## <https://www.javatpoint.com/corejava-interview-questions>

## What is the difference between JVM, JDK, JRE ?

**JDK**

The JDK is a superset of the JRE, and contains everything that is in the JRE, plus tools such as the compilers and debuggers necessary for **developing** applets and applications.

**JRE**

The Java Runtime Environment (JRE) **provides the libraries, the Java Virtual Machine**, and **other components to run** applets and applications written in the Java programming language. The JRE does not contain tools and utilities such as compilers or debuggers for developing applets and applications.

**JVM**

The **Java Virtual Machine** (JVM) is the virtual machine that runs the Java bytecodes line by line. The JVM doesn't understand Java source code; that's why you need compile your \*.java files to obtain \*.class files that contain the bytecodes understood by the JVM.  Indeed, there are specific implementations of the JVM for different systems (Windows, Linux, macOS). JVM is the one that actually calls the main method present in Java code.

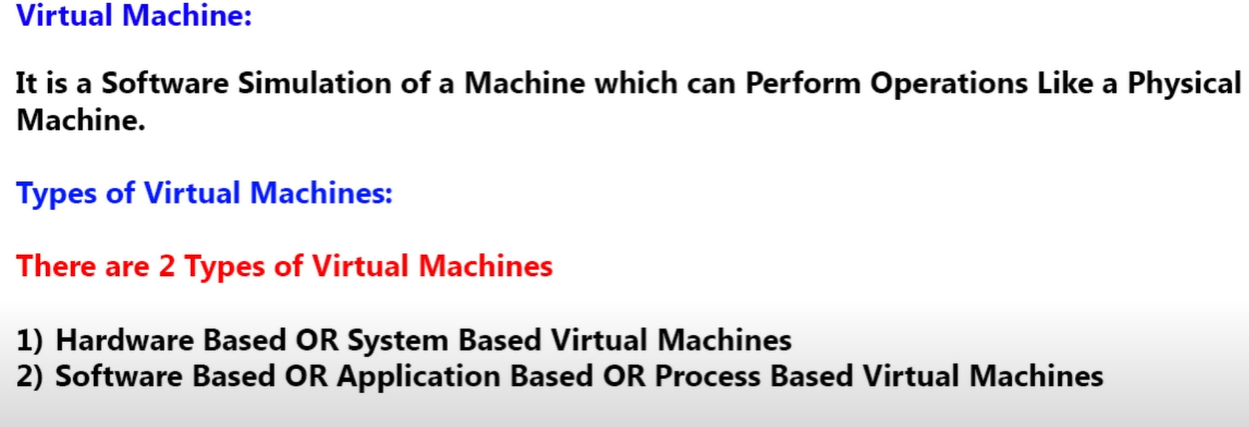
## JVM architecture

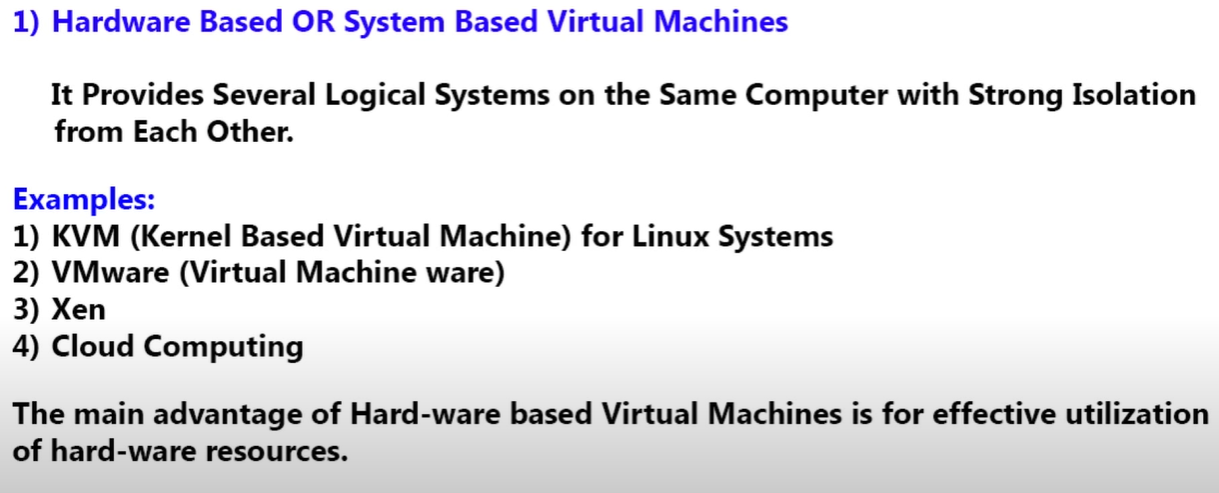
<https://www.youtube.com/watch?v=uq_KF5tmDLM&list=PLd3UqWTnYXOkPLxxK5AV_PsJZh2AC5shI&index=4>

