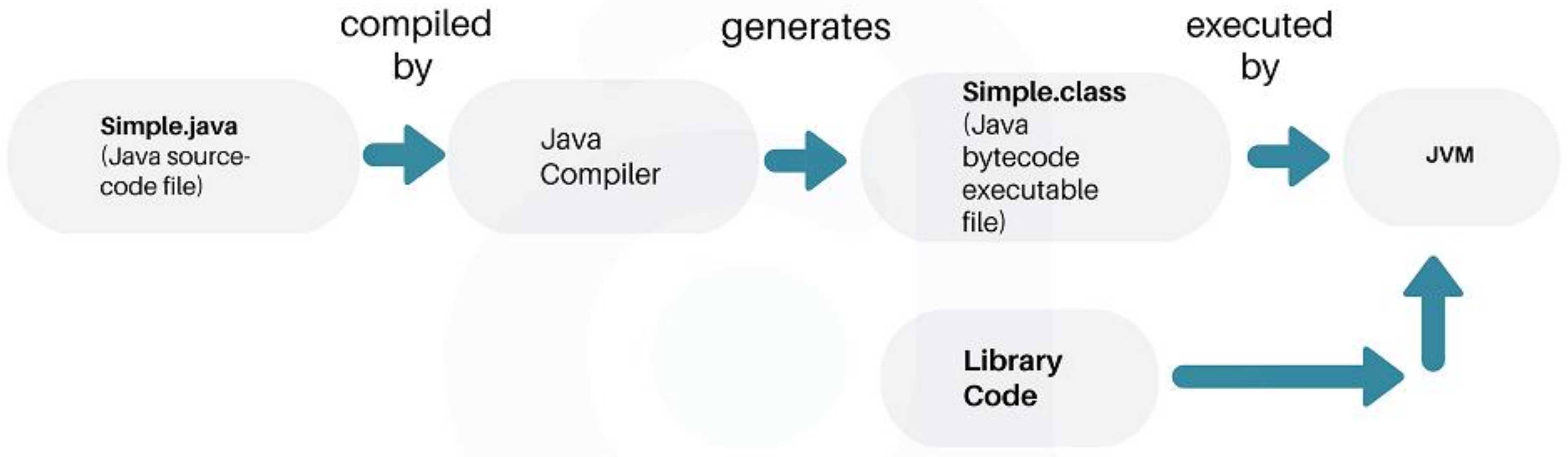
CORE JAVA

AMOL R PATIL - 9822291613

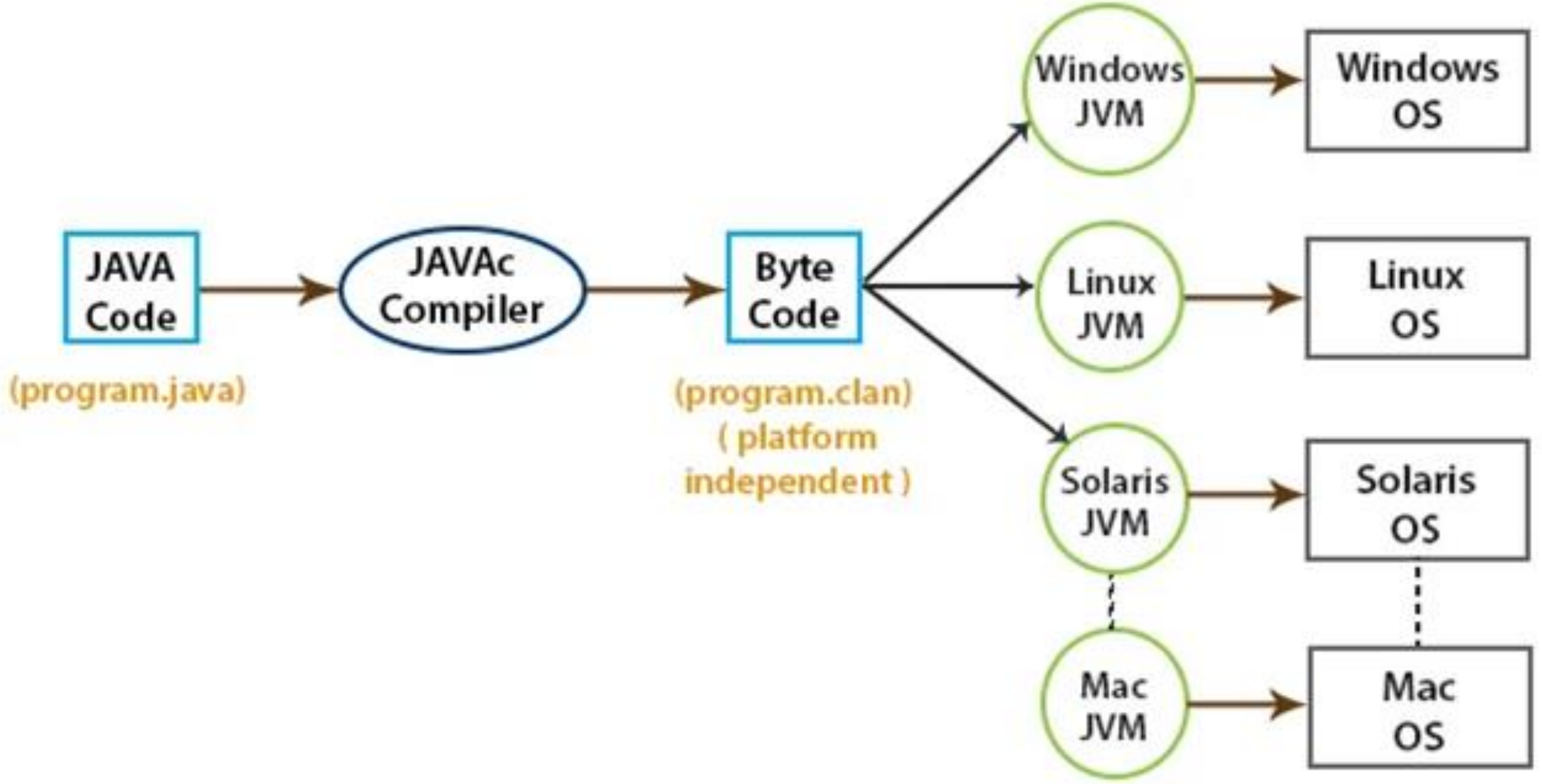
What is Java ?

- Java is a high level, robust, object-oriented and secure programming language.
- Java technology is both a programming language and a platform.
- JAVA was developed by Sun Microsystems Inc in 1991, later acquired by Oracle Corporation.
- It was developed by James Gosling and Patrick Naughton

Execution of Java Program


compiled by
generates
executed by
Simple.java (Java source-code file)
Java Compiler
Simple.class (Java bytecode executable file)
JVM
Library Code

Platform Independence in Java



Java Buzzwords

- Simple
- Distributed
- Dynamic
- Portable
- Robust

Object oriented

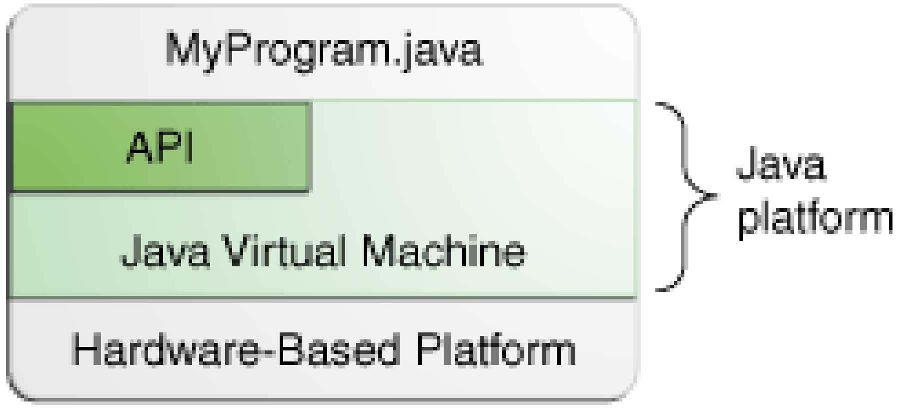
Multithreaded

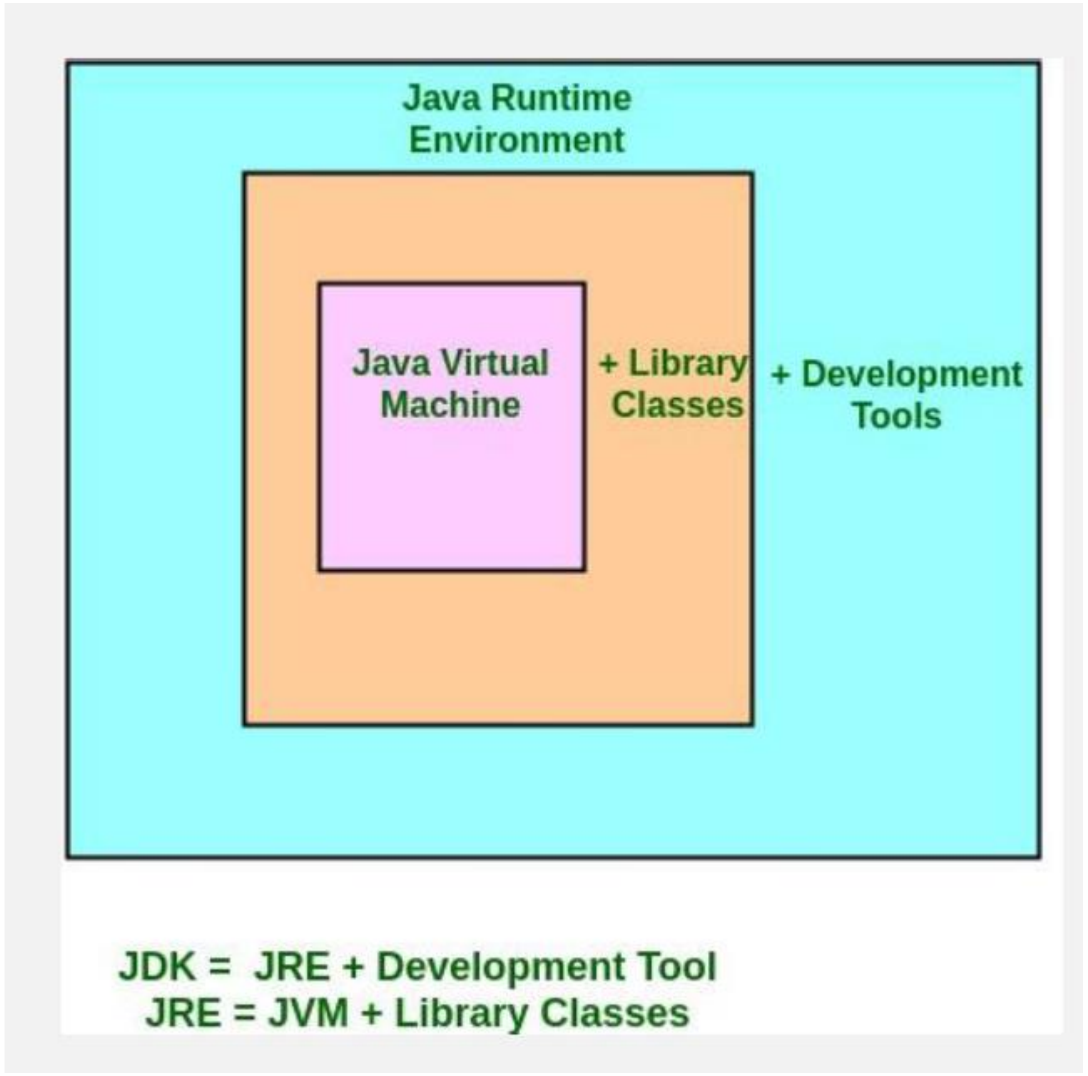
Architecture neutral

High performance

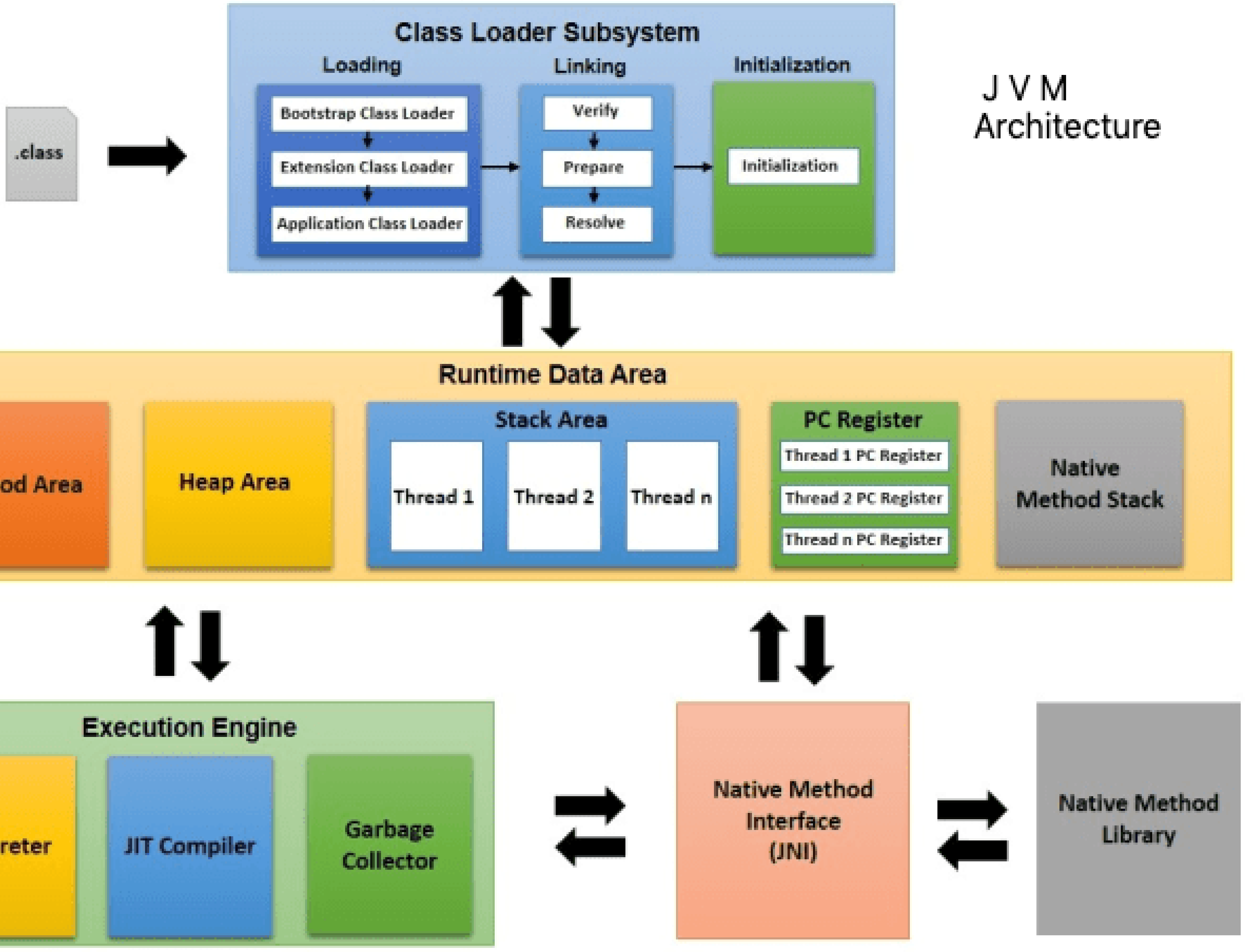
Secure

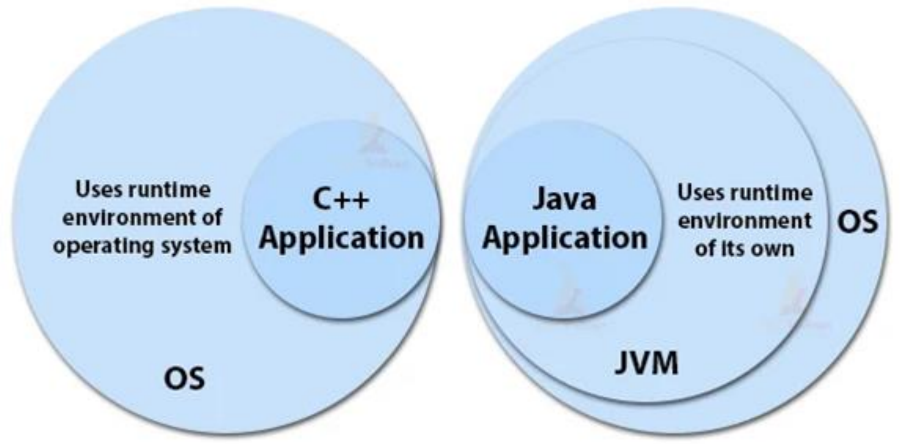
JDK , JRE , JVM


MyProgram.java
API
Java Virtual Machine
Hardware-Based Platform
Java platform


Java Runtime Environment
Java Virtual Machine
+ Library Classes
+ Development Tools
JDK = JRE + Development Tool
JRE = JVM + Library Classes

J V M
Architecture




Uses runtime environment of operating system
C++ Application
OS
Java Application
Uses runtime environment of its own
OS
JVM

Security in Applications

- No explicit pointer
- Java Programs run inside a virtual machine sandbox

Classloader: Classloader in Java is a part of the Java Runtime Environment (JRE) which is used to load Java classes into the Java Virtual Machine dynamically. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.

Bytecode Verifier: It checks the code fragments for illegal code that can violate access rights to objects.

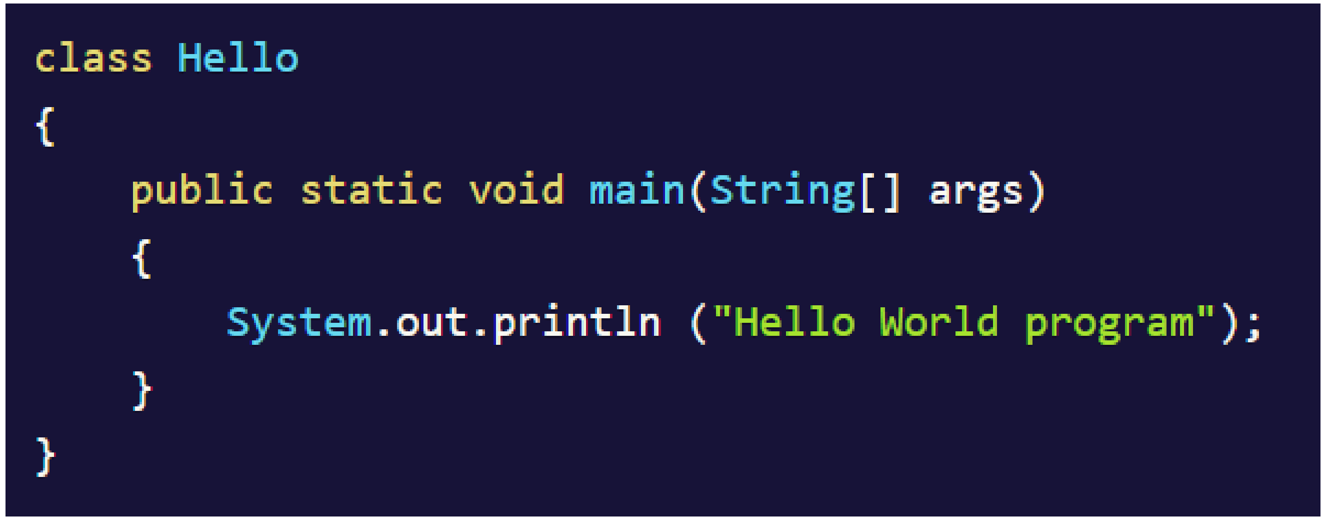
Security Manager: It determines what resources a class can access such as reading and writing to the local disk.

