Object Oriented Programming Concepts

Ms. J J Desai

What is String in Java?

- Generally, String is a sequence of characters.
- But in Java, string is an object that represents a sequence of characters.
- The java.lang.String class is used to create a string object.
- The java.lang.String class implements *Serializable*, *Comparable* and *CharSequence* interfaces.

How to create a string object?

There are two ways to create String object:

- By string literal
- By new keyword

String Literal

Java String literal is created by using double quotes. For Example:

```
String s="welcome";
```

Each time you create a string literal, the JVM checks the "string constant pool" first.

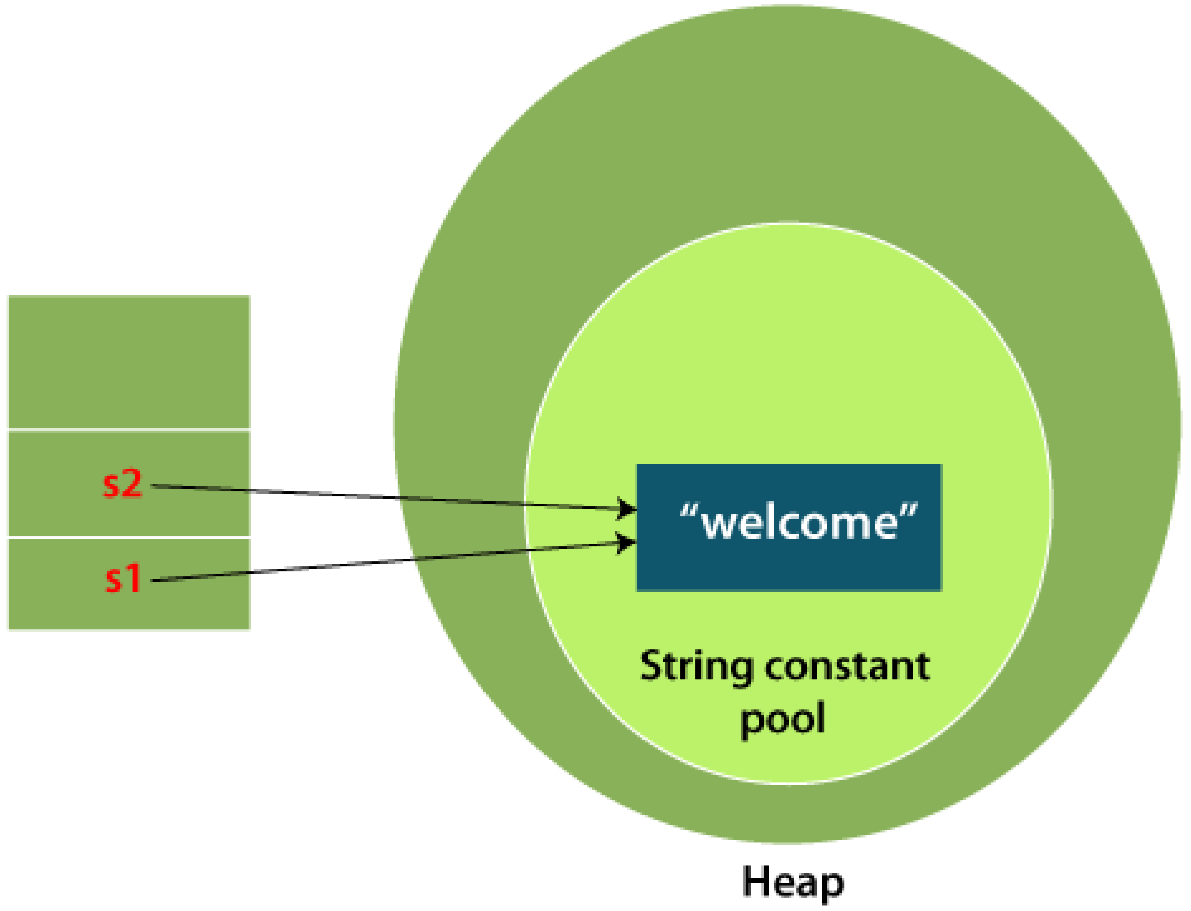
If the string already exists in the pool, a reference to the pooled instance is returned.

If the string doesn't exist in the pool, a new string instance is created and placed in the pool.

Example

String s1="Welcome";

String s2="Welcome";//It doesn't create a new i nstance



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 (non-pool) heap memory. The variable s will refer to the object in a heap (non-pool).

Example

```
public class StringExample{
public static void main(String args[]){
String s1="java";//creating string by Java string literal
char ch[]={'s','t','r','i','n','g','s'};
String s2=new String(ch);//converting char array to string
String s3=new String("example");//creating Java string by new ke
yword
System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
}}
```

Java String class methods

No.	Method	Description
1	char charAt(int index)	It returns char value for the particular index
2	int length()	It returns string length
3	static String format(String format, Object... args)	It returns a formatted string.
4	static String format(Locale l, String format, Object... args)	It returns formatted string with given locale.
5	String substring(int beginIndex)	It returns substring for given begin index.
6	String substring(int beginIndex, int endIndex)	It returns substring for given begin index and end index.
7	boolean contains(CharSequence s)	It returns true or false after matching the sequence of char value.
8	static String join(CharSequence delimiter, CharSequence... elements)	It returns a joined string.
9	static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)	It returns a joined string.
10	boolean equals(Object another)	It checks the equality of string with the given object.

Java String class methods

No.	Method	Description
1	char charAt(int index)	It returns char value for the particular index
2	int length()	It returns string length
3	static String format(String format, Object... args)	It returns a formatted string.
4	static String format(Locale l, String format, Object... args)	It returns formatted string with given locale.
5	String substring(int beginIndex)	It returns substring for given begin index.
6	String substring(int beginIndex, int endIndex)	It returns substring for given begin index and end index.
7	boolean contains(CharSequence s)	It returns true or false after matching the sequence of char value.
8	static String join(CharSequence delimiter, CharSequence... elements)	It returns a joined string.
9	static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)	It returns a joined string.
10	boolean equals(Object another)	It checks the equality of string with the given object.

Object Oriented Programming Concepts

MS. J J DESAI

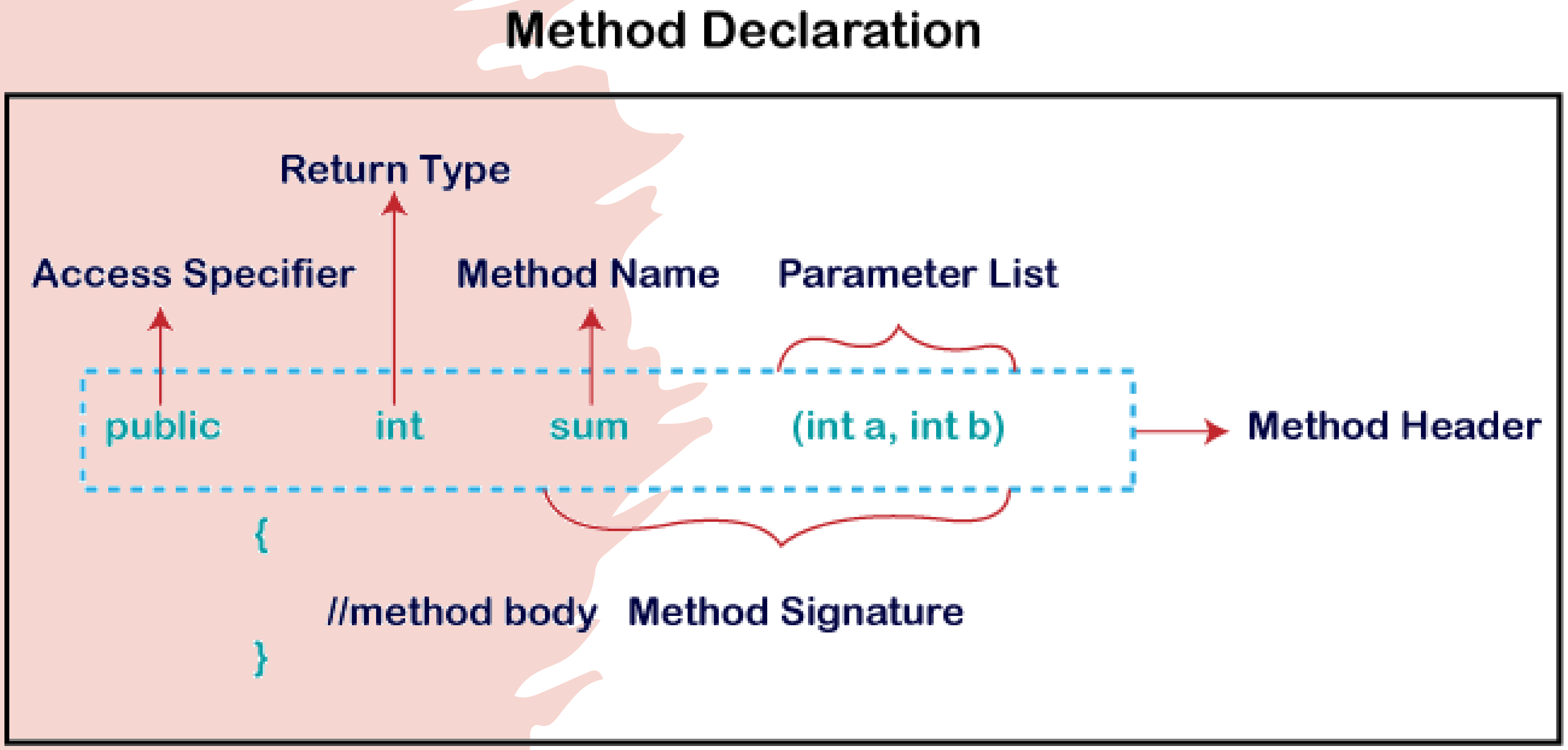
Method

The **method in Java** is a collection of instructions that performs a specific task.

It provides the reusability of code.

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

Method Declaration



Access Specifier

- Access specifier or modifier is the access type of the method.
- It specifies the visibility of the method.
- Java provides **four** types of access specifier:
  - o **Public:** The method is accessible by all classes when we use public specifier in our application.
  - o **Private:** When we use a private access specifier, the method is accessible only in the classes in which it is defined.
  - o **Protected:** When we use protected access specifier, the method is accessible within the same package and subclasses.
  - o **Default:** When we do not use any access specifier in the method declaration, Java uses default access specifier by default. It is visible only from the same package only.

Types of Method

Predefined Method: In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods. It is also known as the **standard library method** or **built-in method**.

User-defined Method: The method written by the user or programmer is known as **a user-defined** method. These methods are modified according to the requirement.

Static Method

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

Example

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

Instance Method

The method of the class is known as an **instance method**.

It is a **non-static** method defined in the class.

Before calling or invoking the instance method, it is necessary to create an object of its class.

