

# Object Oriented Programming

## **COURSE OUTCOMES:**

- ❖ Use object oriented programming concepts to solve real world problems.
- ❖ Demonstrate the user defined exceptions by exception handling keywords (try, catch, throw, throws and finally).
- ❖ Use multithreading concepts to develop inter process communication.
- ❖ Develop java application to interact with database by using relevant software component (JDBC Driver).
- ❖ Solve real world problems using Collections.

# Unit-I

# Syllabus

## **JAVA BASICS:**

Review of Object oriented concepts, History of Java, Java buzzwords, JVM architecture, Data types, Variables, Scope and life time of variables, arrays, operators, control statements, type conversion and casting, simple java program, constructors, methods, Static block, Static Data, Static Method String and String Buffer Classes, Using Java API Document

# History of Java

# History of Java

- Initially developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released first official version in 1995.
- James Gosling also known as **Father of Java**



- The team initiated this project to develop a language for digital devices such as set-top boxes, television, etc.
- Originally C++ was considered to be used in the project but the idea was rejected for several reasons(For instance C++ required more memory).
- **James Gosling** , **Mike Sheridan**, and **Patrick Naughton** initiated the Java language project in June 1991.
- James Gosling and his team started their project “**Greentalk**” and its file extension was **.gt** and later became to known as “**OAK**”.

# Why it's named "Oak"?

- The name **Oak** was used by **Gosling** after an **oak tree** that remained outside his office.
- Also, Oak is an image of solidarity and picked as a national tree of numerous nations like the U.S.A., France, Germany, Romania, etc.
- But they had to later rename it as “**JAVA**” as it was already a trademark by **Oak Technologies**.

# OAK Tree



# Finally named it as Java

- Gosling and his team did a brainstorm session and after the session, they came up with several names such as **JAVA, DNA, SILK, RUBY, etc.**
- **Java** name was decided after much discussion since it was so unique.
- Gosling came up with this name while having a coffee near his office
- Java is **an Island** of Indonesia where the **first coffee** was produced (called java coffee). It is a kind of espresso bean.
- Note : -JAVA is not an acronym  
-It is not extension of C++

# Symbol of Java



# Java Buzzwords

# Java buzzwords

□ Java is a language characterized by buzzwords

◆ **buzzword:** A trendy word or phrase that is used more to impress than explain

□ From Sun Microsystems, the developers of Java:

*“Java is a simple, object-oriented, distributed, interpreted, robust, secure, architecture-neutral, portable, high performance, multi-threaded, and dynamic language.”*

What do all of those terms mean?

- Simple
- Object-oriented
- Distributed
- Interpreted
- Robust
- Secure
- Architecture neutral
- Portable
- High performance
- Multithreaded
- Dynamic

# Simple:

According to Sun, Java language is simple because:

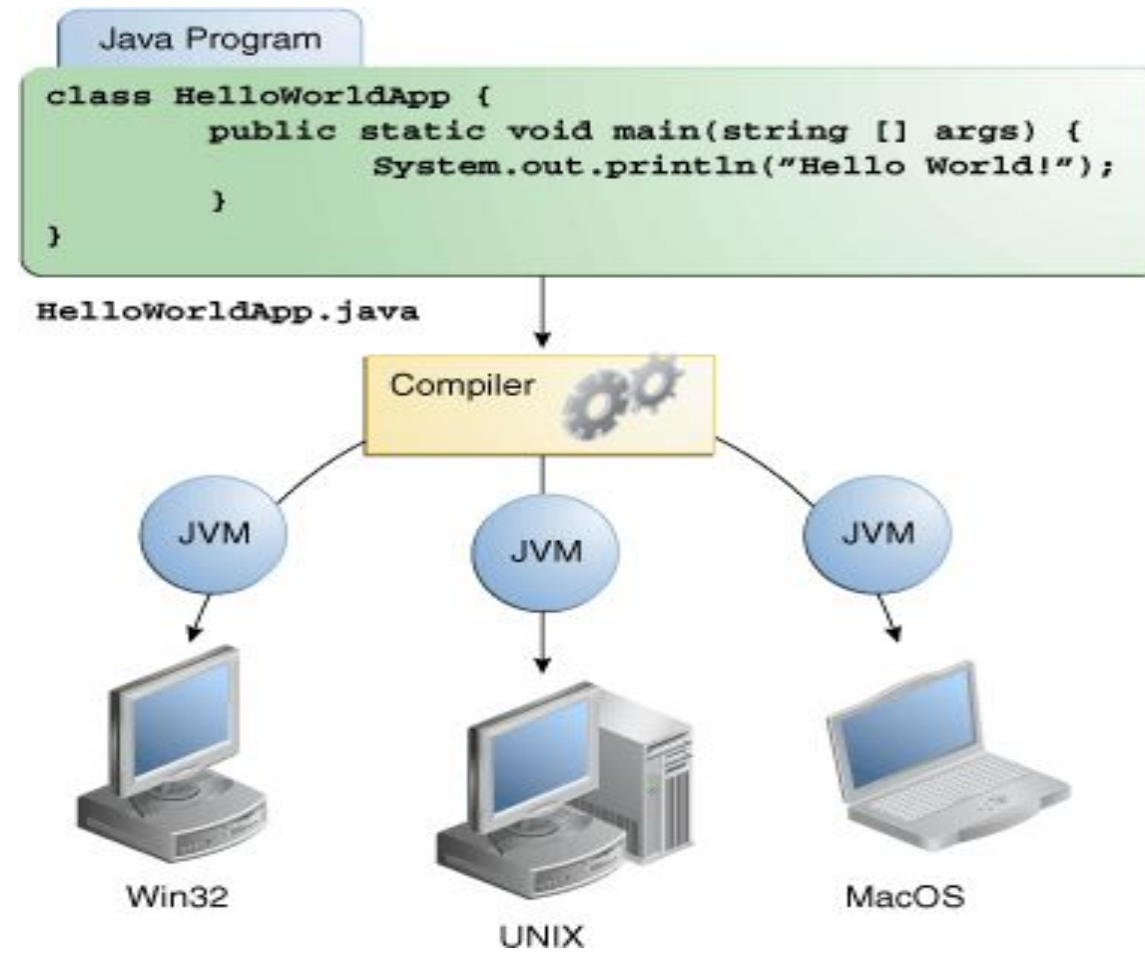
- Most of the syntaxes are based on C/C++ (so easier for programmers to learn it after C++).
- Removed many confusing and/or rarely-used features  
**Ex:** Explicit pointers, operator overloading etc.
- No need to remove unreferenced objects because there is Automatic Garbage Collection in java.

