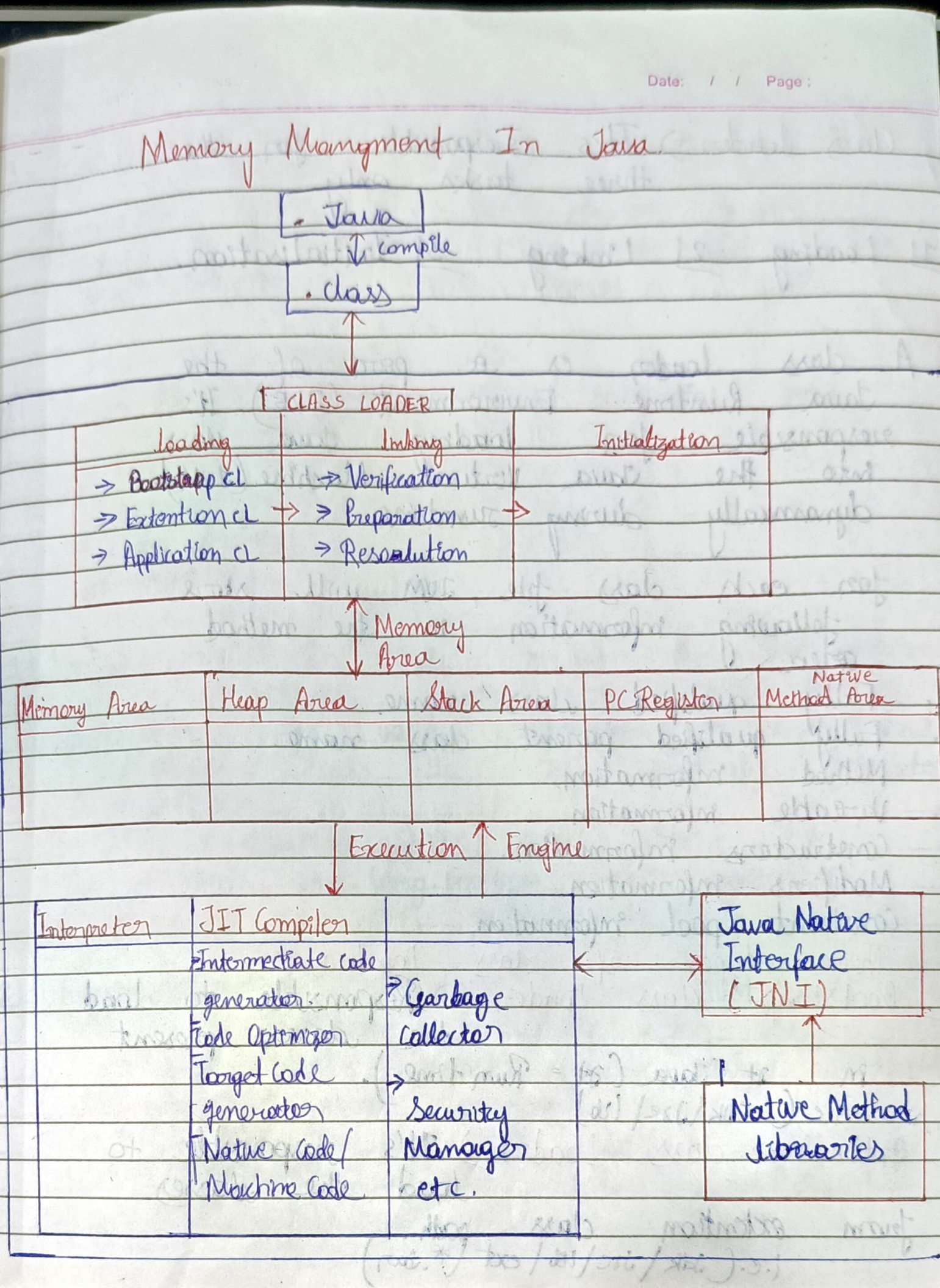
Date: / / Page :

Memory Management In Java



Loader ⇒ It's responsible for the three tasks only.

1] Loading 2] Linking 3] Initialisation.

