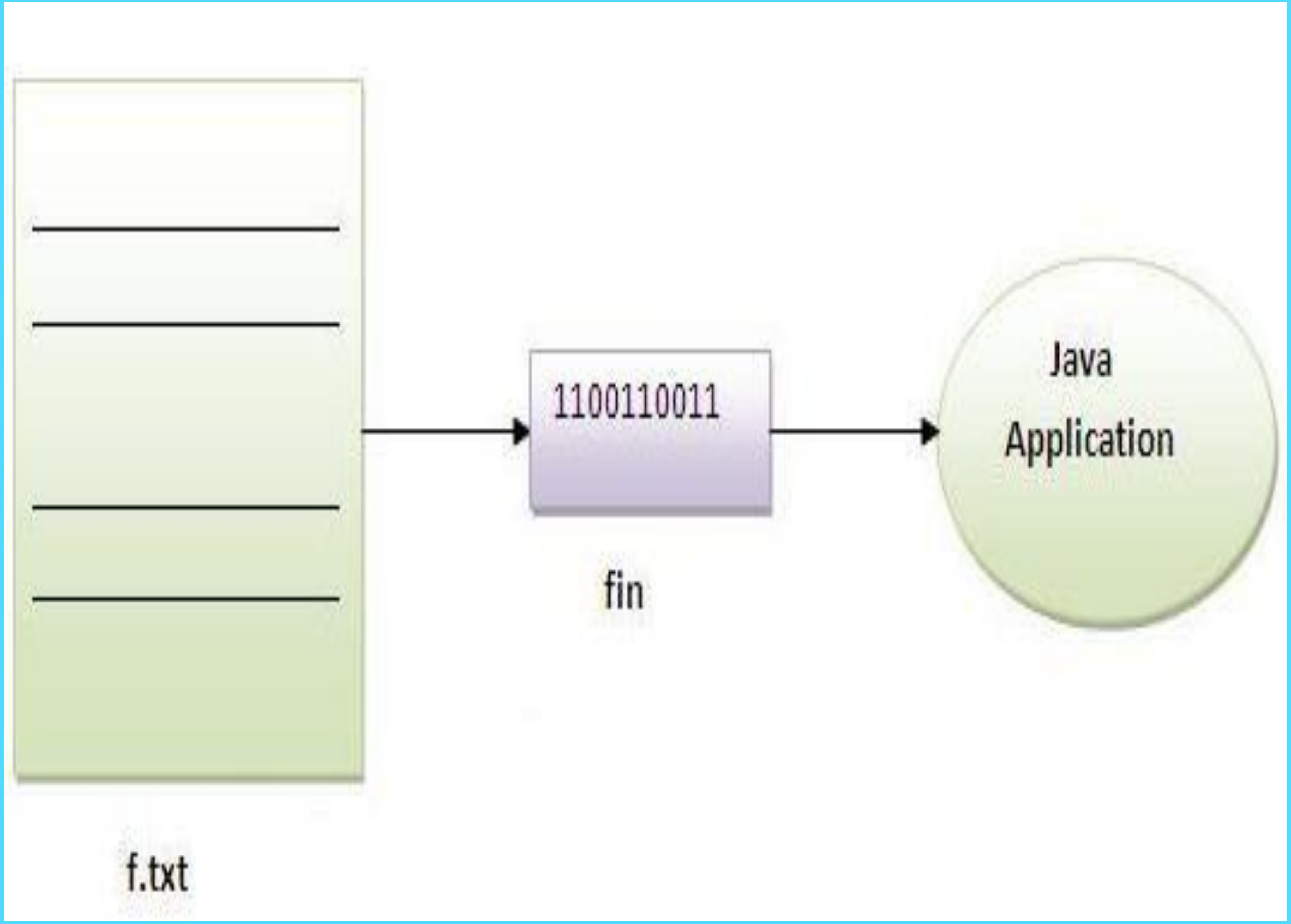
- FileInputStream and FileOutputStream (File Handling):
- FileInputStream and FileOutputStream classes are used to read and write data in file. In another words, they are used for file handling in java.FileOutputStream class:
- A FileOutputStream is an output stream for writing data to a file. If you have to write primitive values then use FileOutputStream. Instead, for character-oriented data, prefer FileWriter. But you can write byte-oriented as well as character-oriented data.



Core Java Tutorial

- FileInputStream class:

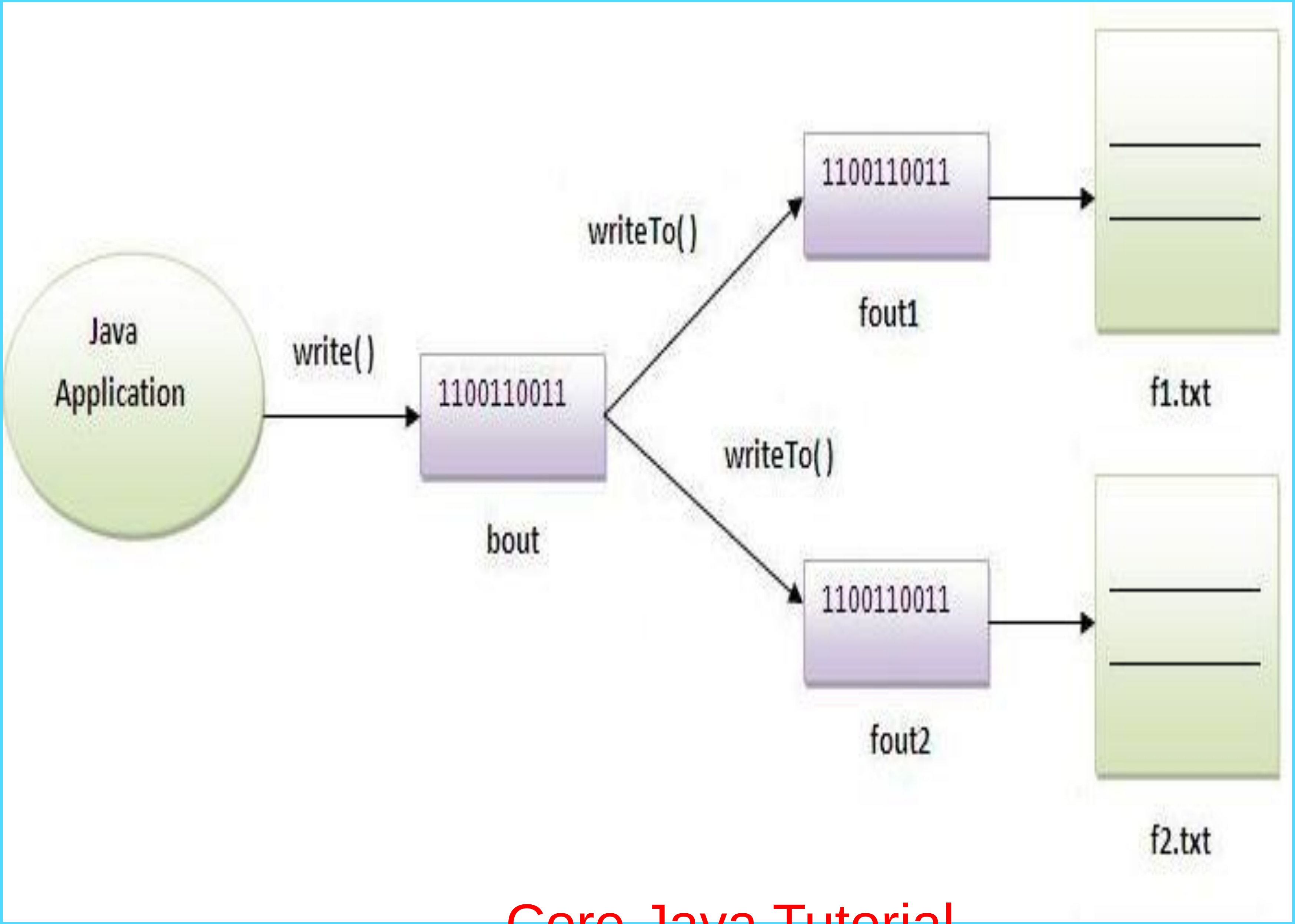
A FileInputStream obtains input bytes from a file.It is used for reading streams of raw bytes such as image data. For reading streams of characters, consider using FileReader.It should be used to read byte-oriented data.For example, to read image etc.


1100110011
fin
Java
Application
f.txt

- ByteArrayOutputStream class:
- In this stream, the data is written into a byte array. The buffer automatically grows as data is written to it. Closing a ByteArrayOutputStream has no effect. Commonly used Constructors of ByteArrayOutputStream class:
- **ByteArrayOutputStream():**creates a new byte array output stream with the initial capacity of 32 bytes, though its size increases if necessary.
- **ByteArrayOutputStream(int size):**creates a new byte array output stream, with a buffer capacity of the specified size, in bytes.

- Commonly used Methods of ByteArrayOutputStream class:

- **public synchronized void writeTo(OutputStream out) throws IOException:** writes the complete contents of this byte array output stream to the specified output stream.


Java
Application
write( )
1100110011
bout
writeTo( )
1100110011
fout1
f1.txt
writeTo( )
1100110011
fout2
f2.txt

- SequenceInputStream class:
- SequenceInputStream class is used to read data from multipule streams. Constructors of SequenceInputStream class:
- **SequenceInputStream(InputStream s1, InputStream s2):**

  creates a new input stream by reading the data of two input stream in order, first s1 and then s2.
- **SequenceInputStream(Enumeration e):**creates a new input stream by reading the data of an enumeration whose type is InputStream.

- BufferedOutputStream class:

- BufferedOutputStream used an internal buffer. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast.
- FileWriter class:
- FileWriter class is used to write character-oriented data to the file. Sun Microsystem has suggested not to use the FileInputStream and FileOutputStream classes if you have to read and write the textual information.

- FileReader class:
- FileReader class is used to read data from the file.

- CharArrayWriter class:
- The CharArrayWriter class can be used to write data to multiple files. This class implements the Appendable interface. Its buffer automatically grows when data is written in this stream. Calling the close() method on this object has no effect.