# Object-oriented:

- Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.
- Java supports OOPs features like
  - ❖ Abstraction
  - ❖ Encapsulation
  - ❖ Polymorphism
  - ❖ Inheritance

# Platform Independent:

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



# Secured:

- No explicit pointer
- To provide **implicit security** Java provide one component inside JVM called Security Manager.
- To provide **explicit security** for the Java applications we are having very good predefined library in the form of **java.Security.package**.
- Web security for web applications we are having JAAS(Java Authentication and Authorization Services) for distributed applications.

# Robust:

- Robust simply means strong.
- Any technology if it is good at two main areas it is said to be ROBUST .
  - ❖ Exception Handling
  - ❖ Memory Allocation .

JAVA is Robust because:

- JAVA is having very good predefined Exception Handling mechanism whenever we are getting exception we are having meaning full information.
- JAVA is having very good memory management system that is Dynamic Memory (at runtime the memory is allocated) Allocation which allocates and deallocates memory for objects at runtime.

# Architecture-neutral:

- There is no implementation dependent features .  
Ex: size of primitive types is fixed.
- In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture.
- But in java, it occupies 4 bytes of memory for both 32 and 64 bit architectures.

So Java is Architecturally neutral

# Compiled and Interpreted

- Usually, a computer language is either compiled or Interpreted. Java combines both this approach and makes it a two-stage system.
- **Compiled:** Java enables the creation of cross-platform programs by compiling into an intermediate representation called Java Bytecode.
- **Interpreted:** Bytecode is then interpreted, which generates machine code that can be directly executed by the machine that provides a Java Virtual machine.

# Portable:

- Java is portable because it facilitates you to carry the Java bytecode to any platform.
- It doesn't require any implementation.

# High-performance:

- Java is faster than traditional interpretation since byte code is "close" to native code still somewhat slower than a compiled language (e.g., C++)
- If any technology having features like Robust, Security, Platform Independent, Dynamic and so on then that technology is high performance.

# Multithreaded

- Multithreaded Programs handled multiple tasks simultaneously, which was helpful in creating interactive, networked programs.
- Java run-time system comes with tools that support multiprocessing synchronization used to construct smoothly interactive systems

# Distributed:

- We can create distributed applications in java.
- RMI is used for creating distributed applications
- We may access files by calling the methods from any machine on the internet.

# Dynamic

- Java is capable of linking in new class libraries, methods, and objects.
- Java programs carry with them substantial amounts of run-time type information that is used to verify and resolve accesses to objects at runtime.
- This makes it possible to dynamically link code in a safe and expedient manner.

# Java Application areas:

- Java is used to develop Desktop Applications such as Media Player, Antivirus etc.
- Java is Used to Develop Web Applications such as facebook.com, irctc.co.in etc.
- Java is Used to Develop Enterprise Application such as Banking applications.
- Java is Used to Develop Mobile Applications. Ex: Android Studio
- Java is Used to Develop Embedded System.
- Java can be used in Bigdata.
- Java is Used to Develop Robotics.
- Java is used to Develop Games etc.