A class loader is a part of the Java Runtime Environment (JRE). It's responsible for loading Java class into the Java Virtual Machine (JVM) dynamically during runtime.

for each class file, JVM will store following information in the method area.

1. Fully qualified class name.
2. Fully qualified parent class name.
3. Method information.
4. Variable information.
5. Constructors information.
6. Modifiers information.
7. Constant pool information.

Three types of class loaders

1] Bootstrap class Loader ⇒ Responsible to load the classes present in rt.jar (rt = Runtime). i.e. (JDK\Jre\lib)

2] Extention class Loader ⇒ It's responsible to load the classes from extention class path. i.e. (Jdk\Jre\lib\ext (*.jar)

3) Application class loader ⇒

Responsible to load the classes from application classpath. It's internally uses environment variable class path.

2) Linking ⇒ In linking three activities are performed.

1. Verification
2. Preparation
3. Resolution

1) Verification ⇒ In this process Byte code verifier checks wether the .class file is generated by valid compiler or not. and whether .class file is properly formated or not.
If verification fails, then JVM will provides "java.lang.VerifyError" exception.
because of this process, Java is secured.

2) Preparation ⇒ In this process JVM will allocate memory for class level static variable & assign default values (not original values)
for ex.⇒ you declear int a=10, but, It allocate int default value 0.

Date: / /

3] Resolution => In, this process symbolic names present in our program are replaced with original memory references from method area.

2] Initialisation => In this process, two activities will be performed.

1] All static variables are assigned with original values.

2] Static blocks will be executed from top to bottom.

Memory Areas

five types of Memory areas.

1] Method Area. 2] Heap Area
3] Stack Area. 4] PC Register
5] Native Method Area.

1] Method Area => • Method area created when JVM is started.

- It stores .class file information and static variable.
- JVM one memory area, therefore multiple

threads can access this area, so it is not thread safe.

2. Heap Area =>
- Heap area is created when JVM is started.
- It's stores objects, instance variables and arrays.
- It can not be accessed by multiple threads, so the data stored in heap area is not thread safe.

3. Stack Area => Whenever a new thread is created, a seperate stack area will to also be ~~true~~ created.
- It stores the current running methods and local variable
- When the method is completed, the corresponding entry from the stack will be removed.
- After Completeting all methods calls, the stack will become empty and that empty stack will be destroyed by the JVM just before terminating the thread.
- The data stored in the stack is available only for the corresponding

thread and not available to the thread, so the remaining area is thread safe.

4] PC Register ⇒ It holds the address of next executing instruction.

for every thread, a separate pc register is created, so it's also thread safe.

5] Native Method Stacks ⇒

All native method calls invoked by the thread will be stored in the corresponding native method stack.

For every thread separated native method stack will be created.

It is also thread safe.

thread and not available to the remaining thread, so the this area is thread safe.

4] PC Register ⇒ It holds the address of next executing instruction.

for every thread, a separate PC register is created, so it's also thread safe.

5] Native Method Stacks ⇒

All native method calls invoked by the thread will be stored in the corresponding native method stack.

For every thread separated native method stack will be created.

It is also thread safe.

Date: / / Page:

Execution Engine.

Execution Engine is responsible to execute Java class file.

It contains mainly two components.

1. Interpreter
2. JIT Compiler.

1. Interpreter ⇒ In Java, a interpreter is a software component that executes Java bytecode line by line.

A Interpreter is a part of JVM and it's read a .class file line by line and then execute it. It's called Interpreter.

byte code ⟶ Interpreter ⟶ Read line by line (Machine code) ⟶ ⟶ execute.

Date: / / Page:

JIT Compiler

JIT is stand for Just-In-Time which means that code gets compiled when it is needed, not before runtime.

The main purpose of JIT compiler is to improve performance.

OOPS

Object Oriented Programming language

Class => Class is a collection of objects and it's class is not a real world entity. It's and It doesn't take any space on memory. and class is also called a blueprint & logical entity.

for ex.

It's class

```
class Animal {

    String dog; // It's object
    String cat
```

And also create, methods, constructors, fields, blocks, Nested class.

Methods => A set of code which is perform a particular task is called a method.

Advantages of Methods =>

1) Code reusability
2) Code optimization.