Java IDE Software's

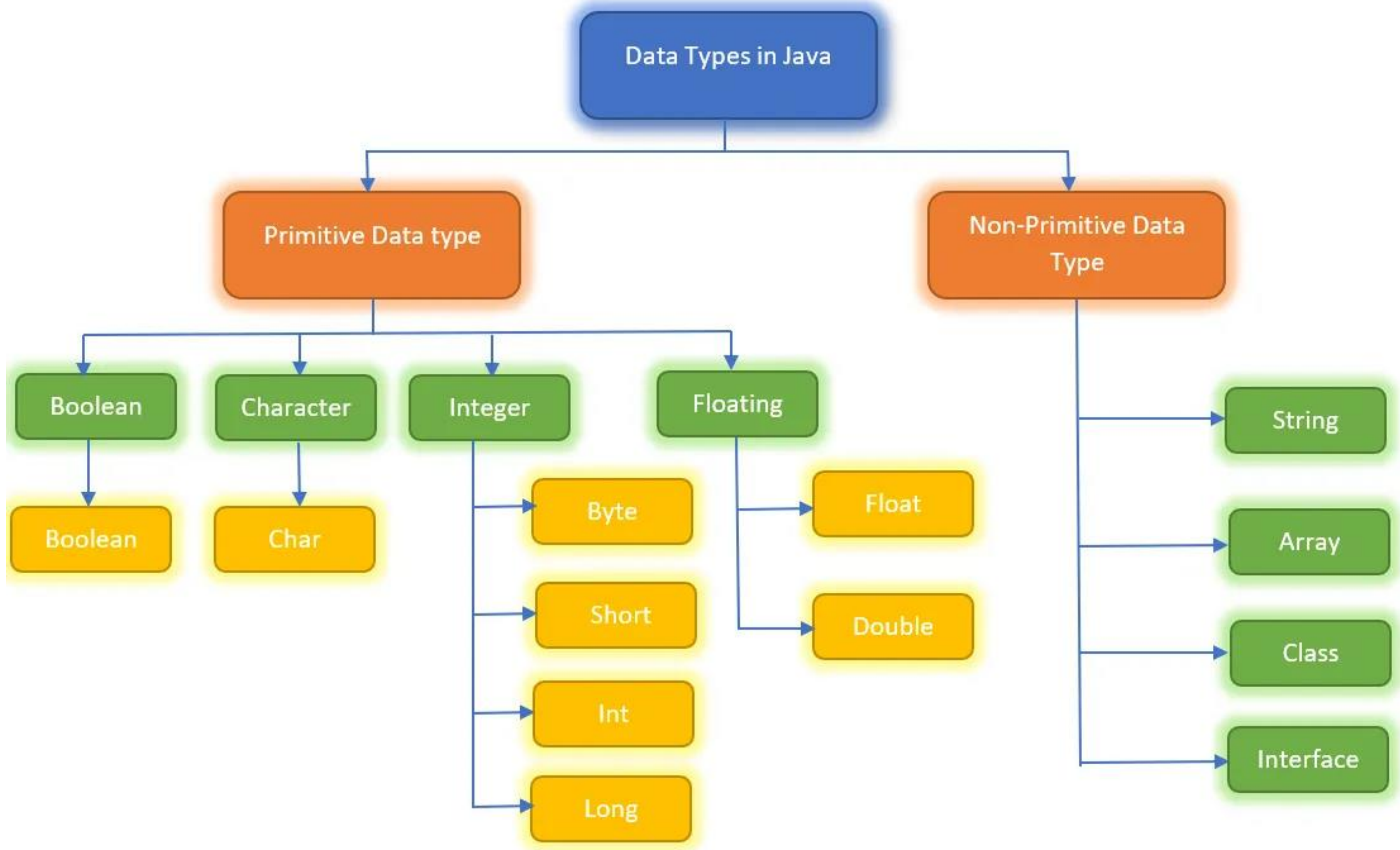
Top 10 Java IDEs

- Eclipse
- NetBeans
- IntelliJ IDEA
- BlueJ
- JDeveloper
- MyEclipse
- Greenfoot
- jGRASP
- jCreator
- DrJava

Java First Program



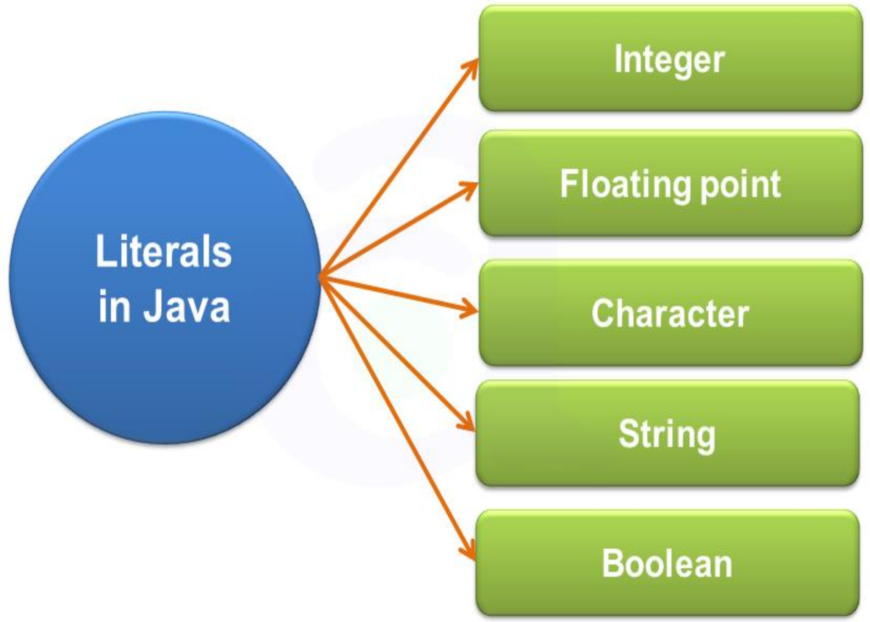
```
class Hello
{
    public static void main(String[] args)
    {
        System.out.println ("Hello World program");
    }
}
```


Data Types in Java
Primitive Data type
Non-Primitive Data Type
Boolean
Character
Integer
Floating
Boolean
Char
Byte
Short
Int
Long
Float
Double
String
Array
Class
Interface

Java Primitive Data Types

Data Type	Size	Description	Default value
byte	1 byte	Stores whole numbers from -128 to 127	0 (zero)
short	2 bytes	Stores whole number from -32768 to 32767	0 (zero)
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647	0 (zero)
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0L
float	4 bytes	Stores fractional numbers up to 6-7 decimal digits	0.0f
double	8 bytes	Stores fractional numbers with up to 15 decimal digits	0.0d
char	2 bytes	Stores single character/letter	'\u0000'
boolean	1 bit	Stores true or false	false

Java Literals



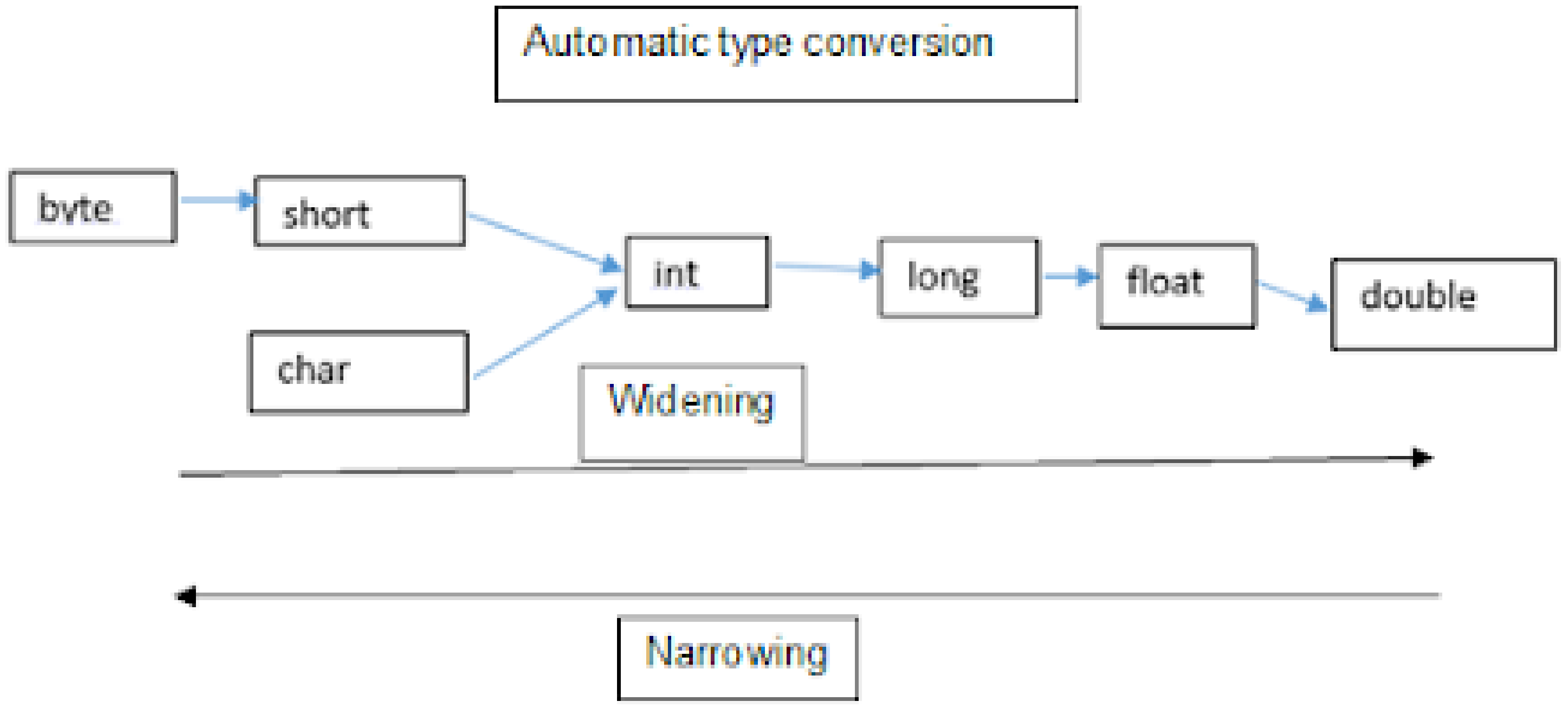
- int a = 101;
- int b = 0146;
- int c = 0x123Face;
- int d = 0b1111;

- double decimalValue = 101.230
- double decimalValue1 = 0123.222;
- double hexaDecimalValue = 1.234e2;

- boolean boolVar1 = true;
- boolean boolVar2 = false;

- char ch = 'a';
- char c = '\u0061';
- String s = "Hello";

Java Typecasting



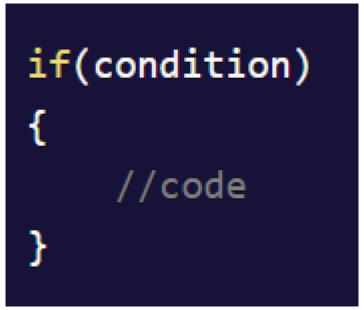
Java Operators

Operators	Precedence
postfix	*expr*++ *expr*--
unary	++*expr* --*expr* +*expr* -*expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

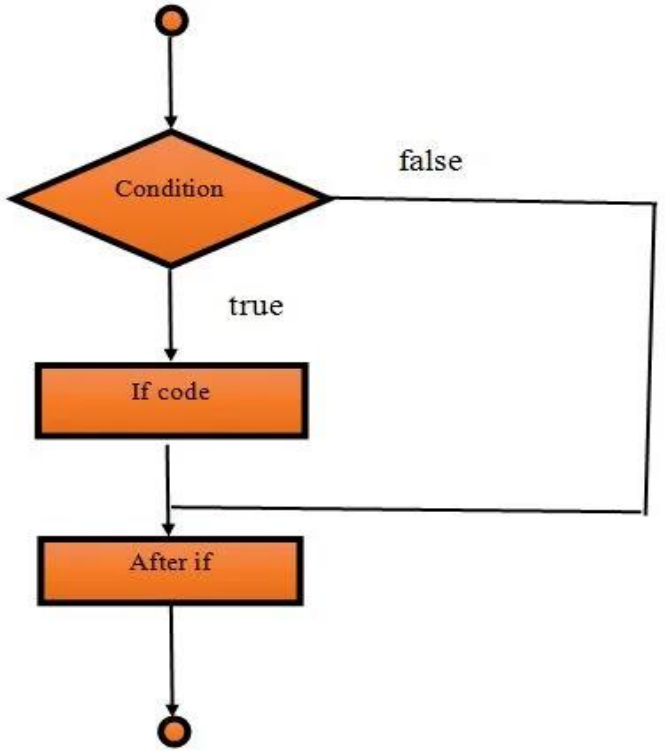
Simple Program Assignment

- Temperature of a city in Fahrenheit degrees is input through the keyboard. Write a program to convert this temperature into Centigrade degrees
- If a four-digit number is input through the keyboard, write a program to obtain the sum of the first and last digit of this number
- Two numbers are input through the keyboard into two locations C and D. Write a program to interchange the contents of C and D.

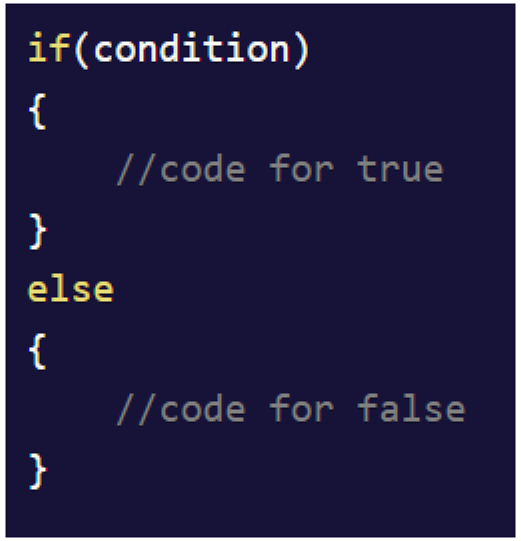
Java Control Statements – if statement



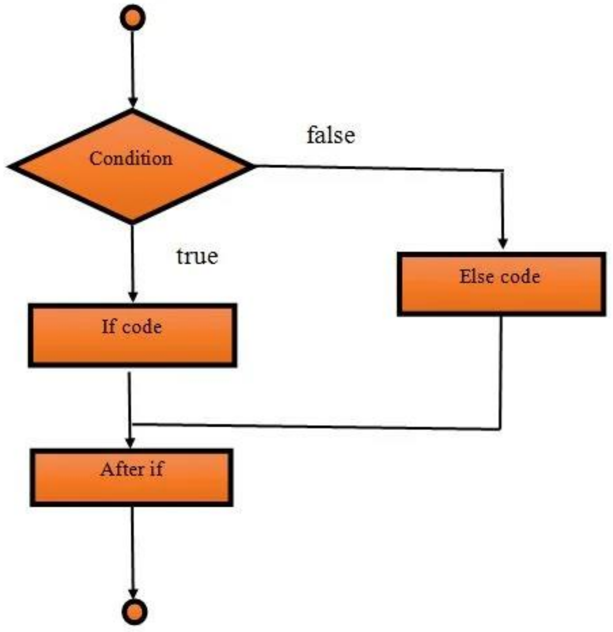
```
if(condition)
{
    //code
}
```



Java Control Statements – if –else statement

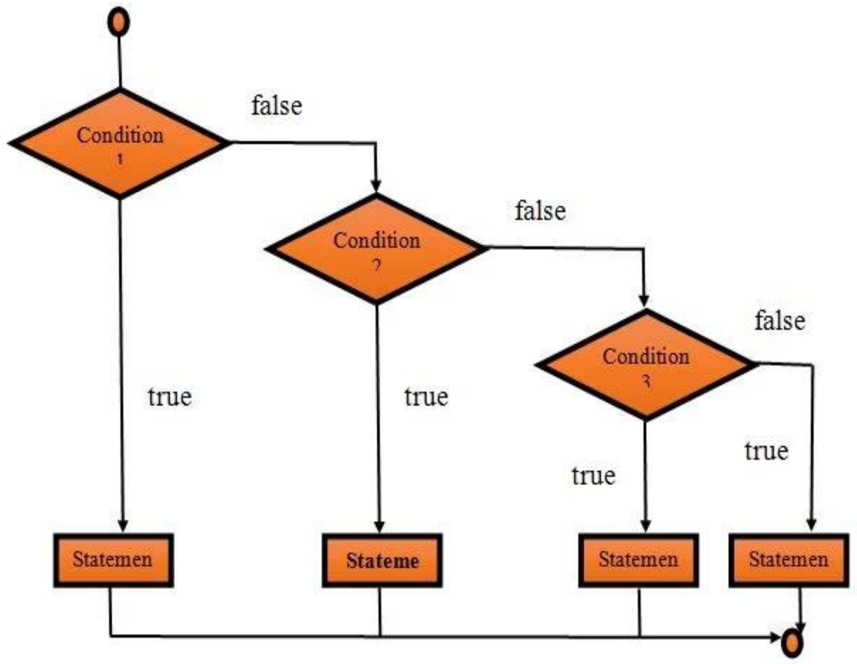


```
if(condition)
{
    //code for true
}
else
{
    //code for false
}
```



Java Control Statements – if –else statement

```
if(condition1)
{
    //code for if condition1 is true
}
else if(condition2)
{
    //code for if condition2 is true
}
else if(condition3)
{
    //code for if condition3 is true
}
...
else
{
    //code for all the false conditions
}
```



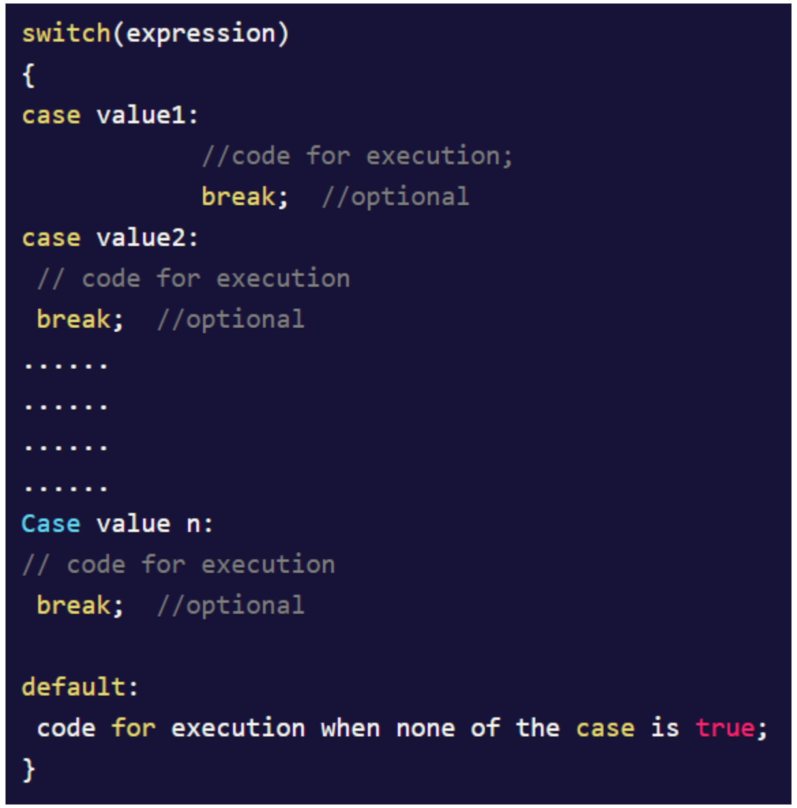
If-Else Statements - Assignment

- Write a program to accept a coordinate point in a XY coordinate system and determine in which quadrant the coordinate point lies.