Example

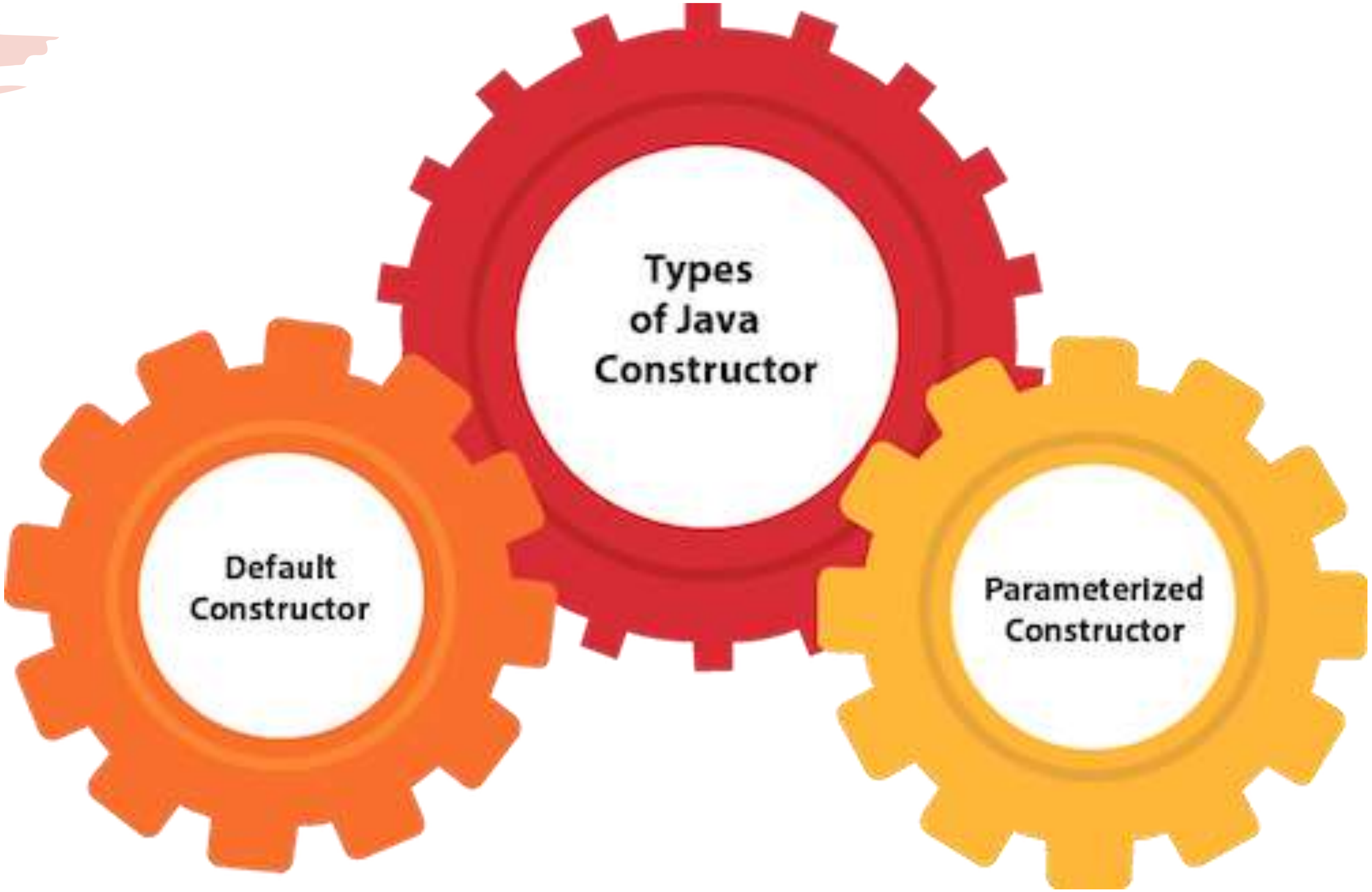
```
public class InstanceMethodExample
{
public static void main(String [] args)
{
//Creating an object of the class
InstanceMethodExample obj = new InstanceMethodExam
ple();
//invoking instance method
System.out.println("The sum is: "+obj.add(12, 13));
}
int s;
//user-
defined method because we have not used static keywor
d
public int add(int a, int b)
{
s = a+b;
//returning the sum
return s;
}
}
```

Constructors in Java

- In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.
- It is a special type of method which is used to initialize the object.
- Every time an object is created using the new() keyword, at least one constructor is called.
- It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

Rules for creating Java constructor

- Constructor name must be the same as its class name
- A Constructor must have no explicit return type
- A Java constructor cannot be abstract, static, final, and synchronized


Types of Java Constructor
Default Constructor
Parameterized Constructor

Example of default constructor

```
class Bike1{
//creating a default constructor
Bike1()
{
   System.out.println("Bike is created");
}
//main method
public static void main(String args[])
{
//calling a default constructor
   Bike1 b=new Bike1();
}
}
```

Example of parameterized constructor

```
class Student4{
    int id;
    String name;
        Student4(int i,String n){
    id = i;
    name = n;
    }
     void display()System.out.println(id+" "+name);}
       public static void main(String args[]){
    //creating objects and passing values
    Student4 s1 = new Student4(111,"Karan");
    Student4 s2 = new Student4(222,"Aryan");
    //calling method to display the values of object
    s1.display();
    s2.display();
  }
}
```

Method Overloading in Java

If a class has multiple methods having same name but different in 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 behavior of the method because its name differs.

Different ways to overload the method

By changing number of arguments

By changing the data type

Method Overloading: changing no. of arguments

```
class Adder{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+
c;}
}
class TestOverloading1{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));

}}
```

Method Overloading: changing data type of arguments

```
class Adder{
static int add(int a, int b){return a+b;}
static double add(double a, double b){r
eturn a+b;}
}
class TestOverloading2{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));

}}
```

Constructor Overloading in Java

```
class Student5{
   int id;
   String name;
   int age;
   //creating two arg constructor
   Student5(int i,String n){
   id = i;
   name = n;
   }
   //creating three arg constructor
   Student5(int i,String n,int a){
   id = i;
   name = n;
   age=a;
   }
Void display()
{System.out.println(id+" "+name+" "+age);}

   public static void main(String args[]){
   Student5 s1 = new Student5(111,"Karan");
   Student5 s2 = new Student5(222,"Aryan",25);
   s1.display();
   s2.display();
  }
}
```

Difference between constructor and method in Java

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

Java Copy Constructor

- There is no copy constructor in Java. However, we can copy the values from one object to another like copy constructor in C++.
- There are many ways to copy the values of one object into another in Java. They are:
  - By constructor
  - By assigning the values of one object into another

By constructor

```
class Student6{
    int id;
    String name;
    //constructor to initialize integer and string
    Student6(int i,String n){
    id = i;
    name = n;
    }
    //constructor to initialize another object
    Student6(Student6 s){
    id = s.id;
    name =s.name;
    }
    void display(){System.out.println(id+" "+nam
e);}

    public static void main(String args[]){
    Student6 s1 = new Student6(111,"Karan");
    Student6 s2 = new Student6(s1);
    s1.display();
    s2.display();
   }
}
```

Copying values without constructor

```
class Student7{
    int id;
    String name;
    Student7(int i,String n){
    id = i;
    name = n;
    }
    Student7(){}
    void display()
{System.out.println(id+" "+name);}
```

```
public static void main(String args
[]){
    Student7 s1 = new Student7(11
1,"Karan");
    Student7 s2 = new Student7();
    s2.id=s1.id;
    s2.name=s1.name;
    s1.display();
    s2.display();
  } }
```

What is a private constructor?

- Java allows us to declare a constructor as private.
- We can declare a constructor private by using the private access specifier.
- Note that if a constructor is declared private, we are not able to create an object of the class.
- Instead, we can use this private constructor in Singleton Design Pattern.

Thank you.

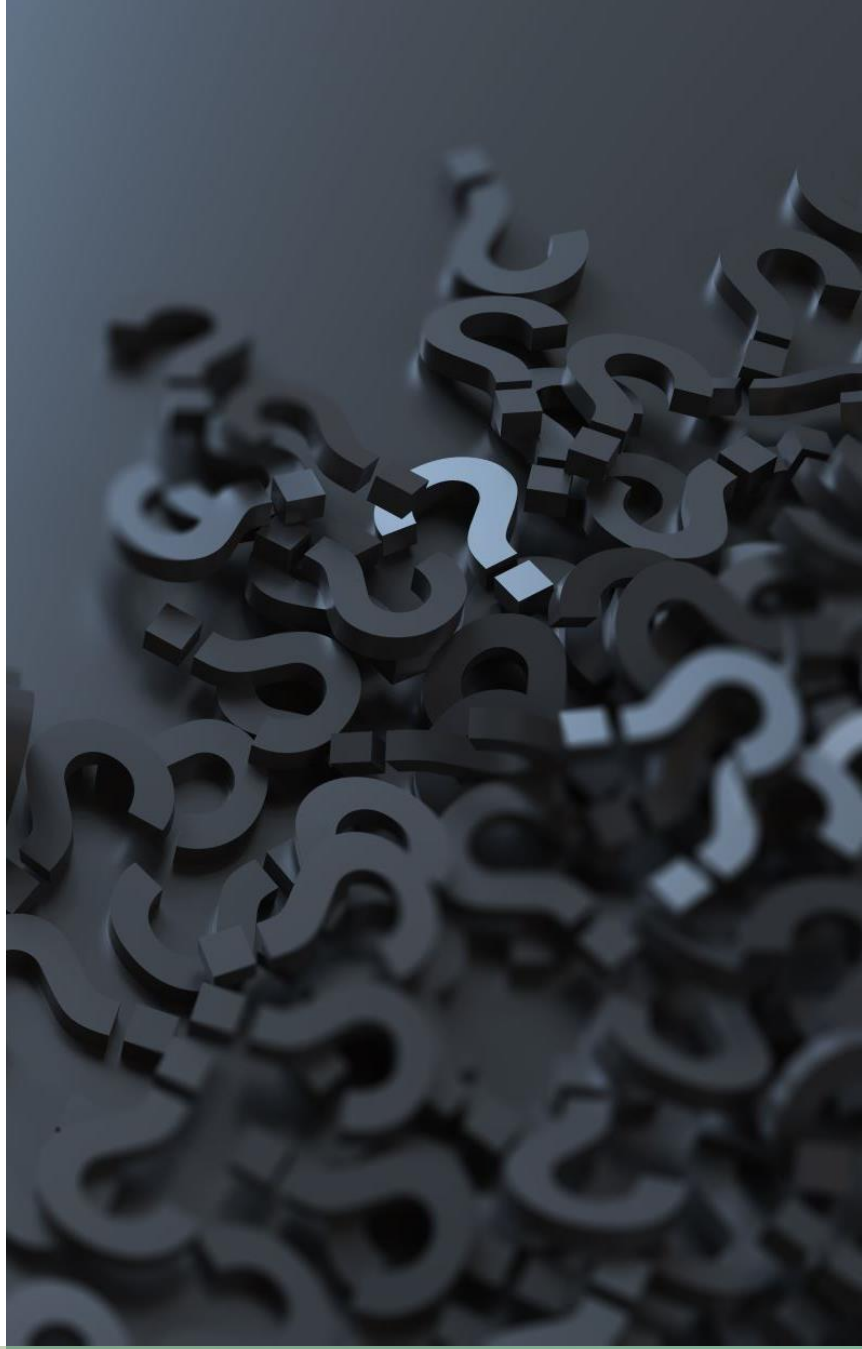
FINAL KEYWORD IN JAVA

INTRODUCTION

- Use of final keyword in JAVA is similar to use of "const" in C++
- In JAVA final keyword can be applied to
  - variable
  - methods
  - class
- Once you declare anything as final in JAVA it cannot be modified further in JAVA.