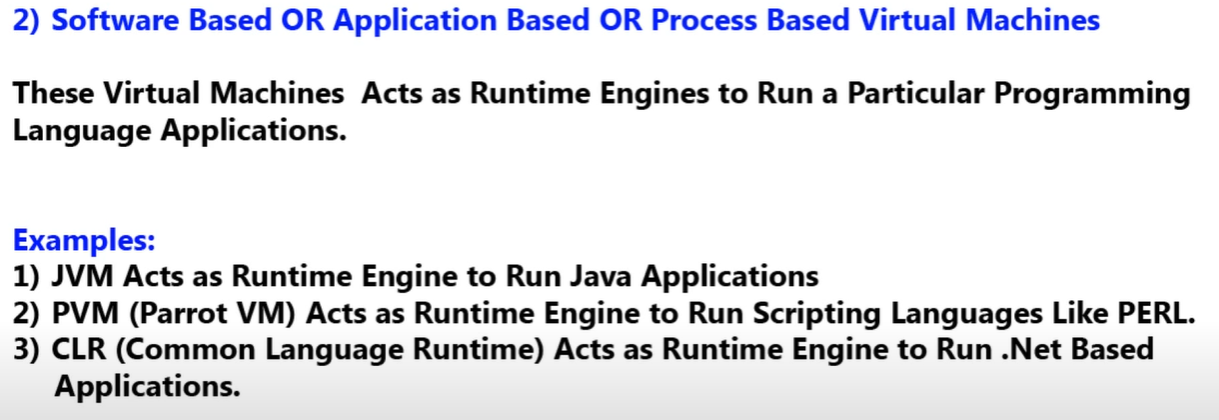
## Virtual machine/ Platform meaning

<https://www.youtube.com/watch?v=p04_pNFECsg&list=PLd3UqWTnYXOkPLxxK5AV_PsJZh2AC5shI&index=2>

Virtual machine (**eg calculator program running on system**) It is a software machine which simulates the functionality of a physical machine.