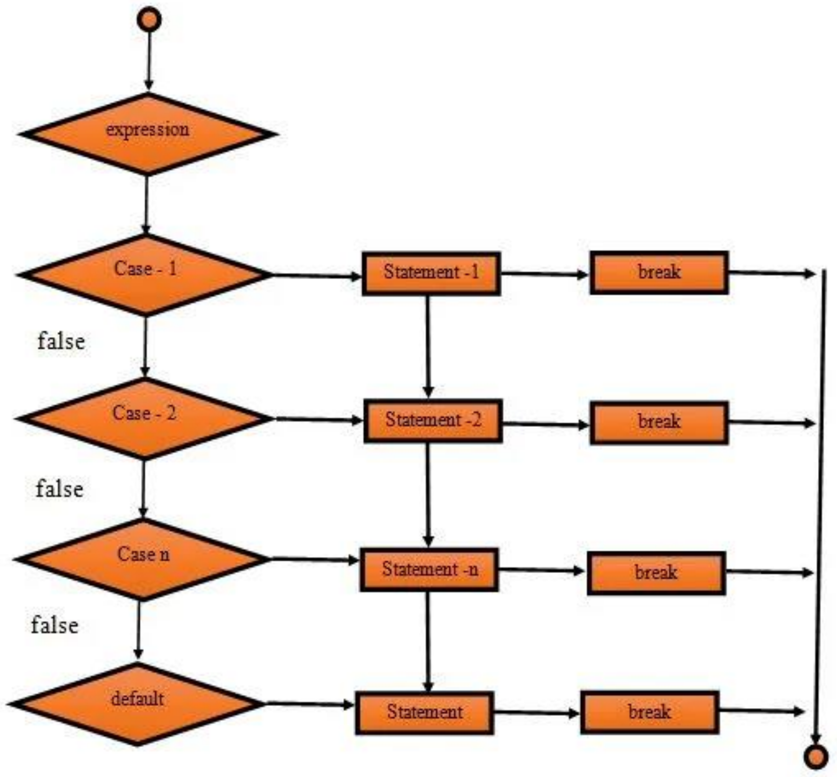
- Write a program to check whether a triangle is Equilateral, Isosceles or Scalene

- Write a program to input electricity unit charges and calculate total electricity bill according to the given condition:
  - For first 50 units Rs. 0.50/unit
  - For next 100 units Rs. 0.75/unit
  - For next 100 units Rs. 1.20/unit
  - For unit above 250 Rs. 1.50/unit
  - An additional surcharge of 20% is added to the bill

switch-case Statement



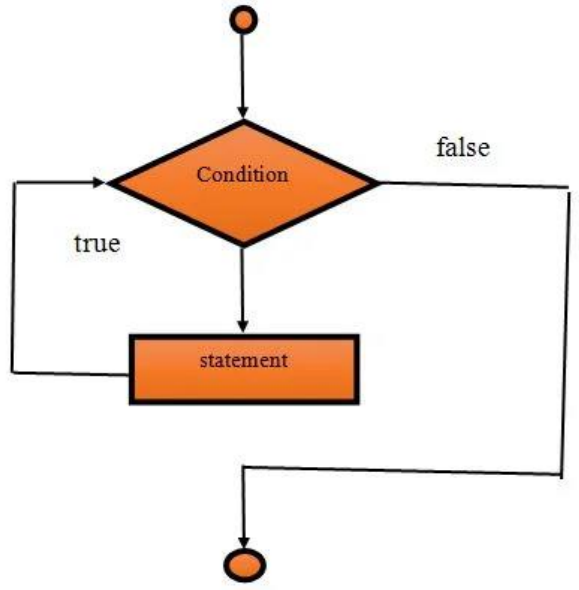
```
switch(expression)
{
case value1:
            //code for execution;
            break;  //optional
case value2:
 // code for execution
 break;  //optional
......
......
......
......
Case value n:
// code for execution
 break;  //optional

default:
 code for execution when none of the case is true;
}
```



Switch-Case Statement - Assignment

- Write a program to create simple calculator using switch Statement

Loop in Java

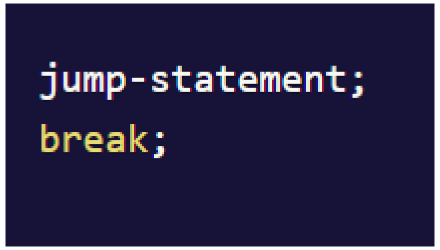


```
while(condition)
{
//code for execution
}
```

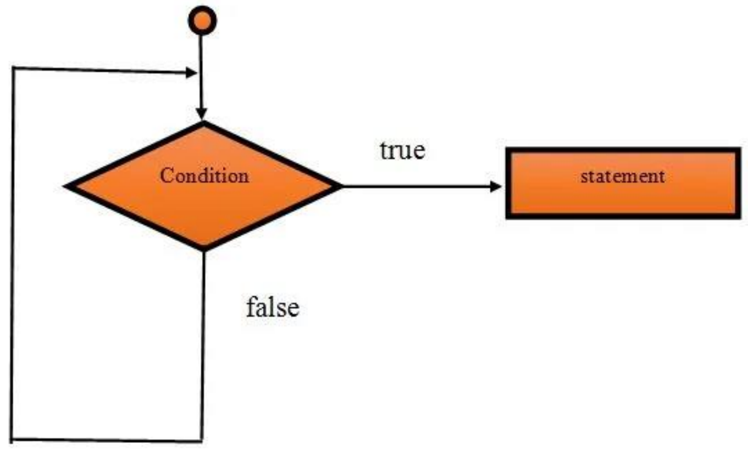
```
do
{
//code for execution
}
while(condition);
```

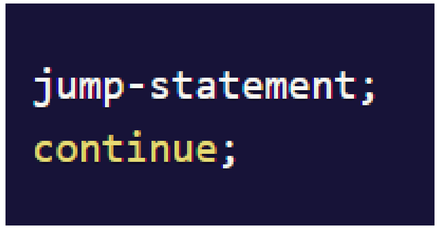
```
for(initialization;condition;increment/decrement)
{
//statement
}
```

continue, break statements



```
jump-statement;
break;
```



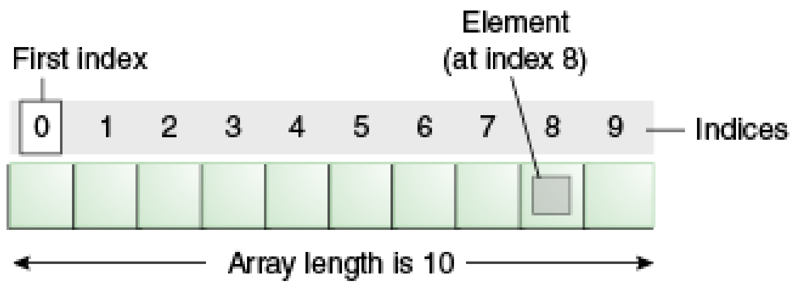


```
jump-statement;
continue;
```

Loop Assignments

- Write a program to check whether a given number is an Armstrong number or not
- Write a program to display the first n terms of Fibonacci series.
- Write a program to display the sum of the series [9 + 99 + 999 + 9999 ...]
- Write a program to find the prime numbers within a range of numbers
- Write a program to print Pyramid of *

Java Arrays

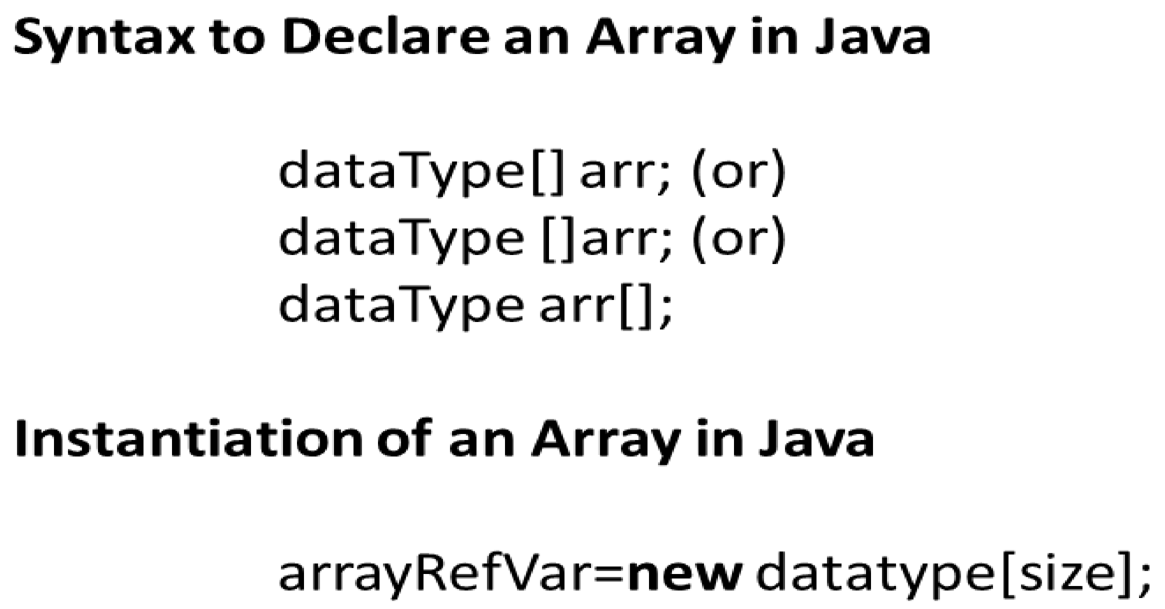


- Java array is an object which contains elements of a similar data type.

- It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array. Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

- Unlike C/C++, we can get the length of the array using the length member. In C/C++, we need to use the sizeof operator.

1 D Array



Syntax to Declare an Array in Java

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

Instantiation of an Array in Java

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

Declaration, Instantiation and Initialization of Java Array
We can declare, instantiate and initialize the java array together by:

```
int a[]={33,3,4,5};//declaration, instantiation and initialization
```

Java Single Dim Array - Exs

- Write a program to find second largest element in an array.

- Write a program to print all unique elements in the array.

- WAP to find the intersection of two arrays.

  - X { 23,-5,89,12,9 }
  - Y { 9,-34,12,100,-5 }

  - Intersection: { -5, 12, 9 }

Java Multi Dim Array

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

Example to instantiate Multidimensional Array in Java

```
int[][] arr=new int[3][3];//3 row and 3 column
```

Java Multi Dim Array - Exs

- Write a program to find sum of diagonal elements of a matrix.
- Matrix multiplication

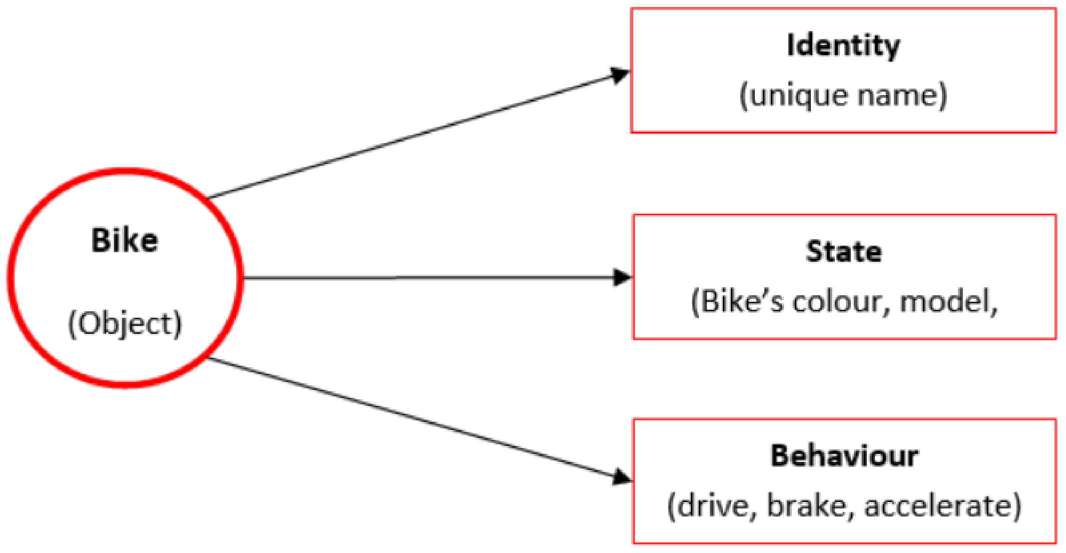
What is Object Oriented Programming ?

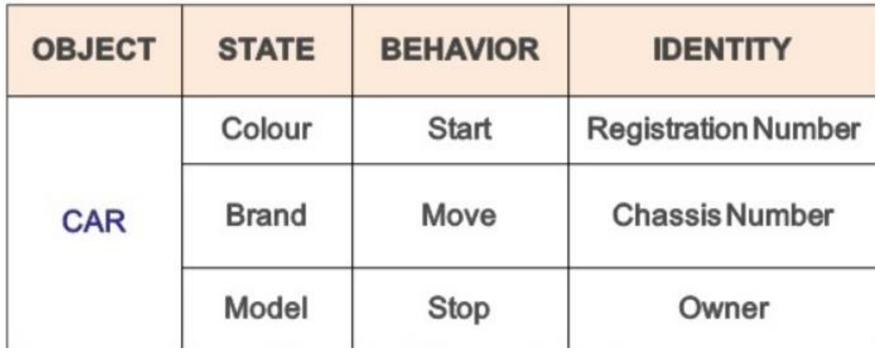
- Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic.

- An object can be defined as a data field that has unique attributes and behavior

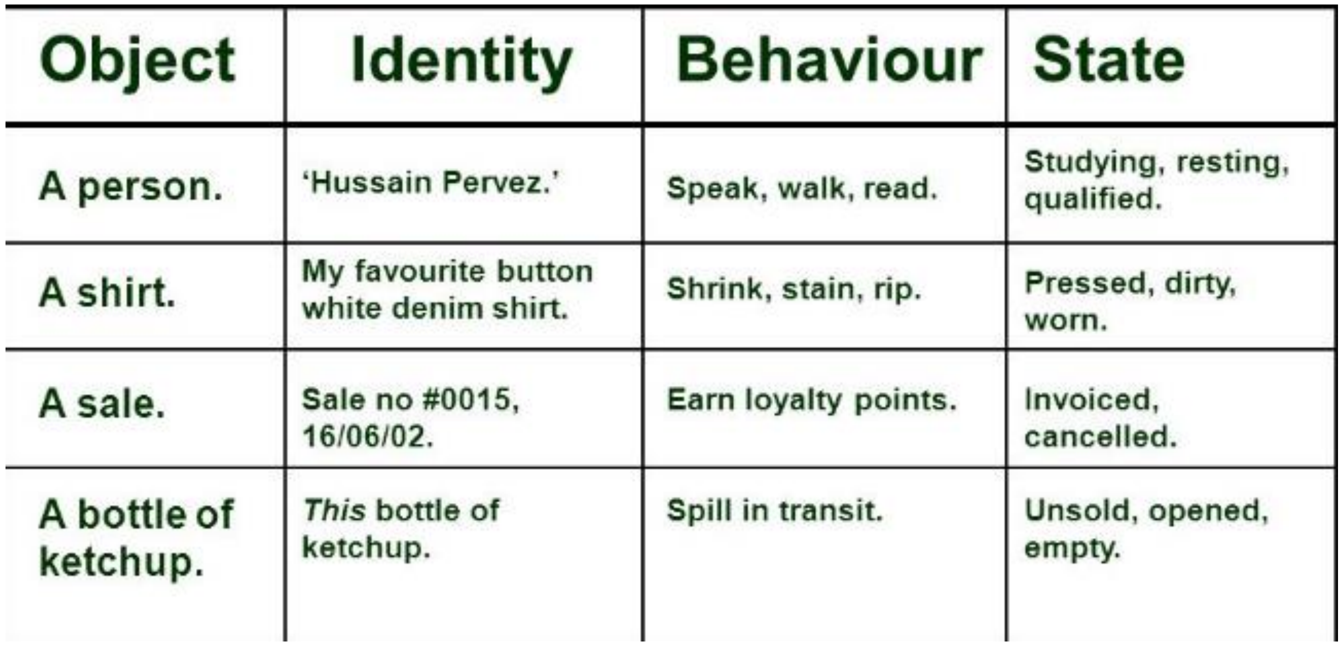
- Everything's an object, right? Well, almost everything. The assertion that everything is an object is the concept which underlies object-oriented programming, or OOP for short.

Characteristics of an OBJECT





OBJECT	STATE	BEHAVIOR	IDENTITY
CAR	Colour	Start	Registration Number
	Brand	Move	Chassis Number
	Model	Stop	Owner



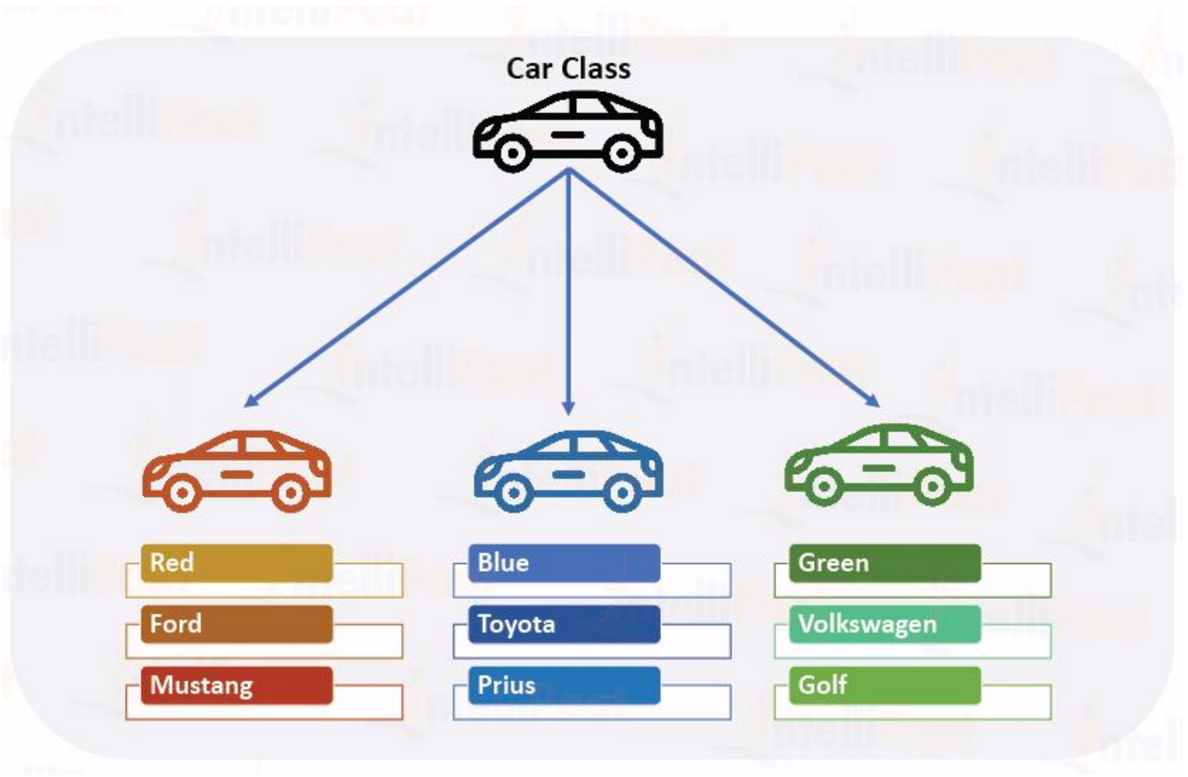
Object	Identity	Behaviour	State
A person.	'Hussain Pervez.'	Speak, walk, read.	Studying, resting, qualified.
A shirt.	My favourite button white denim shirt.	Shrink, stain, rip.	Pressed, dirty, worn.
A sale.	Sale no #0015, 16/06/02.	Earn loyalty points.	Invoiced, cancelled.
A bottle of ketchup.	*This* bottle of ketchup.	Spill in transit.	Unsold, opened, empty.

Class and Objects

Class: Think of a class like a blueprint or a template. It defines the structure and behavior of something. For example, a "Car" class could define what properties (like color and model) a car has and what actions (like starting the engine) it can do. It's a general plan.

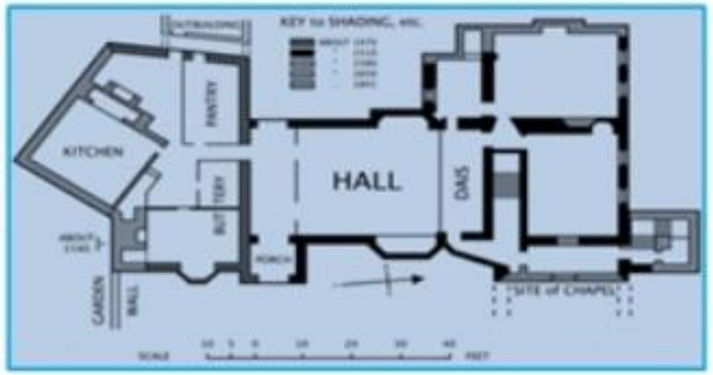
Object: An object is like a specific instance created from that blueprint. If the "Car" class is the blueprint, an object would be an actual car, like a red Toyota. Objects are like real, usable things that follow the instructions laid out in the class.