Syntax =>

access - modifier return type method name
(list of parameters)

{

//

}

Objects =>

Object is an instance of a class. and It's a real world entity. Once a object is a created it's takes space on memory.

Object consists of =>

1) Identity => name

2) State (Attribute => colour, breed & age

3) Behaviour => eat, run etc.

How to create object =>

1) - new keywords
   - new Instance() method

2) - clone() method
   - deserialization
   - factory method.

for example:

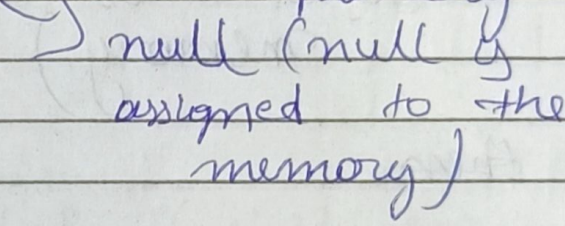
three step to create a object

1] Declaration
2] Instantiation
3] Initialization

1] Declaration: It is declaring a variable name with an object type

for example: Animal buzo; (where "Animal" is className & "buzo" is reference_variable_name.)

working: Animal buzo;



Note:

1. All the objects share the attributes and the behavior of the class
2. Simply declaring a reference variable doesn't create an object.

2] Instantiation: It's a allocated memory for the object. The new keyword is used to create the object.

Initialization →

The new keyword is followed by a call to constructor. This call initializes the new object.

for ex. → Animal bruno = new Animal();

Working → The values are put into the memory that was allocated.

breed, age, color (state/attribute) eat, run (Behaviors)

Memory

(all the states & behaviors of an object is loaded in the memory)

~~then~~

Animal bruno = new Animal();

Date: / / Page:

~~Inheritan~~ Inheritance

Inheritance is the procedure by which one object is acquires all the properties and behaviors of a parent object, and **extends** keywords is use to archive the Inheritance

Or

Inheritance inherite all the properties and behaviors of parents class into child class.

for example →

```
class Animal {          // Parent class or super class
    void eat () {
        sout ("I'm eating");
    }

class Dog extends Animal {  // child class or sub class
    PSVM () {

        Dog d = new Dog ();
        d. eat ();
    }
}
```

Relationship

Dog (child) **IS-A** Animal (Parent)

It's also represents a IS-A relationship.

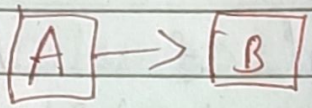
Advantages →

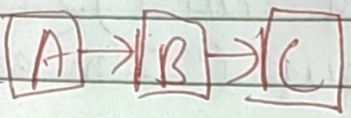
1) Code Reusability
2) We can achieve play polymorphism using inheritance. (means achieve achieve method overriding)

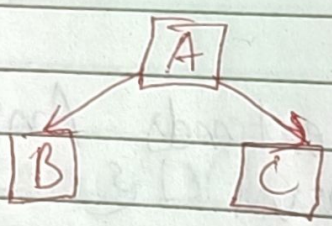
Dis - Advantage →

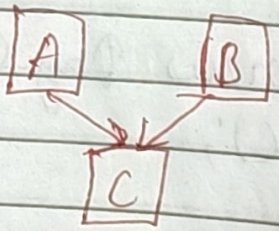
1) Tightly ~~coupled~~ copuled.

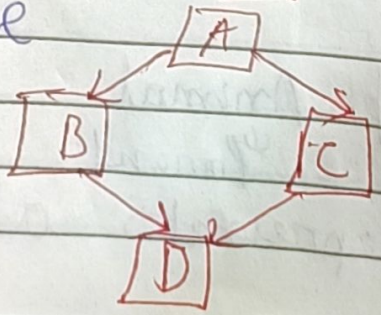
Types of Inheritance (five)

1) Single Inheritance 

2) Multi level Inheritance 

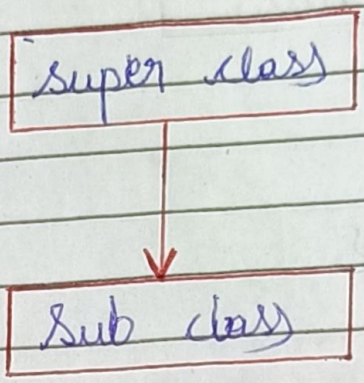
3) Hierarchical Inheritance 

4) Multiple Inheritance 

5) Hybrid Inheritance 

1) Single Inheritance =>

Single inheritance contains only one super class and one subclass is called simple inheritance. Sub-class inherit the features of one super class.