****



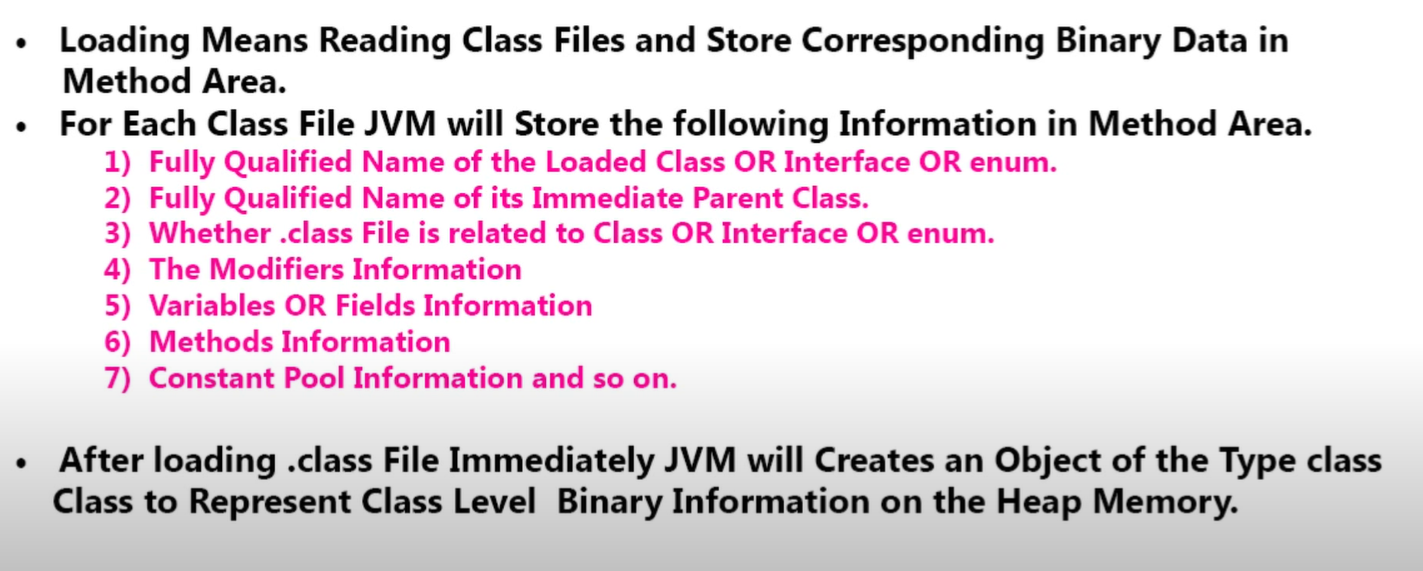
**3 imp components of JVM?**

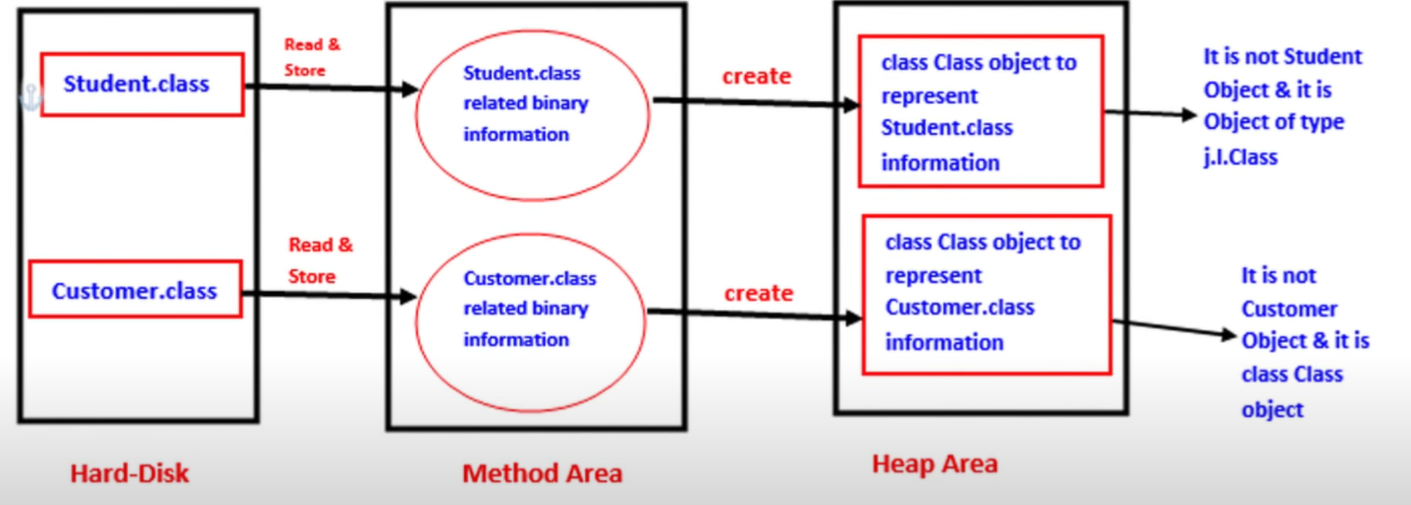
1. Classloader subsystem-> Load .class file into the memory
2. Memory area
3. Execution engine- Run the code line by line