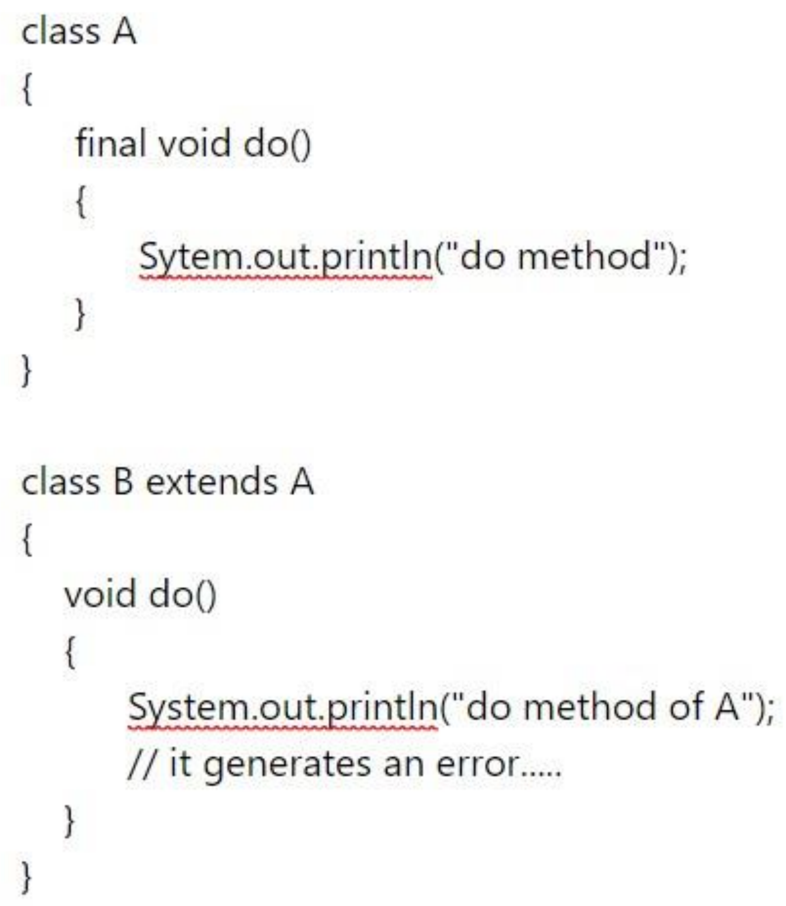
FINAL AT VARIABLE LEVEL

- it is used to make constant in JAVA.
- once you make variable as final you can not change its value.
- you must initialize final variable.
- final variable that does not have any value is called **blank final.**
- A blank final variable can be initialized inside instance-initializer block or inside constructor.
- Example
    - final double PI =3.14;



FINAL AT METHOD LEVEL

- final method can not be override.

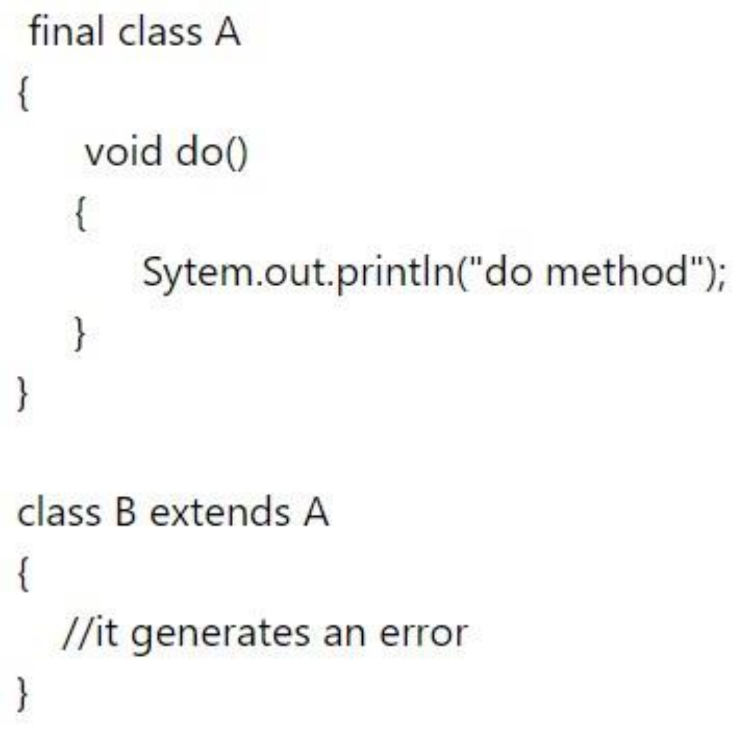


```
class A
{
     final void do()
     {
          Sytem.out.println("do method");
     }
}

class B extends A
{
    void do()
    {
         System.out.println("do method of A");
         // it generates an error.....
    }
}
```

FINAL AT CLASS LEVEL

- final class can not be inherited by other classes.



```
final class A
{
     void do()
    {
         Sytem.out.println("do method");
    }
}

class B extends A
{
   //it generates an error
}
```

static

- Static keyword is applicable to variables, methods and block.
- It is not compulsory to initialize the static variable at the time of its declaration.
- The static variable can be reinitialized.
- Static methods can only access the static members of the class, and can only be called by other static methods.

final

- Final keyword is applicable to class, methods and variables.
- It is compulsory to initialize the final variable.
- The final variable can not be reinitialized.
- Final methods can not be override.

STATIC KEYWORD

Introduction

Static keyword is mainly used for memory management in JAVA.

It can be used with

- variables
- methods
- blocks

It is used to share the same variable and method of a given class.

No objects needed to be created to use static variables or call static methods.

When?

There will be time when you want to define a class member that will be used independently of any object of that class.

To create such member just precede its declaration with the keyword **static**

How?

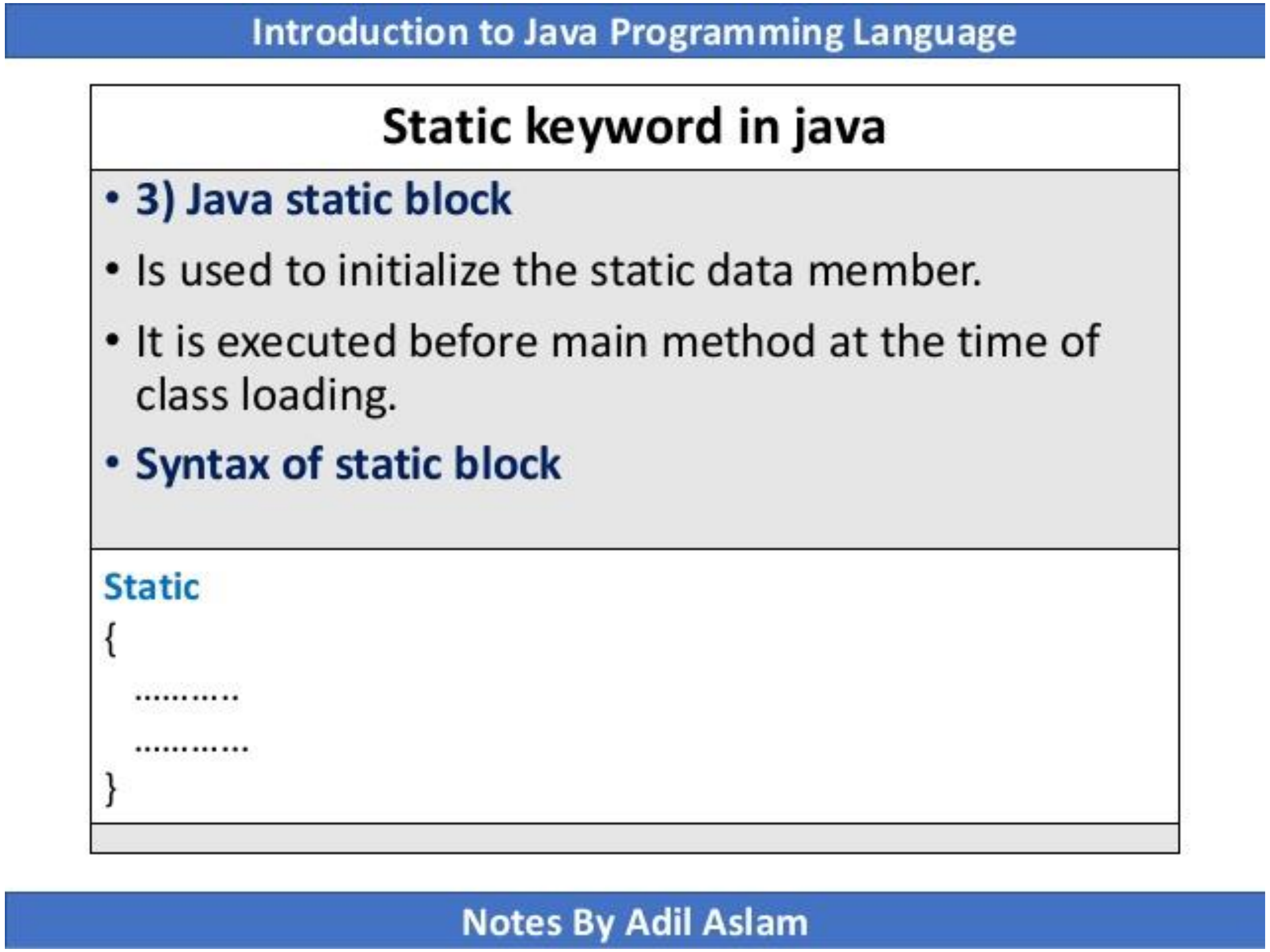
Block

VARIABLES

METHODS

Blocks (static blocks)

if you need to do computation in order to initialize your **static variables** you can declare a static block that gets executed exactly once when class is first time loaded.



Introduction to Java Programming Language

Static keyword in java

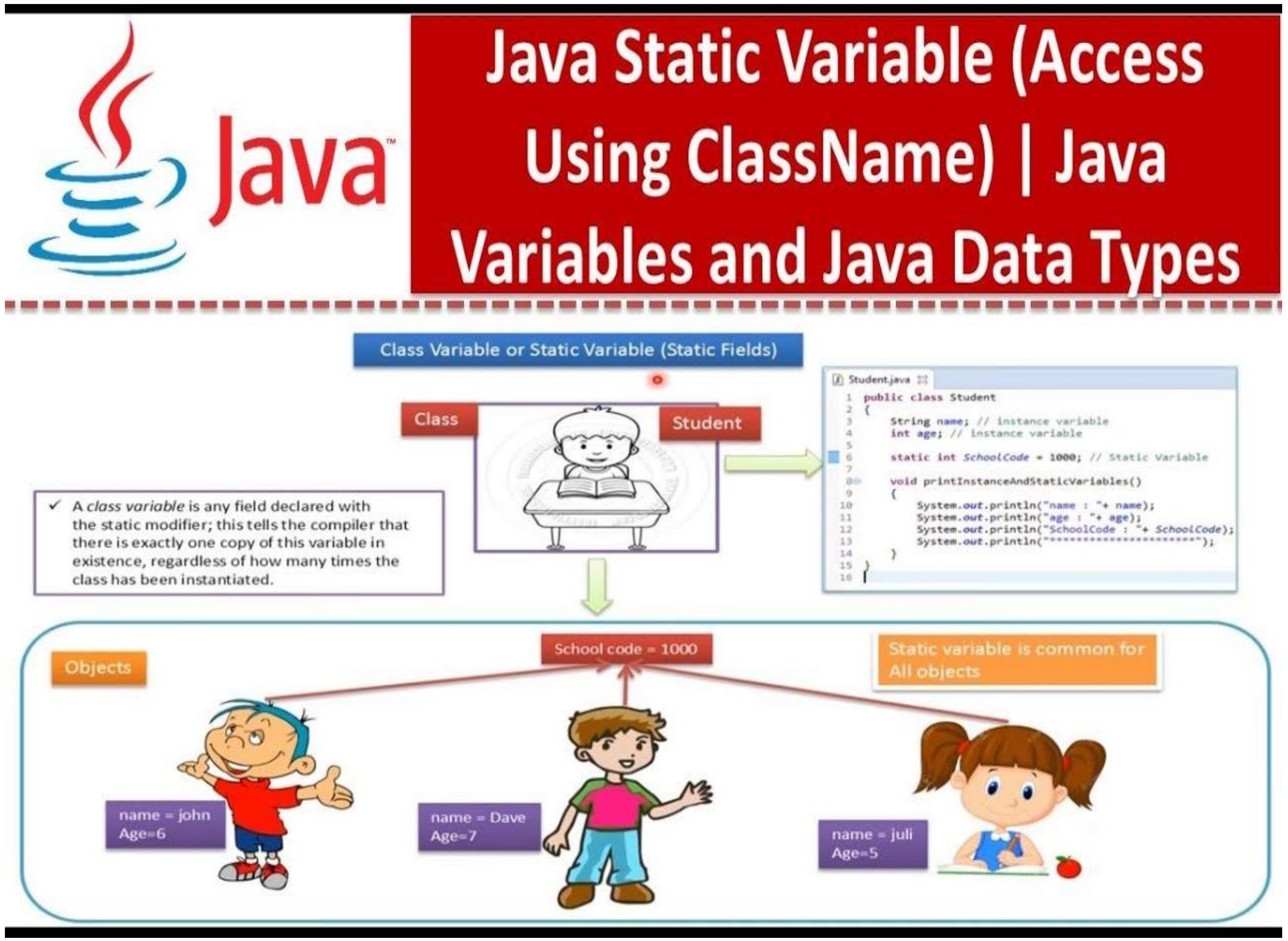
- **3) Java static block**
- Is used to initialize the static data member.
- It is executed before main method at the time of class loading.
- **Syntax of static block**

```
Static
{
   …………
   ………….
}
```

Notes By Adil Aslam

Variable (class variable)

- when a variable is declared as static then only single copy of variable is created and shared among all objects at class level.
- They are actually global variables.
- we can create static variables at class-level only.
- static variable allocate memory only once in **class area at** the time of **class loading.**


Java
Java Static Variable (Access Using ClassName) | Java Variables and Java Data Types
Class Variable or Static Variable (Static Fields)
Class
Student
Objects
School code = 1000
Static variable is common for All objects
name = john
Age=6
name = Dave
Age=7
name = juli
Age=5

✓ A *class variable* is any field declared with the static modifier; this tells the compiler that there is exactly one copy of this variable in existence, regardless of how many times the class has been instantiated.