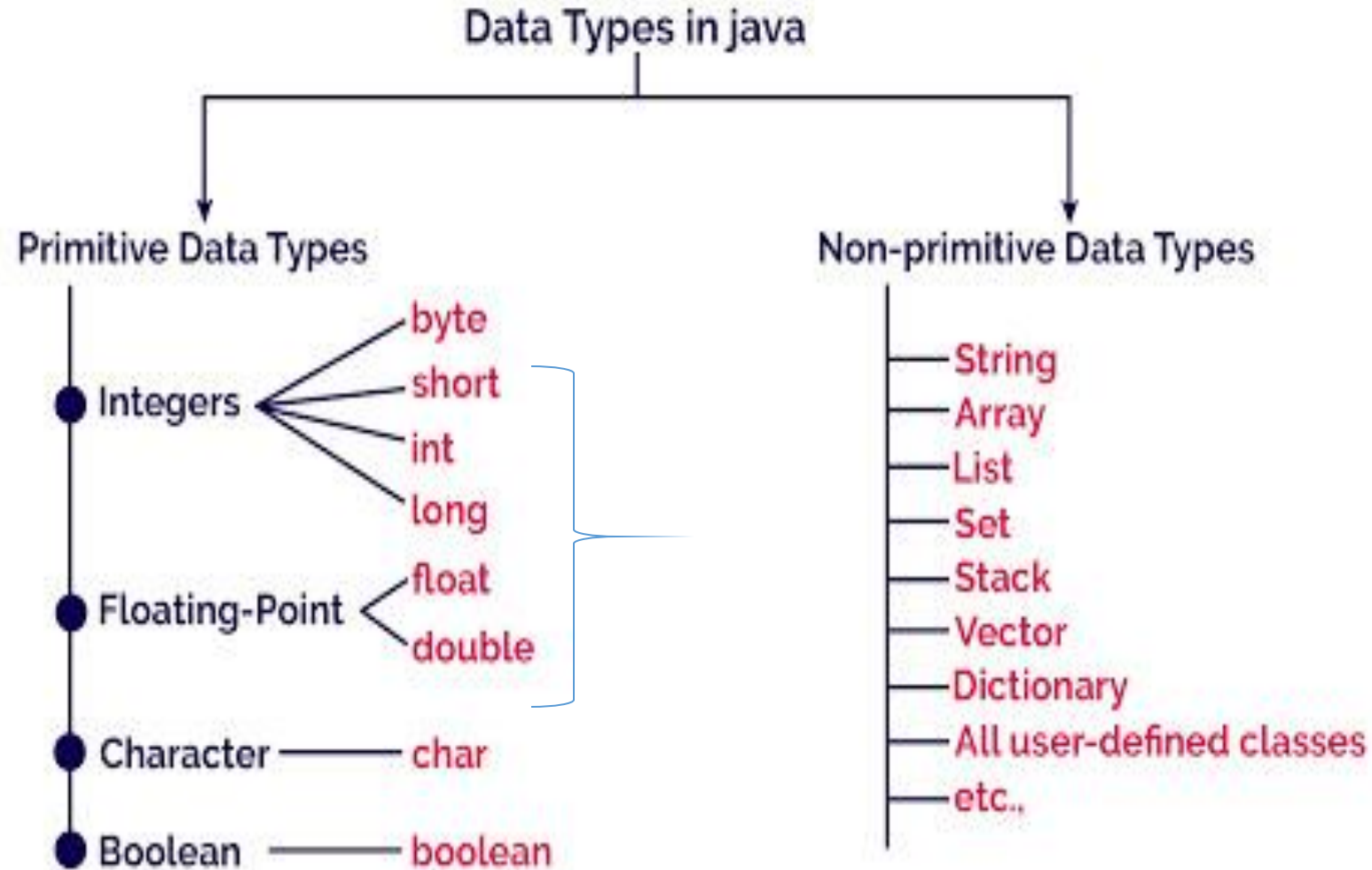
# Data Types

# What is data type ?

- Data types are used to represent the type of the variable and type of the expression.
- Java is **Strictly typed** / **Strongly typed** / **Statically typed**