### Classloader subsystem activities

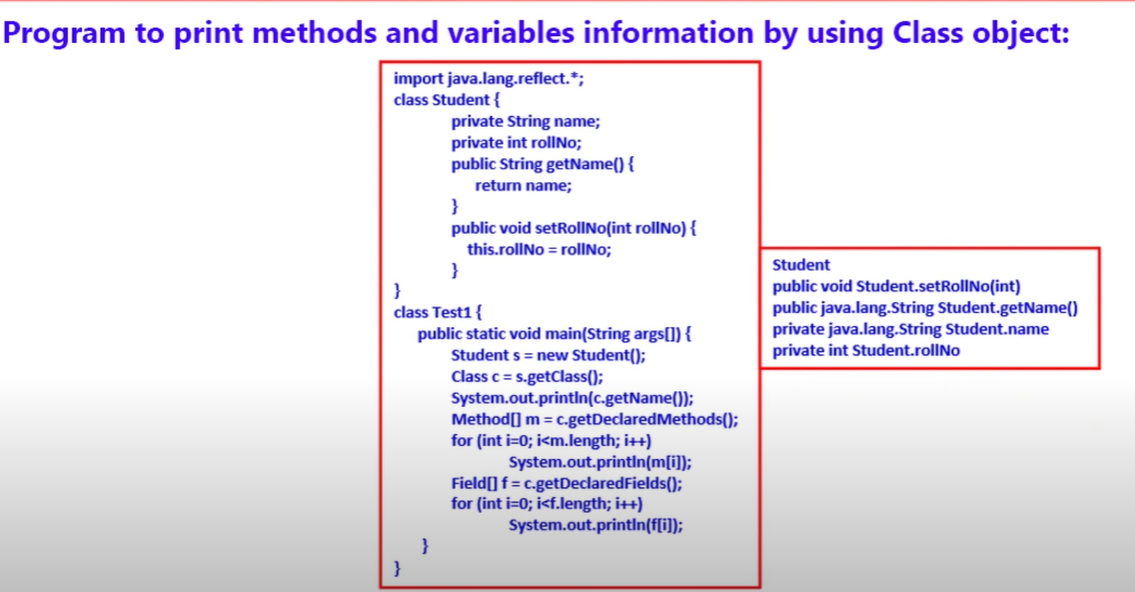
#### Loading

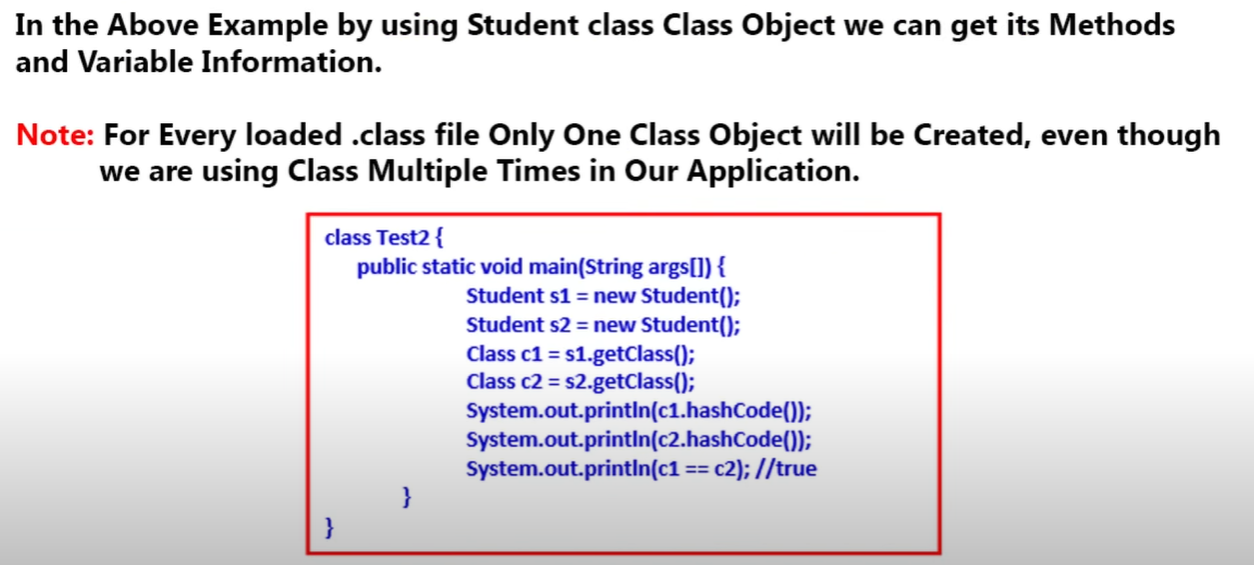
1. Loading- read .class file from hard disk and store binary data inside method area.