- Reading data from keyboard:

There are many ways to read data from the keyboard. For example:

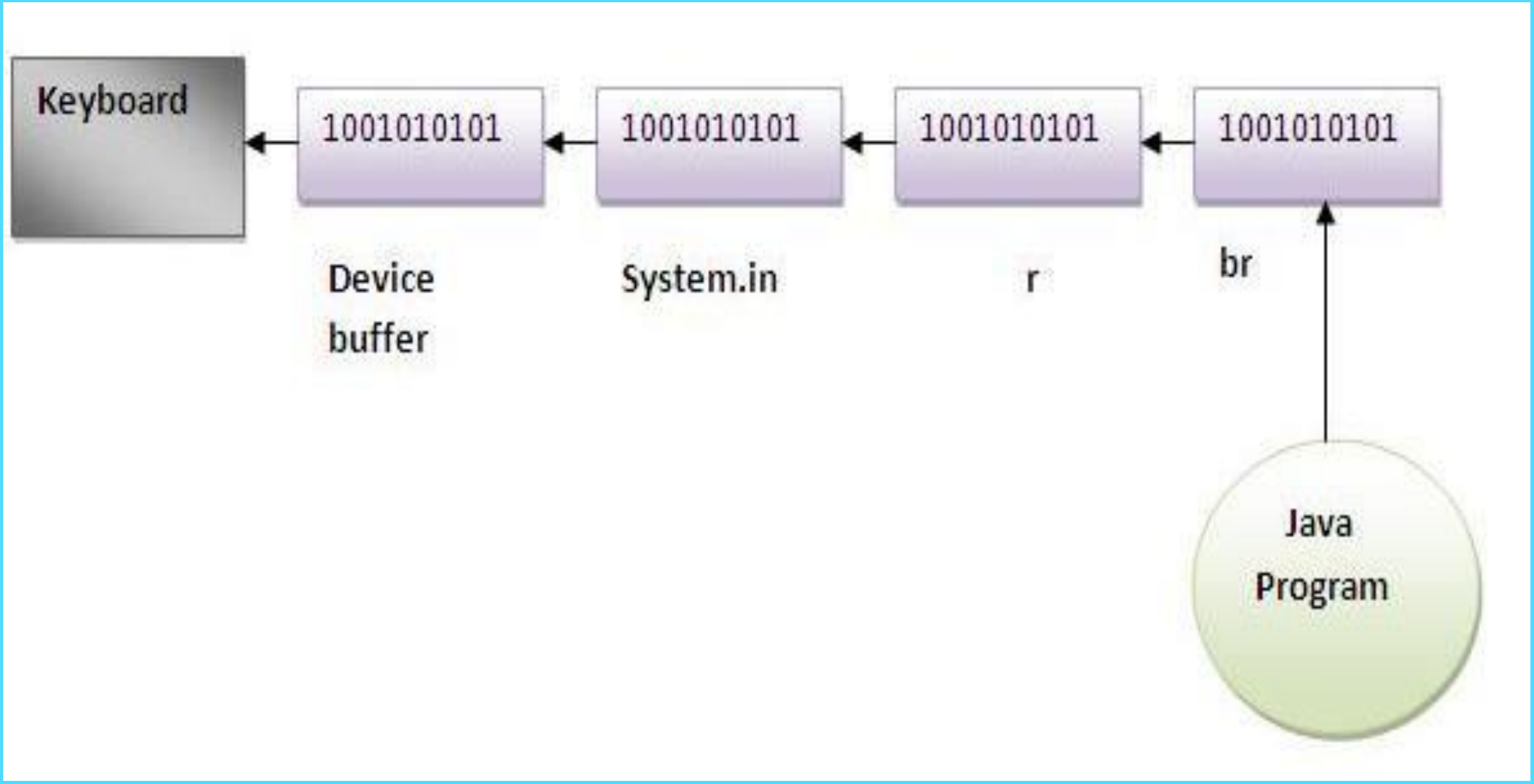
- InputStreamReader

- Scanner
- DataInputStream etc.

- InputStreamReader class:

- InputStreamReader class can be used to read data from keyboard.
- It performs two tasks:
- connects to input stream of keyboard
- converts the byte-oriented stream into character-oriented stream

- BufferedReader class:
- BufferedReader class can be used to read data line by line by readLine() method.



- java.util.Scanner class:

There are various ways to read input from the keyboad, the java.util.Scanner class is one of them. The Scanner class breaks the input into tokens using a delimiter which is whitespace bydefault. It provides many methods to read and parse various primitive values.Commonly used methods of Scanner class:

- There is a list of commonly used Scanner class methods:
- **public String next():** it returns the next token from the scanner.
- **public String nextLine():** it moves the scanner position to the next line and returns the value as a string.
- **public byte nextByte():** it scans the next token as a byte.
- **public short nextShort():** it scans the next token as a short value.
- **public int nextInt():** it scans the next token as an int value.

- **public long nextLong():** it scans the next token as a long value.
- **public float nextFloat():** it scans the next token as a float value.
- **public double nextDouble():** it scans the next token as a double value.

- java.io.PrintStream class:

The PrintStream class provides methods to write data to another stream. The PrintStream class automatically flushes the data so there is no need to call flush() method. Moreover, its methods don't throw IOException.Commonly used methods of PrintStream class:

There are many methods in PrintStream class. Let's see commonly used methods of PrintStream class:

- **public void print(boolean b):** it prints the specified boolean value.
- **public void print(char c):** it prints the specified char value.
- **public void print(char[] c):** it prints the specified character array values.
- **public void print(int i):** it prints the specified int value.
- **public void print(long l):** it prints the specified long value.
- **public void print(float f):** it prints the specified float value.

- **public void print(double d):** it prints the specified double value.
- **public void print(String s):** it prints the specified string value.
- **public void print(Object obj):** it prints the specified object value.
- **public void println(boolean b):** it prints the specified boolean value and terminates the line.
- **public void println(char c):** it prints the specified char value and terminates the line.

- **public void println(char[] c):** it prints the specified character array values and terminates the line.
- **public void println(int i):** it prints the specified int value and terminates the line.
- **public void println(long l):** it prints the specified long value and terminates the line.
- **public void println(float f):** it prints the specified float value and terminates the line.

- **public void println(double d):** it prints the specified double value and terminates the line.
- **public void println(String s):** it prints the specified string value and terminates the line.
- **public void println(Object obj):** it prints the specified object value and terminates the line.
- **public void println():** it terminates the line only.

- **public void printf(Object format, Object... args):** it writes the formatted string to the current stream.
- **public void printf(Locale l, Object format, Object... args):** it writes the formatted string to the current stream.
- **public void format(Object format, Object... args):** it writes the formatted string to the current stream using specified format.

- **public void format(Locale l, Object format, Object... args):** it writes the formatted string to the current stream using specified format.

- Compressing and Uncompressing File

The DeflaterOutputStream and InflaterInputStream classes provide mechanism to compress and uncompress the data in the **deflate compression format**.

- DeflaterOutputStream class:

- The DeflaterOutputStream class is used to compress the data in the deflate compression format. It provides facility to the other compression filters, such as GZIPOutputStream.

- InflaterInputStream class:
- The InflaterInputStream class is used to uncompress the file in the deflate compression format. It provides facility to the other uncompression filters, such as GZIPInputStream class.

- Serialization

Serialization is a machanism of writing the state of an object into a byte stream. It is mainly used in Hibernate, JPA, EJB etc. The reverse operation of the serialization is called deserialization. The String class and all the wrapper classes implements Serializable interface bydefault.

Advantages of Serialization

- It is mainly used to travel object's state on the network.

- About Serializable interface

  Serializable is a marker interface(have no body). It is just used to "mark" Java classes which support a certain capability. It must be implemented by the class whose object you want to persist

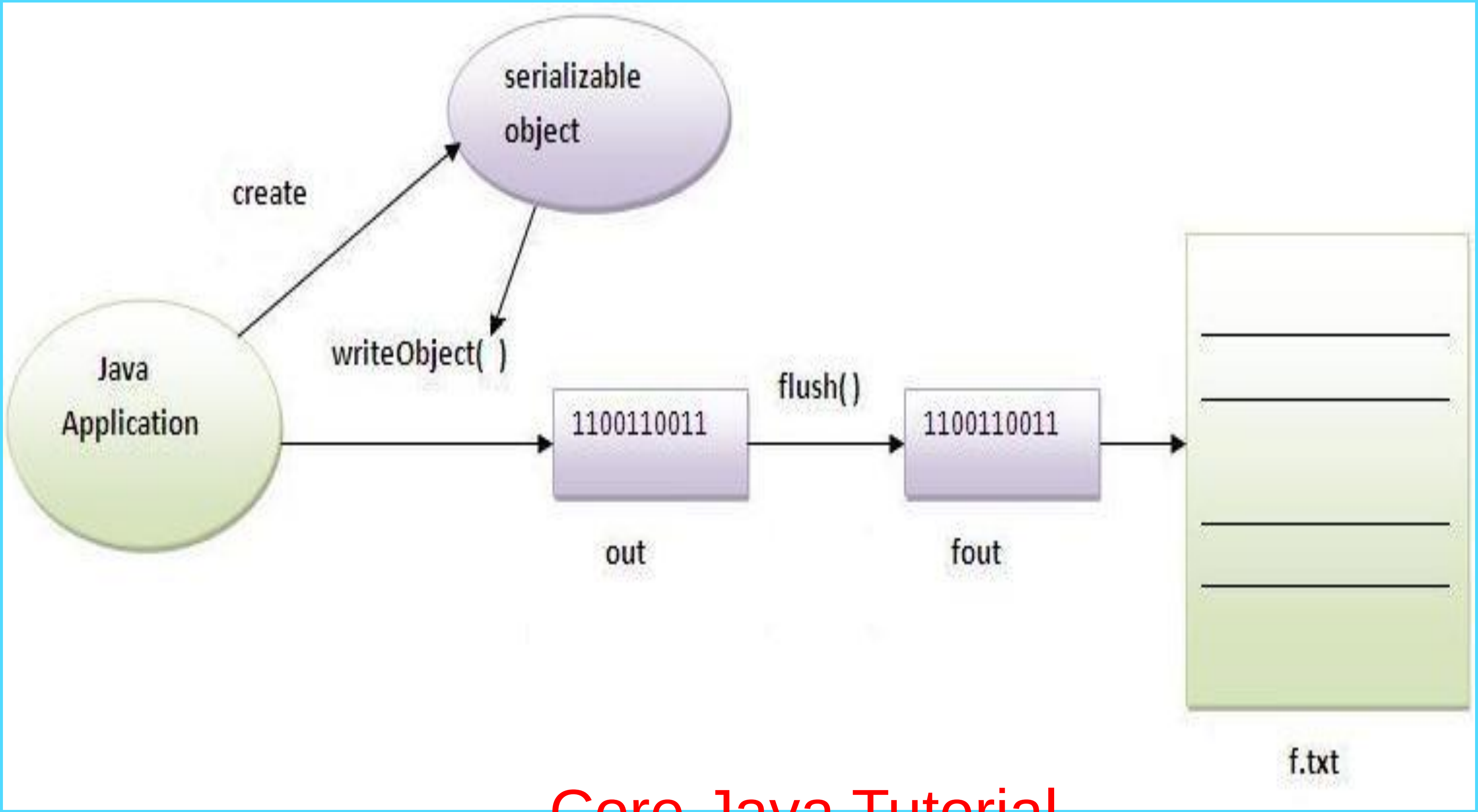
- ObjectOutputStream class:

  An ObjectOutputStream is used to write primitive data types and Java objects to an OutputStream.Only objects that support the java.io.Serializable interface can be written to streams.