So, a class is a plan, and an object is something built based on that plan. Think of a class as a recipe, and an object as the actual dish you cook using that recipe.



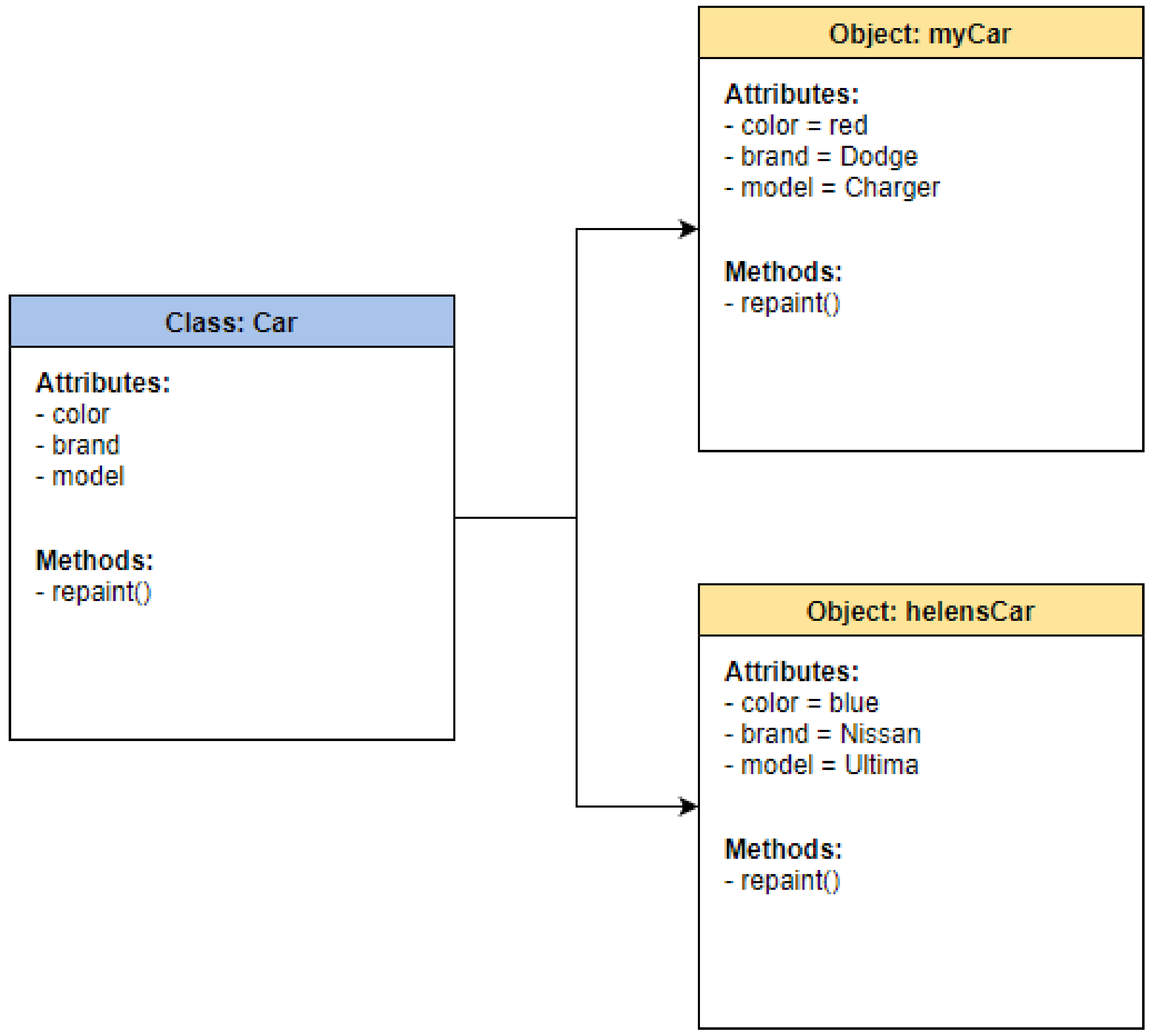
Class In Object Oriented Programming (OOP)

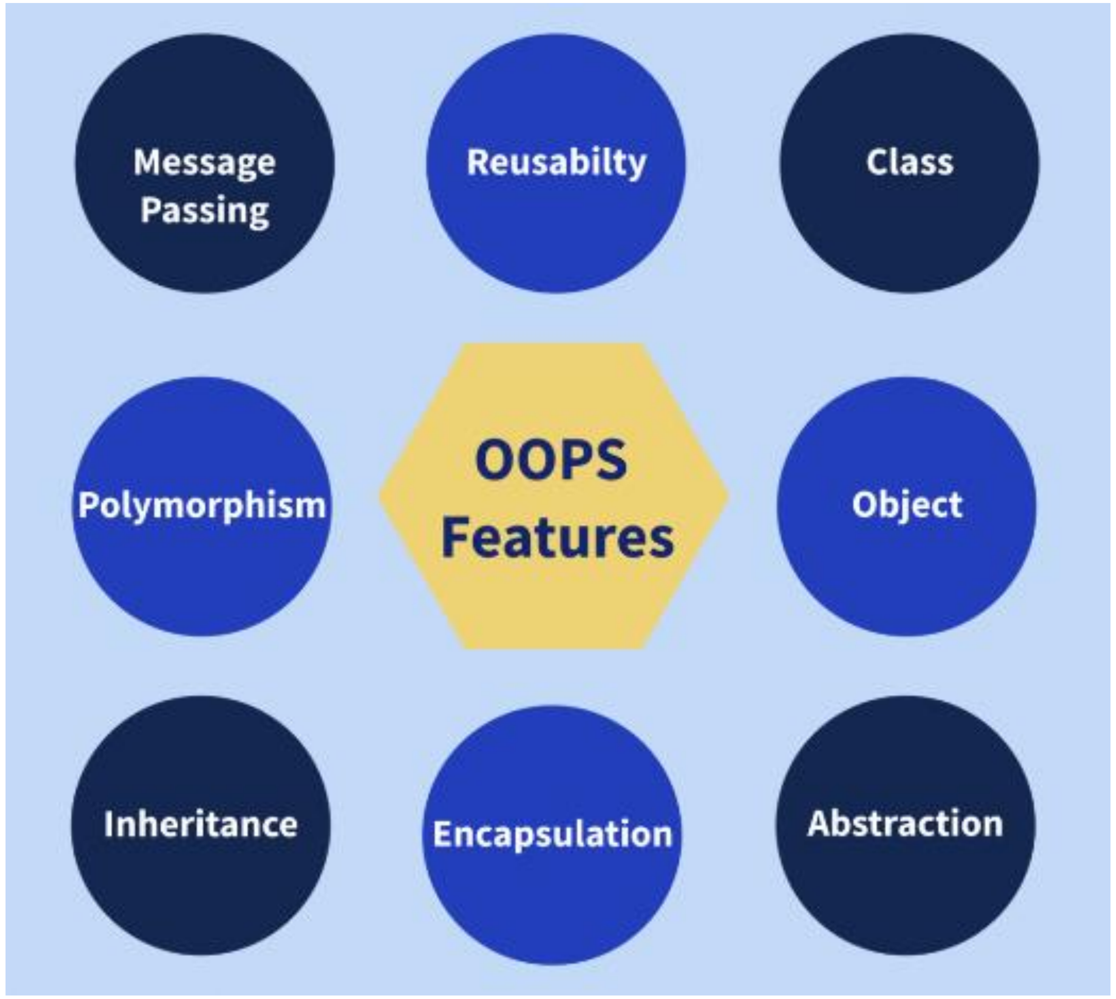
- A **Class** is basically a Description , Data structure , Blueprint / Design / Template for creating an object .
- A **Class** is a program **design time** entity where as **Object** is a program **run time** entity .



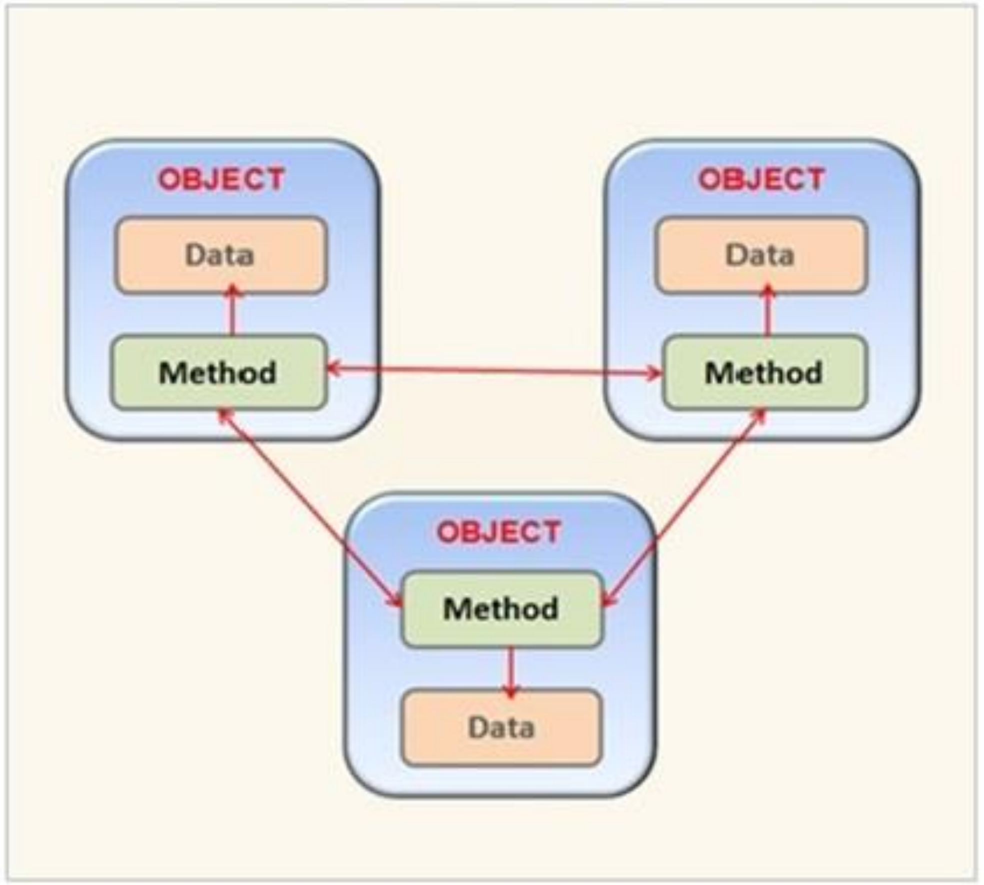
Blueprint / Design / Template

Object

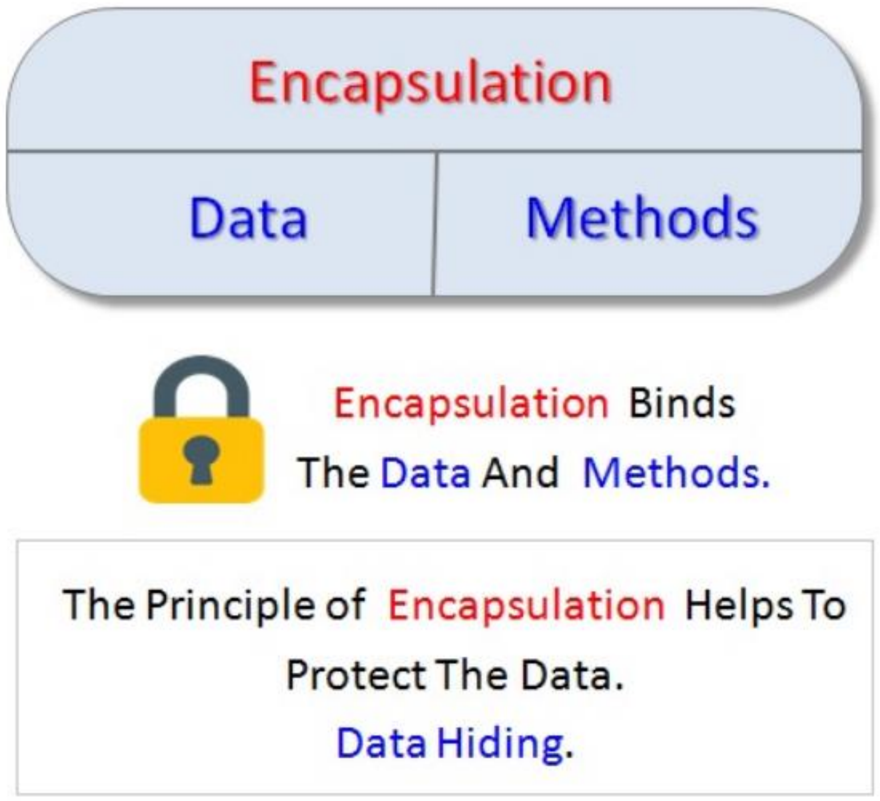

Class: Car
Attributes:
- color
- brand
- model
Methods:
- repaint()
Object: myCar
Attributes:
- color = red
- brand = Dodge
- model = Charger
Methods:
- repaint()
Object: helensCar
Attributes:
- color = blue
- brand = Nissan
- model = Ultima
Methods:
- repaint()



OPP - Program Organization

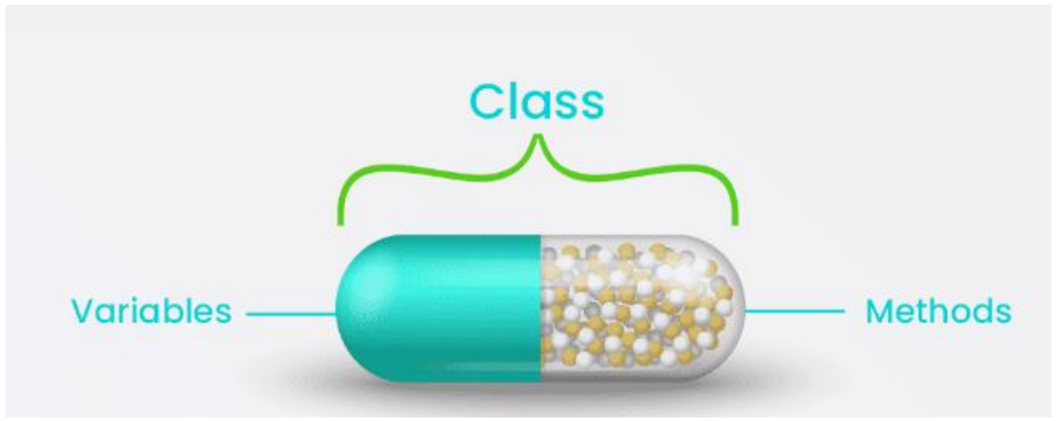


Encapsulation

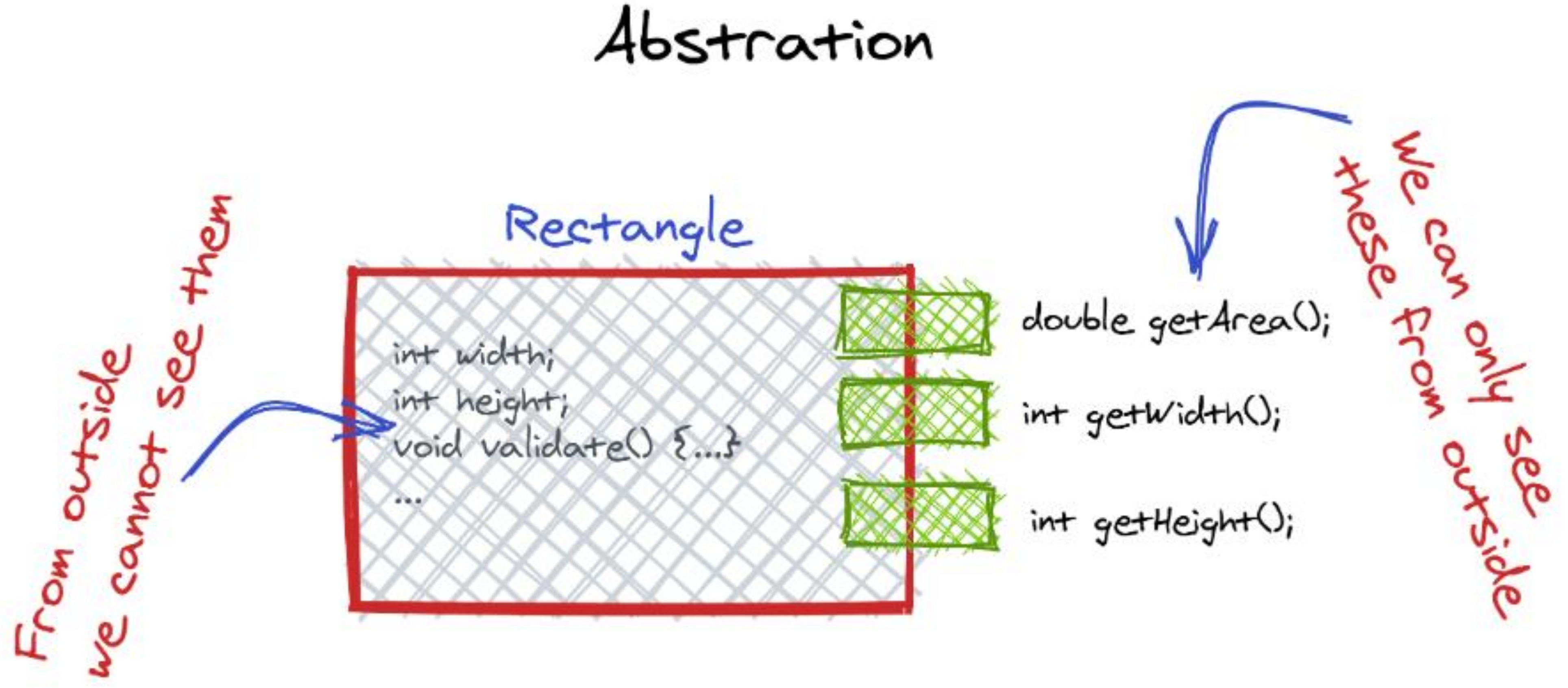


Encapsulation Binds The Data And Methods.

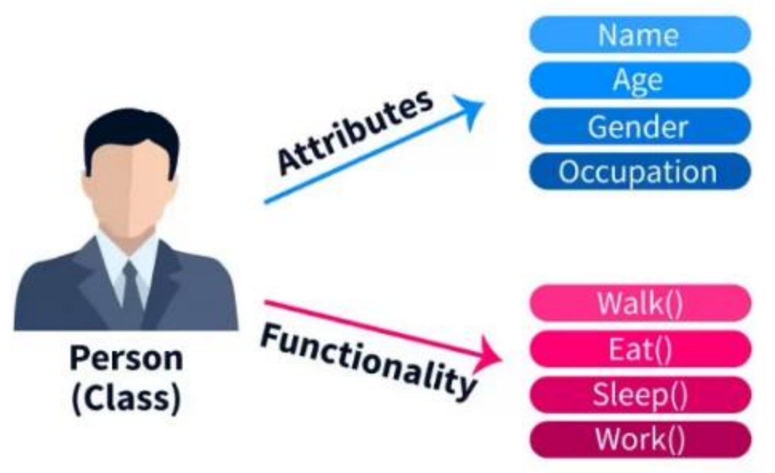
The Principle of Encapsulation Helps To Protect The Data.
Data Hiding.



Abstraction



Java class

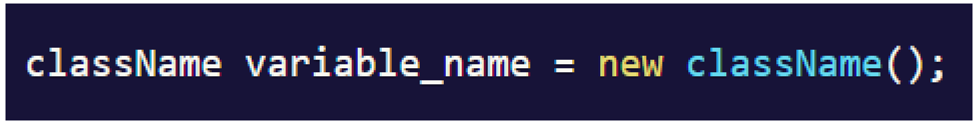


Class is a blueprint or a set of instructions to build a specific type of object. It is a basic concept of Object-Oriented Programming which revolve around the real-life entities. Class in Java determines how an object will behave and what the object will contain.