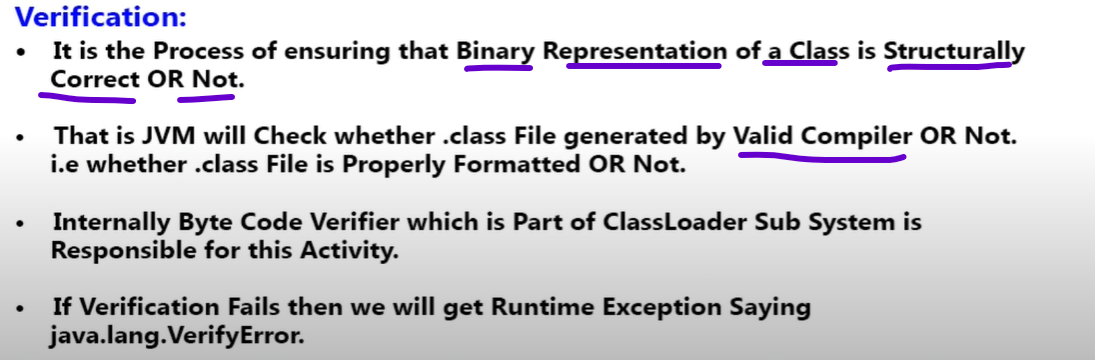


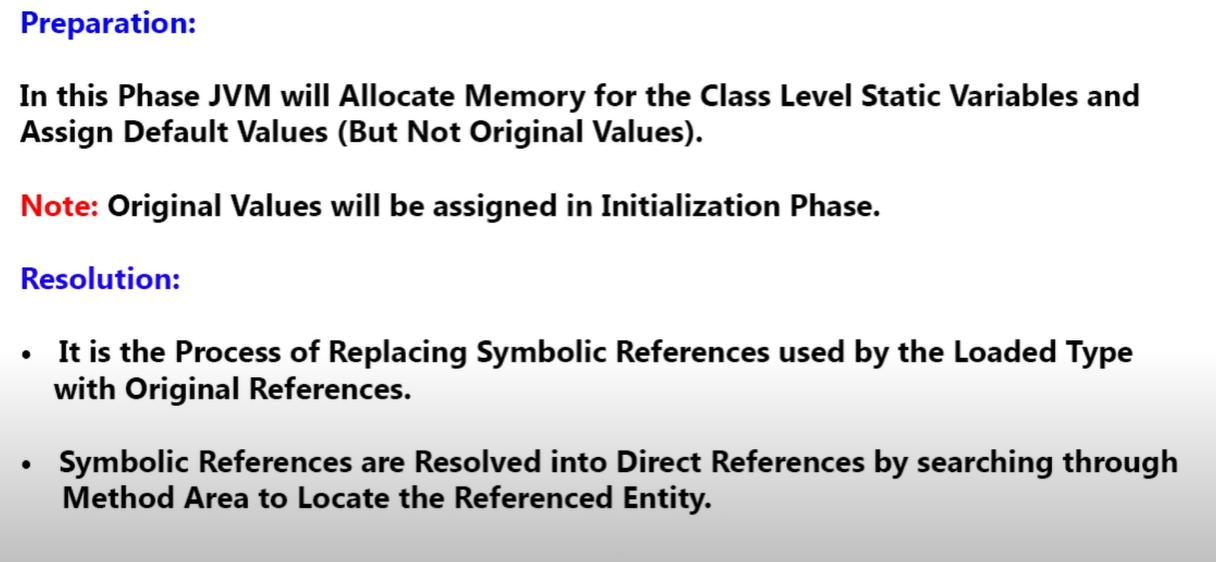




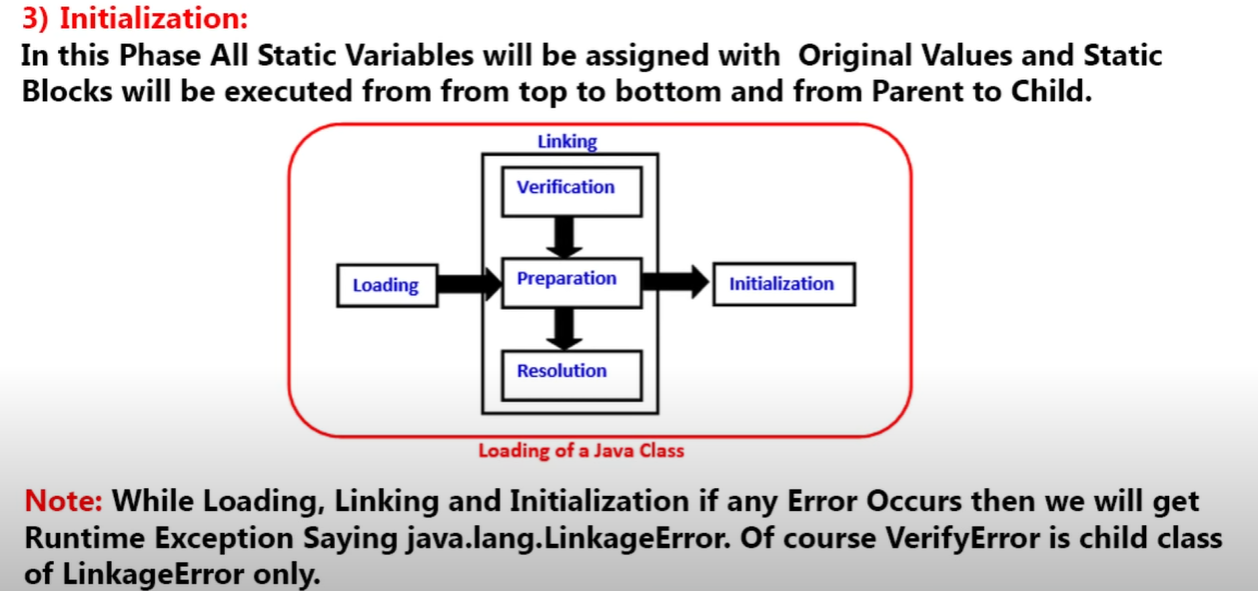


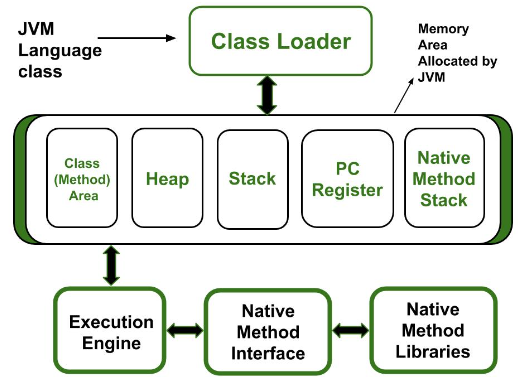
#### Linking (Verification, preparation, resolution)





#### Initialization





Native methods (methods not implemented in java) examples- hashcode, clone

## What is the difference between an object-oriented programming language and object-based programming language?

Object-oriented languages follow all the concepts of OOPs whereas, the object-based language doesn't follow all the concepts of OOPs like inheritance and polymorphism.

Object-oriented languages do not have the inbuilt objects whereas Object-based languages have the inbuilt objects, for example, JavaScript has window object.

Examples of object-oriented programming are Java, C#, Smalltalk, etc. whereas the examples of object-based languages are JavaScript, VBScript, etc.

## What is the purpose of a default constructor?

The purpose of the default constructor is to assign the default value to the objects. The java compiler creates a default constructor implicitly if there is no constructor in the class.

## Does constructor return any value?

A constructor can not return a value because a constructor implicitly returns the reference ID of an object, and since a constructor is also a method and a method can't return more than one values. So we say **explicitly constructor does not have a return value.**

## What do you understand by copy constructor in Java?

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 clone() method of Object class

**copy the values of one object into another using java constructor.**

1. **class** Student{
2. **int** id;
3. String name;
4. Student(**int** id,String name){
5. this.id = id;
6. this.name = name;
7. }
8. //constructor to initialize another object
9. Student(Student s){
10. this.id = s.id;
11. this.name =s.name;
12. }
13. **void** display(){System.out.println(id+" "+name);}
15. **public** **static** **void** main(String args[]){
16. Student s1 = **new** Student(111,"Karan");
17. Student s2 = **new** Student(s1);
18. s1.display();
19. s2.display();
20. }
21. }

## What are the restrictions that are applied to the Java static methods?

Two main restrictions are applied to the static methods.

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

## Why Main Method is Static in Java?

The main() method is the first method you will encounter when studying Java programming and is a standard method that the JVM uses to begin the execution of any Java application. There is no object of the class existing when the Java runtime starts. This is why the main() method must be static for the JVM to load the class into memory and call the main function. If the main method is not static, JVM will be unable to call it since no object of the class is present.