- **Commonly used Constructors:**

- **public ObjectOutputStream(OutputStream out) throws IOException {}**creates an ObjectOutputStream that writes to the specified OutputStream.
- **Commonly used Methods:**
- **public final void writeObject(Object obj) throws IOException {}**write the specified object to the ObjectOutputStream.

- **public void flush() throws IOException {}**flushes the current output stream.



- Deserilization:
- Deserialization is the process of reconstructing the object from the serialized state. It is the reverse operation of serialization.

- ObjectInputStream class:
- An ObjectInputStream deserializes objects and primitive data written using an ObjectOutputStream.

- **Commonly used Constructors:**
- **public ObjectInputStream(InputStream in) throws IOException {}**creates an ObjectInputStream that reads from the specified InputStream.**Commonly used Methods:**
- **public final Object readObject() throws IOException, ClassNotFoundException{}**reads an object from the input stream.

- Serialization with Inheritance

  If a class implements Serilizable then all its subclasses will also be serilizable.

- Externalizable interface:

  The Externalizable interface provides the facility of writing the state of an object into a byte stream in compress format. It is not a marker interface.

  The Externalizable interface provides two methods:

- **public void writeExternal(ObjectOutput out) throws IOException**
- **public void readExternal(ObjectInput in) throws IOException**

- Serialization with Static datamember

- ***Note:*** ***If there is any static data member in a class, it will not be serialized because static is related to class not to instance.***

- ***Rule:*** ***In case of array or collection, all the objects of array or collection must be serializable,if any object is not serialiizable then serialization will be failed.***

- Wrapper classes
- **Introduction**
- **Wrapper classes** are used to convert any data type into an object.
- The primitive data types are not objects; they do not belong to any class; they are defined in the language itself. Sometimes, it is required to convert data types into objects in Java language. A data type is to be converted into an object and then added to a Stack or Vector etc. For this conversion, the designers introduced **wrapper classes**.
- **What are Wrapper classes?**

- As the name says, a wrapper class wraps (encloses) around a data type and gives it an object appearance. Wherever, the data type is required as an object, this object can be used. Wrapper classes include methods to unwrap the object and give back the data type. It can be compared with a chocolate. The manufacturer wraps the chocolate with some foil or paper to prevent from pollution. The user takes the chocolate, removes and throws the wrapper and eats it.

- Observe the following conversion.
- **int k = 100;**
  Integer it1 = new Integer(k);
- The **int** data type **k** is converted into an object, **it1** using **Integer** class. The **it1**object can be used in Java programming wherever **k** is required an object.
- The following code can be used to unwrap (getting back **int** from **Integer** object) the object **it1**.
- **int m = it1.intValue();**
  System.out.println(m*m); // prints 10000
- **intValue()** is a method of **Integer** class that returns an **int** data type.

- **List of Wrapper classes**
- In the above code, **Integer** class is known as a wrapper class (because it wraps around int data type to give it an impression of object). To wrap (or to convert) each primitive data type, there comes a wrapper class. Eight wrapper classes exist in **java.lang** package that represent 8 data types. Following list gives.

Primitive Data Types	Wrapper Classes
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

- **Importance of Wrapper classes**
- To convert simple data types into objects, that is, to give object form to a data type; here constructors are used.

- The transient keyword

The transient keyword is used in serialization. If you define any data member as transient, it will not be serialized. Let's take an example, I have declared a class as Student, it has three data members id, name and age. If you serialize the object, all the values will be serialized but I don't want to serialize one value, e.g. age then we can declare the age datamember as transient.

- **Java InetAddress Class**
- Java InetAddress Class is used to encapsulate the two thing.
- **1. Numeric IP Address**
- **2. The domain name for that address.**
-
- The InetAddress can handle both IPv4 and IPv6 addresssses. It has no visible constructors so to create its object, the user have to use one of the available in-built static methods.
-

- The commonly used InetAddress in-built methods are:
-
- **(1) getLocalHost():** It returns the InetAddress object that represents the local host contain the name and address both. If this method unable to find out the host name, it throw an UnknownHostException.
-
- **Syntax:**
- Static InetAddress getLocalHost() throws UnknownHostException
-

- **(2) getByName():** It returns an InetAddress for a host name passed to it as a parameter argument. If this method unable to find out the host name, it throw an UnknownHostException.
-
- **Syntax:**
- Static InetAddress getByName(String host_name) throws UnknownHostException
-

- **(3) getAllByName():** It returns an array of an InetAddress that represent all of the addresses that a particular name resolve to it. If this method can’t find out the name to at least one address, it throw an UnknownHostException.
-
- **Syntax:**
- Static InetAddress[] getAllByName(String host_name) throws UnknownHostException
-
-

- Networking:

Networking is a concept of connecting two or more computing devices together so that we can share resources.

Advantages:

- sharing resources
- centralize software management
-

- Networking Terminology:

There are some networking terminologies given below:

- IP Address
- Protocol
- Port Number
- Connection-oriented
- Connectionless protocol
- Socket

- IP Address:

IP address is a unique number assigned to a node of a network e.g. 192.168.0.1 . It is composed of octets that range from 0 to 255.Protocol:

A protocol is a set of rules basically that is followed for communication. For example:

- TCP
- FTP
- Telnet
- SMTP
- POP etc.

- **Network Programming Introduction:**
-
- As we all know that Computer Network means a group of computers connect with each other via some medium and transfer data between them as and when require.
-
- Java supports Network Programming so we can make such program in which the machines connected in network will send and receive data from other machine in the network by programming.
-

- The first and simple logic to send or receive any kind of data or message is we must have the address of receiver or sender. So when a computer needs to communicate with another computer, it`s require the other computer’s address.
-
- Java networking programming supports the concept of socket. A socket identifies an endpoint in a network. The socket communication takes place via a protocol.

- The Internet Protocol is a lower-level, connection less (means there is no continuing connection between the end points) protocol for delivering the data into small packets from one computer (address) to another computer (address) across the network (Internet). It does not guarantee to deliver sent packets to the destination.
-

- IP addresses are written in a notation using numbers separated by dots is called dotted-decimal notation. There are four 8 bits value between 0 and 255 are available in each IP address such as 127.0.0.1 means local-host, 192.168.0.3 etc.
- It`s not an easy to remember because of so many numbers, they are often mapped to meaningful names called domain names such as mail.google.com There is a server on Internet who is translate the host names into IP addresses is called DNS (Domain Name Server).

- **NOTE:** Internet is the global network of millions of computer and the any computer may connect the Internet through LAN (Local Area Network), Cable Modem, ISP (Internet Service Provider) using dialup.

- When a user pass the URL like java2all.com in the web-browser from any computer, it first ask to DNS to translate this domain name into the numeric IP address and then sends the request to this IP address. This enables users to work with domain names, but the internet operates on IP addresses.
- Here in java2all.com the "com" domain is reserved for commercial sites; then "java2all" is the company name.
- The Higher-level protocol used in with the IP are TCP (Transmission Control Protocol) and UDP (User Datagram Protocol).

- The TCP enables two host to make a connection and exchange the stream of data, so it`s called Stream-based communication. TCP guarantees delivery of data and also guarantees that streams of data will be delivered in the same order in which they are sent. The TCP can detect the lost of transmission and so resubmit them and hence the transmissions are lossless and reliable.
-
-

- The UDP is a standard, directly to support fast, connectionless host-to-host datagram oriented model that is used over the IP and exchange the packet of data so it`s called packet-based communication. The UDP cannot guarantee lossless transmission.
-
- JAVA supports both TCP and UDP protocol families.

- The java.net package provides support for the two common network protocols:
- **TCP:** TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.
- **UDP:** UDP stands for User Datagram Protocol, a connection-less protocol that allows for packets of data to be transmitted between applications.
-

- Socket Programming:

  Socket programming is performed for communication between the machines. Socket programming can be connection-oriented or connectionless. Socket and ServerSocket classes are used for connection-oriented socket programming. The client in socket programming must know two information:

- IPaddress of Server, and
- Port number.

- Socket class:

A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket. Commonly used methods of Socket class:

- public InputStream getInputStream()
- public OutputStream getOutputStream()
- public synchronized void close()

- ServerSocket class:

The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients.Commonly used methods of ServerSocket class:

- public Socket accept()
- public InputStream getInputStream()
- public OutputStream getOutputStream()
- public synchronized void close()

- Socket Programming:
- Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to a server.
- When the connection is made, the server creates a socket object on its end of the communication. The client and server can now communicate by writing to and reading from the socket.

- The java.net.Socket class represents a socket, and the java.net.ServerSocket class provides a mechanism for the server program to listen for clients and establish connections with them.
- The following steps occur when establishing a TCP connection between two computers using sockets:
- The server instantiates a ServerSocket object, denoting which port number communication is to occur on.

- The server invokes the accept() method of the ServerSocket class. This method waits until a client connects to the server on the given port.
- After the server is waiting, a client instantiates a Socket object, specifying the server name and port number to connect to.

- The constructor of the Socket class attempts to connect the client to the specified server and port number. If communication is established, the client now has a Socket object capable of communicating with the server.
- On the server side, the accept() method returns a reference to a new socket on the server that is connected to the client's socket.

- After the connections are established, communication can occur using I/O streams. Each socket has both an OutputStream and an InputStream. The client's OutputStream is connected to the server's InputStream, and the client's InputStream is connected to the server's OutputStream.
- TCP is a twoway communication protocol, so data can be sent across both streams at the same time. There are following usefull classes providing complete set of methods to implement sockets.
-

- ServerSocket Class Methods:
- The **java.net.ServerSocket** class is used by server applications to obtain a port and listen for client requests
- The ServerSocket class has four constructors:

SN	Methods with Description
1	**public ServerSocket(int port) throws IOException** Attempts to create a server socket bound to the specified port. An exception occurs if the port is already bound by another application.
2	**public ServerSocket(int port, int backlog) throws IOException** Similar to the previous constructor, the backlog parameter specifies how many incoming clients to store in a wait queue.
3	**public ServerSocket(int port, int backlog, InetAddress address) throws IOException** Similar to the previous constructor, the InetAddress parameter specifies the local IP address to bind to. The InetAddress is used for servers that may have multiple IP addresses, allowing the server to specify which of its IP addresses to accept client requests on
4	**public ServerSocket() throws IOException** Creates an unbound server socket. When using this constructor, use the bind() method when you are ready to bind the server socket

- Here are some of the common methods of the ServerSocket class:

SN	Methods with Description
1	**public int getLocalPort()** Returns the port that the server socket is listening on. This method is useful if you passed in 0 as the port number in a constructor and let the server find a port for you.
2	**public Socket accept() throws IOException** Waits for an incoming client. This method blocks until either a client connects to the server on the specified port or the socket times out, assuming that the time-out value has been set using the setSoTimeout() method. Otherwise, this method blocks indefinitely
3	**public void setSoTimeout(int timeout)** Sets the time-out value for how long the server socket waits for a client during the accept().
4	**public void bind(SocketAddress host, int backlog)** Binds the socket to the specified server and port in the SocketAddress object. Use this method if you instantiated the ServerSocket using the no-argument constructor.