Java object

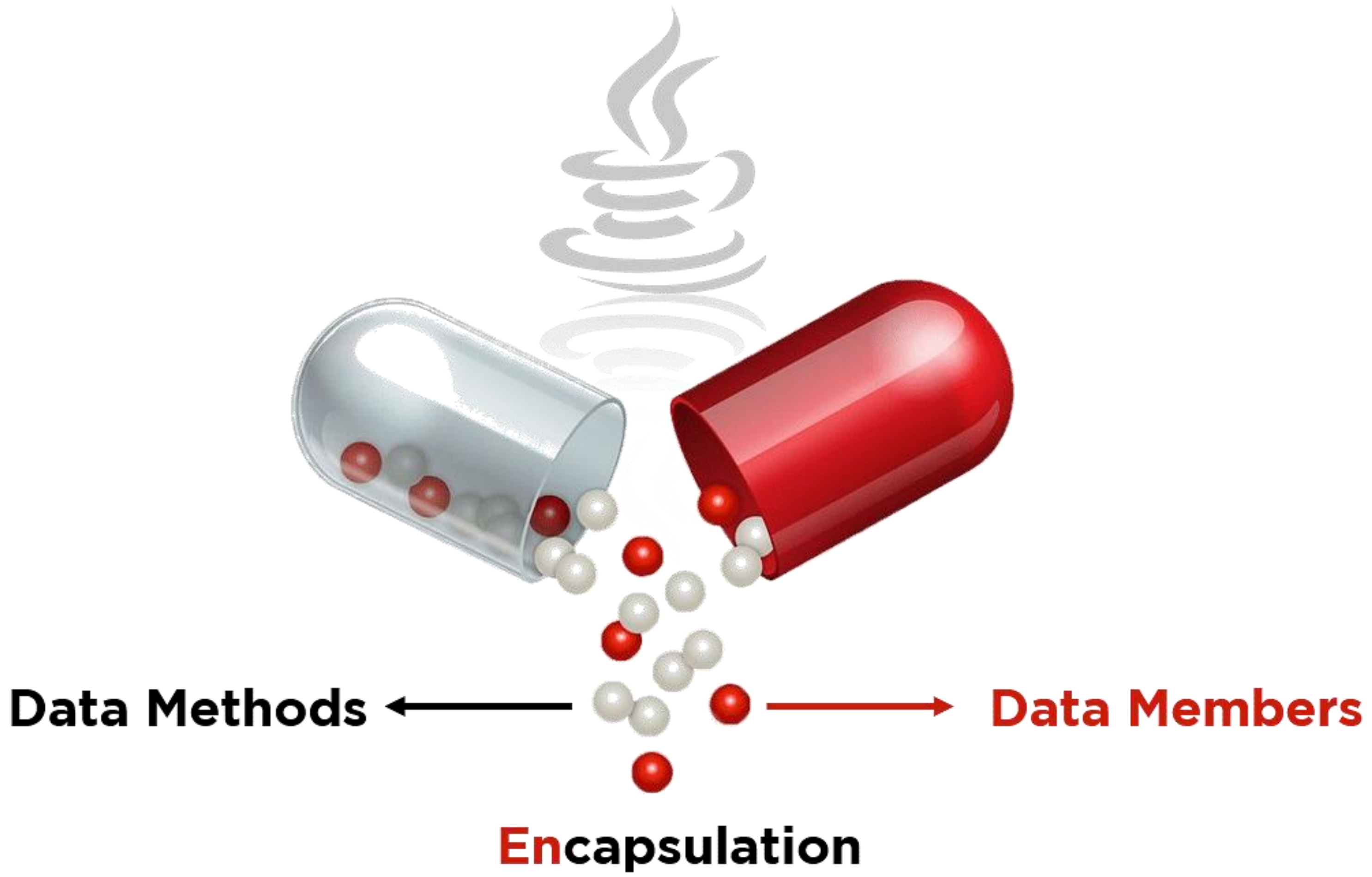
Object is an instance of a class. An object in OOPS is nothing but a self-contained component which consists of methods and properties to make a particular type of data useful. For example color name, table, bag, barking. When you send a message to an object, you are asking the object to invoke or execute one of its methods as defined in the class.

From a programming point of view, an object in OOPS can include a data structure, a variable, or a function. It has a memory location allocated. Java Objects are designed as class hierarchies.



```
className variable_name = new className();
```

Encapsulation in Java



Java Access Specifiers

Access Specifier	Inside Class	Inside Package	Outside package subclass	Outside package
Private	Yes	No	No	No
Default	Yes	Yes	No	No
Protected	Yes	Yes	Yes	No
Public	Yes	Yes	Yes	Yes

Message Passing

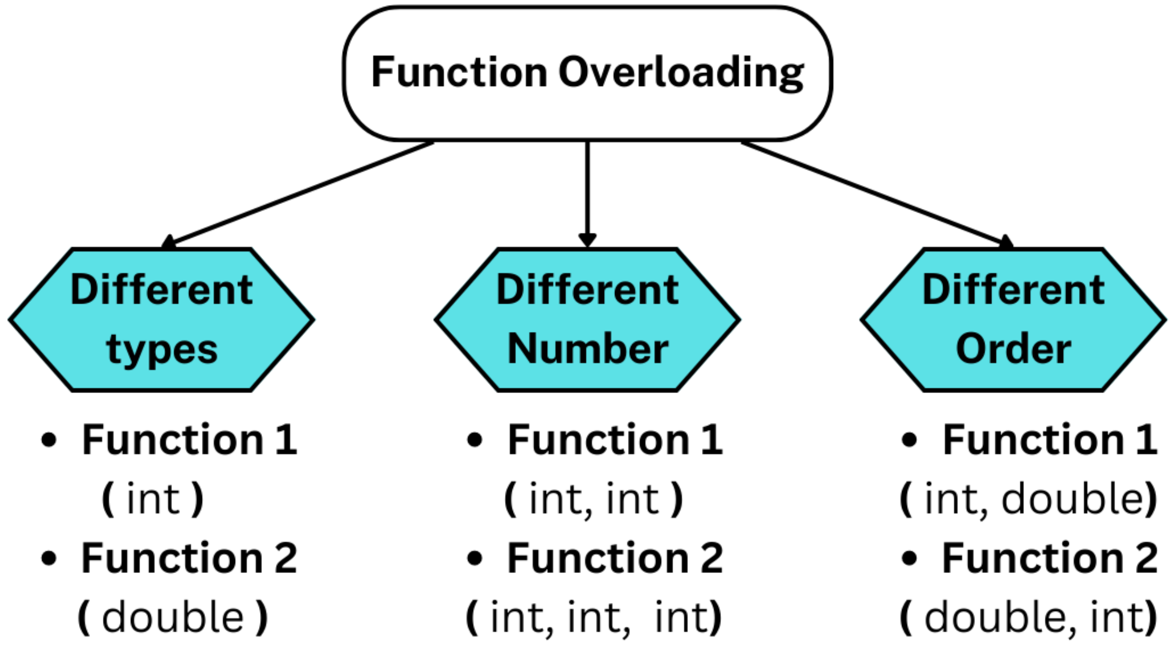
- A request for an object to perform one of its operations is called a message.
- Operation means method/function. A message cannot go automatically it creates an interface, which means it creates an interface for an object. The interface provides the abstraction over the message means hide the implementation.
- An interface is a set of operations that a given object can perform.
- All communication between objects is done via message is called message passing,

Constructors

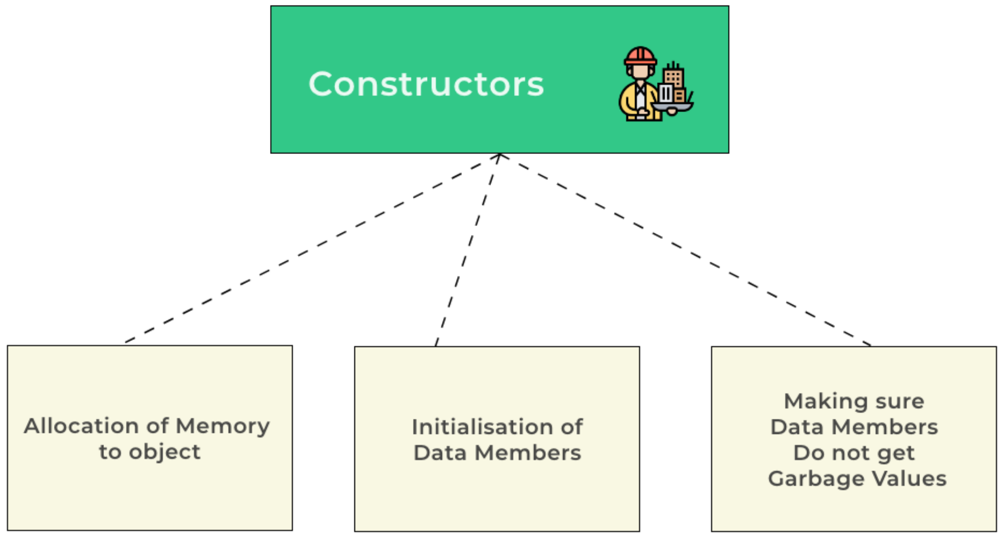
- A constructor is a special method of a class or structure in object-oriented programming that initializes a newly created object of that type.
- Whenever an object is created, the constructor is called automatically.
- Constructors are not called explicitly and are invoked only once during their lifetime

Function Overloading

- Java lets you specify more than one function of the same name in the same scope. These functions are called overloaded functions, or overloads. Overloaded functions enable you to supply different semantics for a function, depending on the types and number of its arguments.



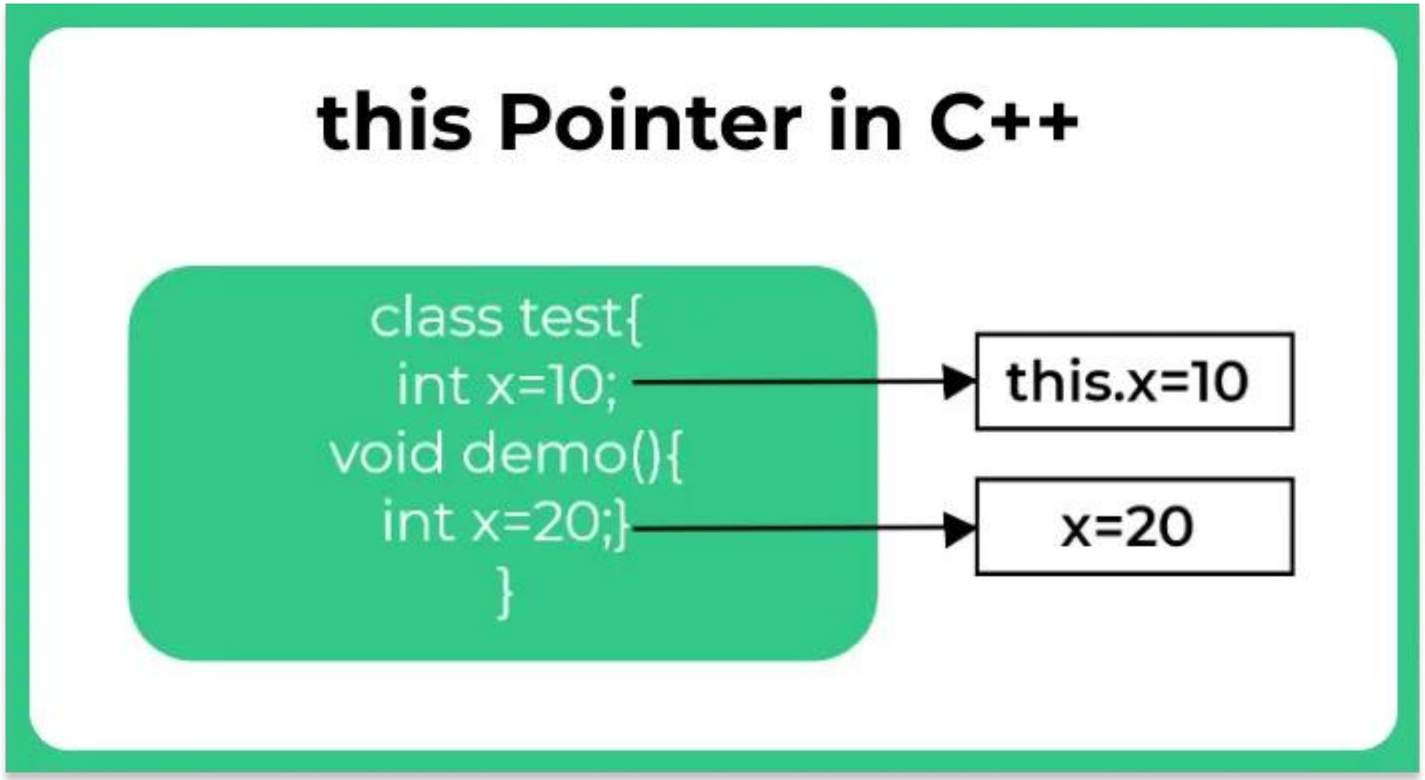
Constructors in C++


Constructors
Allocation of Memory to object
Initialisation of Data Members
Making sure Data Members Do not get Garbage Values

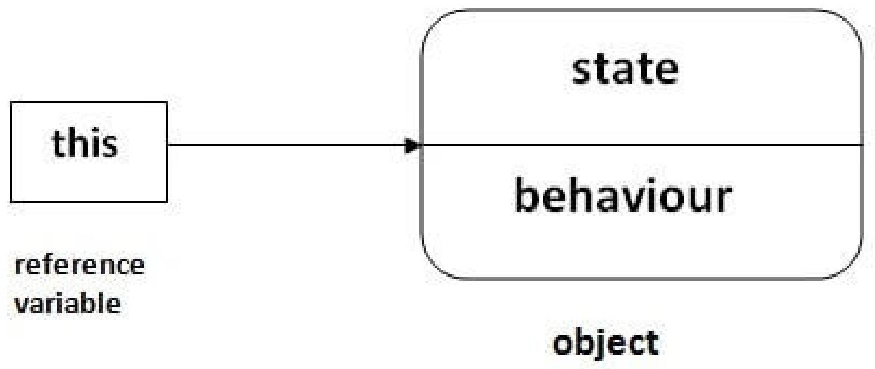
Constructors are a unique class functions that do the job of initialising every object. Whenever, any object is created the constructor is called by the compiler.

this Pointer

In C++ programming, this is a keyword that refers to the current instance of the class



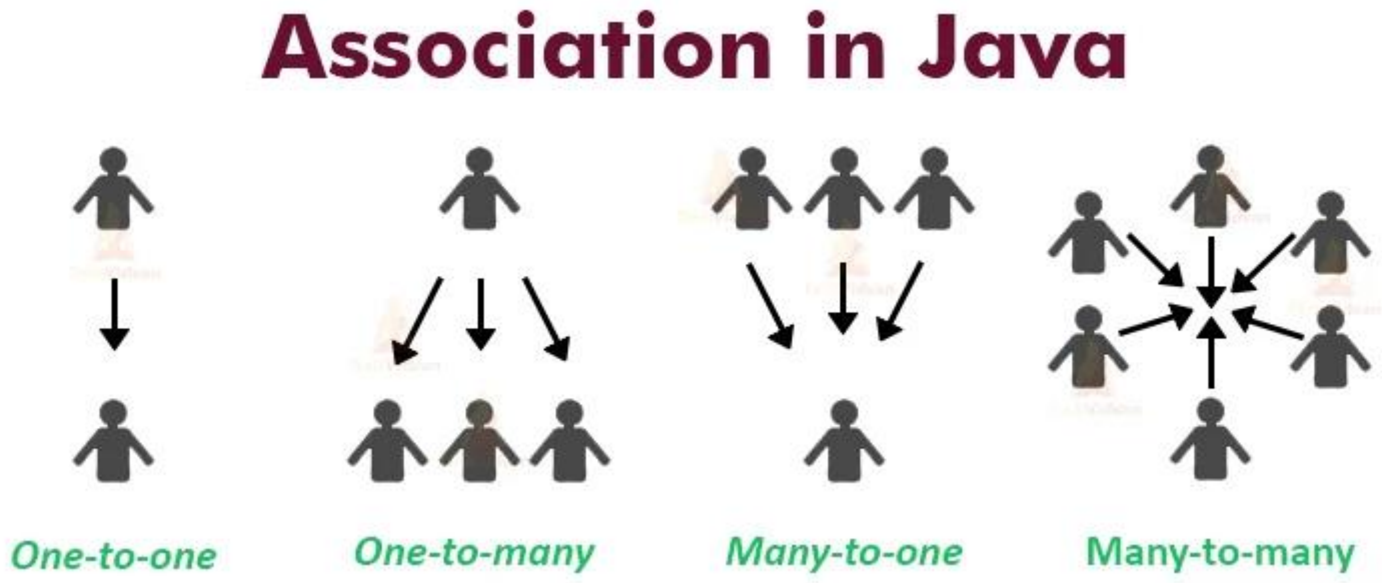
this keyword in Java



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

Association

- Association is a relation between two separate classes which establishes through their Objects



Forms of Relationships

- There are main forms of relationships in OOP paradigm.
  - IS-A (Inheritance)   HAS-A (Association)

- IS-A Relationship   Car IS A Vehicle   Cat IS AN Animal   Manager IS AN Employee
- All these examples denote that sub type is in the form of super type creating is-a relationship.

- HAS-A Relationship Hotel HAS Rooms   School HAS Departments Employee HAS Address
- This represents the ownership of an entity in another. One entity belongs to another...

- There are $\boxed{2}$ forms of Association that are possible in Java:

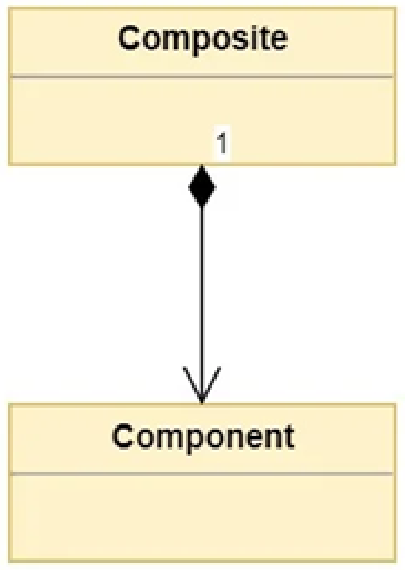
- a) Aggregation — loose coupling, weak
- b) Composition — tight coupling, strong

Composition

The composition means having a container object that contains other objects.

Composition is a concept that models a has a relationship. This means that a class Composite can contain an object of another class Component. This relationship means that a Composite has a Component.

Classes that contain objects of other classes are usually referred to as composites, where classes that are used to create more complex types are referred to as components



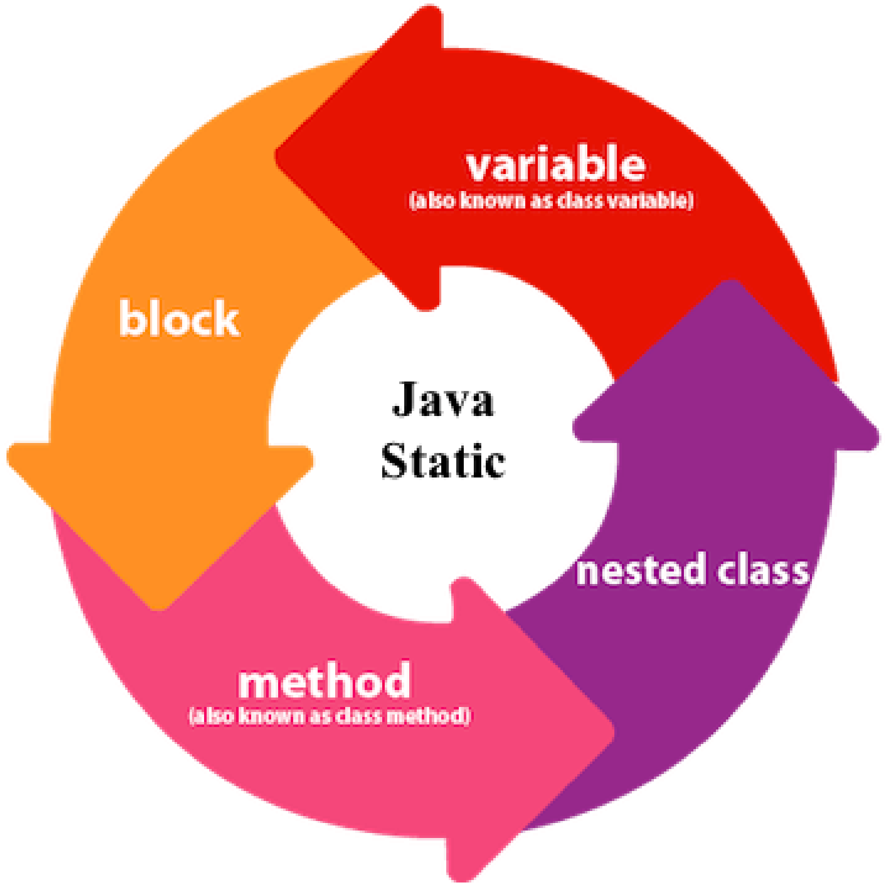
Aggregation

Aggregation is an association that describes “has a” relationship between objects. For example, a classroom and student are linked with “has a” relationship

Difference Between Composition & Aggregation :-

Aggregation is an association between two objects which describes the “has a” relationship while the composition is the most specific type of aggregation that implies ownership. Thus, this is the main difference between aggregation and composition.
In aggregation, destroying the owning object does not affect the containing object. However, in composition, destroying the owning object affects the containing object. Hence, this is another difference between aggregation and composition.

