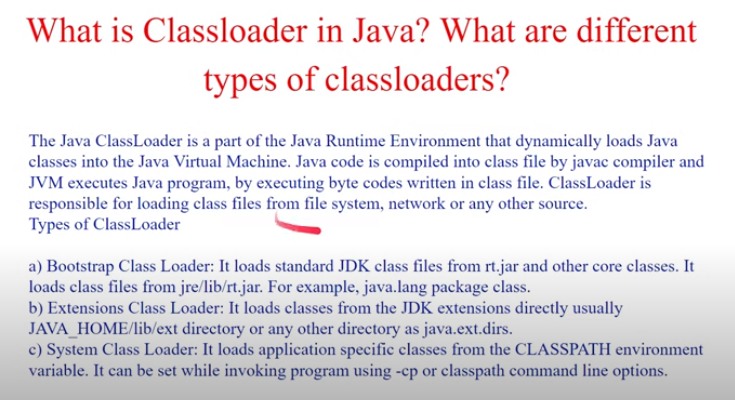
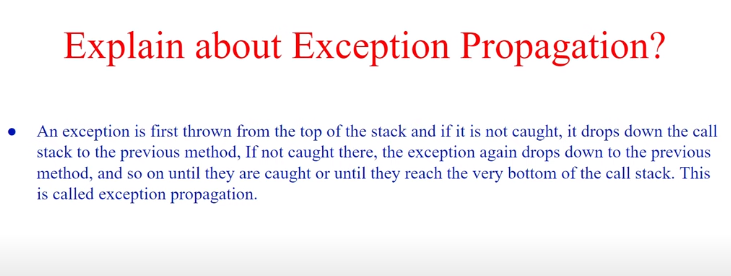
**Servlet**?

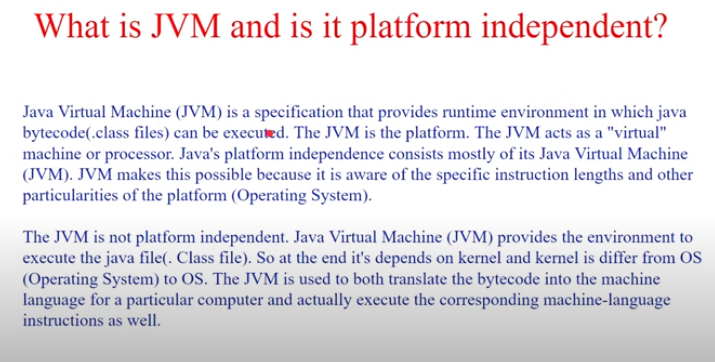
* Servlet is a technology which is used to create a web application.
* Servlet is an API that provides many interfaces and classes including documentation.
* Servlet is an interface that must be implemented for creating any Servlet.
* Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
* Servlet is a web component that is deployed on the server to create a dynamic web page.

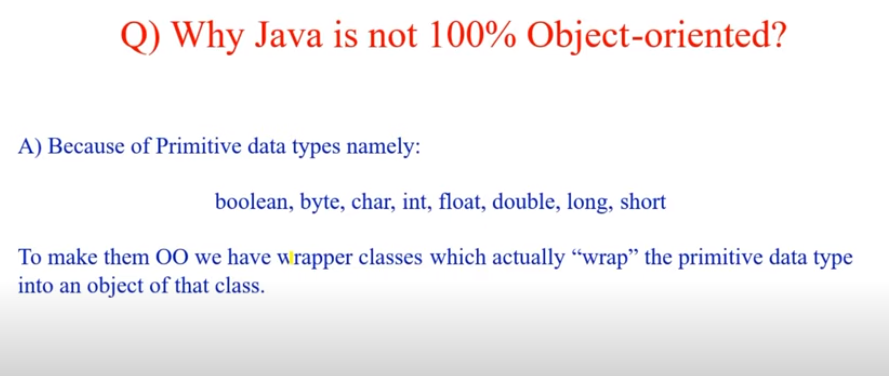
**Difference between abstract class and interface:**