Student.java

```
public class Student
{
    String name; // instance variable
    int age; // instance variable

    static int SchoolCode = 1000; // Static Variable

    void printInstanceAndStaticVariables()
    {
        System.out.println("name : "+ name);
        System.out.println("age : "+ age);
        System.out.println("SchoolCode : "+ SchoolCode);
        System.out.println("**********************");
    }
}
```

INSTANCE VARIABLE

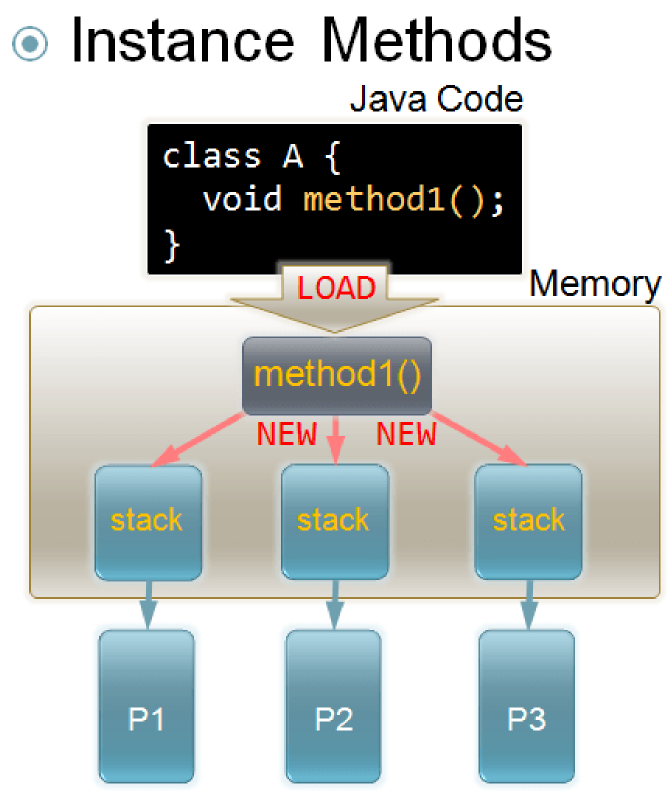
- Variables that the object contains are called instance variables.
- instance variable have different copies for all objects of the class.
- they are specific to instance of the class.
- Each object has its own copy of instance variables of the class.
- so changes made to the instance variable of one object have no effect on the instance variable of another.

CLASS VARIABLE

- they are global.
- they can be used by all instances of the class.
- they are used for
- keeping track of global state
- communicating between different objects of same class.
- static keyword is used to declare class variable.
- we can access class variable through instance or through class itself.
- When a variable is declared as static, then a single copy of variable is created and shared among all objects at class level.

Method (class method , static method)

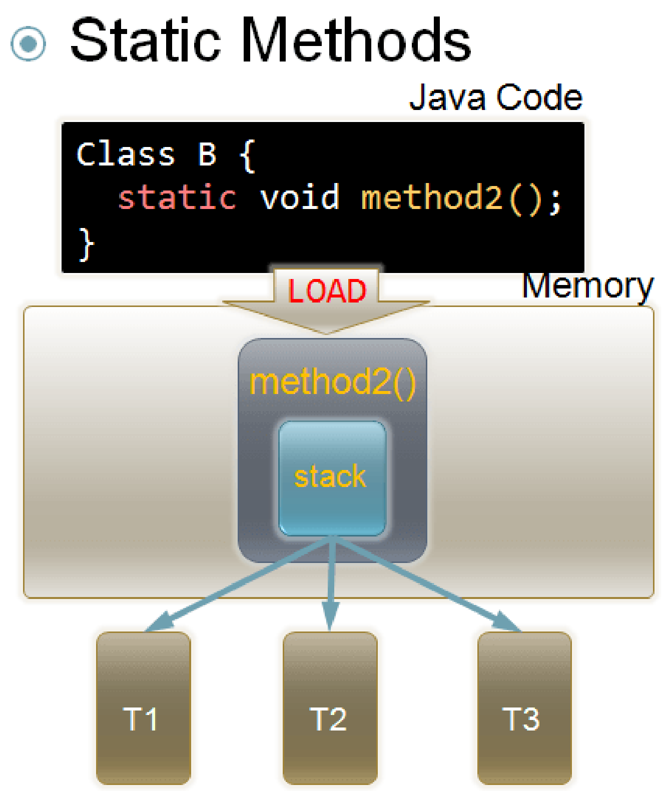
- when a method is declared as static it belongs to class.
- static method can only access static data members of same class or other class.
- for static method memory allocation happens only once during **loading of class.**



Instance Methods

Java Code

```
class A {
  void method1();
}
```



Static Methods

Java Code

```
Class B {
  static void method2();
}
```

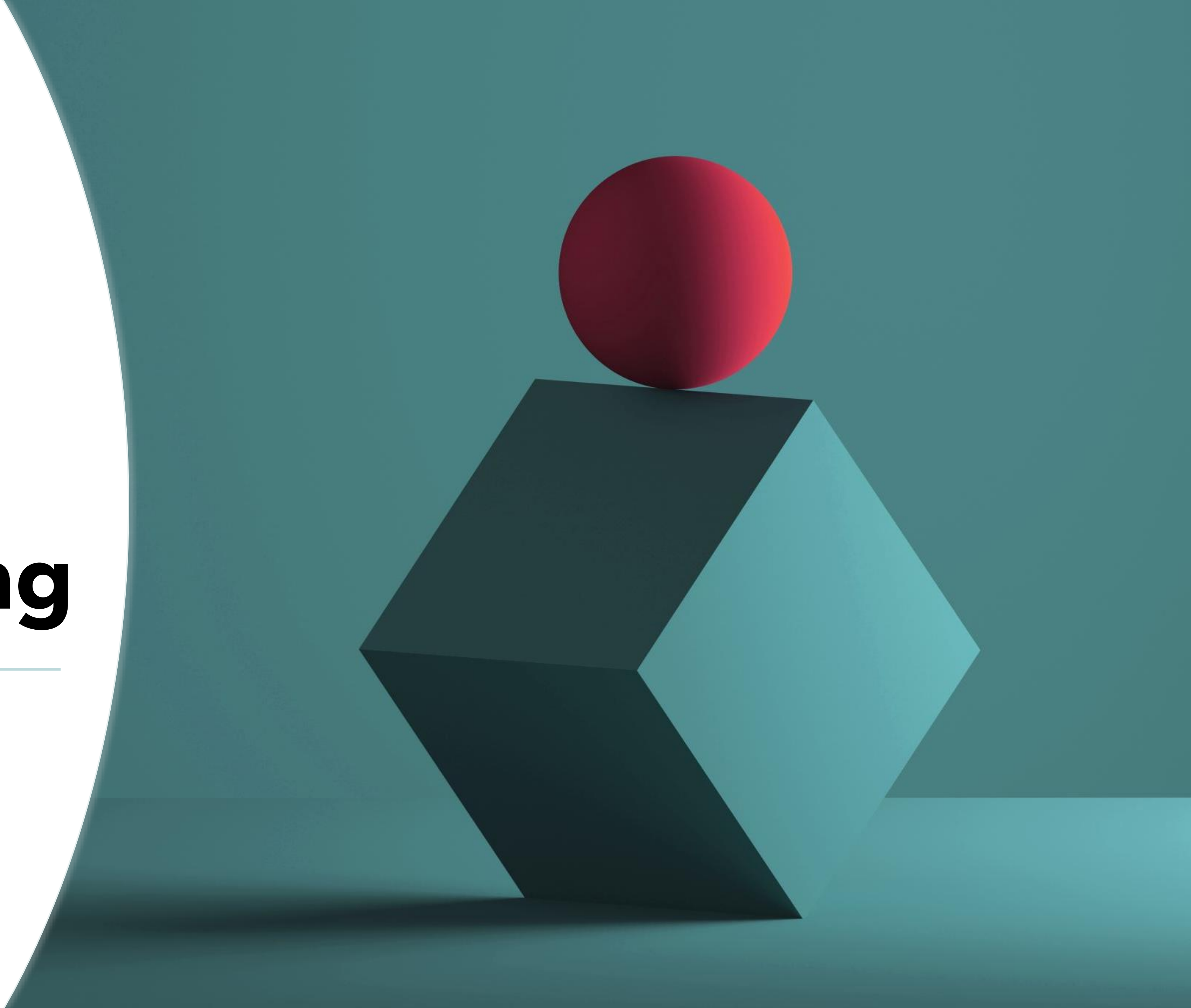
INSTANCE METHOD	STATIC METHOD (CLASS METHOD)
Method that is invoked with respect to an instance of a class.	Method that is invoked without reference to a particular object.
Instance method affect particular instance of class	Class Method affect the class as a whole.
Instance method operates on instance variable of specific object.	Static method does not use instance variables of any object.
by default all method declared in class are instance method.	static keyword is used to declare method as a static
Example ◦ void display() { }	**Example** ◦ static void display(){ }

Note:

static data members can be accessed without creating an instance of class.

we can access static members of class using class name in which they are define along with dot operator

Object Oriented Programming



ABSTRACTION

it is defined as the process of specifying only the required characteristics of an object ignoring irrelevant details.

it separates **specification** from implementation of the class.

specification defines what services are provided through the interface.

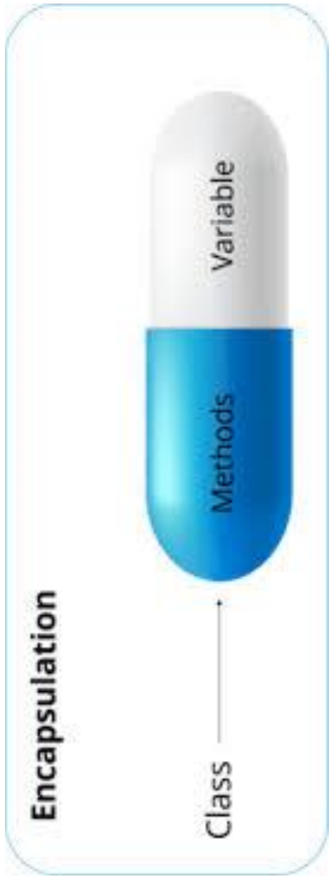
Example

Car as an object. interface of car is clutch , break , accelerator

we know what happens if we press paddle of break or accelerator but we do not know its internal implementation. which is abstracted from driver.

it is the process of representing simplified version of real world objects in your classes and objects.

Encapsulation



Encapsulation is the process of wrapping of data and code in a single unit to keep them safe from outside interference and misuse.