Java static members



static variables

Static variables defined as a class member can be accessed without object of that class. Static variable is initialized once and shared among different objects of the class. All the object of the class having static variable will have the same instance of static variable.

Static variable	Instance Variable
Represent common property	Represent unique property
Accessed using class name (can be accessed using object name as well)	Accessed using object
Allocated memory only once	Allocated new memory each time a new object is created

Java 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 the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.

Static block in Java

- Static block is used to initialize static data members. Static block executes even before main() method.

- It executes when the class is loaded in the memory.

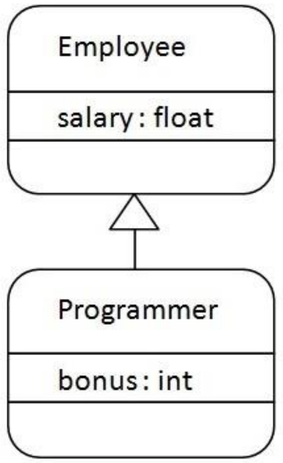
- A class can have multiple Static blocks, which will execute in the same sequence in which they are programmed

Inheritance in Java

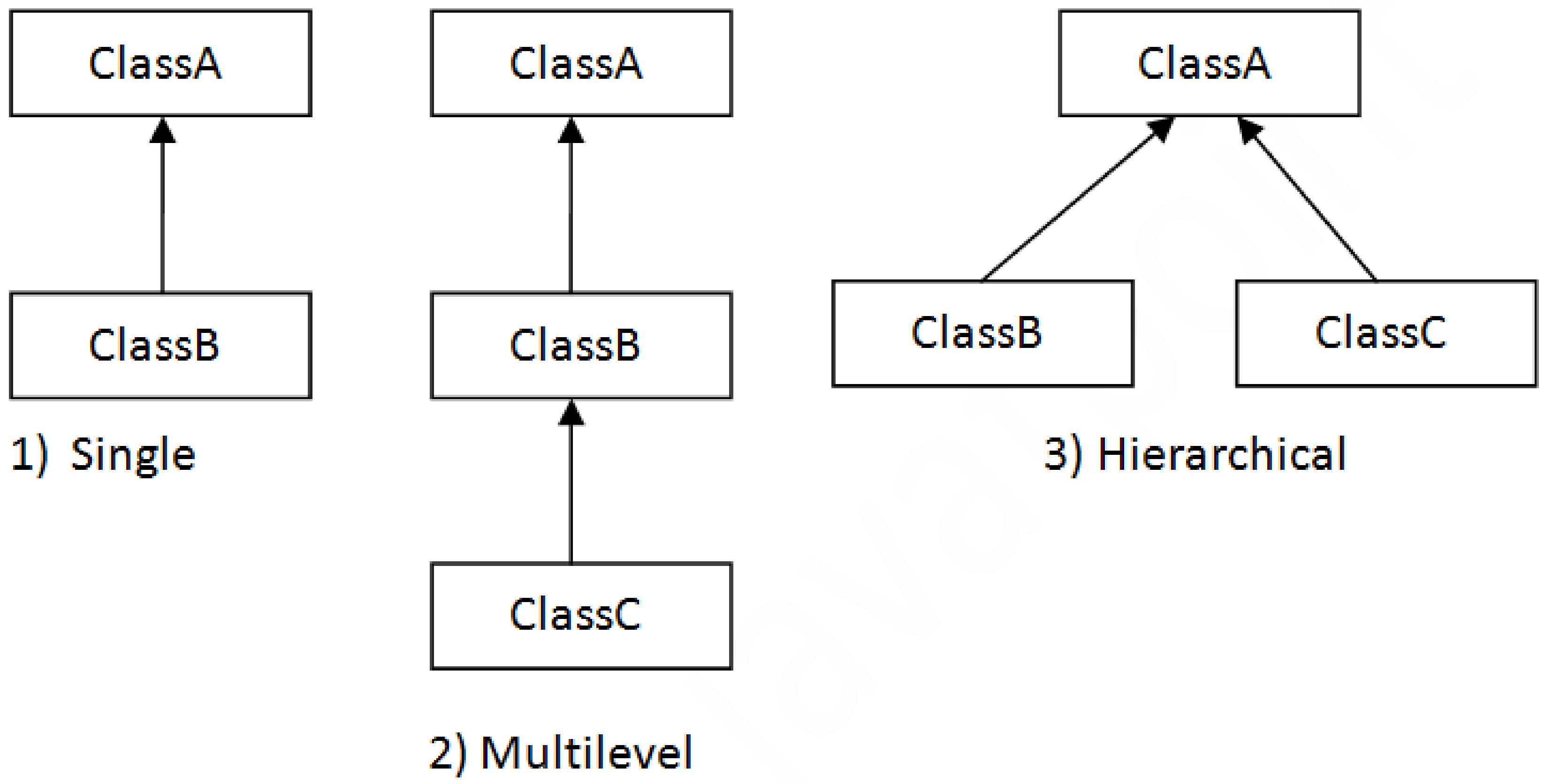
- Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

- The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

- Inheritance represents the IS-A relationship which is also known as a
- *parent-child* relationship.

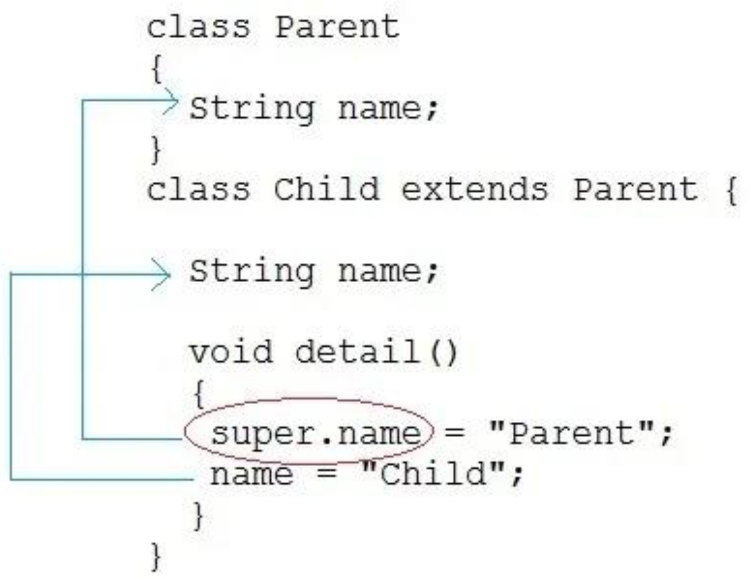


Types of Inheritances



super keyword

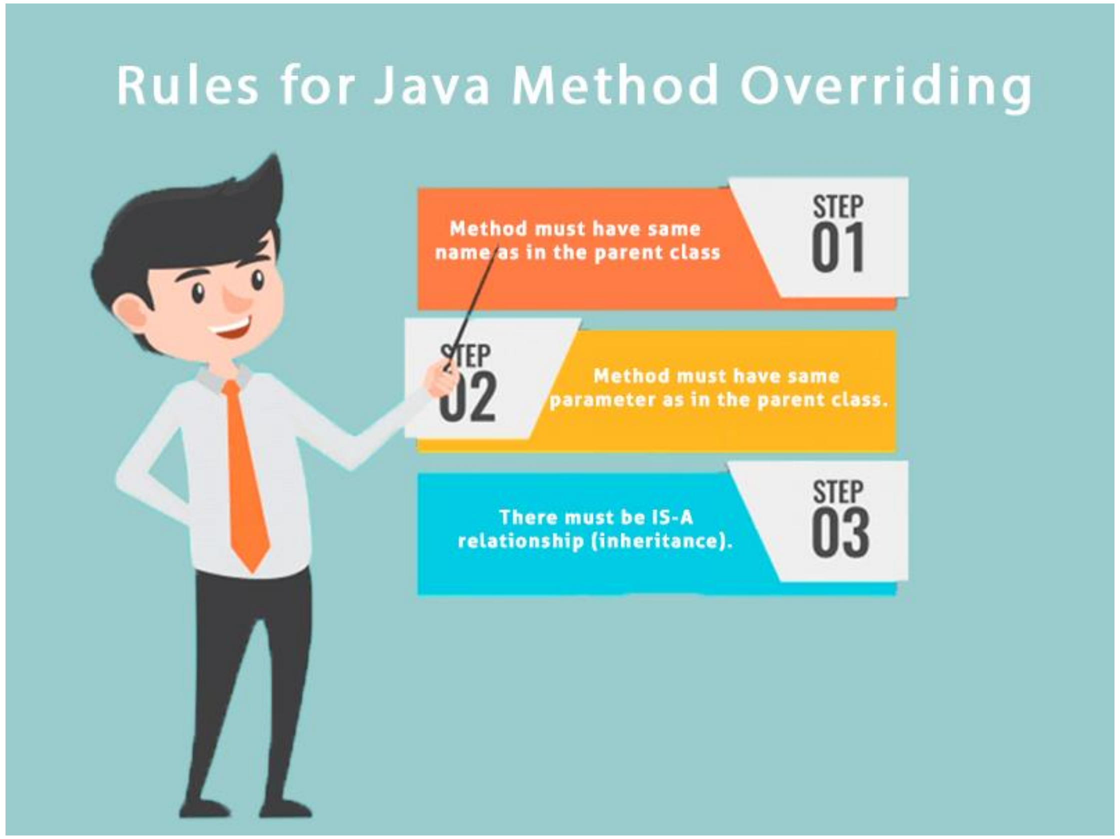
In Java, super keyword is used to refer to immediate parent class of a child class. In other words super keyword is used by a subclass whenever it need to refer to its immediate super class.



```
class Parent
{
  String name;
}
class Child extends Parent {

  String name;

  void detail()
  {
   super.name = "Parent";
   name = "Child";
  }
}
```

Method Overriding

If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java.


Rules for Java Method Overriding
Method must have same name as in the parent class
STEP 01
STEP 02
Method must have same parameter as in the parent class.
There must be IS-A relationship (inheritance).
STEP 03

Method Overriding

- When we override a method, it's recommended to put @Override, immediately above the method definition.
- The @Override statement is not required, but it's a way to get the compiler to flag an error, if you don't actually properly override this method.
- We'll get an error, if we don't follow the overriding rules correctly.
- We can't override static methods, only instance methods can be overridden

Method Overriding Rules

A method will be considered overridden, if we follow these rules.

- It must have the same name and same arguments.
- The return type can be a subclass of the return type in the parent class.
- It can't have a lower access modifier. In other words, it can't have more restrictive access privileges.
- For example, if the parent's method is protected, then using private in the child's overridden method is not allowed. However, using public for the child's method would be allowed, in this example.

Method Overriding vs. Overloading

OVERRIDING

same name
same parameters

```
class Dog {

    public void bark() {
        System.out.println("woof");
    }
}

class GermanShepherd extends Dog {

    @Override
    public void bark() {
        System.out.println("woof woof woof");
    }
}
```

OVERLOADING

same name
different parameters

```
class Dog {

    public void bark() {
        System.out.println("woof");
    }

    public void bark(int number) {
        for(int i = 0; i < number; i++) {
            System.out.println("woof");
        }
    }
}
```

Method Overriding vs. Overloading

Method Overloading	**Method Overriding**
Provides functionality to reuse a method name with different parameters.	Used to override a behavior which the class has inherited from the parent class.
Usually in a single class but may also be used in a child class.	**Always in two classes** that have a child-parent or IS-A relationship.
Must have different parameters.	**Must have** the same parameters and same name.
May have different return types.	**Must have** the same return type or covariant return type(child class).
May have different access modifiers(private, protected, public).	**Must NOT** have a lower modifier but may have a higher modifier.
May throw different exceptions.	**Must NOT** throw a new or broader checked exception.

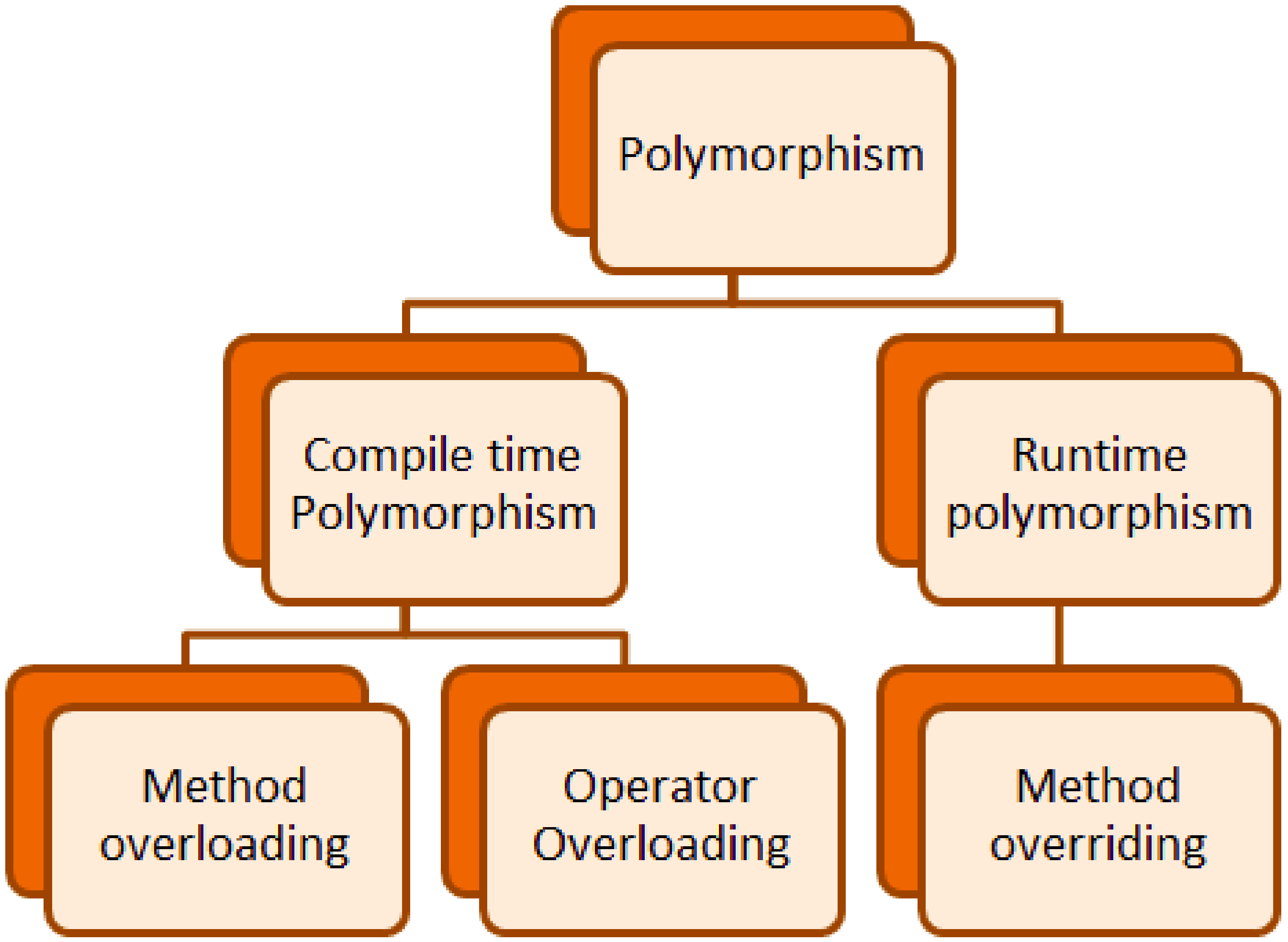
final modifier

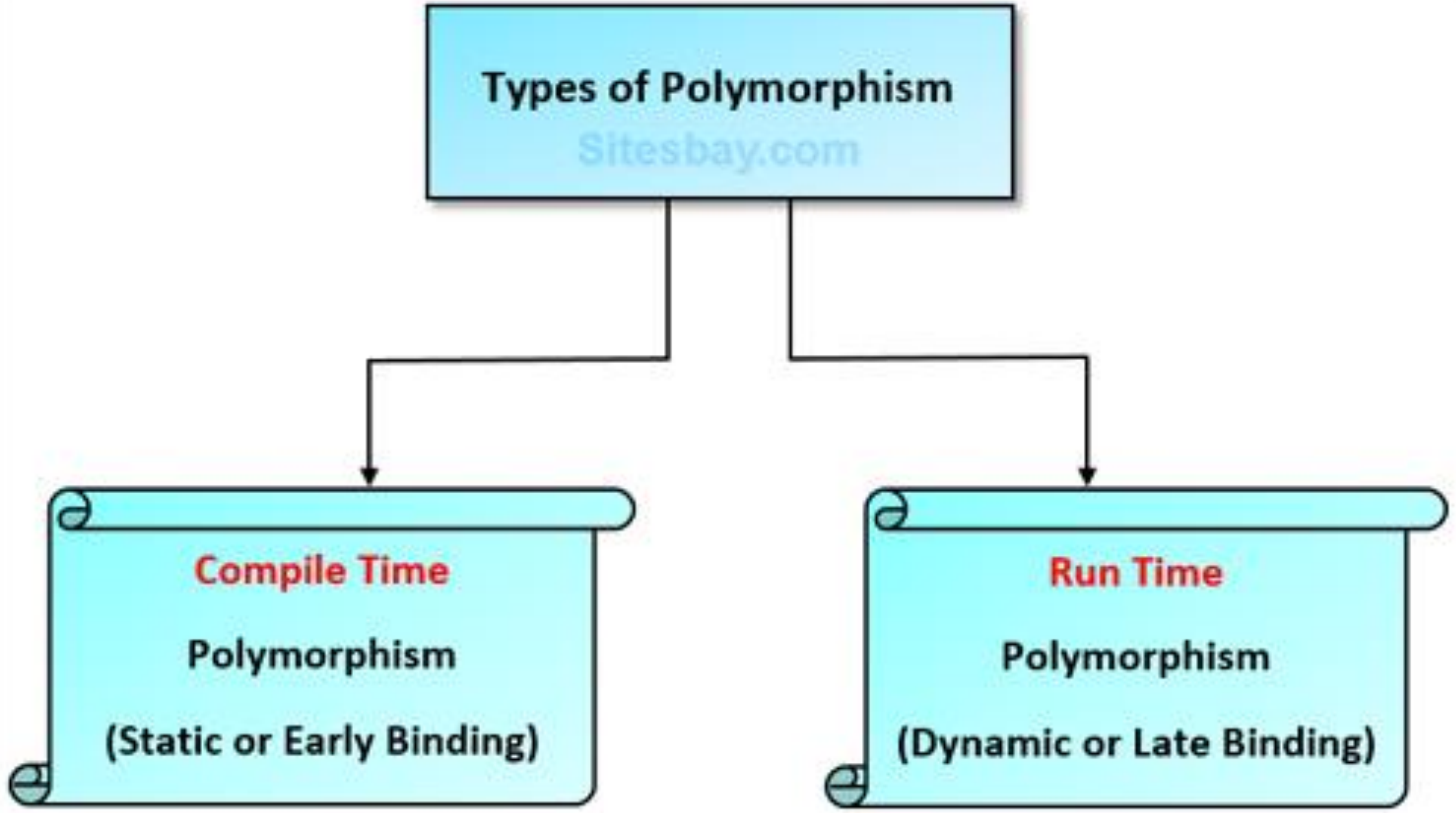
Final modifier is used to declare a field as final. It can be used with variable, method or a class.