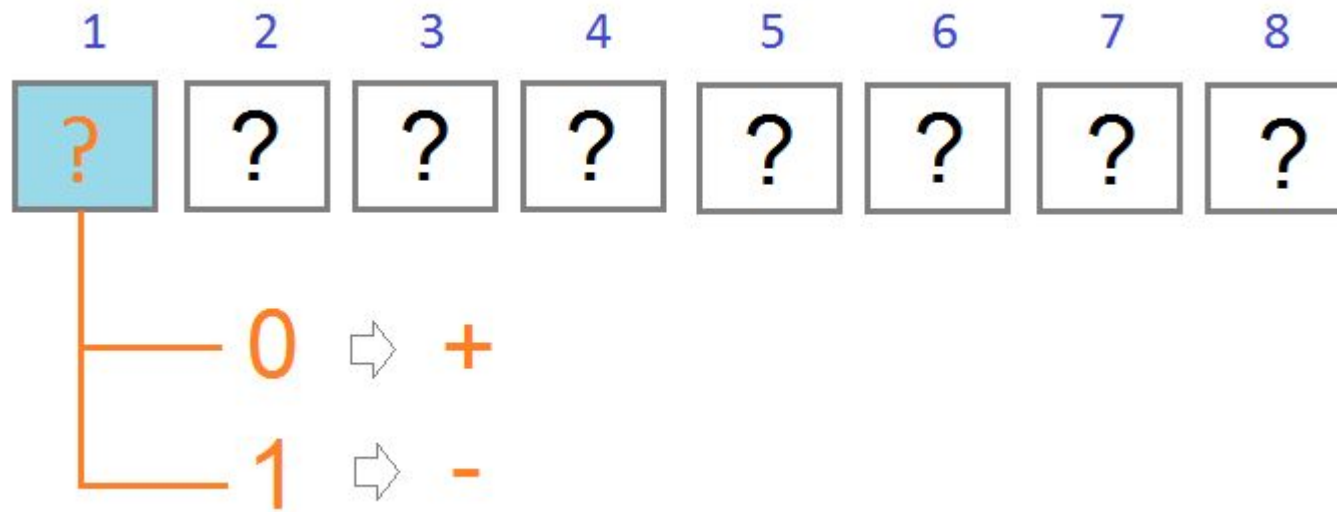
## Java is a Strongly Typed

- Every variable has a type
- Every expression has a type
- All assignments are checked for type compatibility at compile time



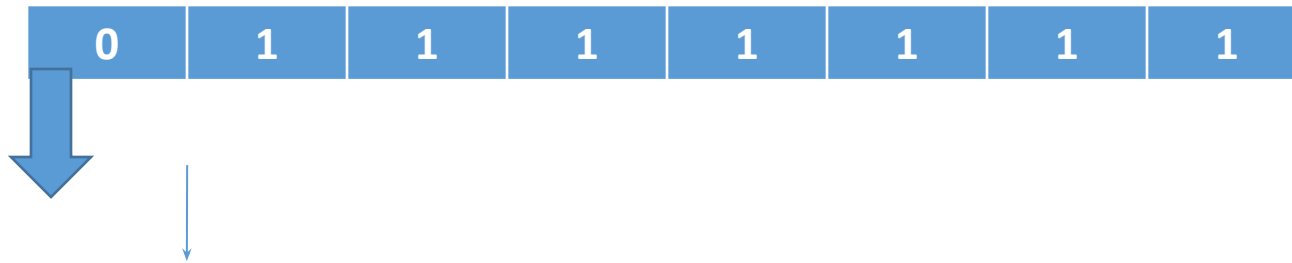
# Byte

- **byte** a byte is composed of 8 consecutive **bits** in the memory of computer. Each **bit** is a binary number of 0 or 1.



**Range:**

Max value=127



MSB=0----> +ve value

Value=?  $1+2+4+8+16+32+64=127$

## Range:

Min value=-128



MSB=1----> -ve value

Value? All negative values will be calculated in 2's complement form

0000000--->1111111

+1

-----

10000000 ==>-128

# Application area

- ❑ Used to store data into Files
- ❑ Network

# short data type

- The short data type is a 16-bit signed two's complement integer.
- Rarely used data type.
- **Size:** 2 bytes
- **Range:**-32,768 to 32767.
- **Default value:** 0

# int Data type

- int data type is the preferred data type when we create variables with a numeric value.
- It will take 32 bits or 4bytes of memory.
- **Range:-**2147483648 to 2147483647
- **Default value:** 0

# long Data type

- Used when int is not large enough to hold the value, it has wider range than int data type.
- **Size** :8 bytes or 64 bits
- **Range**:-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
- **Default value**: 0

## float Data type

- A float data type is a **single precision** number format that occupies 4 bytes or 32 bits of memory.
- A float data type in Java stores a decimal value with 6-7 total digits of precision.
- Ex: float f=10.25212121212f
- But it will take only **10.252121.**
- **Range:**  $3.4e-038$  to  $3.4e+038$
- **Default value:** 0.0

# double Data type

- The double data type is a **double-precision** 64-bit IEEE 754 floating point.
- In Java any floating value by default it's a **double** type
- 12-13 digits precisions it will take.
- EX: double d=10.2521212121213434332222;
- Output: **10.252121212121343**

## char data type

- The char data type is a single 16-bit Unicode character.
- Ex: `char c='A';`
- Unsigned data type.
- **Range:** 0 to 65535
- **Default value:** `\u0000`

# Why 16bits for char type??

- In C/C++ uses only ASCII characters and to represent all ASCII characters 8-bits is enough.
- Java uses Unicode system, to represent Unicode system 8 bit is not enough.
- Unicode= ASCII + Other language symbols.