It a protective wrapper that prevents the code and data from being arbitrarily accessed by other code defined outside the wrapper.

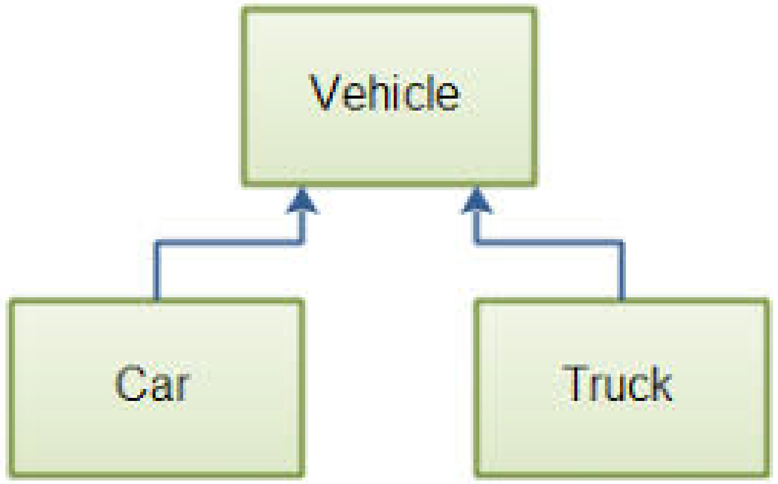
In java encapsulation can be achieved using **class** concept.

A class defines the structure and behavior that will be shared by set of **objects**.

Objects are sometime referred to as an instance of a class.

class is a logical concept and object is physical reality.

The main advantage of encapsulation is to secure data and methods from outside the class by using different access specifiers(**Private**).



INHERITANCE

- Process by which one object can acquires the properties of another object.
- it supports the concept of hierarchical classification.
- Using inheritance object need only to define those qualities that makes it unique within its class and it can inherit general characteristics from its parents.
- Main advantage of inheritance is code re usability.

POLYMORPHISM

- meaning many forms
- it is a feature that allows one interface to be used for a general class of actions.
- specific action is determined by exact nature of situation.
- it is an ability of an object to change its behavior according to how it is being used.

Polymorphism can be achieved in java by two ways

Compile-time polymorphism

- Method to be invoked is determined at the compile time.
- Method overloading-multiple methods having same name but different signature.

Runtime polymorphism

- Method to be invoked is determined at runtime.
- Method overriding-subclass contain a method with same name and signature as in the super class.

CLASS

- Abstract datatype
- it is set of data and code which acts on data encapsulated as a single unit with an identifiable name.
- **Super class**
    - class whose properties are inherited .
- **Subclass**
    - class which inherits the properties of the other class.

POP Vs OOP

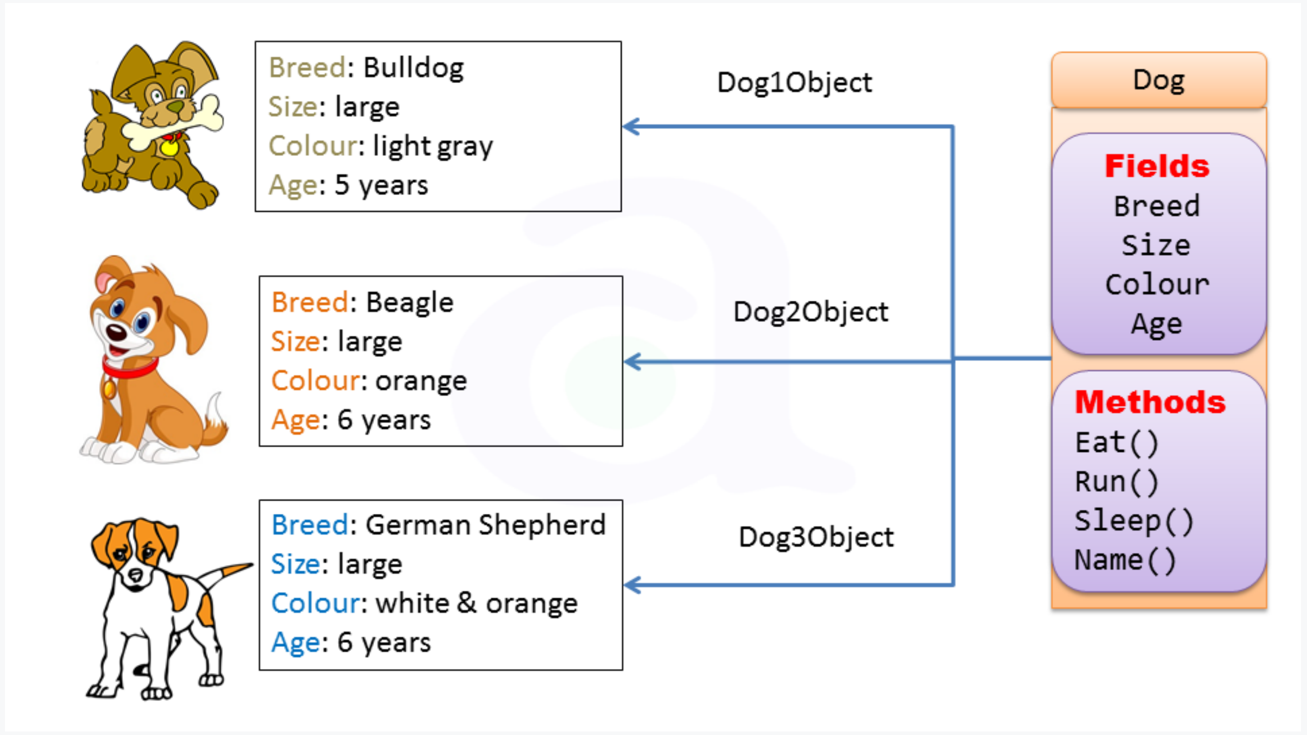
Procedure Oriented Programming	Object Oriented Programming
In POP, program is divided into small parts called functions.	In OOP, program is divided into parts called Objects.
In POP, importance is given to Functions	In OOP, importance is given to data rather than procedures.
Top down Approach	Bottom Up approach
POP does not have any access specifier	OOP has access specifiers : public , protected, private
To add new data and functions in POP is not easy.	OOP provides an easy way to add new data and functions.
POP is less secure	OOP is more secure with the concepts of Data Hiding
Overloading is not possible	Function and Operator overloading is possible
In POP, data can move freely from function to function in the system	In OOP , object can communicate with each other using member functions.
C,FORTRAN	C++,JAVA, VB.NET

C++ Vs Java

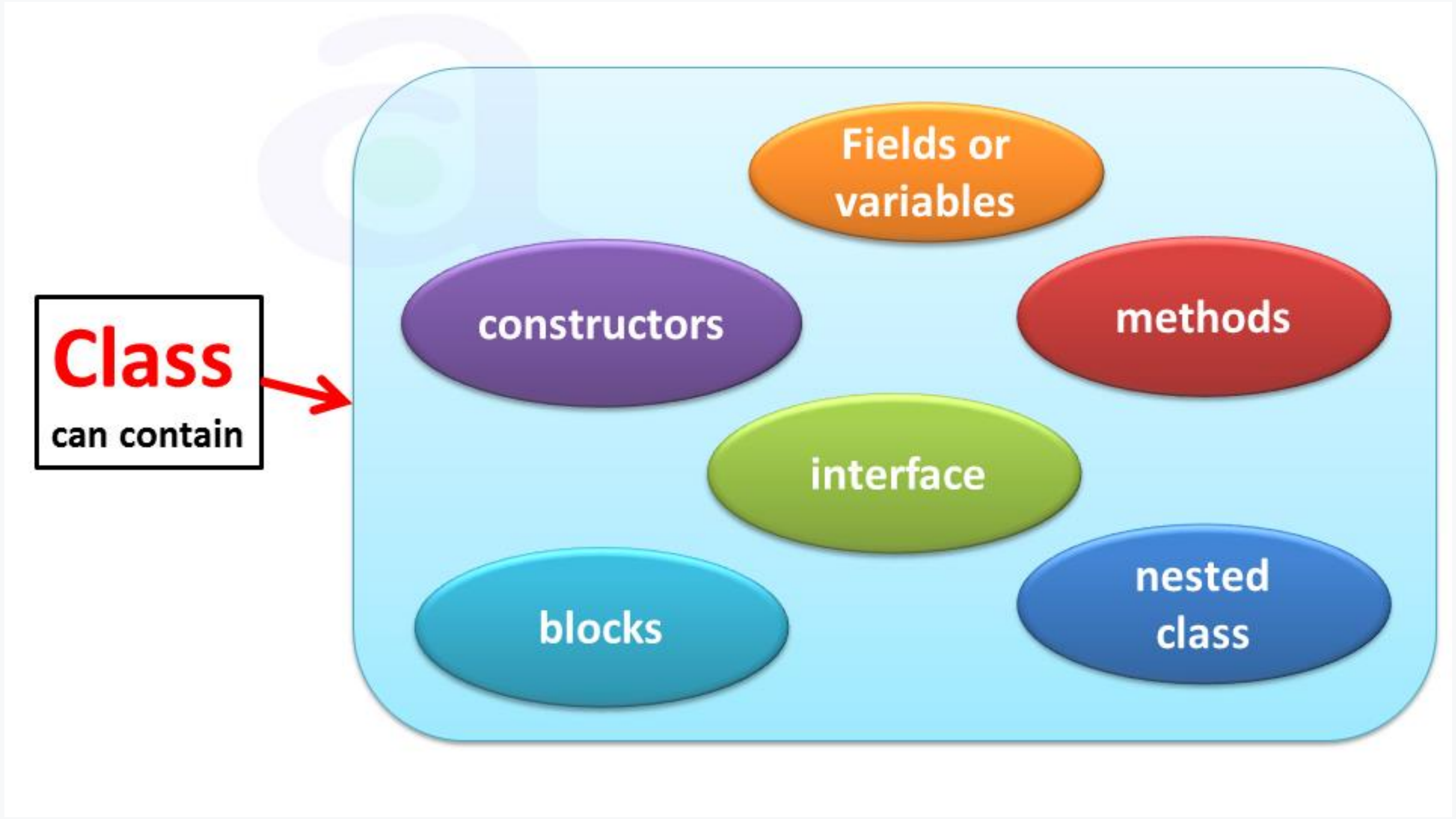
C++	Java
C++ is platform-dependent.	Java is platform-independent.
C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in window, web-based, enterprise and mobile applications.
C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by interfaces in java.
C++ supports operator overloading.	Java doesn't support operator overloading.
C++ doesn't have built-in support for threads.	Java has built-in thread support.
C++ supports pointers. You can write pointer program in C++.	Java has restricted pointer support in java programming.