- If we declare a variable as final then it prevents its content from being modified. The variable acts like a constant. Final field must be initialized when it is declared. Final variable that is not initialized at the time of declaration is called blank final variable. Java allows to declare a final variable without initialization but it should be initialized by the constructor only

- If we declare a method as final then it prevents it from being overridden.

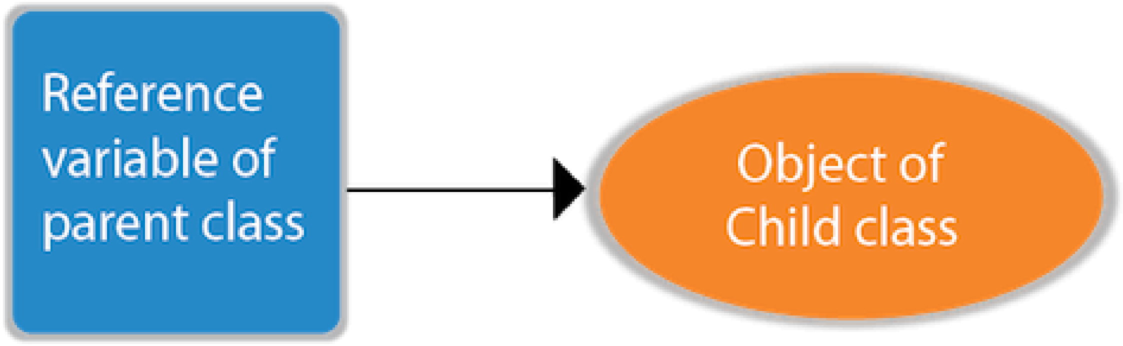
- If we declare a class as final the it prevents from being inherited. We can not inherit final class in Java.


Polymorphism
Compile time Polymorphism
Runtime polymorphism
Method overloading
Operator Overloading
Method overriding


Types of Polymorphism
Sitesbay.com
Compile Time
Polymorphism
(Static or Early Binding)
Run Time
Polymorphism
(Dynamic or Late Binding)

Dynamic Polymorphism

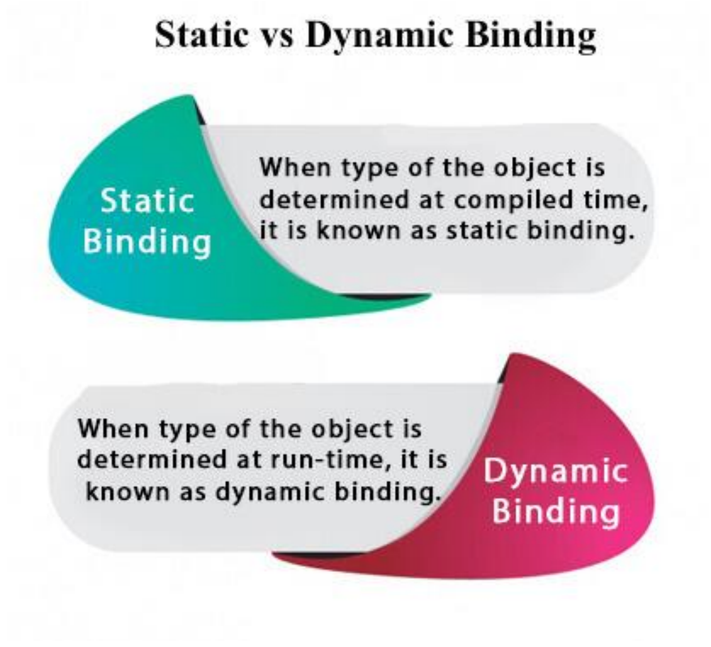
Runtime polymorphism or Dynamic Method Dispatch is a process in which a call to an overridden method is resolved at runtime rather than compile-time.



static binding and dynamic binding

Connecting a method call to the method body is known as binding.
There are two types of binding

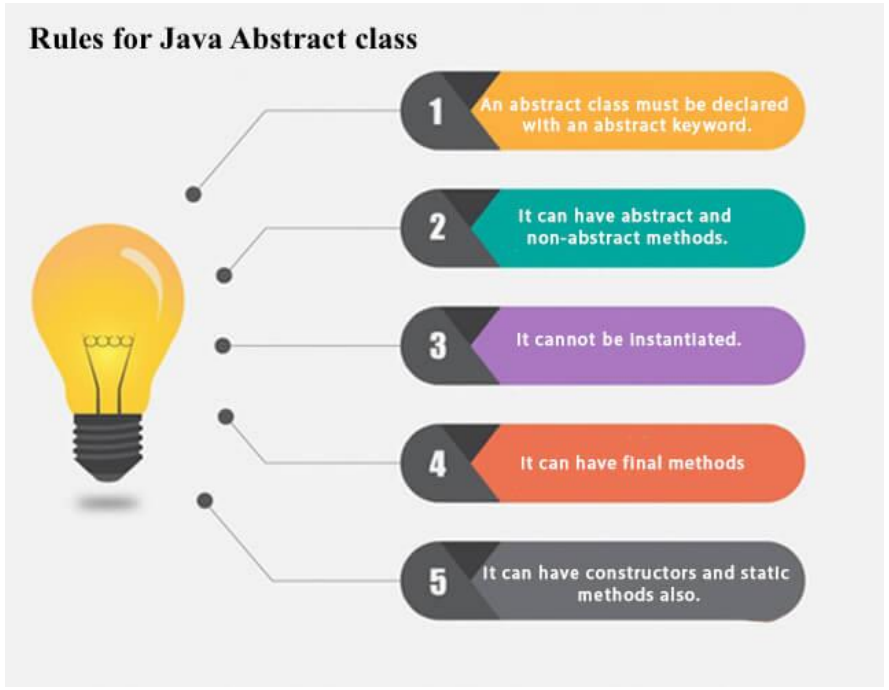
- Static Binding (also known as Early Binding).
- Dynamic Binding (also known as Late Binding).


Static vs Dynamic Binding
Static Binding
When type of the object is determined at compiled time, it is known as static binding.
When type of the object is determined at run-time, it is known as dynamic binding.
Dynamic Binding

Abstract class

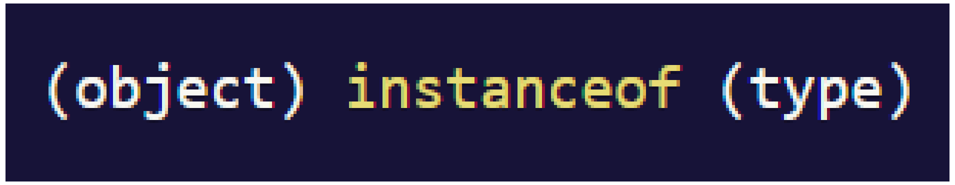
Abstraction is a process of hiding the implementation details and showing only functionality to the user.

A class which is declared using abstract keyword known as abstract class. An abstract class may or may not have abstract methods. We cannot create object of abstract class.


Rules for Java Abstract class
1 An abstract class must be declared with an abstract keyword.
2 It can have abstract and non-abstract methods.
3 It cannot be instantiated.
4 It can have final methods
5 It can have constructors and static methods also.

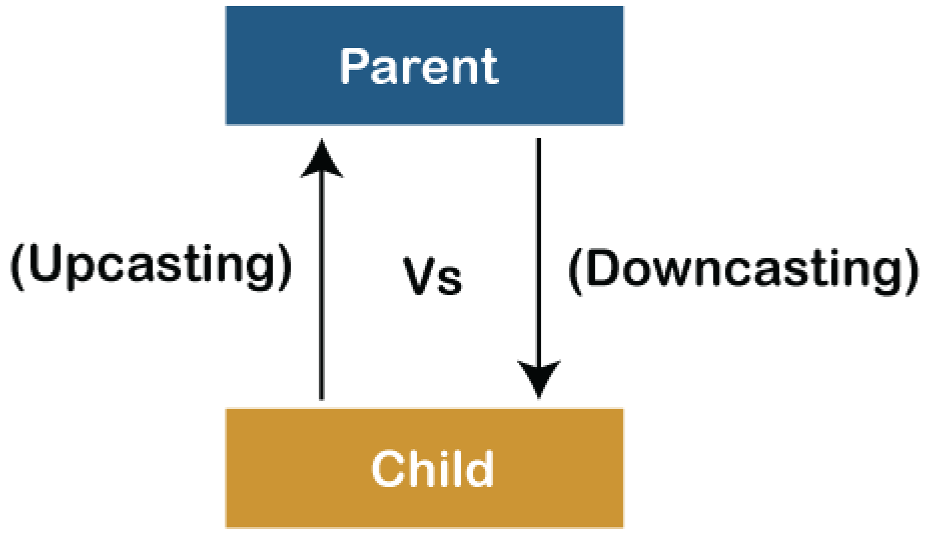
instance of operator

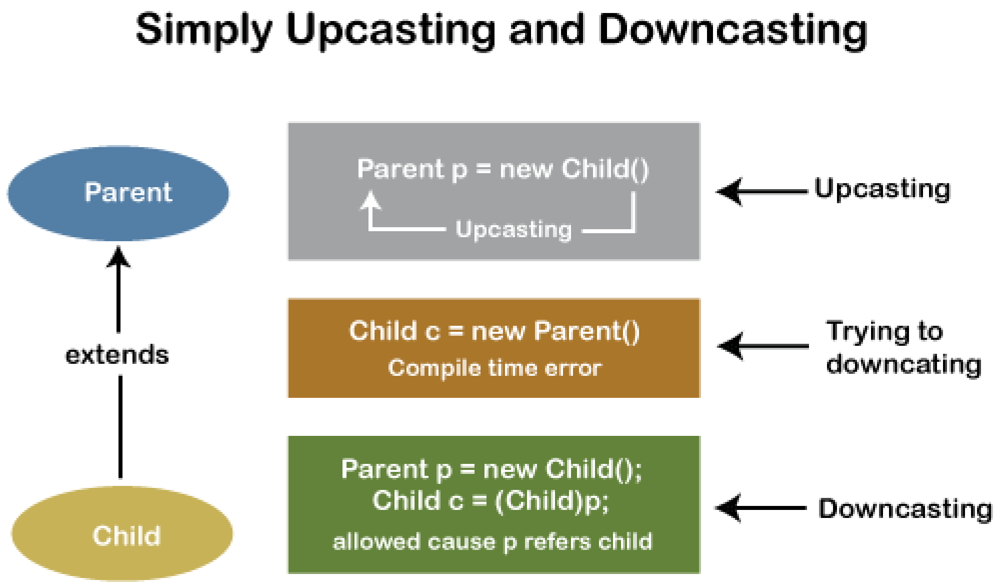
In Java, instanceof is an operator which is used to check object reference. It checks whether the reference of an object belongs to the provided type or not. It returns either true or false, if an object reference is of specified type then it return true otherwise false.



```
(object) instanceof (type)
```

Downcasting and Upcasting





var type

var, is a special contextual keyword in Java, that lets our code take advantage of Local Variable Type Inference.

By using var as the type, we're telling Java to figure out the compile-time type for us.

Local Variable Type Inference was introduced in Java 10.

One of the benefits is to help with the readability of the code, and to reduce boilerplate code.

It's called Local Variable Type Inference for a reason, because:

- It can't be used in field declarations on a class.
- It can't be used in method signatures, either as a parameter type or a return type.
- It can't be used without an assignment, because the type can't be inferred in that case.
- It can't be assigned a null literal, again because a type can't be inferred in that case.

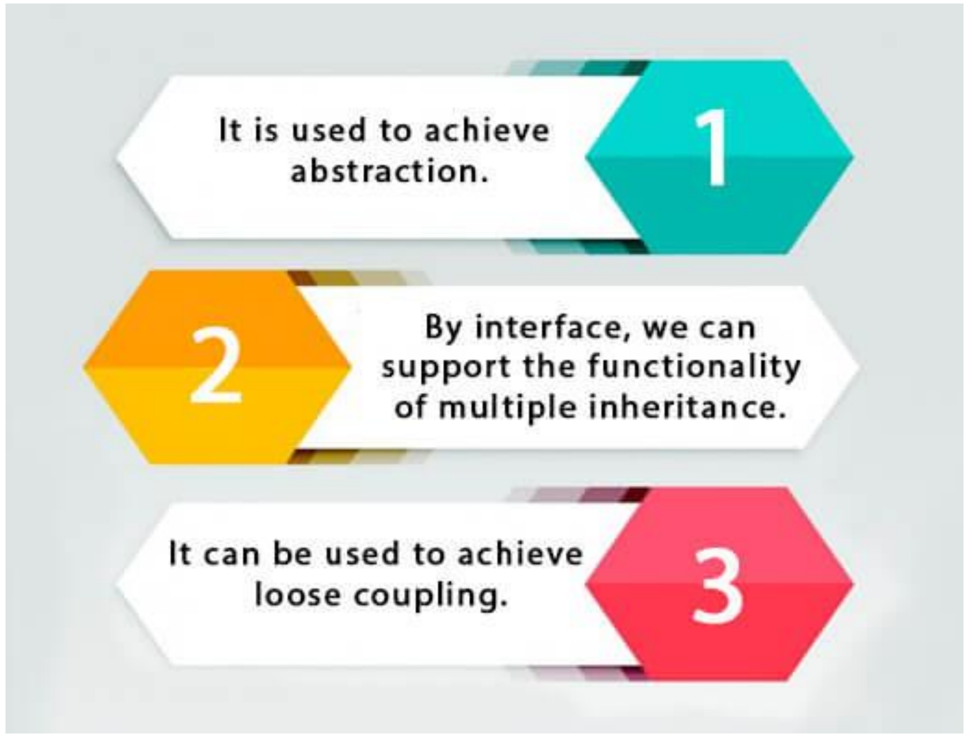
Java Interfaces

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

The interface in Java is *a mechanism to achieve abstraction*. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

Since Java 8, we can have **default and static methods** in an interface.

Since Java 9, we can have **private methods** in an interface.


It is used to achieve abstraction.
1
2
By interface, we can support the functionality of multiple inheritance.
It can be used to achieve loose coupling.
3

Purpose of Java Interfaces

An interface in Java is a specification of method prototypes. Whenever you need to guide the programmer or, make a contract specifying how the methods and fields of a type should be you can define an interface

Multiple inheritance – Java doesn’t support multiple inheritance, using interfaces you can achieve multiple inheritance –

Abstraction – Abstraction is a process of hiding the implementation details from the user, only the functionality will be provided to the user. In other words, the user will have the information on what the object does instead of how it does it.

Since all the methods of the interface are abstract and user doesn’t know how a method is written except the method signature/prototype. Using interfaces, you can achieve (complete) abstraction.

Loose coupling – Coupling refers to the dependency of one object type on another, if two objects are completely independent of each other and the changes done in one doesn’t affect the other both are said to be loosely coupled.

You can achieve loose coupling in Java using interfaces –

Default Methods in Interface

- With the release of Java 8, we can now add methods with implementation inside an interface. These methods are called default methods.

```
public default void getSides() {
        // body of getSides()
}
```

- The Java 8 also added another feature to include static methods inside an interface.

- With the release of Java 9, private methods are also supported in interfaces. private methods are used as helper methods that provide support to other methods in interfaces.

Functional Interface

- An Interface that contains exactly one abstract method is known as functional interface. It can have any number of default, static methods but can contain only one abstract method.
- Functional Interface is also known as Single Abstract Method Interfaces or SAM Interfaces
- Runnable, ActionListener, and Comparable are some of the examples of functional interfaces.
- Functional interfaces are used and executed by representing the interface with an annotation called @FunctionalInterface.

Lambda Expression

- Lambda Expression is an anonymous (nameless) function. In other words, the function which doesn’t have the name, return type and access modifiers. Lambda Expression is also known as anonymous functions or closures.
- The primary Objective of introducing Lambda Expression is to promote the benefits of functional programming in Java

```
(parameters) -> expression

(parameters) -> { statements; }

() -> expression
```

Abstract class vs. Interface

Abstract class	Interface
Abstract class is a class which contain one or more abstract methods, which has to be implemented by its sub classes.	Interface is a Java Object containing method declaration but no implementation. The classes which implement the Interfaces must provide the method definition for all the methods.
Abstract class is a Class prefix with an abstract keyword followed by Class definition.	Interface is a pure abstract class which starts with interface keyword.
Abstract class can also contain concrete methods.	Whereas, Interface contains all abstract methods and final variable declarations.
Abstract classes are useful in a situation that Some general methods should be implemented and specialization behavior should be implemented by child classes.	Interfaces are useful in a situation that all properties should be implemented.

Nested classes

- Java inner class or nested class is a class that is declared inside the class or interface.
- We use inner classes to logically group classes and interfaces in one place to be more readable and maintainable.
- Additionally, it can access all the members of the outer class, including private data members and methods.

There are two types of nested classes you can create in Java.

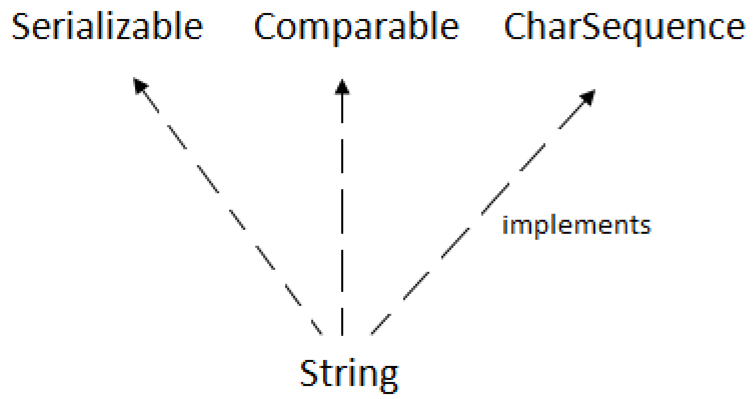
Non-static nested class (inner class)
Static nested class

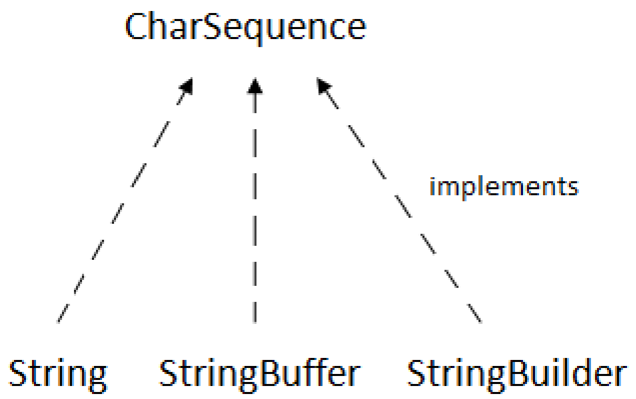
Nested classes

Type	Description
Member Inner Class	A class created within class and outside method.
Anonymous Inner Class	A class created for implementing an interface or extending class. The java compiler decides its name.
Local Inner Class	A class was created within the method.
Static Nested Class	A static class was created within the class.
Nested Interface	An interface created within class or interface.