## boolean Data type

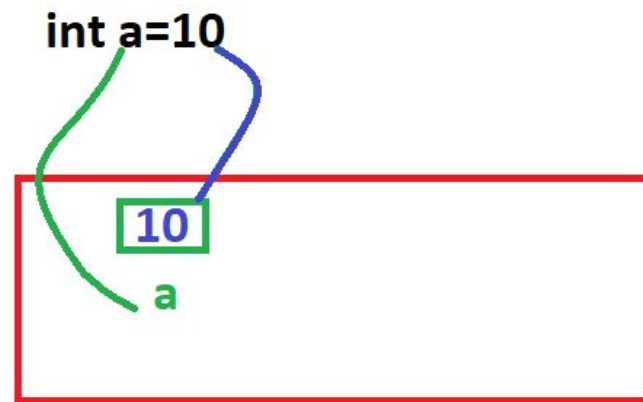
- The boolean data type has only two possible values: true and false.
- Use this data type for simple flags that track true/false conditions.
- This data type represents one bit of information, but its "size" isn't something that's precisely defined.

- Default value is **false**
- Ex: **boolean b=true;**
- **Boolean b=0;** → Not allowed

# Variables

# What is a variable??

- A variable is a name of the memory location. It is the small unit of storage in a program.



- The value stored in a variable can be varied during program execution.

## □ Syntax:

`datatype variable_name= value;`

**Ex:** `int a= 10;`  
`char c='A';`

## Note:

□ **Java is strongly typed.** So before using any variable in it is mandatory to declare with specific data type

# Naming convention of a variable

□ As per documentation all user variables are small case.

□ All constant variables should be in uppercase letter

**Ex: final double PI=3.141592653589793238;**

**final int DATABASE\_VERSION=1;**

□ **Note:**

Above rules are optional. But it is highly recommended to follow Java coding standards

# Types of Variables

□ There are three types of variables in Java:

- ❖ Local Variables
- ❖ Instance Variables
- ❖ Static Variables

# Local variables

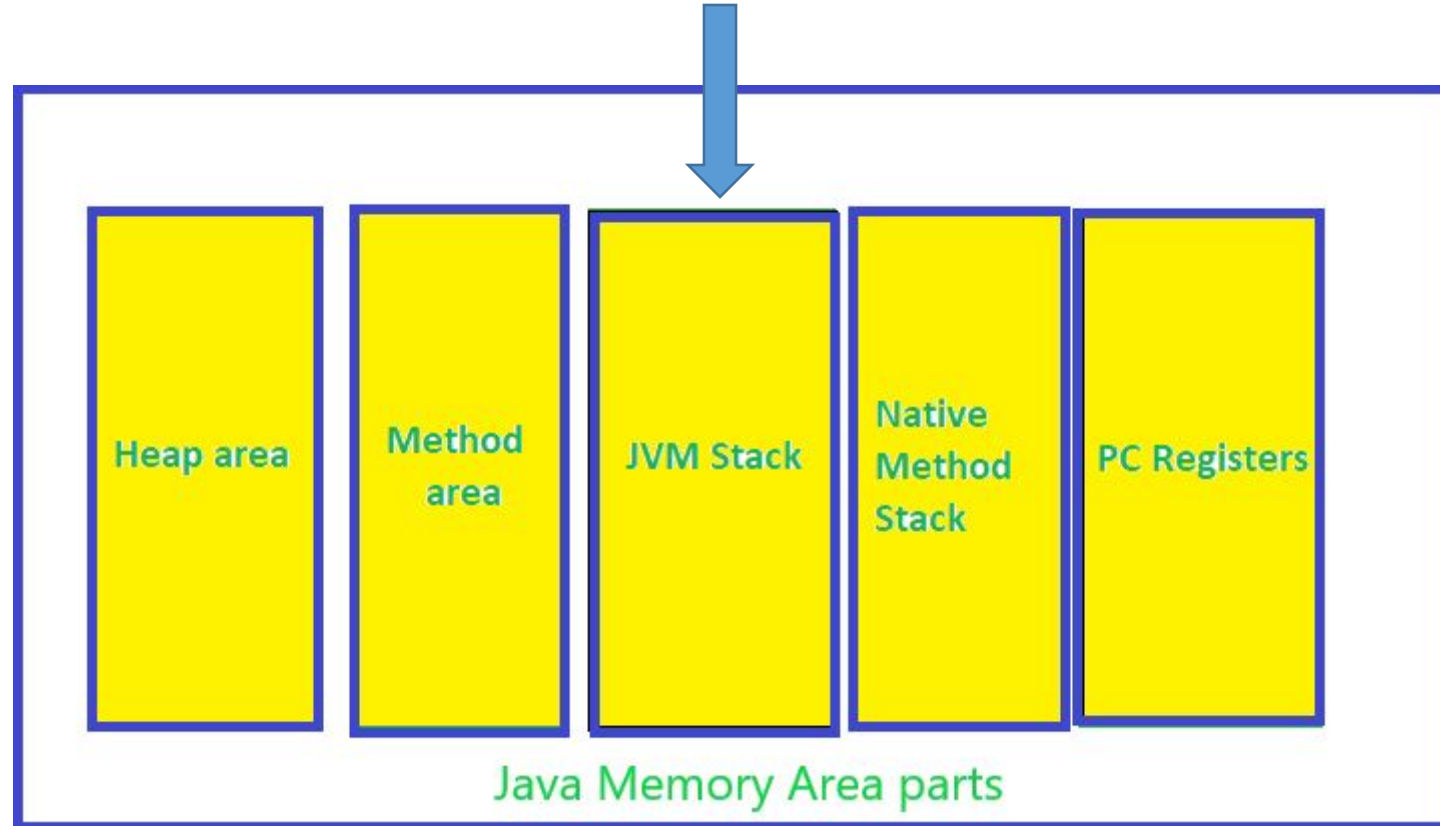
- The variables which are declare inside a method or inside a block or inside a constructor is called local variables.