FUNDAMENTALS OF CLASS

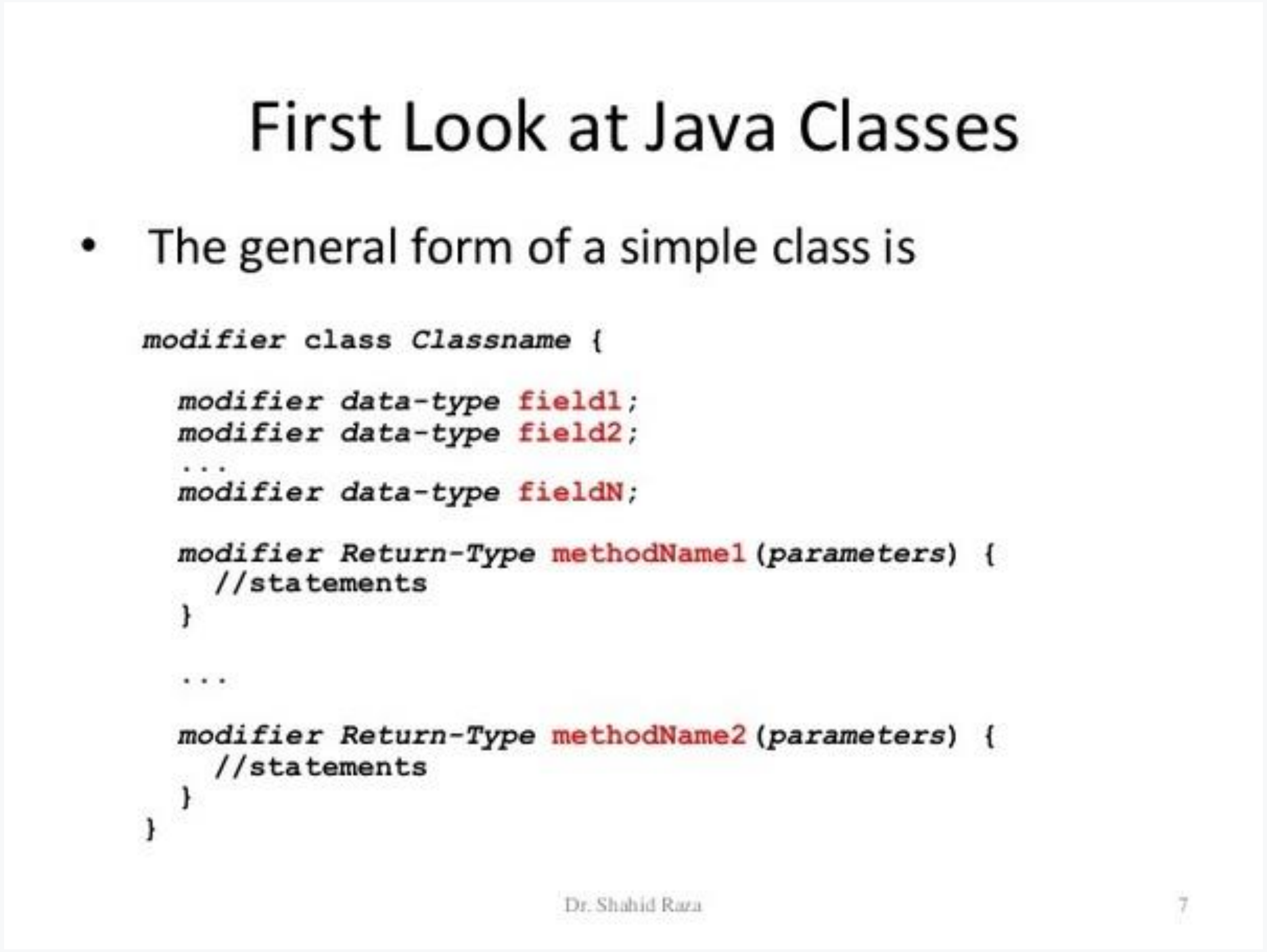
- The class is at the core of JAVA.
- It is used to define new data type.
- Class is user defined blueprint from which objects can be created.
- It represents the set of properties or methods that are common to all objects of same type.



Class Declaration includes



- Variables and methods defined within a class are called **members** of the class.
- Variables defined within a class are called instance variables because each instance of the class contains its own copy of these variables



First Look at Java Classes

- The general form of a simple class is

```
modifier class Classname {

  modifier data-type field1;
  modifier data-type field2;
  ...
  modifier data-type fieldN;

  modifier Return-Type methodName1(parameters) {
    //statements
  }

  ...

  modifier Return-Type methodName2(parameters) {
    //statements
  }
}
```

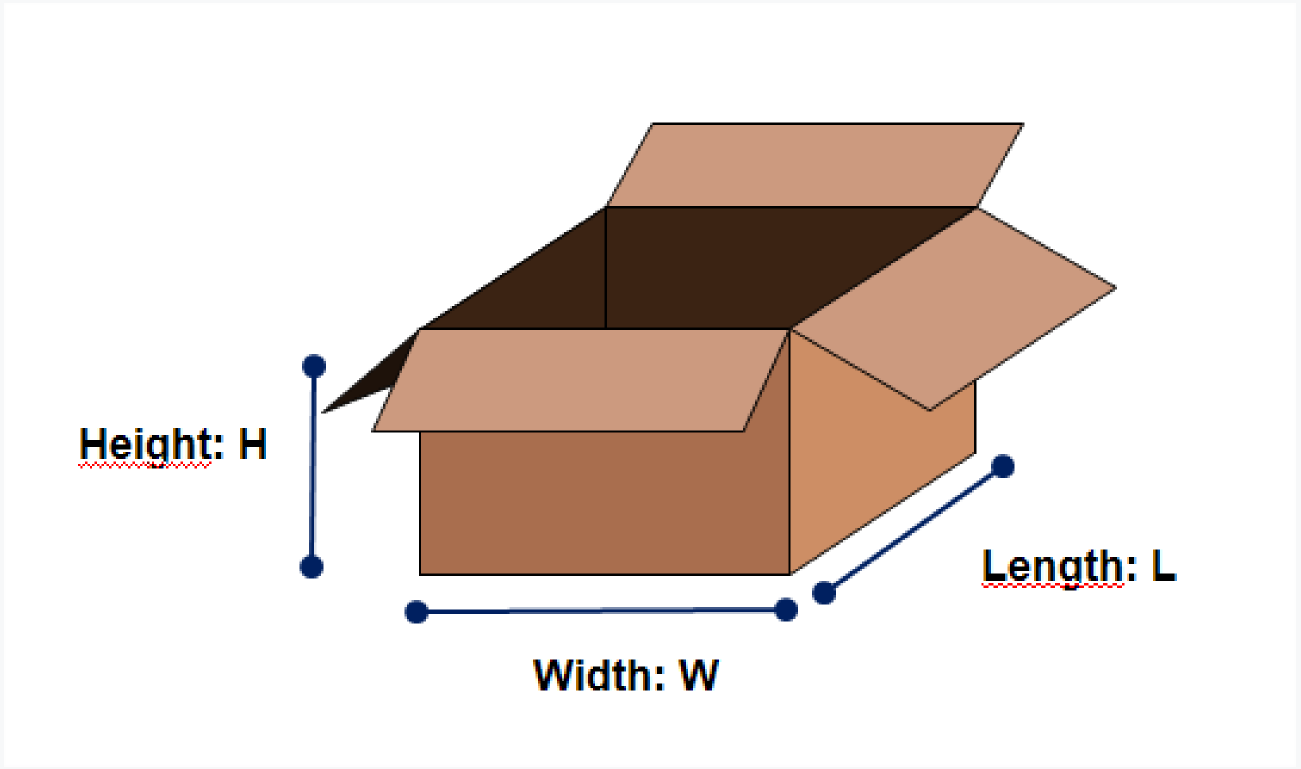
Dr. Shahid Raza 7

Example

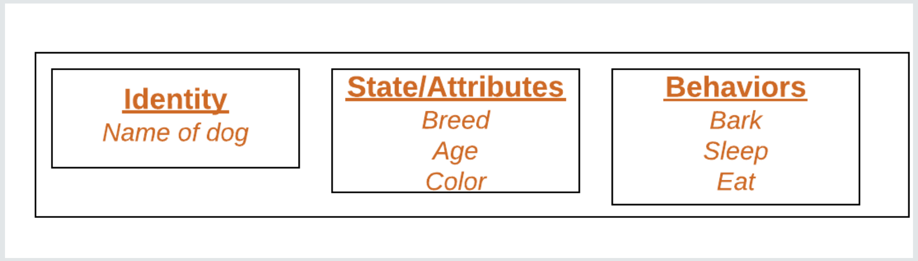
```
class Box
{
      //Instance variables
      double width;
      double height;
      double length;

     //member methods

       double volumeBox()
       {
             double vol = width * height
* length;
             return vol;
       }
}
```



Object



- it is an instance of a class.
- it is used to model/represent real world objects we finds in everyday life.
- An object in java has following characteristics.
  - State
  - Behavior
  - Identity
- **State**
  - *it represents attributes and properties of an object.*
- **Behavior**
  - it is represented by methods(functionality) of an object which operates on data.
- **Identity**
  - Unique **name/ID to an object.**
  - this id is not visible to the external user.
  - it is internally used by the JVM to identify each object uniquely.

Creating Object

- **General Syntax**
- class_name ref_var_name;
- ref_var_name = new class_name();
  - OR
- class_name ref_var_name = new class_name();

- Box b1 = new Box();
- **new** operator dynamically allocate memory for an object at runtime
- JVM automatically supply default constructor to create object.
- b1 is a reference to an object of type Box.

Access Object Data

- dot operator is used to access the data of objects through reference variable.
- **Syntax**
- ✓ **<object_refrence>.<variablename>**
- **Example**
- b1.width
- b1.height
- b1.length

- you can create more than one object of class.
- Any changes to data of one object do not have any effect on data of another object.

Attributes of class

These are properties that determines the state ,appearance of an object.

INSTANCE VARIABLE

- Variables that the object contains are called instance variables.
- instance variable have different copies for all objects of the class.
- they are specific to instance of the class.
- Each object has its own copy of instance variables of the class.
- so changes made to the instance variable of one object have no effect on the instance variable of another.

CLASS VARIABLE

- they are global.
- they can be used by all instances of the class.
- they are used for
- keeping track of global state
- communicating between different objects of same class.
- static keyword is used to declare class variable.
- we can access class variable through instance or through class itself.
- When a variable is declared as static, then a single copy of variable is created and shared among all objects at class level.

STRING IN JAVA

STRING

- It is a sequence of character enclosed within double quotes
- In Java every strings you create is consider as an object of type String Class
- System.out.println("Hello world");
- here, "Hello world" is a String object.
- ✓ Object of type String are **immutable**.

Once String object is created its content cannot be altered.

Basic aim of String Handling is to store string data into memory, manipulating the string data and retrieve part of string etc.

String handling provides lots of concepts that can be performed on String such String Concatenation , Comparison and find substring etc.

In C/C++ String is simply an array of characters but in JAVA String is a class encapsulated under **java.lang** Package.

String objects are stored in special memory area know as **String Constant Pool** inside the Heap memory.

Different Ways to Create String

USING **STRING LITERALS**

USING **NEW KEYWORD**

Using **String Literals**

String s = "Hello";