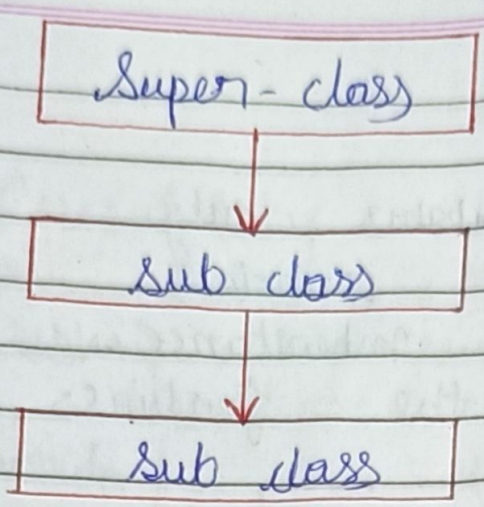


Syntax =>

```
class A {

}
class B extends A {

}
```

2) Multilevel Inheritance =>

Multi-level Inheritance is a chain of inheritance. multi-level Inheritance contains one super class and multiple sub-classes. is called a multi-level Inheritance.



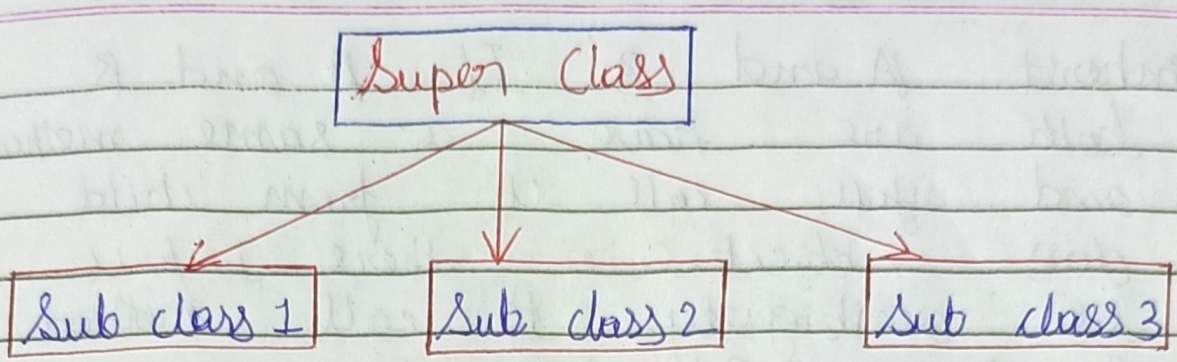
Syntax =>

```
class A {

}
class B extends A {

}
class C extends B {

}
```

3) Hierarchical Inheritance =>

A hierarchical Inheritance which is contains one super class and two or more subclasses. and sub-classes directly inherits a super-class class is know as a hierarchical Inheritance.

Date: / / Page :



Syntax ⇒

```
class A {
}
class B extends A {
}
class C extends A {
}
class D extends A {
}
```

1) Multiple Inheritance ⇒ In Multiple Inheritance is which is contains one or more super class and one subclass. but Java is not support multiple inheritance.

for example, where A, C and C are three classes. The C class

Date: / / Page:

inherit A and B. If A and B both are have a same method and you call it from child class object, so there will be ambiguity to call method of A or B class.

And compiler there will be compile time error.

for example:-

```
class A {
}
class B {
}
class C extends A,B {
}
```

and if you want to achieve multiple inheritance, so you will using Interfaces.