Ex: class Student

```
{  
    public void studentInfo()  
    {  
        // local variables  
        String name="Ramu";  
        int age = 26;  
        System.out.println("Student name : " + name);  
    }  
}
```

- These variables are created when the block is entered or the function is called and destroyed after exiting from the block or when the call returns from the function.
- Hence The scope of local variables are inside a **method** or inside a **constructor** or inside a **block**.
- JVM wont provide any initial values for local variables. So initialization of local Variable is Mandatory. If not it will generate compile time error.

- Access modifiers (public , private , protected , default) are not allowed for local variables.
- Local variables will be stored in **stack** area.



**Example:**

```
public class Test
{
    public static void main(String[] args)
    {
        Test t=new Test();
        t.m1();
    }
    public void m1()
    {
        //Declaring variable inside method..
        int i=10;
        System.out.println(i);
    }
    public Test()
    {
        //Declaring variable inside constructor..
        int j=20;
        System.out.println(j);
    }
    {
        //Declaring variable inside block..
        int k=30;
        System.out.println(k);
    }
}
```

# Instance Variables

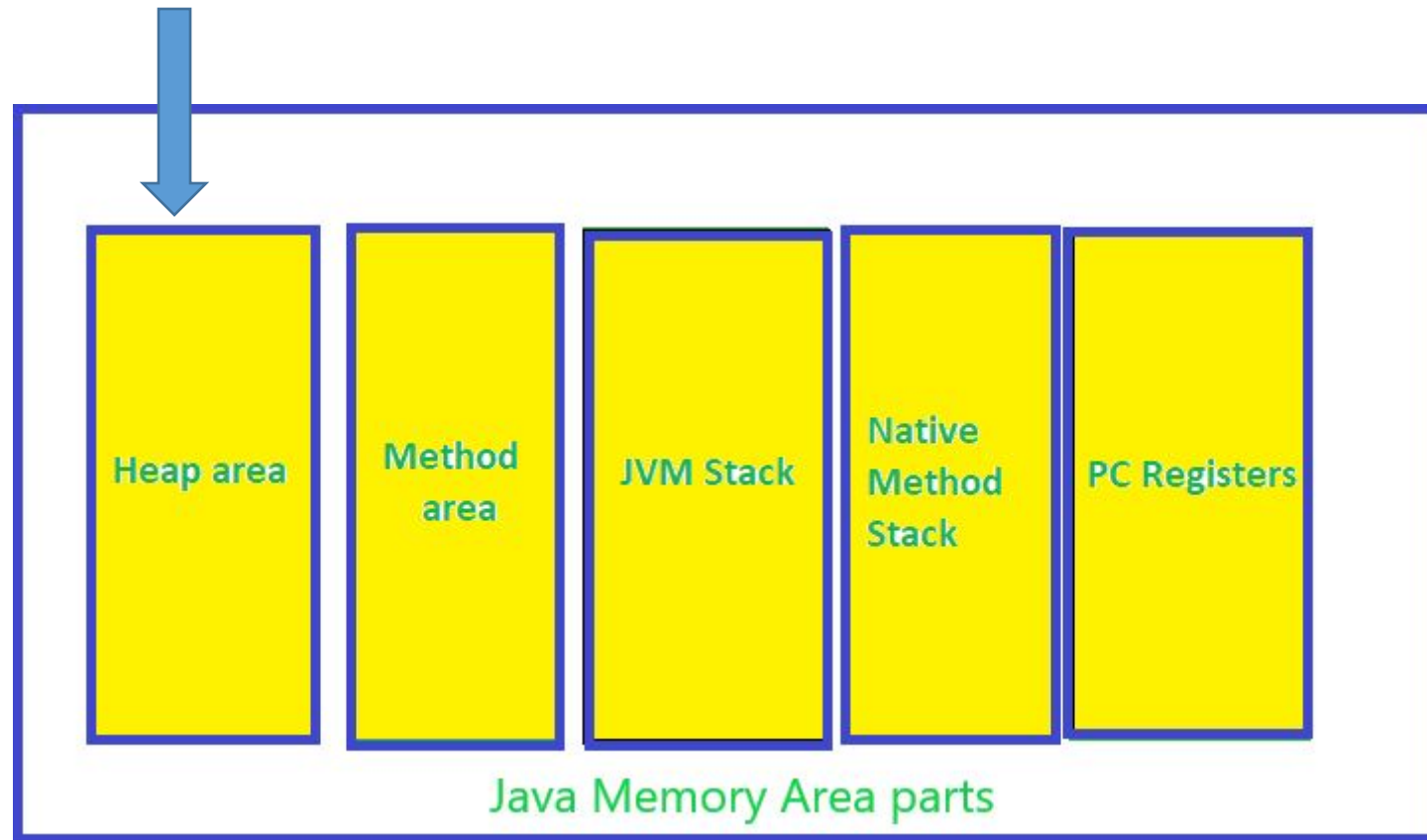
- Instance variables are declared inside class and outside of methods or constructor or block.