- S ocket Class Methods:
- The **java.net.Socket** class represents the socket that both the client and server use to communicate with each other. The client obtains a Socket object by instantiating one, whereas the server obtains a Socket object from the return value of the accept() method.
- The Socket class has five constructors that a client uses to connect to a server:

SN	Methods with Description
1	**public Socket(String host, int port) throws UnknownHostException, IOException.** This method attempts to connect to the specified server at the specified port. If this constructor does not throw an exception, the connection is successful and the client is connected to the server.
2	**public Socket(InetAddress host, int port) throws IOException** This method is identical to the previous constructor, except that the host is denoted by an InetAddress object.
3	**public Socket(String host, int port, InetAddress localAddress, int localPort) throws IOException.** Connects to the specified host and port, creating a socket on the local host at the specified address and port.
4	**public Socket(InetAddress host, int port, InetAddress localAddress, int localPort) throws IOException.** This method is identical to the previous constructor, except that the host is denoted by an InetAddress object instead of a String
5	**public Socket()** Creates an unconnected socket. Use the connect() method to connect this socket to a server.

- When the Socket constructor returns, it does not simply instantiate a Socket object but it actually attempts to connect to the specified server and port.
- Some methods of interest in the Socket class are listed here. Notice that both the client and server have a Socket object, so these methods can be invoked by both the client and server.

SN	Methods with Description
1	**public void connect(SocketAddress host, int timeout) throws IOException** This method connects the socket to the specified host. This method is needed only when you instantiated the Socket using the no-argument constructor.
2	**public InetAddress getInetAddress()** This method returns the address of the other computer that this socket is connected to.
3	**public int getPort()** Returns the port the socket is bound to on the remote machine.
4	**public int getLocalPort()** Returns the port the socket is bound to on the local machine.
5	**public SocketAddress getRemoteSocketAddress()** Returns the address of the remote socket.
6	**public InputStream getInputStream() throws IOException** Returns the input stream of the socket. The input stream is connected to the output stream of the remote socket.
7	**public OutputStream getOutputStream() throws IOException** Returns the output stream of the socket. The output stream is connected to the input stream of the remote socket
8	**public void close() throws IOException** Closes the socket, which makes this Socket object no longer capable of connecting again to any server

SN	Methods with Description
1	**static InetAddress getByAddress(byte[] addr)** Returns an InetAddress object given the raw IP address .
2	**static InetAddress getByAddress(String host, byte[] addr)** Create an InetAddress based on the provided host name and IP address.
3	**static InetAddress getByName(String host)** Determines the IP address of a host, given the host's name.
4	**String getHostAddress()** Returns the IP address string in textual presentation.
5	**String getHostName()** Gets the host name for this IP address.
6	**static InetAddress InetAddress getLocalHost()** Returns the local host.
7	**String toString()** Converts this IP address to a String.

- URL class:

The URL class represents a URL. URL is an acronym for Uniform Resource Locator. It points to a resource on the World Wide Web. For example:
http://www.javatpoint.com/sonoojaiswal/index.jsp

A URL contains many informations:

- **Protocol:** In this case, http is the protocol.

- **Server name or IP Address:** In this case, www.javatpoint.com is the server name.

- **Port Number:** It is an optional attribute. If we write http//ww.javatpoint.com:80/praveen/ , 80 is the port number.
- **File Name or directory name:** In this case, index.jsp is the file name.

- Commonly used methods of URL class:
- **public String getProtocol():** it returns the protocol of the URL.
- **public String getHost():** it returns the host name of the URL.
- **public String getPort():** it returns the Port Number of the URL.
- **public String getFile():** it returns the file name of the URL.

- URLConnection class:

The URLConnection class represents a communication link between the URL and the application. This class can be used to read and write data to the specified resource reffered by the URL.How to get the object of URLConnection class:

The openConnection() method of URL class returns the object of URLConnection class.

Syntax:

```
public URLConnection openConnection()throws IOException{
    }
  }
```

- Displaying all the data of a webpage by URLConnecton class

The URLConnection class provides many methods, we can display all the data of a webpage by using the getInputStream() method. The getInputStream() method returns all the data of the specified URL in the stream that can be read and displayed.

- InetAddress class:
- The java.net.InetAddress class represents an IP address. The Inet Address class provides methods to get the IP of any host name.
- Commonly used methods of InetAddress class:
- **public static InetAddress getByName(String host) throws UnknownHostException:** it returns the IP of the given host.

- **public static InetAddress getLocalHost() throws UnknownHostException:** it returns the LocalHost IP and name.
- **public String getHostName():** it returns the host name of the IP address.
- **public String getHostAddress():** it returns the IP address in string format.

- DatagramSocket and DatagramPacket (Networking)
- DatagramSocket class

The DatagramSocket class represents a connection-less socket for sending and receiving datagram packets. Datagram is basically an information but there is no gurantee of its content, arrival or arrival time. The DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

- Commonly used Constructors of DatagramSocket class
- **DatagramSocket() throws SocketEeption:** it creates a datagram socket and binds it with the available Port Number on the localhost machine.
- **DatagramSocket(int port) throws SocketEeption:** it creates a datagram socket and binds it with the given Port Number.

- **DatagramSocket(int port, InetAddress address) throws SocketEeption:** it creates a datagram socket and binds it with the specified port number and host address.

- DatagramPacket class:

  The DatagramPacket is message that can be sent or received. If you send multiple packet, it may arrive in any order. Moreover, packet delivery is not guaranteed.

- Commonly used Constructors of DatagramPacket class
- **DatagramPacket(byte[] barr, int length):** it creates a datagram packet. This constructor is used to receive the packets.

- **DatagramPacket(byte[] barr, int length, InetAddress address, int port):** it creates a datagram packet. This constructor is used to send the packets.

- **Java Collection Framework**

  Collection Framework provides an architecture to store and manipulate the group of objects. All the operations that you perform on a data such as searching, sorting, insertion, deletion etc. can be performed by Java Collection Framework.

- Collection simply means a single unit of objects. Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc).

- **What is Collection**

Collection represents a single unit of objects i.e. a group.

What is framework?

- provides readymade architecture.
- represents set of classes and interface
- is optional.

- **Collection framework**

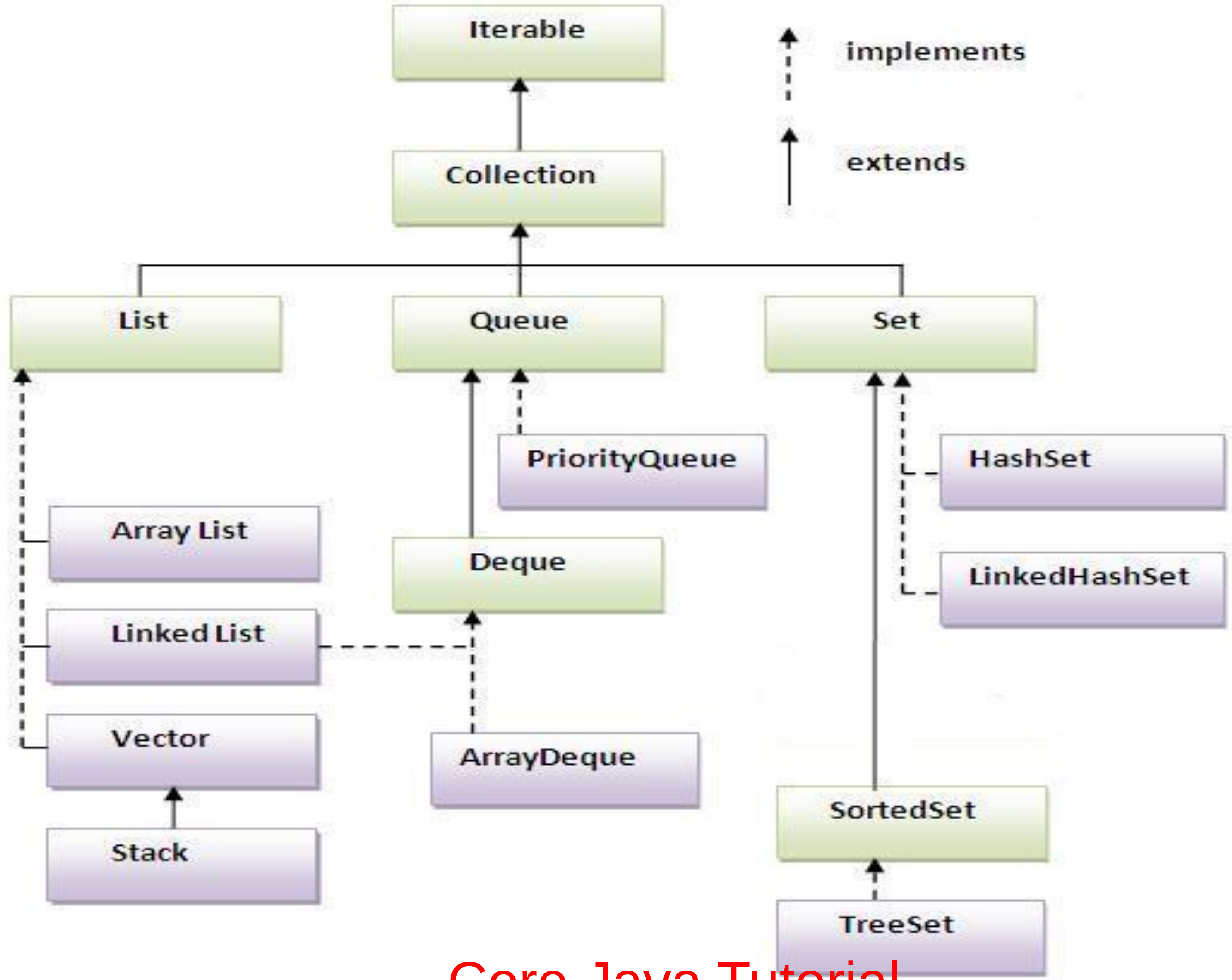
Collection framework represents a unified architecture for storing and manipulating group of object.

It has:

- Interfaces and its implementations i.e. classes
- Algorithm

- Hierarchy of Collection Framework

Let us see the hierarchy of collection frame work. The java.util package contains all the classes and interfaces for Collection framework.