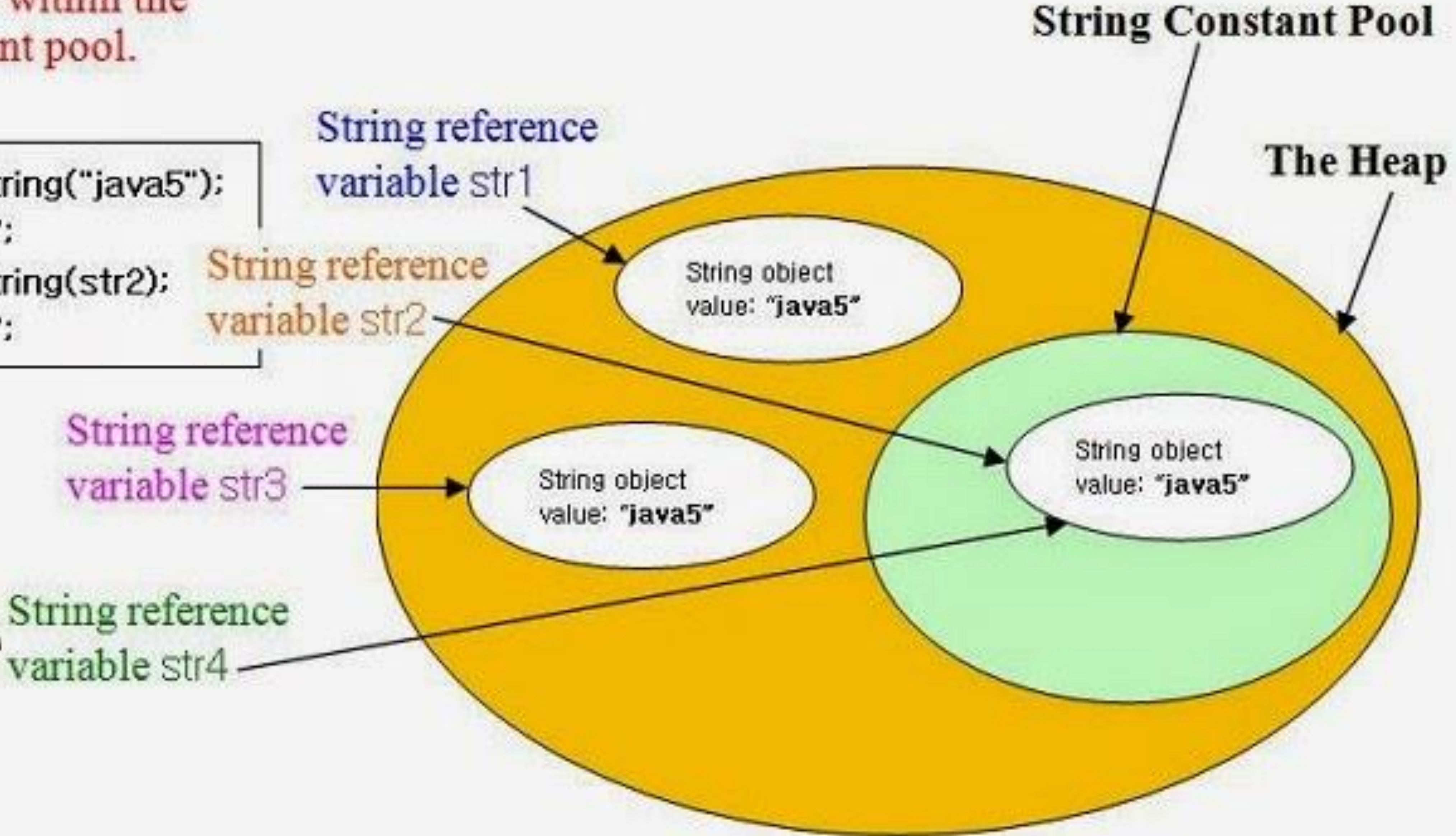
each time you create a String literal ,the JVM checks the string constant pool in memory first ,if string already exist in memory pool the only reference to the pooled instance is return.

if String doesnot exist in the pool then new String object is created and placed in the pool.

4

The str4 reference will be pointed to the existing object with the value “java5 within the string constant pool.

```
String str1 = new String("java5");
String str2 = "java5";
String str3 = new String(str2);
String str4 = "java5";
```


String Constant Pool
The Heap
String reference variable str1
String reference variable str2
String reference variable str3
String reference variable str4
String object value: "java5"
String object value: "java5"
String object value: "java5"

Using **new Keyword**

```
String s = new String("Welcome");
```

in above case JVM will create new String Object in normal (non-pool) heap memory and variable **s** will refer to that object in memory.

Length of a String

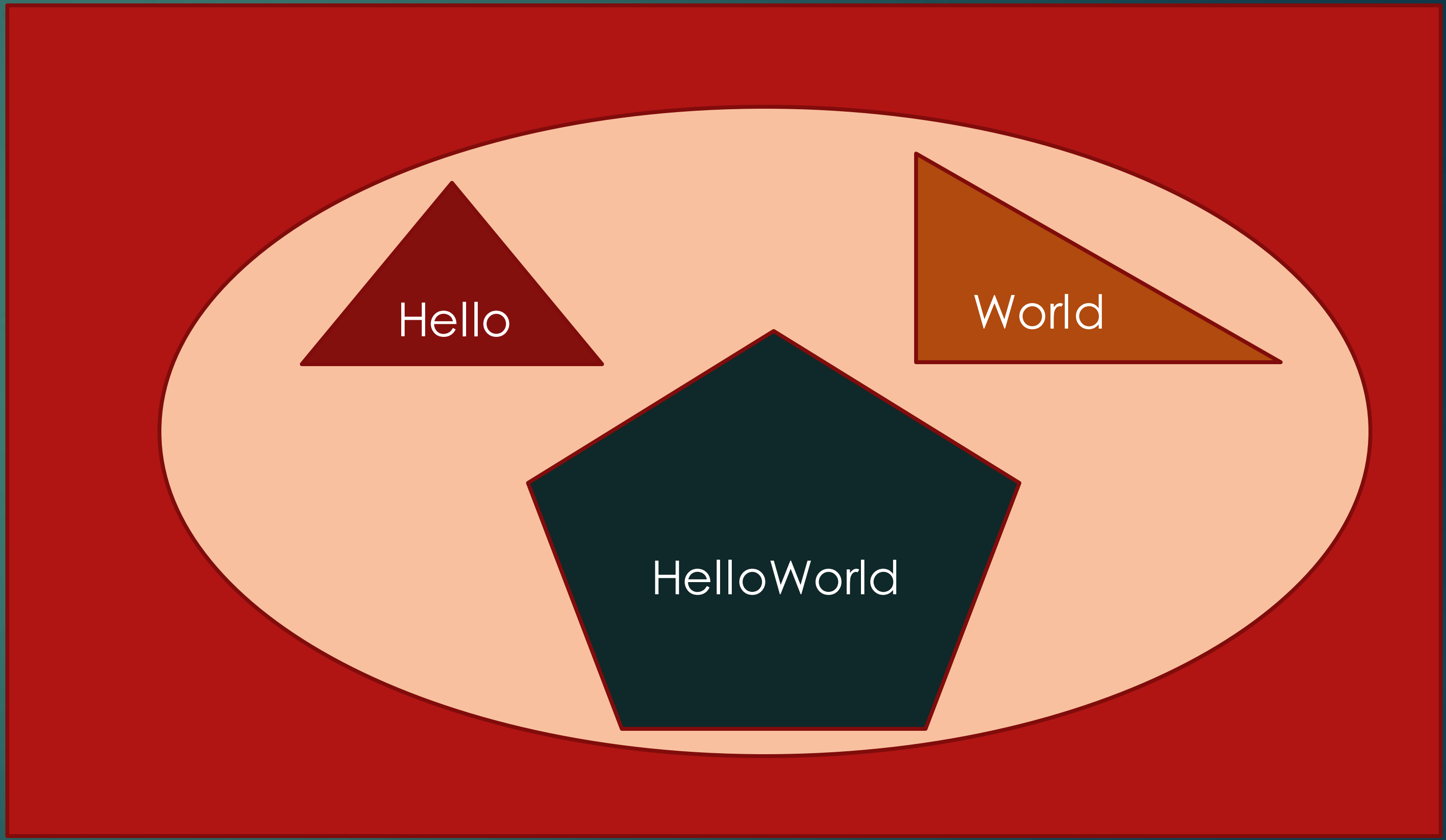
- As String is class in Java you can find out length of a String using a function called **length()** in String class.
- **Example**
  - **char a[] ={'w','e','l','l','c','o','m','e'};**
  - **String s = new String(a);**
  - **System.out.println("length is" + s.length());**

String Concatenation

- Concatenated String forms a new String which is combination of multiple Strings.
- There are two ways to concat Strings in JAVA
  - Using + (string concatenation) operator
  - using concat() method.

Using + String concatenation operator

- String s ="Hello" + "World";
- System.out.println(s);



Note: String concatenation operator can concatenate not only String types but primitive values also.

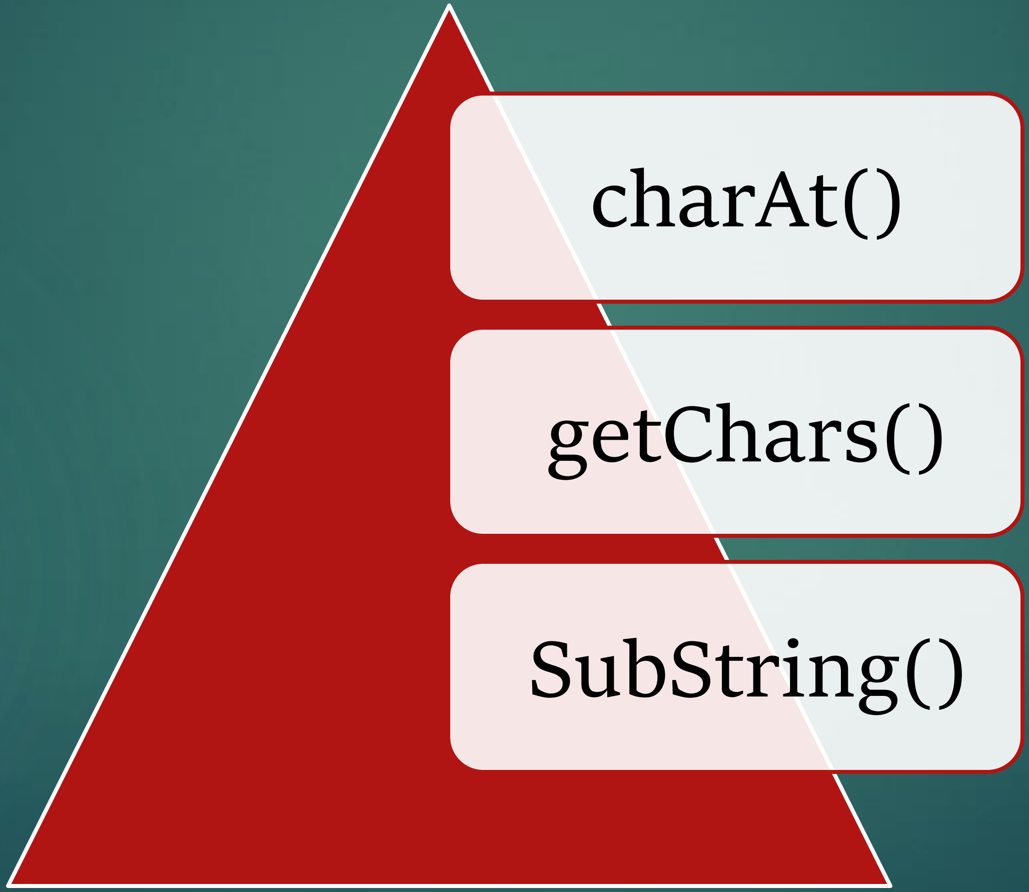
Using Concat() Method

- **Syntax**
  - public String concat(String s);
- this method concatenates the specified string at the end of current string.
- **Example**
  - String s1 ="Hello";
  - String s2 =s1.concat("world");
  - System.out.println(s2);

Immutable String

- it means un-modifiable or unchangeable.
- Once String object is created its data or state cannot be changed but a new String is created.
- **Example**
  - String s = "Indian";
  - s.concat("Cricketer");
  - System.out.println(s);

Character Extraction



charAt()

Syntax:

- **char charAt(int where)**
- where is the index of character that you want to extract.
- where must be non negative value and specify location within String.