Ex: class A

```
{  
    int a;    //instance variable  
    public static void main(String[] args)  
    {  
    }  
}
```

- Instance variables are created when an object of the class is created and destroyed when the object is destroyed.
- We are able to access instance variables only inside the class any number of methods.

□ Instance variables are stored in heap area.



- Initialization of Instance Variable is **not mandatory** JVM automatically allocates default values.

### Example:

```
public class Sample
{
    int i=10; //With initializing
    int j; //Without initializing
    boolean b;
    public static void main(String[] args)
    {
        Sample s=new Sample();
        System.out.println(s.i); //10
        System.out.println(s.j); // 0
        System.out.println(s.b); // false
    }
}
```

□ Instance variables are not allowed inside static area directly. But using object it will allow.

**Ex:**

```
public class Sample
{
    int i=100;
    public static void main(String[] args)
    {
        System.out.println(i); // Not accessible..
        Sample s=new Sample();
        System.out.println(s.i); // Accessible..
    }
}
```

- Unlike static variable, instance variables have their own separate copy i.e, if any updation done in the instance variable that will not reflect on other objects.

### Example:

```
public class Instance
{
    int i=10;
    public static void main(String[] args)
    {
        int j=10;
        System.out.println("Before updating object-1 variable value..");
        Instance obj1=new Instance();
        System.out.println("Object-1 : i="+obj1.i);
        obj1.i=200;
        System.out.println("After updating object-1 variable value..");
        System.out.println("Object-1 : i="+obj1.i);
        Instance obj2=new Instance();
        System.out.println("Object-2 : i="+obj2.i);
        Instance obj3=new Instance();
        System.out.println("Object-3 : i="+obj3.i);
    }
}
```

## Output:

Before updating object-1 variable value..

Object-1 : i=10

After updating object-1 variable value..