Iterable
implements
extends
Collection
List
Queue
Set
PriorityQueue
HashSet
Array List
Deque
LinkedHashSet
Linked List
Vector
ArrayDeque
SortedSet
Stack
TreeSet

- Commonly used methods of Collection interface
- **There are many methods declared in the Collection interface. They are as follows:**

Method	**Description**
public boolean add(object element)	is used to insert an element in this collection.
public boolean addAll(collection c)	is used to insert the specified collection elements in the invoking collection.
public boolean remove(object element)	is used to delete an element from this collection.
public boolean removeAll(Collection c)	is used to delete all the elements of specified collection from the invoking collection.

Method	Description
public boolean retainAll(Collection c)	is used to delete all the elements of invoking collection except the specified collection.
public int size()	return the total number of elements in the collection.
public void clear()	removes the total no of element from the collection.
public boolean contains(object element)	is used to search an element.
public boolean containsAll(collection c)	is used to search the specified collection in this collection.
public Iterator iterator()	returns an iterator.

- Iterator interface

- Iterator interface provides the facility of iterating the elements in forward direction only.

- **Methods of Iterator interface**

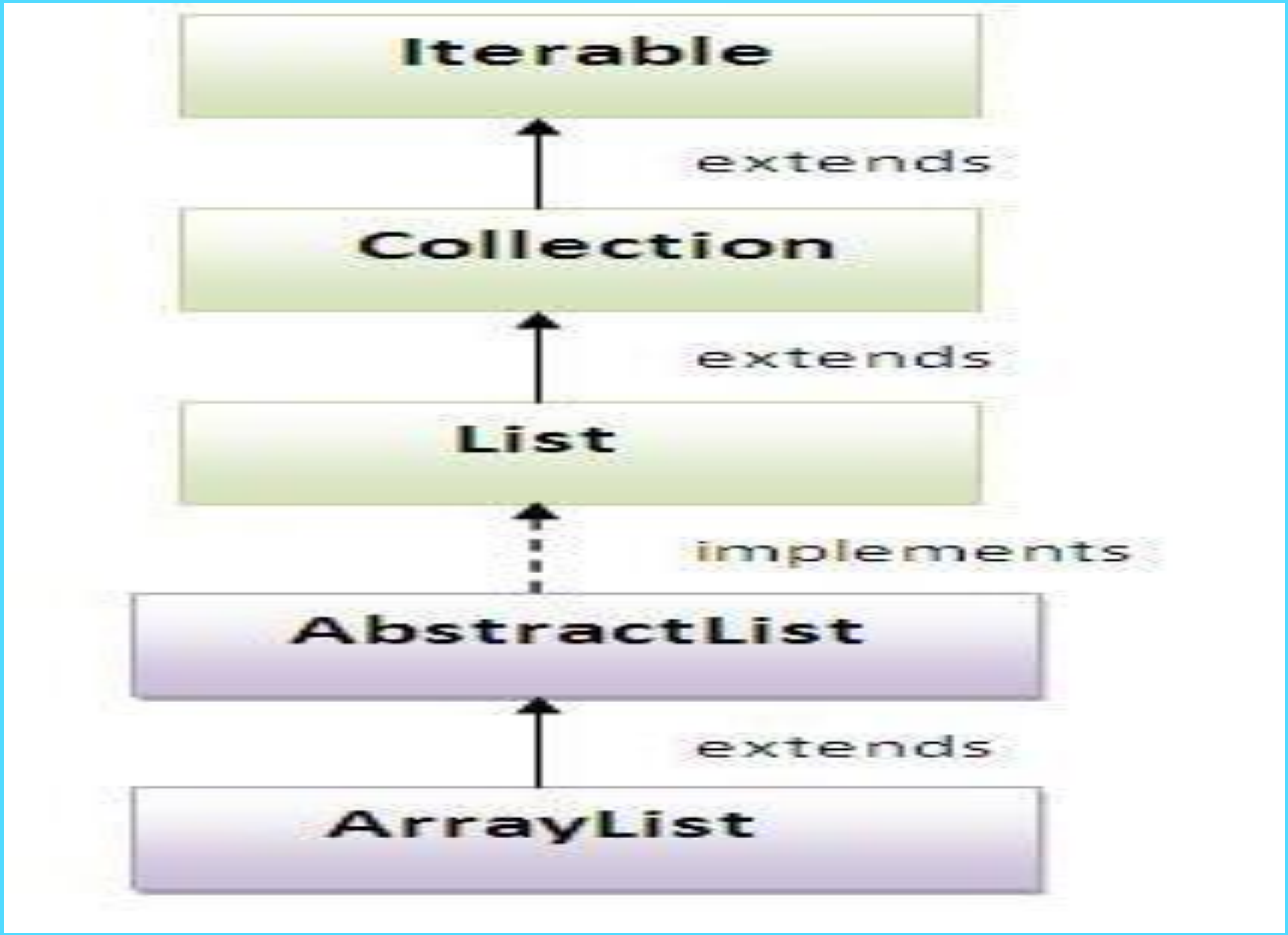
There are only three methods in the Iterator interface. They are:

- **public boolean hasNext() :** it returns true if iterator has more elements.
- **public object next()** : it returns the element and moves the cursor pointer to the next element.
- **public void remove() :** it removes the last elements returned by the iterator. It is rarely used.
-

- ArrayList class:

  uses a dynamic array for storing the elements.It extends AbstractList class and implements List interface.
- can contain duplicate elements.
- maintains insertion order.
- not synchronized.
- random access because array works at the index basis.
- manipulation slow because a lot of shifting needs to be occured.

- Hierarchy of ArrayList class:



- Two ways to iterate the elements of collection:
- By Iterator interface.
- By for-each loop.

- Linked List (Doubly Linked List)

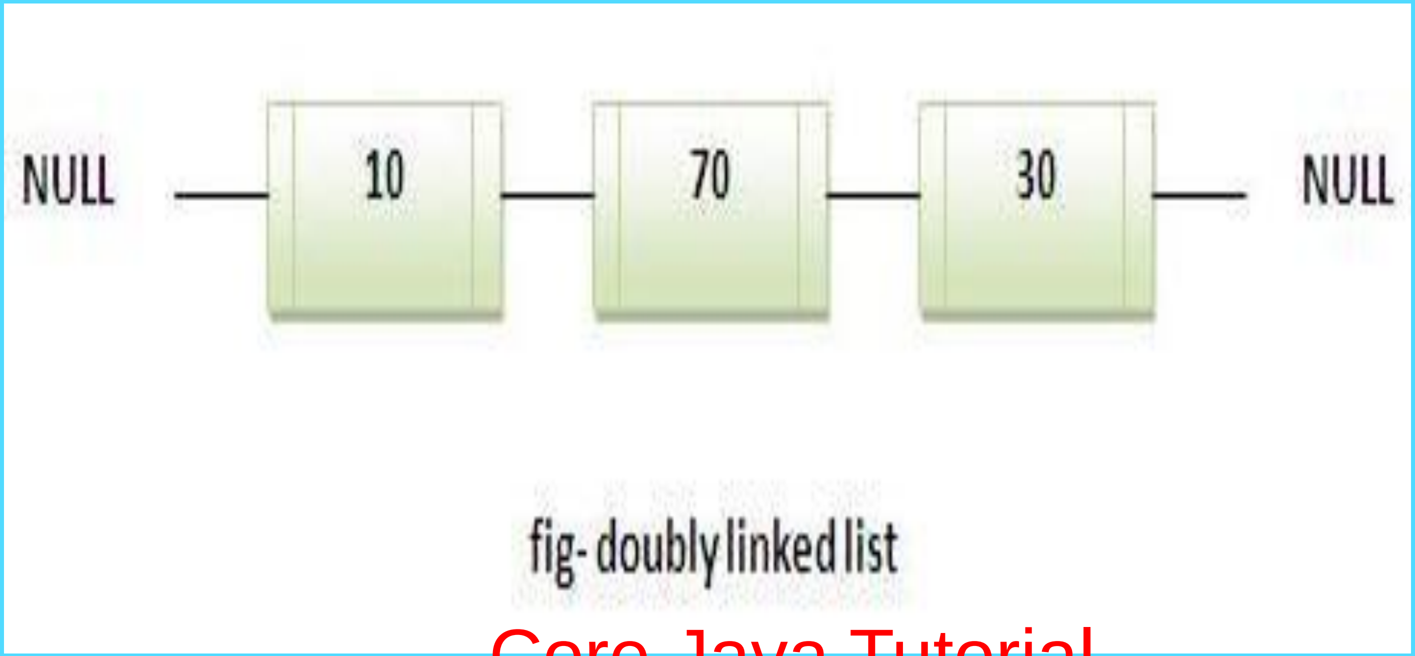


fig- doubly linked list

- LinkedList class:
- uses doubly linked list to store the elements. It extends the AbstractList class and implements List and Deque interfaces.
- can contain duplicate elements.
- maintains insertion order.
- not synchronized.
- No random access.
- manipulation fast because no shifting needs to be occured.
- can be used as list, stack or queue.

- List Interface:
- List Interface is the subinterface of Collection.It contains methods to insert and delete elements in index basis.It is a factory of ListIterator interface.