Date: / / Page:

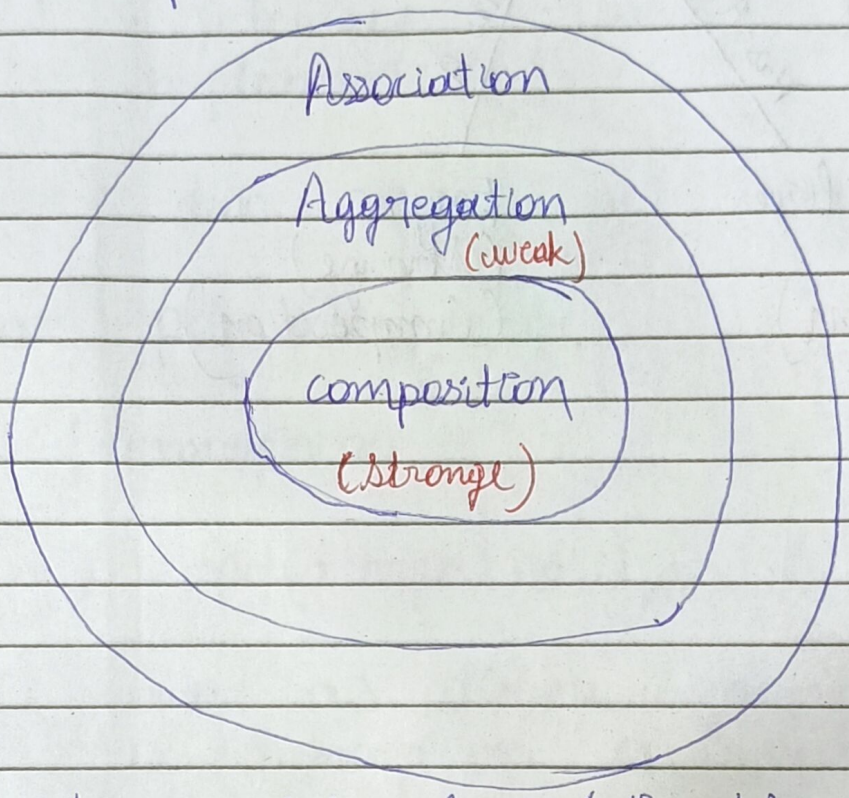
Is-A Relationship ⇒

The Is-A Relationship is also know as a ~~inherita~~ inheritance.

1) If you want to achieve Is-A relationship so you will using extends keyword.
2) And It is used ~~do~~ for code reusability in Java
3) Is-A relationship is tightly coupled, which means change one entity will effect another entity.