Object-1 : i=200

Object-2 : i=10

Object-3 : i=10

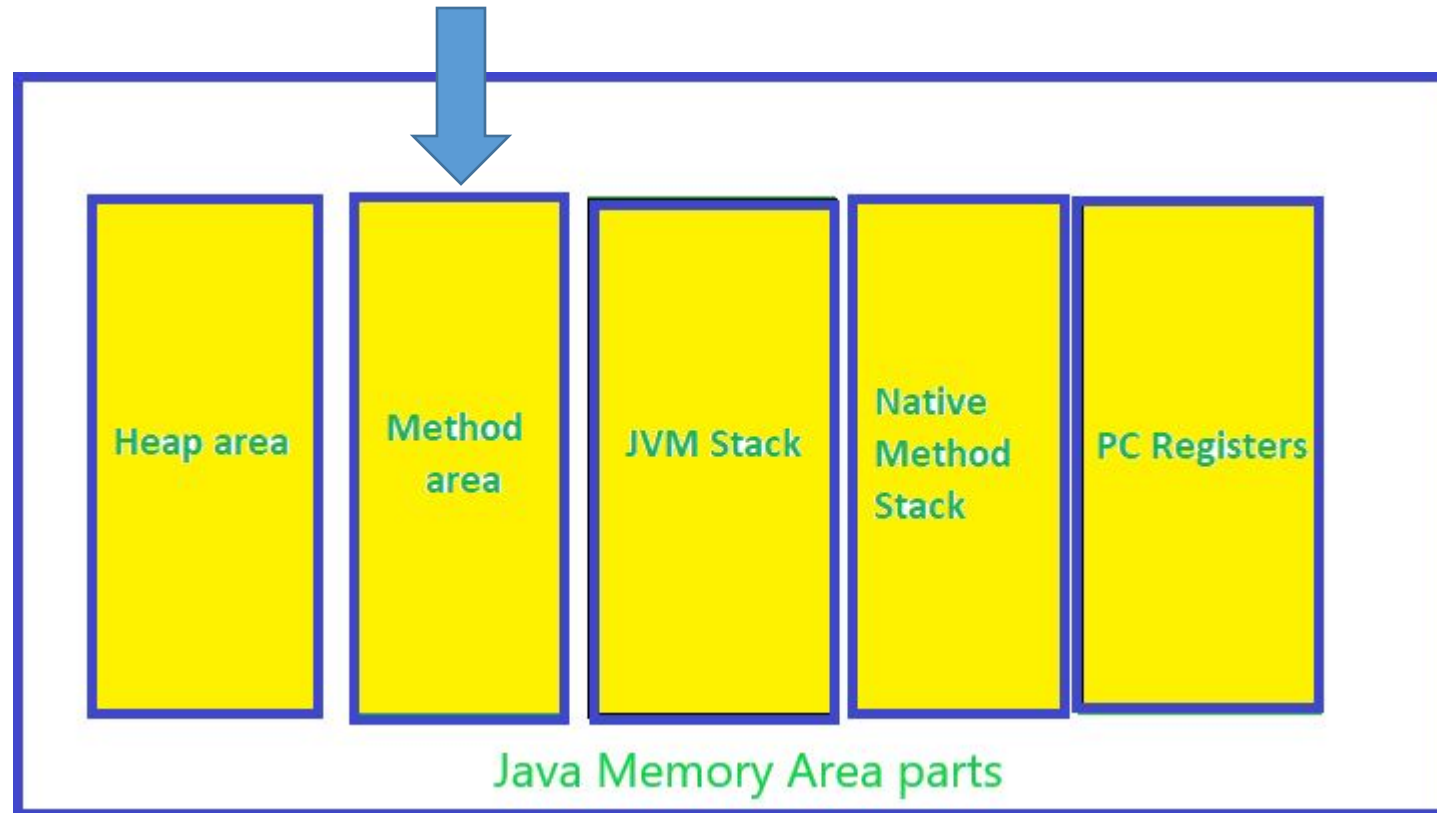
# Static variables

- If any variable declared inside class and out side methods with “**static**” key word is called static variable.

Ex: public class Test  
{  
    **static int a=10;**  
    public static void main(String[] args)  
    {  
    }  
}

- **static** variables are also called as **class variable** because they are associated with the class and common for all the instances of the class.

- **static** variables are created at class loading time and destroyed at class unloading.
- All static variables will be stored within **method** area.



□ Static variables are can be accessed from any area (instance or static) directly.

**Example:**

```
public class Sample
{
    static int i=100;
    public static void main(String[] args)
    {
        // static area..
        System.out.println(i);
        Sample s=new Sample();
        s.m1();
    }
    public void m1()
    {
        // instance area..
        System.out.println(i);
    }
}
```

□ Static variables can be accessed by using objects and using class name.

### Example:

```
public class Sample
{
    static int i=100;
    public static void main(String[] args)
    {
        System.out.println(i); //Direct access
        Sample s=new Sample();
        System.out.println(s.i); // Using object
        System.out.println(Sample.i); // Using class name
    }
}
```

- Initialization of static variables is not mandatory. JVM provides default values.
- Static variables maintain same copy for all objects in the class i.e, if any updating made in the static variable that will reflect on other objects.

### Example:

```
public class StaticVar
{
    static int i=10;
    public static void main(String[] args)
    {
        System.out.println("Before updating object-1 variable value..");
        StaticVar obj1=new StaticVar();
        System.out.println("Object-1 : i="+obj1.i);
        obj1.i=200;
        System.out.println("After updating object-1 variable value..");
        System.out.println("Object-1 : i="+obj1.i);
        StaticVar obj2=new StaticVar();
        System.out.println("Object-2 : i="+obj2.i);
        StaticVar obj3=new StaticVar();
        System.out.println("Object-3 : i="+obj3.i);
    }
}
```

### Output:

Before updating object-1 variable value..

Object-1 : i=10

After updating object-1 variable value..

Object-1 : i=200

Object-2 : i=200

Object-3 : i=200