Commonly used mehtods of List Interface:

- **public void add(int index,Object element);**
- **public boolean addAll(int index,Collection c);**

- **public object get(int Index position);**
- **public object set(int index,Object element);**
- **public object remove(int index);**
- **public ListIterator listIterator();**
- **public ListIterator listIterator(int i);**

- ListIterator Interface:

ListIterator Interface is used to traverse the element in backward and forward direction.Commonly used mehtods of ListIterator Interface:

- **public boolean hasNext();**
- **public Object next();**
- **public boolean hasPrevious();**
- **public Object previous();**

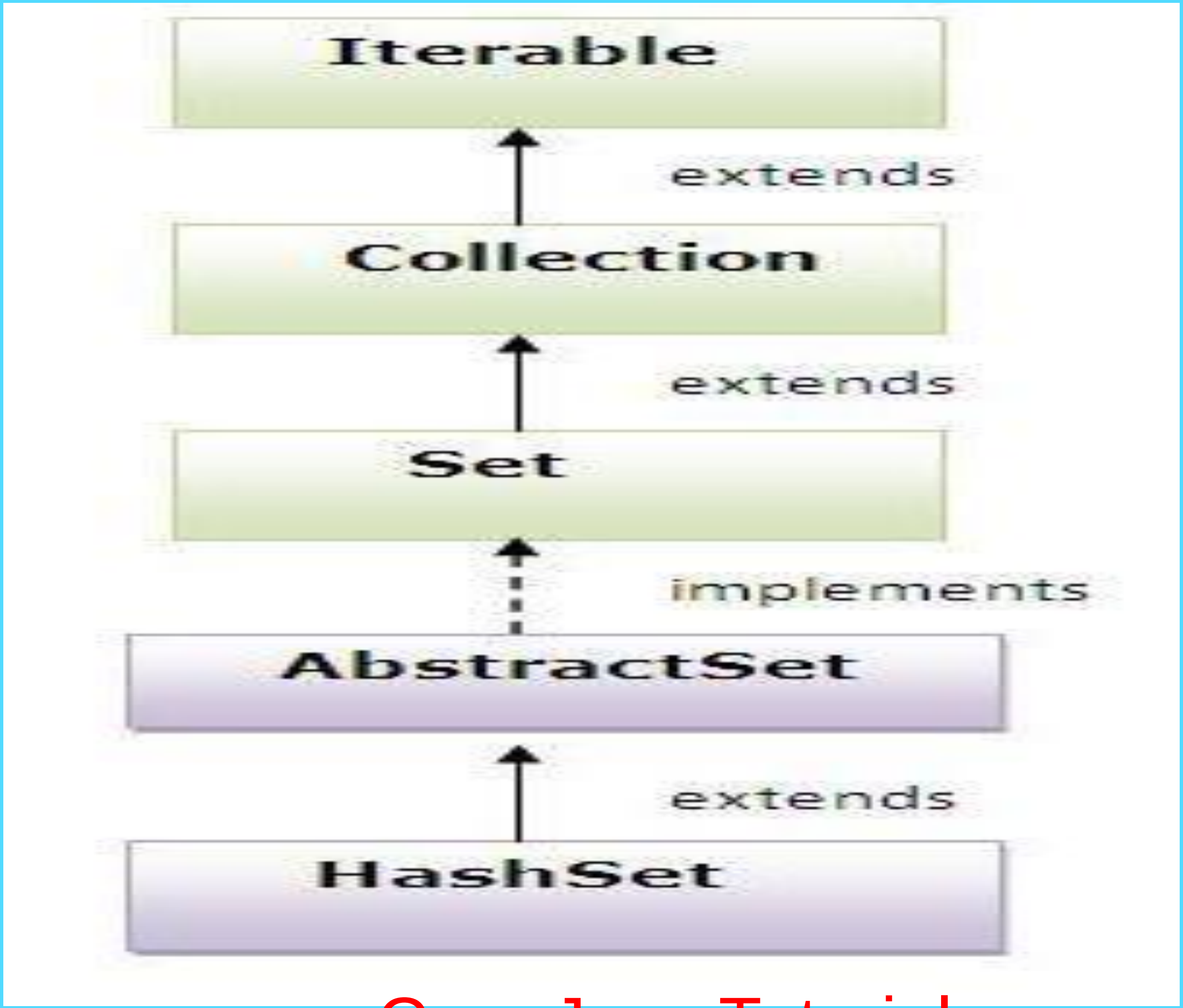
- Difference between List and Set:

- List can contain duplicate elements whereas Set contains unique elements only.
- HashSet class:

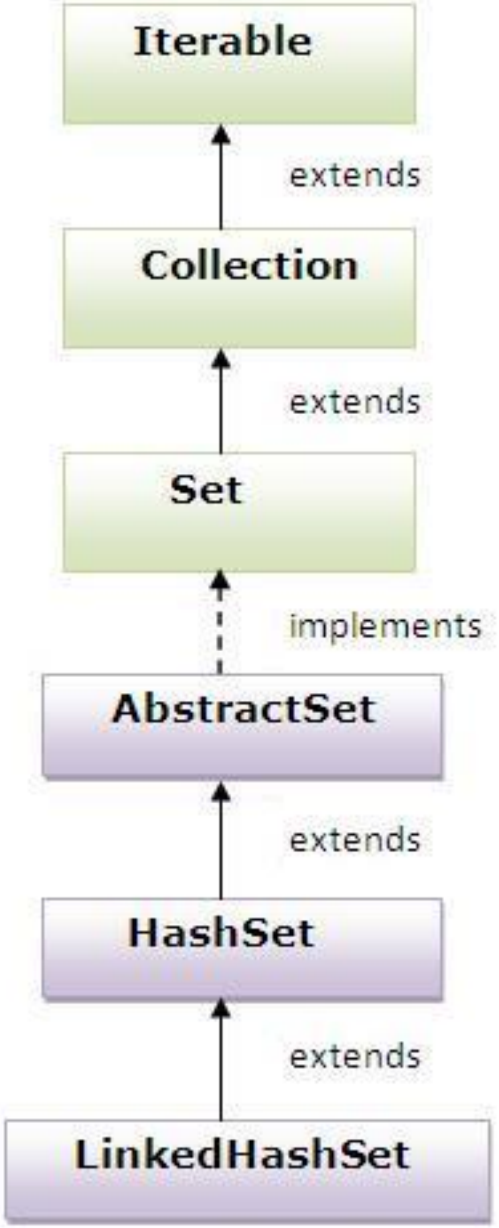
uses hashtable to store the elements.It extends AbstractSet class and implements Set interface.

contains unique elements only.

Hierarchy of HashSet class:



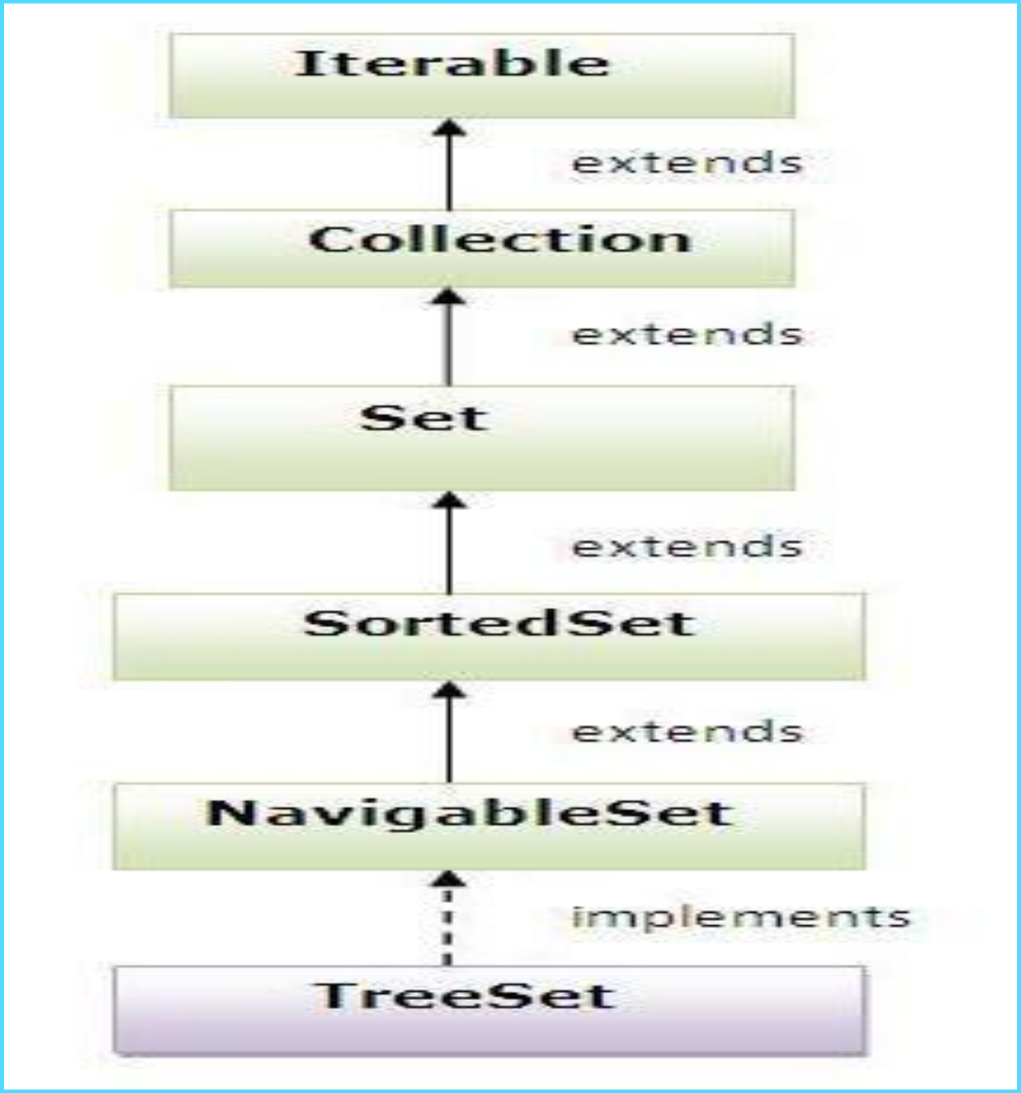
- LinkedHashSet class:
- contains unique elements only like HashSet. It extends HashSet class and implements Set interface.
- maintains insertion order.
- Hierarchy of LinkedHashSet class:


Iterable
extends
Collection
extends
Set
implements
AbstractSet
extends
HashSet
extends
LinkedHashSet

- TreeSet class:

  contains unique elements only like HashSet. The TreeSet class implements NavigableSet interface that extends the SortedSet interface.
- maintains ascending order.

- Hierarchy of TreeSet class:



- Queue Interface:
- The Queue interface basically orders the element in FIFO(First In First Out)manner.Methods of Queue Interface :
- public boolean add(object);
- public boolean offer(object);
- public remove();
- public poll();
- public element();
- public peek();
- PriorityQueue class:
- The PriorityQueue class provides the facility of using queue. But it does not orders the elements in FIFO manner.

- Map Interface

  A map contains values based on the key i.e. key and value pair.Each pair is known as an entry.Map contains only unique elements.
- Commonly used methods of Map interface:
- **public Object put(object key,Object value):** is used to insert an entry in this map.
- **public void putAll(Map map):** is used to insert the specifed map in this map.

- **public Object remove(object key):**is used to delete an entry for the specified key.
- **public Object get(Object key):**is used to return the value for the specified key.
- **public boolean containsKey(Object key):**is used to search the specified key from this map.
- **public boolean containsValue(Object value):**is used to search the specified value from this map.