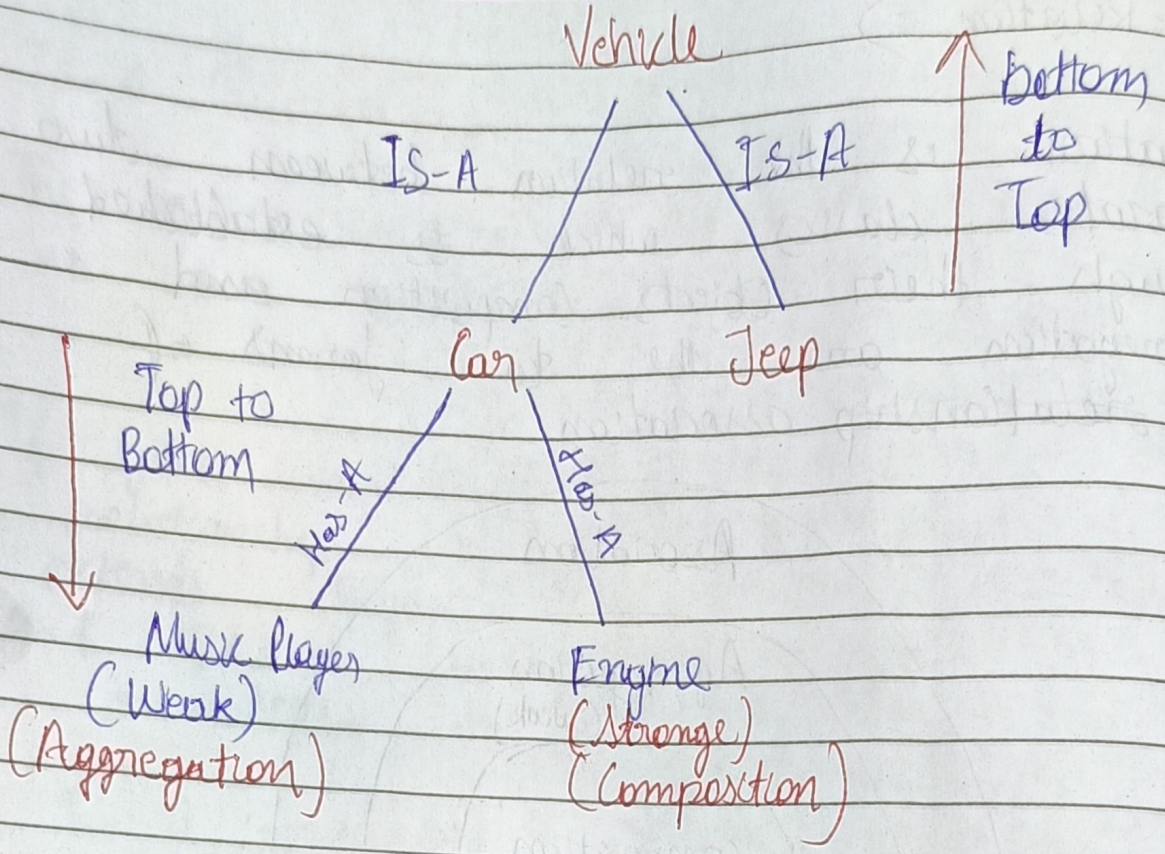
Has-A-Relation =>

Association is the relation between two separate classes which is established through their Objects. Composition and Aggregation are the two forms of relationship association



It's present a HAS-A relationship

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


Vehicle
IS-A
IS-A
Bottom to Top
Car
Jeep
Top to Bottom
HAS-A
HAS-A
Music Player
(Weak)
(Aggregation)
Engine
(Strong)
(Composition)

Date: / / Page :

Poly morphism

polymorphism allows us to perform a single actions in different ways.
for example =)

polymorphism allows you to define one interface and have multiple implementations.

There are two types of poly-Morphism

1) Compile Time - poly Morphism
2) Runtime Polymorphism

1) Compile Time polymorphism =)

It's also known as static polymorphism.
This type of polymorphism is archieved by method overloading.
And the method overloading is in which contains multiple methods in the same class with same name but d/f parameters is know as method over-loading.

for example =) ~~class A~~

```
class A {
    void show() {
        sout("A");
    }

    void show(int a) {
        sout("A");
    }

PSVM {
    A a = new A();
    a.show();
}
```

Runtime - polymorphism (Method Overriding)

Runtime - polymorphism is also known as dynamic polymorphism.
This type of polymorphism is achieved by method Overriding.
The method Overriding which is contains multiple methods in b/t class but method name is same and with same parameters. And It's also achieve Inheritance Is-A relation.

Abstraction =>

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

There are two ways to achieve abstraction in Java

1) Abstract class (0-100%)
2) Interface (100%)

1) Abstract class =>

A class which is declared as abstract is known as an abstract class.
It can have abstract and non-abstract methods.
It needs to

1) A method without body (no implementation) is known as abstract method.
2) A method must always be declared in an abstract class.

Date: / / Page :

3) Abstract classes cannot be instantiated, means we can't create an object of abstract class.

4) If a regular class It can have a constructor and static methods also.

5) Abstract method in an abstract class are meant to be overridden in derived class otherwise compile-time error will be thrown.

2) ~~Interf~~

Date: / / Page:

2) Interface ⇒

A interface in Java is a blueprint of a behaviour.
A Java interface contains static constant and abstract method.

The interface is use to archieve abstraction. It means all the methods in an interface are declared with an empty body.

1) It's use to archieve abstraction.
2) It supports multiple inheritance.
3) It can be used to archieve loose coupling.
4) Implements key-word is used to and achieve inheritance
5) We are creating static method body only.
6) And compiler by default variable takes a public static final

Encapsulation

Encapsulation is one of the best concept. It's used to wrapping a data members and member methods in a single unit. and it's also used for a security purpose with hiding the data (information, details).

for example

```
class emp {

private int empId;

public void setEmpId (int empId1) {
    empId = empId1;

public int getEmpId () {

    return empId;
  }

}
```