## Can we make constructors static?

As we know that the static context (method, block, or variable) belongs to the class, not the object. Since Constructors are invoked only when the object is created, there is no sense to make the constructors static. However, if you try to do so, the compiler will show the compiler error.

## Can we make the abstract methods static in Java?

Because "abstract" means: "Implements no functionality", and "static" means: "There is functionality even if you don't have an object instance". And that's a logical contradiction

## Can we declare the static variables and methods in an abstract class?

Yes, we can declare static variables and methods in an abstract method. As we know that there is no requirement to make the object to access the static context, therefore, we can access the static context declared inside the abstract class by using the name of the abstract class

## What is **this** keyword in java?

The **this** keyword is a reference variable that refers to the current object. There are the various uses of this keyword in Java. It can be used to refer to current class properties such as instance methods, variable, constructors, etc. It can also be passed as an argument into the methods or constructors. It can also be returned from the method as the current class instance.

## Can we assign the reference to **this** variable?

No, this cannot be assigned to any value because it always points to the current class object and this is the final reference in Java. However, if we try to do so, the compiler error will be shown. Consider the following example.

1. **public** **class** Test
2. {
3. **public** Test()
4. {
5. **this** = **null**;
6. System.out.println("Test class constructor called");
7. }
8. **public** **static** **void** main (String args[])
9. {
10. Test t = **new** Test();
11. }
12. }

## Can **this** keyword be used to refer static members?

Yes, It is possible to use this keyword to refer static members because this is just a reference variable which refers to the current class object. However, as we know that, it is unnecessary to access static variables through objects, therefore, it is not the best practice to use this to refer static members.

## What are the advantages of passing this into a method instead of the current class object itself?

two main advantages of passing this into a method instead of the current class object.

1. this is a final variable. Therefore, this cannot be assigned to any new value whereas the current class object might not be final and can be changed.
2. this can be used in the synchronized block.

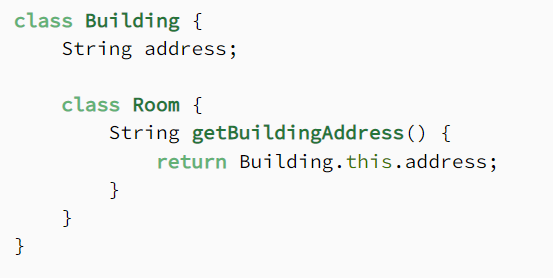
## Composition vs aggregation?

<https://www.baeldung.com/java-composition-aggregation-association>

**Composition** is a strong kind of “**has-a” relationship** because the containing object **owns** it. Therefore, the **objects' lifecycles are tied**. It means that if we d**estroy the owner object, its members also will be destroyed with it. But vice versa is not true.**

For example, a room belongs to a building, or in other words **a building has a room.** So basically, whether we call it “belongs-to” or “has-a” is only a matter of point of view.

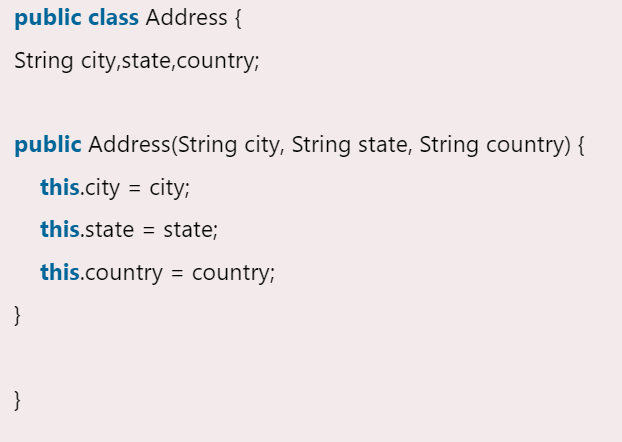
In Java, we can model this with a non-static inner class.



All inner class objects store an implicit reference to their containing object. As a result, we don't need to store it manually to access it.

**Aggregation** is also a “has-a” relationship. What distinguishes it from composition, that it doesn't involve owning. As a result, the lifecycles of the **objects aren't tied: every one of them can exist independently of each other**. Aggregation can be defined as the relationship between two classes where the aggregate class contains a reference to the class it interacts.

For example, The aggregate class Employee having various fields such as age, name, and salary also contains an object of Address class having various fields such as Address-Line 1, City, State, and pin-code. In other words, we can say that Employee (class) has an object of Address class.