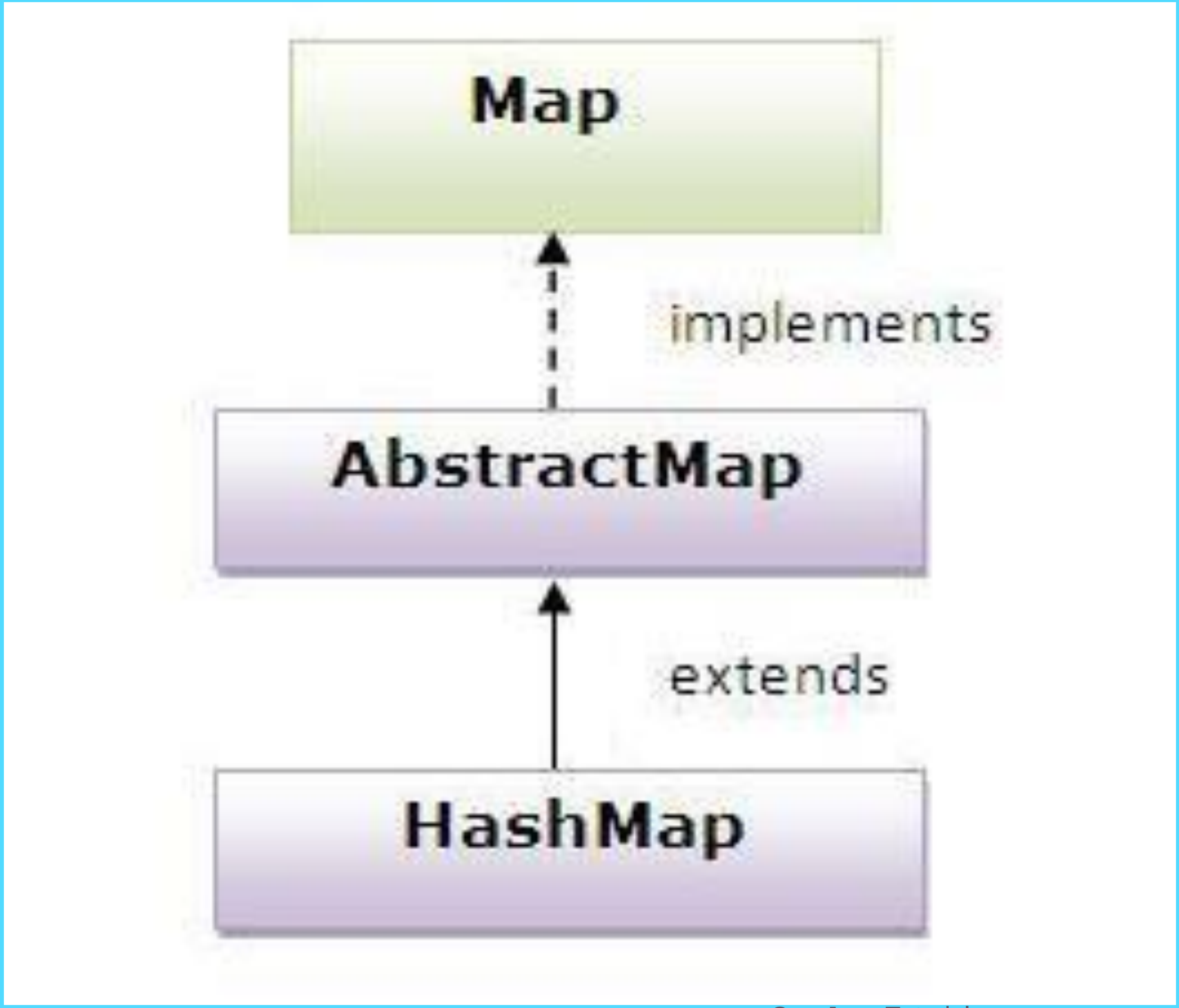
- **public Set keySet():**returns the Set view containing all the keys.
- **public Set entrySet():**returns the Set view containing all the keys and values.
- Entry
- Entry is the subinterface of Map.So we will access it by Map.Entry name.It provides methods to get key and value.Methods of Entry interface:
- **public Object getKey():** is used to obtain key.
- **public Object getValue():**is used to obtain value.

- HashMap class:

  A HashMap contains values based on the key. It implements the Map interface and extends AbstractMap class.

- It contains only unique elements.
- It may have one null key and multiple null values.
- It maintains no order.

- Hierarchy of HashMap class:



- What is difference between HashSet and HashMap?

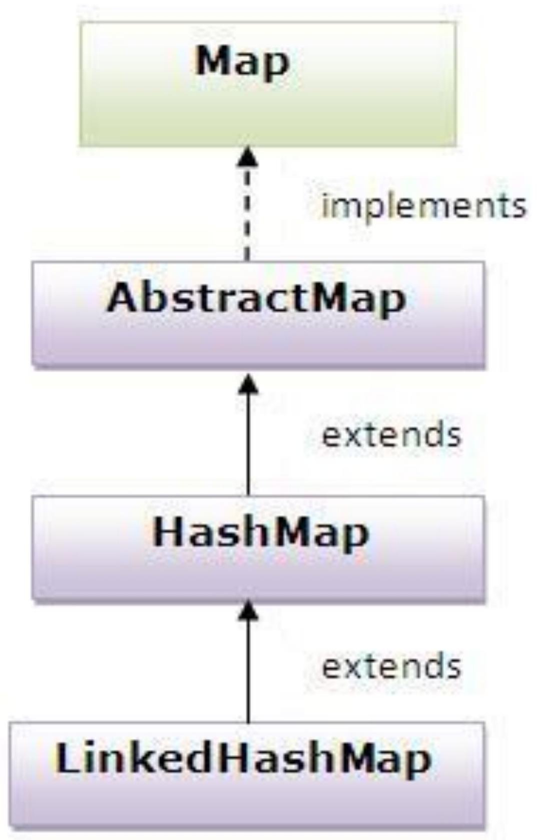
  HashSet contains only values whereas HashMap contains entry(key and value).

- LinkedHashMap class:

  A LinkedHashMap contains values based on the key. It implements the Map interface and extends HashMap class.

- It contains only unique elements.
- It may have one null key and multiple null values.
- It is same as HashMap instead maintains insertion order.

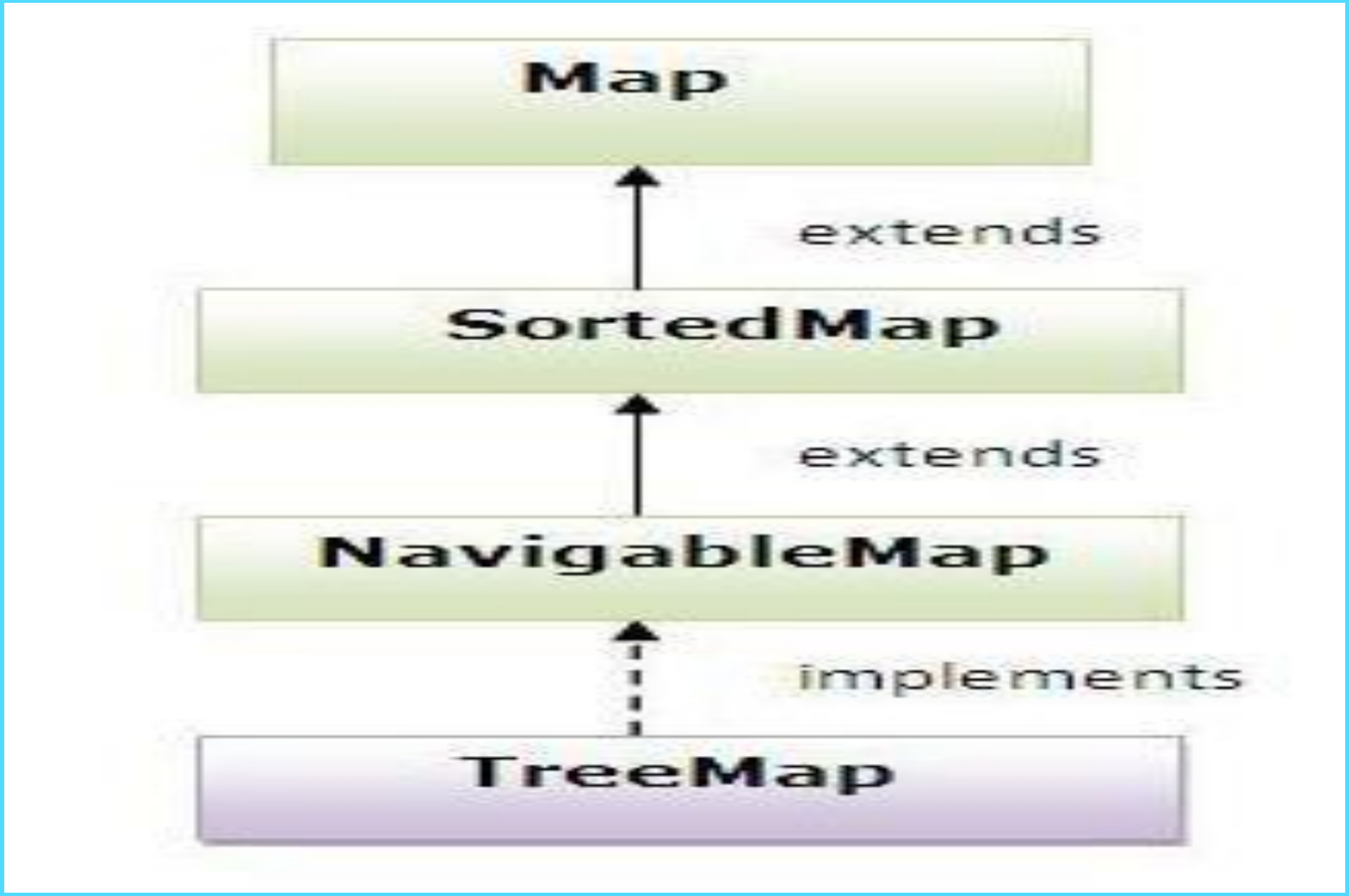
• Hierarchy of LinkedHashMap class:



- TreeMap class

  A TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.
- It contains only unique elements.
- It cannot have null key but can have multiple null values.
- It is same as HashMap instead maintains ascending order.

- Hierarchy of TreeMap class:



- What is difference between HashMap and TreeMap?

Hash Map	Tree Map
1) HashMap is can contain one null key.	TreeMap connot contain any null key.
2) HashMap maintains no order.	TreeMap maintains ascending order.

- Hashtable

  A Hashtable is an array of list.Each list is known as a bucket.The position of bucket is identified by calling the hashcode() method.A Hashtable contains values based on the key. It implements the Map interface and extends Dictionary class.

- It contains only unique elements.
- It may have not have any null key or value.
- It is synchronized.

- What is difference between HashMap and Hashtable?

Hash Map	Hash Table
1) HashMap is not synchronized.	Hashtable is synchronized.
2) HashMap can contain one null key and multiple null values.	Hashtable cannot contain any null key nor value.

- Sorting
- We can sort the elements of:String objects
- Wrapper class objects
- User-defined class objects
- **Collections** class provides static methods for sorting the elements of collection.If collection elements are of Set type, we can use TreeSet.But We cannot sort the elements of List.Collections class provides methods for sorting the elements of List type elements.