Example

- String s ="Hello";
- char ch = s.charAt(1);

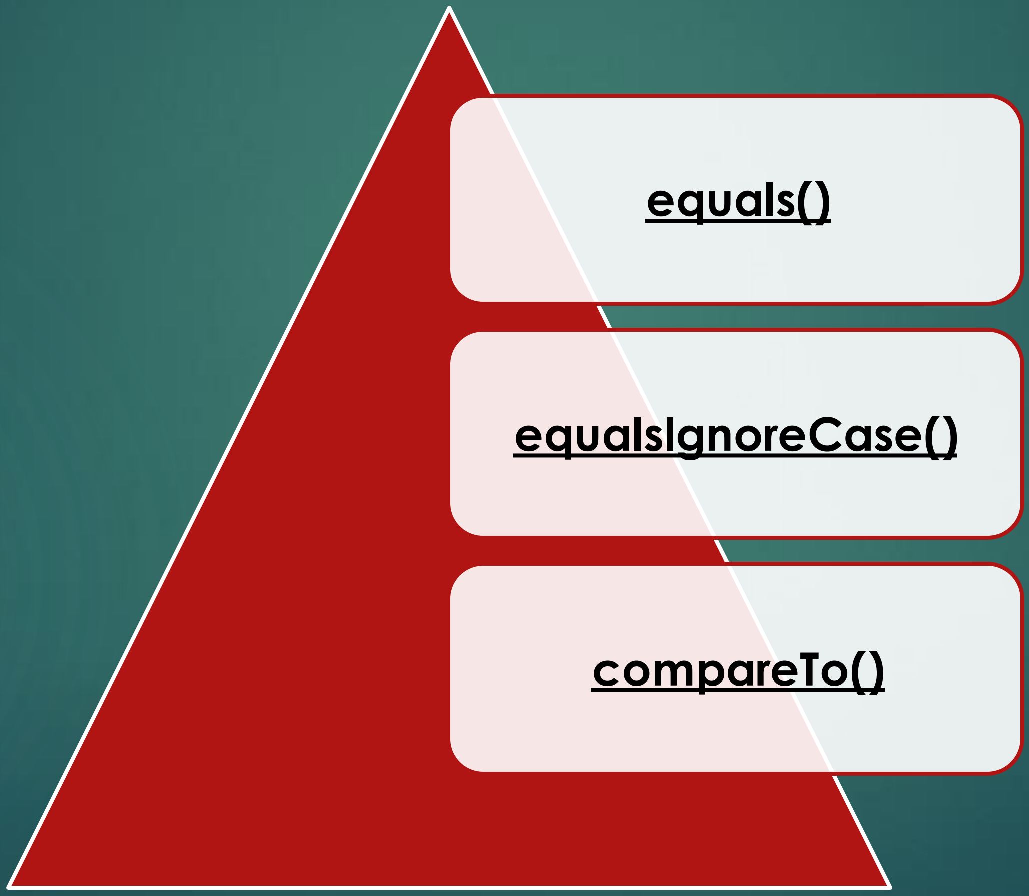
getChars()

- If you need to extract more than one char at a time.
- **Syntax**
  - **void getChars(int sourceStart,int sourceEnd,char target[],int targetstart)**
    - **sourceStart** specify the index of the begining of substring
    - substring contains character from **sourceStart** through **sourceEnd-1**
    - the array that will receive the characters is specified by **target**,
    - index within target at which substring will be copied is specified by targetstart.
- **Example**
  - String s ="This is java demo";
  - char ch[]=new char[5];
  - s.getchars(8,12,ch,0);
  - System.out.println(ch);

substring()

- it is used to extract substring from given string.
- **Syntax**
  - **String substring(int startindex)**
    - startindex specifies the index at which the substring will begin.
  - **String substring(int startindex,int endindex)**
    - startindex specifies the begining index,endindex specify stopping point.

String Comparisions



equals()

- To compare two strings for equality
- **Syntax**
  - **boolean equals(String str)**
  - str is the String object being compared with the invoking String object
  - it returns true if both the string contain same characters in same order ,and false otherwise.
  - this comparison is case-sensitive.
- **Example**
  - String s1 = "Hello"
  - String s2 = "Hello"
  - System.out.println(s1.equals(s2))

equalsIgnoreCase()

- To compare two String for equality but not case-sensitive.
- **Syntax**
  - **boolean equalsIgnoreCase(Object Str)**
- **Example**
  - String s1 ="Hello"
  - String s2 ="heLLo"
  - System.out.println(s1.equalsIgnoreCase(s2))

compareTo()

- **Syntax**
  - **int compareTo(String str)**
    - str is the string being compared with the invoking String.
    - if return value is
      - Less than zero The invoking string is less than *str*.
      - Greater than zero The invoking string is greater than *str*.
      - Zero The two strings are equal.

equals versus ==

- equals() method compares the characters inside the String object.
- The == operator compares two object references to see whether they refer to same instance
- **Example**
  - String s1 ="Hello";
  - String s2 = new String(s1);
  - System.out.println(s1.equals(s2));
  - System.out.println((s1==s2));

StringBuffer

- In JAVA String represent fixed length ,immutable character sequences.
- But StringBuffer represent growable and writable character sequences.
- StringBuffer will automatically grow to make room for such additions and often have more characters pre-allocated than needed.

StringBuffer Constructor

- **StringBuffer()**
  - it is a default constructor that reserves room for 16 characters.
- **StringBuffer(int size)**
  - it accepts an integer argument that explicitly set the size of the buffer.
- **StringBuffer(String s)**
  - it accept String argument that set the initial content of the StringBuffer and reserves room for 16 more characters.

length()

- Current length of StringBuffer can be found via the length() method.
- **Syntax**
  - int length()
- **Example**
  - StringBuffer sb = new StringBuffer("Hello");
  - System.out.println(sb.length());

capacity()

- Total allocated space/memory can be found using the **capacity()** method.
- **Syntax**
  - int capacity();
- **Example**
  - StringBuffer sb = new StringBuffer("Hello");
  - System.out.println(sb.capacity());

ensureCapacity()

- If you want to preallocate room for a certain no of characters after a StringBuffer has been constructed we can use this method to set the size of buffer.
- **Syntax**
  - **void ensureCapacity(int mincapacity)**
  - mincapacity specifies the minimun size of the buffer.

setLength()

- it is used to set the length of string within a StringBuffer object.
- **Syntax**
  - **void setLength(int len)**
  - len specifies length of String and it must be nonnegative value.
  - if you increase the length than the current length then null characters are added to the end.
  - if you decrease the length than current length then content will be lost..

charAt() and setCharAt()

- **Syntax**
  - char charAt(int where)
  - where specifies the index of character being obtained.
- **Syntax**
  - void setCharAt(int where , char ch)
  - where specifies index of character being set.
  - ch specifies the value of character
- **Example**
  - StringBuffer sb = new StringBuffer("Hello");
  - System.out.println(sb.charAt(1));
  - sb.setCharAt(1,'i');
  - System.out.println(sb);

append() Method

- this method append String representation of any other type of data to the end of the invoking StringBuffer object.
- **Syntax**
  - **StrinBuffer append(String str)**
  - **StringBuffer append(int num)**
- **Example**
  - StringBuffer sb = new StringBuffer("Hello");
  - System.out.println(sb.append("a=").append(12));

insert() Method

- This method insert one String into another.
- **Syntax**
  - **StringBuffer insert(int index,String str)**
  - index specifies the point at which str will be inserted in invoking StringBuffer object.

reverse() Method

- it reverse the character within a StringBuffer object.
- **Syntax**
  - StringBuffer reverse()
- **Example**
  - StringBuffer sb = new StringBuffer("abc");
  - sb.reverse();
  - System.out.println(sb);

delete() Method

- we can delete characters within a StringBuffer object.
- **Syntax**
  - **StringBuffer delete(int startIndex,int endIndex)**
  - startIndex is the index of the first character to remove and endIndex specifies the index one past the last character to remove.
- **Example**
  - StringBuffer sb = new StringBuffer("Wellcome");
  - sb.delete(1,5);
  - System.out.println(sb);

replace() Method

- We can replace one set of characters with another set inside a StringBuffer object.
- **Syntax**
  - **StringBuffer replace(int startIndex, int endIndex,String str);**

String v/s StringBuffer

String	StringBuffer
String class is immutable.	StringBuffer is mutable
String is slow and consume more memory when you concate too many Strings because everytime it creates new instance.	StringBuffer is fast and consume less memory when you concate strings.
String class overrides the equals() method of Object class.	StringBuffer doesnot override the equals() method of object class.

Method Overloading

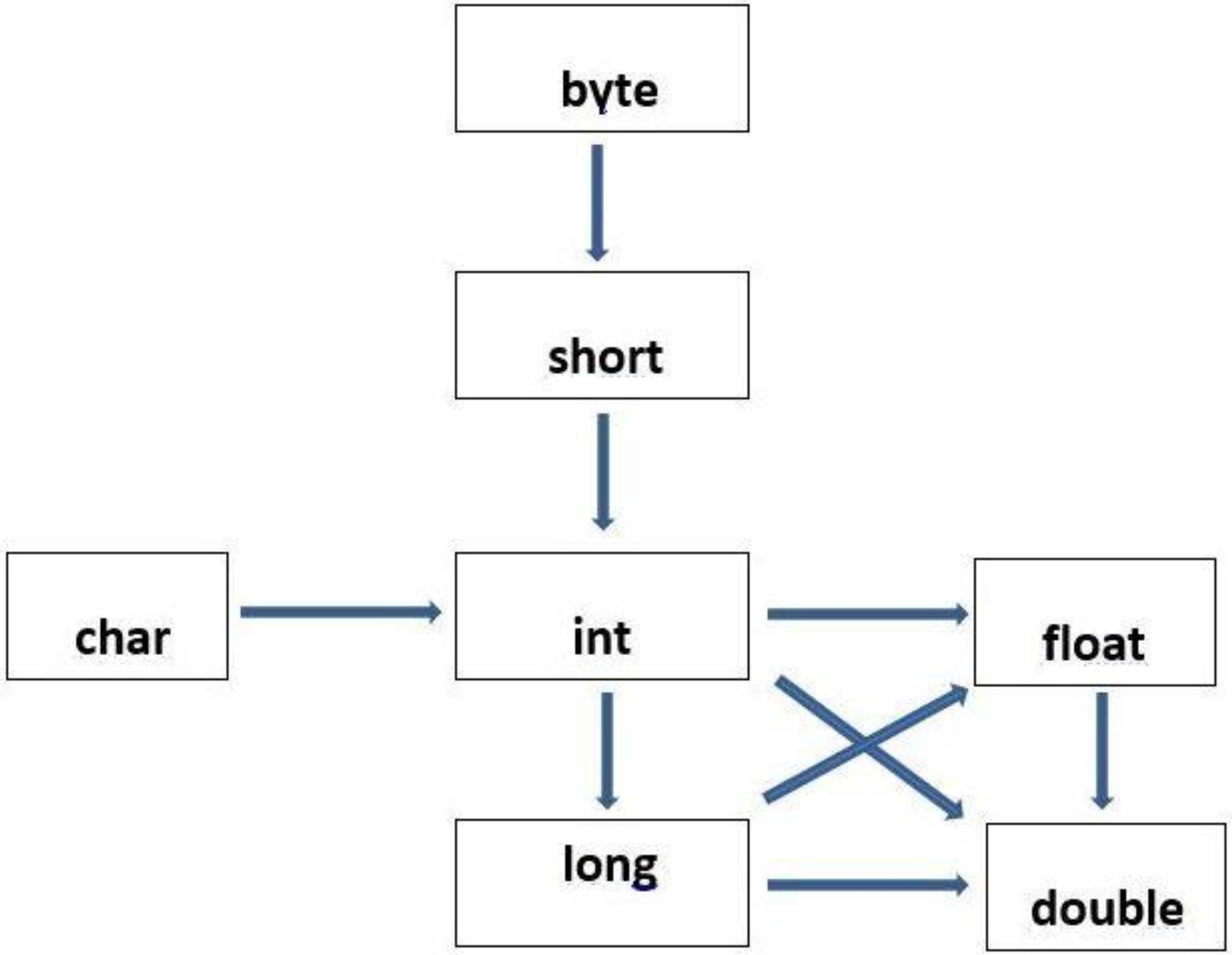
Introduction

- It is a process in JAVA through which we can define two or more methods in the same class that share the same name ,but their parameter declaration are different.
- It is one of the way through which JAVA supports **Polymorphism.**
- When overloaded method is invoked ,Java use the type and/or number of arguments as its guide to determine which version of the overloaded method to actually call.
- Overloaded methods may have different return types, but return type alone is insufficient to distinguish two versions of a method.

Note:

- When an overloaded method is called, Java looks for a match between the arguments used to call the method and the method’s parameters. However, this match need not always be exact. In some cases, Java’s automatic type conversions can play a role in overload resolution.
- return types do not play a role in overload resolution.

TYPE PRAMOTION


byte
short
char
int
float
long
double