1. **public** **class** Emp {
2. **int** id;
3. String name;
4. Address address;
5. **public** Emp(**int** id, String name,Address address) {
6. **this**.id = id;
7. **this**.name = name;
8. **this**.address=address;
9. }
11. **void** display(){
12. System.out.println(id+" "+name);
13. System.out.println(address.city+" "+address.state+" "+address.country);
14. }
16. **public** **static** **void** main(String[] args) {
17. Address address1=**new** Address("gzb","UP","india");
18. Address address2=**new** Address("gno","UP","india");
20. Emp e=**new** Emp(111,"varun",address1);
21. Emp e2=**new** Emp(112,"arun",address2);
23. e.display();
24. e2.display();
26. }
27. }

**Association**

Association is the weakest relationship between the three. **It isn't a “has-a” relationship**, none of the objects are parts or members of another. **Association only means that the objects “know” each other.**

## Why does Java not support pointers?

1. **Memory access via pointer arithmetic**: this is fundamentally unsafe. Java has a robust security model and disallows pointer arithmetic for the same reason. It would be impossible for the Virtual Machine to ensure that code containing pointer arithmetic is safe without expensive runtime checks.

2. **Security**: By not allowing pointers, Java effectively provides another level of abstraction to the developer. No pointer support make Java more secure because they point to memory location or used for memory management that loses the security as we use them directly.

3. **Passing argument by reference**: Passing a reference which allows you to change the value of a variable in **the caller's scope**. Java doesn't have this, but it's a pretty rare use case and can easily be done in other ways. **This is in general equivalent to changing a field in an object scope that both the caller and callee can see.**

4. **Manual memory management**: you can use pointers to manually control and allocate blocks of memory . This is useful for some bigger applications like games, device drivers etc. but for general purpose Object Oriented programming it is simply not worth the effort. Java instead provides very good automatic Garbage Collection (GC) which takes care of memory management

## Object cloning?

* 1. This method belongs to the Object class, which is a base class of every class created in Java.
  2. This method helps to create a copy of the object, but if the class doesn’t support a cloneable interface then it leads to the exception, " CloneNotSupportedException".  
     The syntax for the clone method is-

image

* 1. We need to implement the clone() method in a class whose object has to be cloned, we also need to handle CloneNotSupportedException and calling of the clone() method.

Code available at <https://github.com/djoshi712/Core_Java> (core\_part1)

## What are the advantages of defining packages in Java?

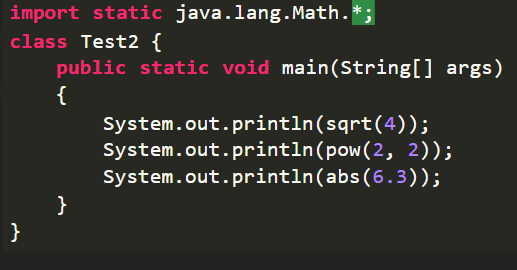
Package in Java is a mechanism to encapsulate a group of classes, sub packages and interfaces. Packages are used for:

* Preventing naming conflicts. For example there can be two classes with name Employee in two packages, college.staff.cse.Employee and college.staff.ee.Employee
* Making searching/locating and usage of classes, interfaces, enumerations and annotations easier
* Providing controlled access: protected and default have package level access control. A protected member is accessible by classes in the same package and its subclasses. A default member (without any access specifier) is accessible by classes in the same package only.
* Packages can be considered as data encapsulation (or data-hiding).

## Static import

In Java, static import concept is introduced in 1.5 version. With the help of static import, we can access the static members of a class directly without class name or any object. For Example: we always use sqrt() method of Math class by using Math class i.e. Math.sqrt(), but by using static import we can access sqrt() method directly.

According to SUN microSystem, it will improve the code readability and enhance coding. But according to the programming experts, it will lead to confusion and not good for programming. If there is no specific requirement then we should not go for static import.



Some of the rules are listed below:

## Can subclass overriding method declare an exception if parent class method doesn't throw an exception?

**If the superclass method does not declare an exception**

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

**If the superclass method declares an exception**

If the superclass method declares an exception, subclass overridden method can declare **same**, **subclass exception or no exception** but cannot declare parent exception.

<https://www.javatpoint.com/exception-handling-with-method-overriding>