- Method of Collections class for sorting List elements
- **public void sort(List list):** is used to sort the elements of List. List elements must be of Comparable type.
- **Note: String class and Wrapper classes implements the Comparable interface.So if you store the objects of string or wrapper classes, it will be Comparable.**

- Comparable interface

Comparable interface is used to order the objects of user-defined class.This interface is found in java.lang package and contains only one method named compareTo(Object).It provide only single sorting sequence i.e. you can sort the elements on based on single datamember only.For instance it may be either rollno,name,age or anything else.

- Syntax:
- **public int compareTo(Object obj):** is used to compare the current object with the specified object.We can sort the elements of:String objects
- Wrapper class objects
- User-defined class objects
- **Collections** class provides static methods for sorting the elements of collection. If collection elements are of Set type, we can use TreeSet. But We cannot sort the elements of List. Collections class provides methods for sorting the elements of List type elements.

- Method of Collections class for sorting List elements

- **public void sort(List list):** is used to sort the elements of List. List elements must be of Comparable type.
- ***Note: String class and Wrapper classes implements the Comparable interface.So if you store the objects of string or wrapper classes, it will be Comparable.***

- Comparator interface
- **Comparator interface** is used to order the objects of user-defined class.
- This interface is found in java.util package and contains 2 methods compare(Object obj1,Object obj2) and equals(Object element).
- It provides multiple sorting sequence i.e. you can sort the elements based on any data member. For instance it may be on rollno, name, age or anything else.

- **syntax of compare method**
- **public int compare(Object obj1,Object obj2):** compares the first object with second object.
- **Collections** class provides static methods for sorting the elements of collection. If collection elements are of Set type, we can use TreeSet. But We cannot sort the elements of List. Collections class provides methods for sorting the elements of List type elements.

- **Method of Collections class for sorting List elements**
- **public void sort(List list,Comparator c):** is used to sort the elements of List by the given comparator.

- Properties class in Java
- The **properties** object contains key and value pair both as a string. It is the subclass of Hashtable.
- It can be used to get property value based on the property key. The Properties class provides methods to get data from properties file and store data into properties file. Moreover, it can be used to get properties of system.
- **Advantage of properties file**
- **Easy Maintenance:** If any information is changed from the properties file, you don't need to recompile the java class. It is mainly used to contain variable information i.e. to be changed.

- Methods of Properties class
- The commonly used methods of Properties class are given below.

Method	Description
public void load(Reader r)	loads data from the Reader object.
public void load(InputStream is)	loads data from the InputStream object
public String getProperty(String key)	returns value based on the key.
public void setProperty(String key,String value)	sets the property in the properties object.
public void store(Writer w, String comment)	writers the properties in the writer object.
public void store(OutputStream os, String comment)	writes the properties in the OutputStream object.

Method	Description
storeToXML(OutputStream os, String comment)	writers the properties in the writer object for generating xml document.
public void storeToXML(Writer w, String comment, String encoding)	writers the properties in the writer object for generating xml document with specified encoding.

- New Features in Java
- There are many new features that have been added in java. There are major enhancement made in Java5, Java6 and Java7 like **auto-boxing**, **generics**, **var-args**, **java annotations**, **enum**, **premain method** etc.

- **J2SE 4 Features**

  The important feature of J2SE 4 is assertions. It is used for testing.
- **Assertion** (Java 4)

- **J2SE 5 Features**

  The important features of J2SE 5 are generics and assertions. Others are auto-boxing, enum, var-args, static import, for-each loop enhanced for loop etc.
- **For-each loop (Java 5)**
- **Varargs (Java 5)**
- **Static Import (Java 5)**
- **Autoboxing and Unboxing (Java 5)**
- **Enum (Java 5)**
- **Covariant Return Type (Java 5)**
- **Annotation (Java 5)**
- **Generics (Java 5)**

- **JavaSE 6 Features**
- The important feature of JavaSE 6 is premain method (also known as instrumentation).
- Instrumentation (premain method) (Java 6)
- **JavaSE 7 Features**
- The important features of JavaSE 7 are try with resource, catching multiple exceptions etc.
- String in switch statement (Java 7)
- Binary Literals (Java 7)
- The try-with-resources (Java 7)
- Caching Multiple Exceptions by single catch (Java 7)
- Underscores in Numeric Literals (Java 7)

- For-each loop (Advanced or Enhanced For loop):
- The for-each loop introduced in Java5. It is mainly used to traverse array or collection elements. The advantage of for-each loop is that it eliminates the possibility of bugs and makes the code more readable.

  Advantage of for-each loop:
- It makes the code more readable.
- It elimnates the possibility of programming errors.
- Syntax of for-each loop:
- **for**(data_type variable : array | collection){}

- Variable Argument (Varargs):
- The varrags allows the method to accept zero or muliple arguments. Before varargs either we use overloaded method or take an array as the method parameter but it was not considered good because it leads to the maintenance problem. If we don't know how many argument we will have to pass in the method, varargs is the better approach.
- Advantage of Varargs:

We don't have to provide overloaded methods so less code.

- Syntax of varargs:

The varargs uses ellipsis i.e. three dots after the data type. Syntax is as follows:

- return_type method_name(data_type... variableName){}

- Rules for varargs:
- While using the varargs, you must follow some rules otherwise program code won't compile. The rules are as follows:There can be only one variable argument in the method.
- Variable argument (varargs) must be the last argument.

- Examples of varargs that fails to compile:

- **void** method(String... a, **int**... b){}//Compile time error
- **void** method(**int**... a, String b){}//Compile time error

- Static Import:

- The static import feature of Java 5 facilitate the java programmer to access any static member of a class directly. There is no need to qualify it by the class name. Advantage of static import:
- Less coding is required if you have access any static member of a class oftenly.

Disadvantage of static import:

- If you overuse the static import feature, it makes the program unreadable and un maintainable.

- What is the difference between import and static import?
- The import allows the java programmer to access classes of a package without package qualification whereas the static import feature allows to access the static members of a class without the class qualification. The import provides accessibility to classes and interface whereas static import provides accessibility to static members of the class.

- Autoboxing and Unboxing:

  The automatic conversion of primitive data types into its equivalent Wrapper type is known as boxing and opposite operation is known as unboxing. This is the new feature of Java5. So java programmer doesn't need to write the conversion code.Advantage of Autoboxing and Unboxing:

- No need of conversion between primitives and Wrappers manually so less coding is required.
- The automatic conversion of wrapper class type into corresponding primitive type, is known as Unboxing.

- Enum
- An enum is a data type which contains fixed set of constants. It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY and SATURDAY) , directions (NORTH, SOUTH, EAST and WEST) etc.

- The enum constants are static and final implicitely. It is available from Java 5. Enums can be thought of as classes that have fixed set of constants.

Points to remember for Enum:

- enum improves type safety
- enum can be easily used in switch
- enum can be traversed
- enum can have fields, constructors and methods
- enum may implement many interfaces but cannot extend any class because it internally extends Enum class

- What is the purpose of values() method in enum?
- The java compiler internally adds the values() method when it creates an enum. The values() method returns an array containing all the values of the enum.

- **Defining enum:**

- The enum can be defined within or outside the class because it is similar to a class

- Initializing specific value to the enum constants:

  The enum constants have initial value that starts from 0, 1, 2, 3 and so on. But we can initialize the specific value to the enum constants by defining fields and constructors. As specified earlier, Enum can have fields, constructors and methods.

- Java Annotation
- **Annotation** is a tag that represents the metadata. It is attached with class, interface, methods or fields to indicate some additional information that can be used by java compiler and JVM.
- Built-In Annotations
- There are several built-in annoations. Some annotations are applied to java code and some to other annotations.

- **Built-In Annotations that are applied to java code**
- @Override
- @SuppressWarnings
- @Deprecated
-

- Understanding Built-In Annotations that are applied to java code
- Let's understand the built-in annotations first.
- @Override
- @Override annotation assures that the subclass method is overriding the parent class method. If it is not so, compile time error occurs.
- Sometimes, we does the silly mistake such as spelling mistakes etc. So, it is better to mark @Override annotation that provides assurity that method is overridden.

- @SuppressWarnings
- If you remove the @SuppressWarnings("unchecked") annotation, it will show warning at compile time because we are using non-generic collection.
- @Deprecated
- @Deprecated annoation marks that this method is deprecated so compiler prints warning. It informs user that it may be removed in the future versions. So, it is better not to use such methods.
- @SuppressWarnings annotation: is used to suppress warnings issued by the compiler.

- Generics in Java

The **Java Generics** programming is introduced in J2SE 5 to deal with type-safe objects.

Before generics, we can store any type of objects in collection i.e. non-generic. Now generics, forces the java programmer to store specific type of objects.

- **Advantage of Java Generics**
- There are mainly 3 advantages of generics. They are as follows:
- **Type-safety :** We can hold only a single type of objects in generics. It doesn’t allow to store other objects.
- **Type casting is not required:** There is no need to typecast the object.

- Before Generics, we need to type cast.

```
List list = new ArrayList();
list.add("hello");
String s = (String) list.get(0);//typecasting
```

- After Generics, we don't need to typecast the object.

```
List<String> list = new ArrayList<String>();
list.add("hello");
String s = list.get(0);
```

- **Compile-Time Checking:** It is checked at compile time so problem will not occur at runtime. The good programming strategy says it is far better to handle the problem at compile time than runtime.

```
List<String> list = new ArrayList<String>();
list.add("hello");
list.add(32);//Compile Time Error
```

- **Syntax to use generic collection**

ClassOrInterface<Type>

Simple example to use Generics

ArrayList<String>

- Understanding Type Parameters

The type parameters naming conventions are important to learn generics thoroughly. The commonly type parameters are as follows:

- T - Type
- E - Element
- K - Key
- N - Number
- V - Value
- S,U,V etc. - 2nd, 3rd, 4th types

- Assertion:
- Assertion is a statement in java. It can be used to test your assumptions about the program.While executing assertion, it is believed to be true. If it fails, JVM will throw an error named AssertionError. It is mainly used for testing purpose.Advantage of Assertion:
- It provides an effective way to detect and correct programming errors.Syntax of using Assertion:

- way to use assertion.
- **assert** expression;
-

If you use assertion, It will not run simply because assertion is disabled by default. To enable the assertion,**-ea** or **-enableassertions** switch of java must be used.Compile it by: **javac AssertionExample.java**Run it by: **java -ea AssertionExample**

- Where not to use Assertion:
- There are some situations where assertion should be avoid to use. They are:According to Sun Specification, assertion should not be used to check arguments in the public methods because it should result in appropriate runtime exception e.g. IllegalArgumentException, NullPointerException etc.
- Do not use assertion, if you don't want any error in any situation.