Java Strings

String is Sequence of Characters represented as an Object

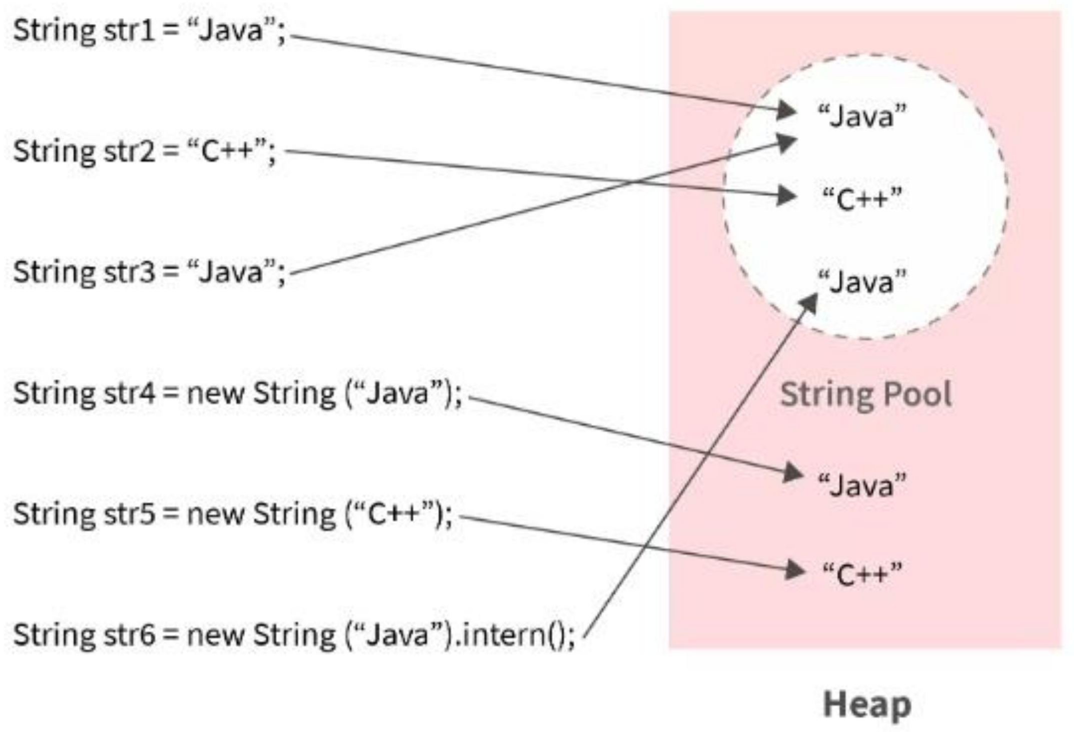




The Java String is immutable which means it cannot be changed. Whenever we change any string, a new instance is created.

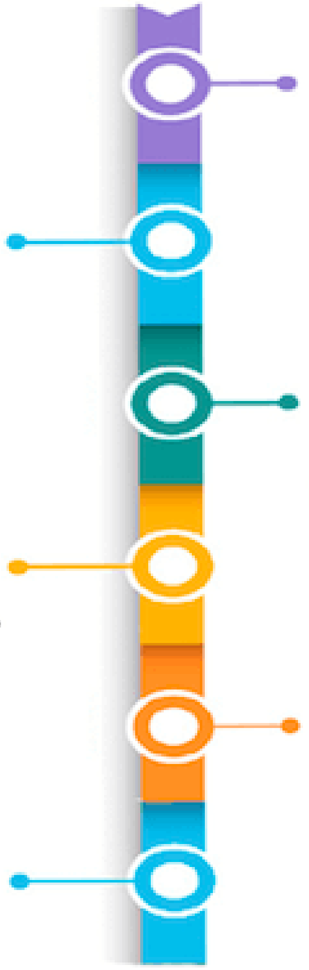
For mutable strings, you can use StringBuffer and StringBuilder classes.

String Pool


String str1 = "Java";
String str2 = "C++";
String str3 = "Java";
String str4 = new String ("Java");
String str5 = new String ("C++");
String str6 = new String ("Java").intern();
"Java"
"C++"
"Java"
String Pool
"Java"
"C++"
Heap

- All Strings are stored in the String Pool (or String Intern Pool) that is allocated in the Java heap.
- String pool is an implementation of the String Interring Concept.
- String Interning is a method that stores only a copy of each distinct string literal.
- The distinct values are stored in the String pool.
- String pool is an example of the Flyweight Design Pattern

StringBuffer vs String



String class is immutable.

StringBuffer class is mutable.

String is slow and consumes more memory when you concat too many strings because every time it creates new instance.

StringBuffer is fast and consumes less memory when you cancat strings.

String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.

StringBuffer class doesn't ' override the equals() method of Object class.

StringBuffer vs StringBuilder

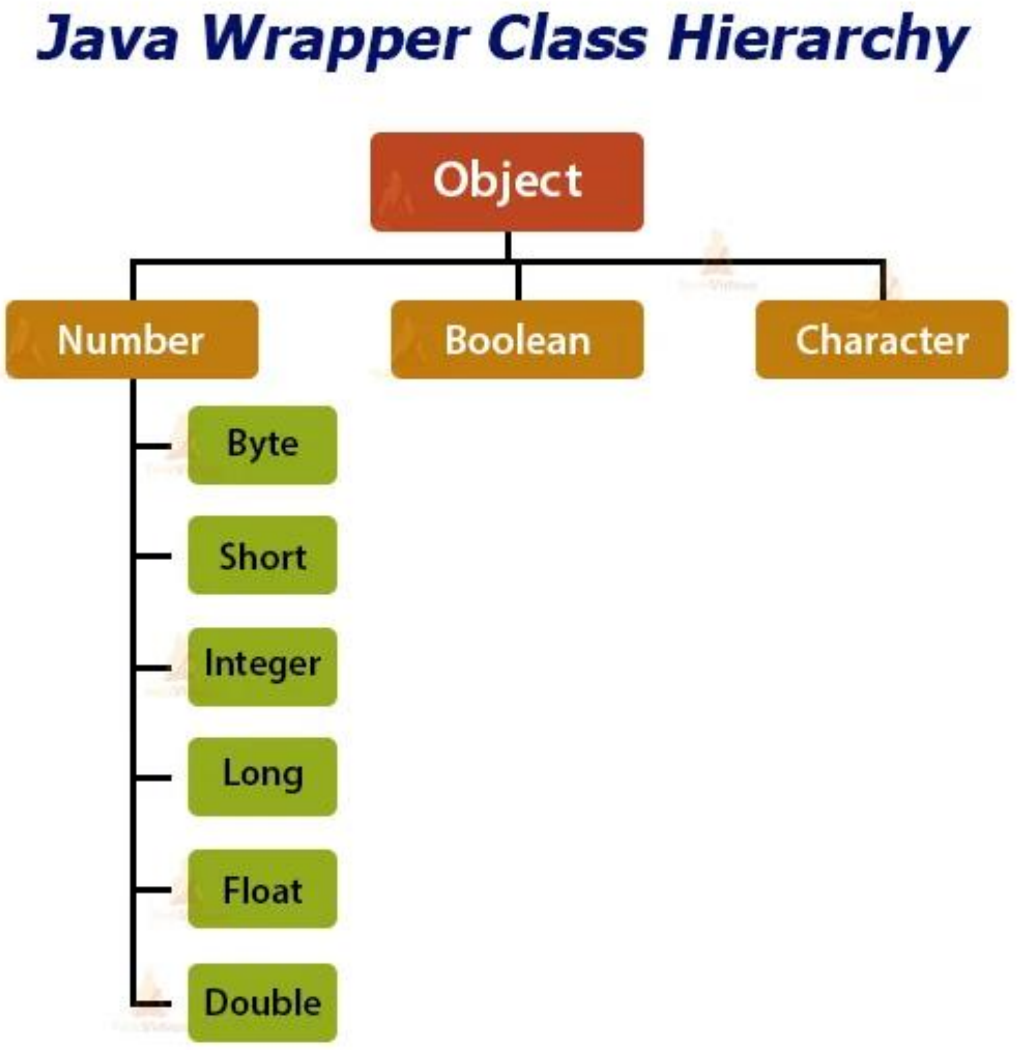
StringBuffer	StringBuilder
1. Thread-Safe	1. Not Thread-Safe
2. Synchronized	2. Not Synchronized
3. Since Java 1.0	3. Since Java 1.5
4. Slower	4. Faster

Java Object class

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

• Java Object getClass()	returns the class name of the object
• Java Object hashCode()	returns the hashcode value of the object
• Java Object toString()	converts an object into the string
• Java Object equals()	checks if two objects are equal
• Java Object clone()	creates a copy of the object

Java Wrapper Classes



Java Exception

An expectation is an unexpected event that occurs while executing the program, that disturbs the normal flow of the code

. It can occur for various reasons say-

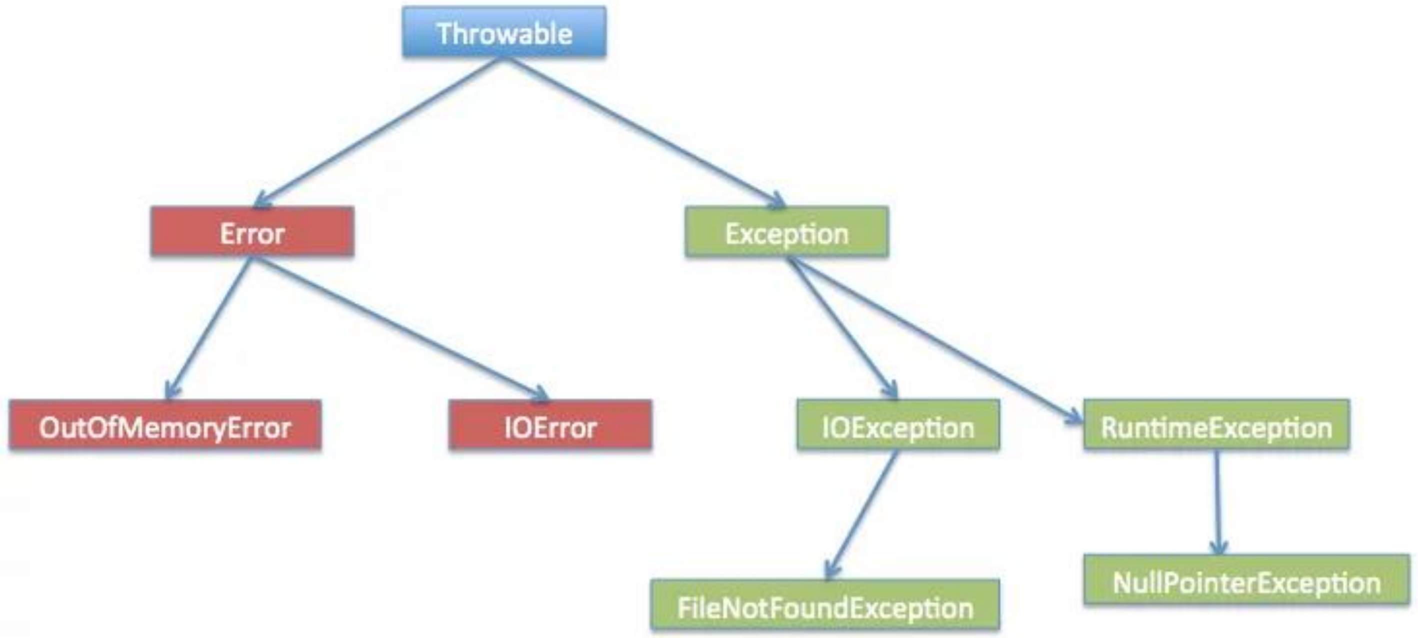
- A user has entered an invalid data
- File not found
- A network connection has been lost in the middle of communications
- The JVM has run out of a memory

Exception Handling

The Exception Handling in Java is a mechanism using which the normal flow of the application is maintained. To do this, we employ a powerful mechanism to handle runtime errors or exceptions in a program.

Java exception handling provides a meaningful message to the user about the issue rather than a system generated message, which may not be understandable to a user

Exception Hierarchy



Types of Exceptions

Checked Exception

These are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using the throws keyword.

Unchecked Exception

These are the exceptions that are not checked at compile time.

Program on Loops

1. Count of prime no’s in given range

2. WAP to generate the max number using the digits of entered number
   4936 --> 9643

3. Enter the number and add the alternate digit from it.
   no=28416 ==> 6+4+2 = 12
   1+8 = 9

4. Write a program to print the Floyd's Triangle.

```
1
01
101
0101
10101
```

Program on Arrays

1. Write a program to find second largest element in an array.
2. Write a program to count frequency of each element in an array.
3. Write a program to print all unique elements in the array.
4. Write a program to count total number of duplicate elements in an array.

Inheritance

Design and develop inheritance for a given case study, identify objects and relationships and implement inheritance wherever applicable.

Employee class has Emp_name, Emp_id,
Address, Mail_id, and Mobile_no as members. Inherit the classes: Programmer, Team Lead, Assistant Project Manager and Project Manager from employee class.

Add Basic Pay (BP) as
the member of all the inherited classes . with 97% of BP as DA, 10 % of BP as HRA, 12% of BP as PF, 0.1% of BP for staff club fund. Generate pay slips for the employees with their gross and net salary

What does it mean when Java defaults the data type to an int?

So, what effect does int, being the default value, have on our code?

Looking at the scenarios we just looked at in summary, we know the following:

This statement works because the result is an int, and assigning it to an int variable is fine.

```
int myTotal = (myMinIntValue / 2);
```

This statement doesn't work, because the expression (myMinShortValue / 2) is an int, and an int can't be assigned to a short, because the compiler won't guess the result.

```
short myNewShortValue = (myMinShortValue / 2);
```

What does it mean when Java defaults the data type to an int?

This statement works, because the result of (-128 / 2) is an int, but when calculations use only literal values, the compiler can determine the result immediately, and knows the value fits into a short.

```java
short myNewShortValue = (-128 / 2);
```

And finally, this code works because we tell the compiler we know what we're doing by using this cast, and the compiler doesn't give an error.

```java
short myNewShortValue = (short) (myMinShortValue / 2);
```

Leap Year Calculator – Method Ex.

Write a method isLeapYear with a parameter of type int named year.

> The parameter needs to be greater than or equal to 1 and less than or equal to 9999.
> If the parameter is not in that range return false.

Otherwise, if it is in the valid range, calculate if the year is a leap year and return true if it is a leap year, otherwise return false.

To determine whether a year is a leap year, follow these steps:
1. If the year is evenly divisible by 4, go to step 2. Otherwise, go to step 5.
2. If the year is evenly divisible by 100, go to step 3. Otherwise, go to step 4.
3. If the year is evenly divisible by 400, go to step 4. Otherwise, go to step 5.
4. The year is a leap year (it has 366 days). The method isLeapYear needs to return true.
5. The year is not a leap year (it has 365 days). The method isLeapYear needs to return false.

The following years are not leap years:
1700, 1800, 1900, 2100, 2200, 2300, 2500, 2600
This is because they are evenly divisible by 100 but not by 400.

The following years are leap years:
1600, 2000, 2400
This is because they are evenly divisible by both 100 and 400.

Examples of input/output:

isLeapYear(-1600); → should return false since the parameter is not in range (1-9999)

isLeapYear(1600); → should return true since 1600 is a leap year

isLeapYear(2017); → should return false since 2017 is not a leap year

isLeapYear(2000); → should return true because 2000 is a leap year

DecimalComparator Method-Ex

Write a method areEqualByThreeDecimalPlaces with two parameters of type double.

The method should return boolean and it needs to return true if two double numbers are the same up to three decimal places. Otherwise, return false.

EXAMPLES OF INPUT/OUTPUT:

areEqualByThreeDecimalPlaces(-3.1756, -3.175); → should return true since numbers are equal up to 3 decimal places.

areEqualByThreeDecimalPlaces(3.175, 3.176); → should return false since numbers are not equal up to 3 decimal places

areEqualByThreeDecimalPlaces(3.0, 3.0); → should return true since numbers are equal up to 3 decimal places.

areEqualByThreeDecimalPlaces(-3.123, 3.123); → should return false since numbers are not equal up to 3 decimal places.

Create two methods with the same name: convertToCentimeters

- The first method has one parameter of type int, which represents the entire height in inches. You'll convert inches to centimeters, in this method, and pass back the number of centimeters, as a double.
- The second method has two parameters of type int, one to represent height in feet, and one to represent the remaining height in inches. So if a person is 5 foot, 8 inches, the values 5 for feet and 8 for inches would be passed to this method. This method will convert feet and inches to just inches, then call the first method, to get the number of centimeters, also returning the value as a double.

Sum Calculator – class and object ex.

Write a class with the name **SimpleCalculator.** The class needs **two fields (instance variables)** with names **firstNumber** and **secondNumber** both of type **double**.

```
SimpleCalculator calculator = new SimpleCalculator();
calculator.setFirstNumber(5.0);
calculator.setSecondNumber(4);
System.out.println("add= " + calculator.getAdditionResult());
System.out.println("subtract= " + calculator.getSubtractionResult());
calculator.setFirstNumber(5.25);
calculator.setSecondNumber(0);
System.out.println("multiply= " + calculator.getMultiplicationResult());
System.out.println("divide= " + calculator.getDivisionResult());
```

Rectangle – class and object ex.

Write a class with the name **Rectangle.** The class needs **two fields (instance variables)** with names **length** and **width** both of type **double**.

```
Rectangle r=new Rectangle();

    r.setLength(10.5);//check with negative values.
    r.setBreadth(5.5);

    System.out.println("Area "+r.area());
    System.out.println("Perimeter "+r.perimeter());
    System.out.println("Is Square "+r.isSquare());

    System.out.println("Length "+r.getLength());
    System.out.println("Breadth "+r.getBreadth());
```

Cylinder – class and object ex.

Write a class with the name **Cylinder.** The class needs **two fields (instance variables)** with names **radius** and **height** both of type **double**.

```
Cylinder c=new Cylinder();
    c.setHeight(10);
    c.setRadius(7);
    c.setDimensions(10, 7);

    System.out.println("LidArea "+c.lidArea());
    System.out.println("Circumference "+c.perimeter());
    System.out.println("totalSurfaceArea "+c.drumArea());
    System.out.println("Volume "+c.volume());
    System.out.println("Height"+c.getHeight());
    System.out.println("Radius"+c.getRadius());
```

Point - Constructor

You have to represent a point in 2D space. Write a class with the name **Point.** The class needs **two fields (instance variables)** with name **x** and **y** of type **int**.
The class needs to have two constructors. The first constructor does not have any parameters (no-arg constructor). The second constructor has parameters **x** and **y** of type **int** and it needs to initialize the fields.

To find a distance between points **A(xA,yA)** and **B(xB,yB)**, we use the formula:
d(A,B)=√ (xB − xA) * (xB - xA) + (yB − yA) * (yB - yA)

```
Point first = new Point(6, 5);
Point second = new Point(3, 1);
System.out.println("distance(0,0)= " + first.distance());
System.out.println("distance(second)= " + first.distance(second));
System.out.println("distance(2,2)= " + first.distance(2, 2));
Point point = new Point();
System.out.println("distance()= " + point.distance());
```

Use Math.sqrt to calculate the square root √.

Composition

Class Customer : ID , Name , Gender, Age
Class Account : accountNo , Customer, balance

```
Account account = new Account(1, new Customer(2, “Rohit", ‘m’ , 45) ,10000.0);
        System.out.println(account);
        account.withdraw(100);
        System.out.println(account);
        System.out.println(account.getCustomerName());
        account.deposit(500);
        System.out.println(account);
```

Composition

Point class : X , Y with constructor and getter , setter methods

```
double distance(int x, int y)
double distance(Point another)
double distance()                    -- from 0,0
```

Circle class : Point center, int radius

```
Circle myCircle = new Circle(new Point(5, 8), 6);
    System.out.println(myCircle.distance(new Circle(new Point(30,46),2)));
    System.out.println(myCircle.getArea());
    System.out.println(myCircle.getCircumference());
```

- Write a class with the name Circle. The class needs one field (instance variable) with name radius of type double.

  - The class needs to have one constructor with parameter radius of type double and it needs to initialize the fields.
  - In case the radius parameter is less than 0 it needs to set the radius field value to 0.
  - Write the following methods (instance methods):
  - Method named getRadius without any parameters, it needs to return the value of radius field.
  - Method named getArea without any parameters, it needs to return the calculated area (radius * radius * PI). For PI use Math.PI constant.

- Write a class with the name Cylinder that extends Circle class. The class needs one field (instance variable) with name height of type double.

  - The class needs to have one constructor with two parameters radius and height both of type double. It needs to call parent constructor and initialize a height field.

  - In case the height parameter is less than 0 it needs to set the height field value to 0.

  Write the following methods (instance methods):

  - Method named getHeight without any parameters, it needs to return the value of height field.
  - Method named getVolume without any parameters, it needs to return the calculated volume. To calculate volume multiply the area with height.