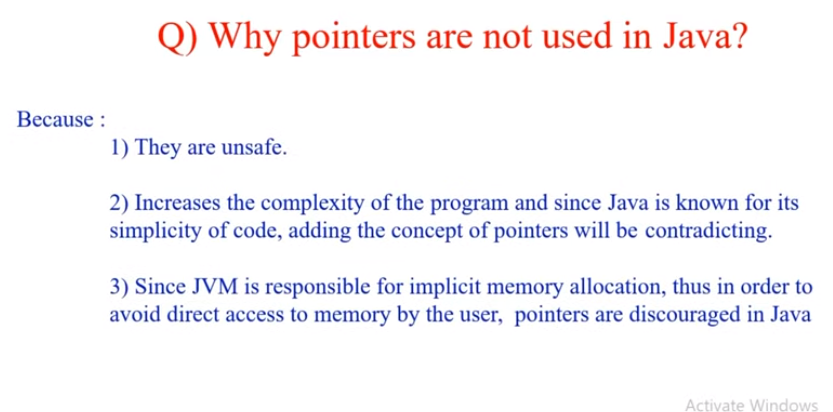
|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| 1) Abstract class can **have abstract and non-abstract** methods. | Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also. |
| 2) Abstract class **doesn't support multiple inheritance**. | Interface **supports multiple inheritance**. |
| 3) Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variables**. |
| 4) Abstract class **can provide the implementation of interface**. | Interface **can't provide the implementation of abstract class**. |
| 5) The **abstract keyword** is used to declare abstract class. | The **interface keyword** is used to declare interface. |
| 6) An **abstract class** can extend another Java class and implement multiple Java interfaces. | An **interface** can extend another Java interface only. |
| 7) An **abstract class** can be extended using keyword "extends". | An **interface** can be implemented using keyword "implements". |
| 8) A Java **abstract class** can have class members like private, protected, etc. | Members of a Java interface are public by default. |
| 9)**Example:** public abstract class Shape{ public abstract void draw(); } | **Example:** public interface Drawable{ void draw(); } |

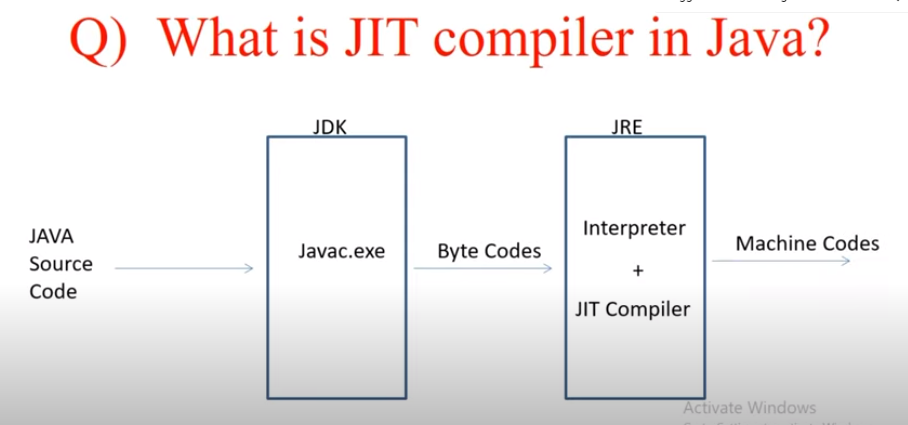


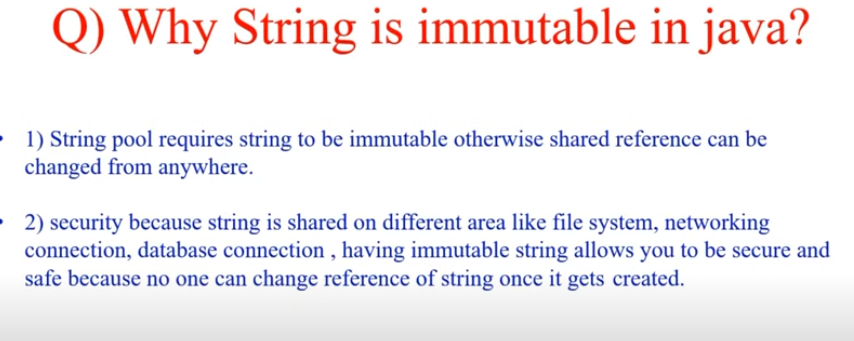


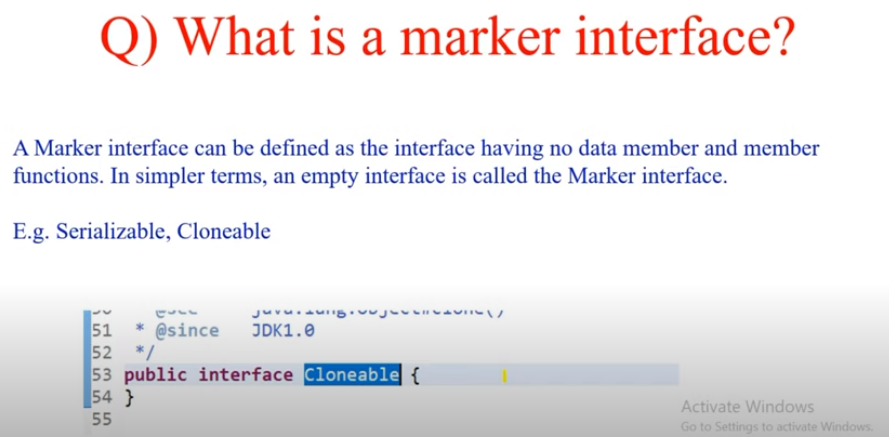


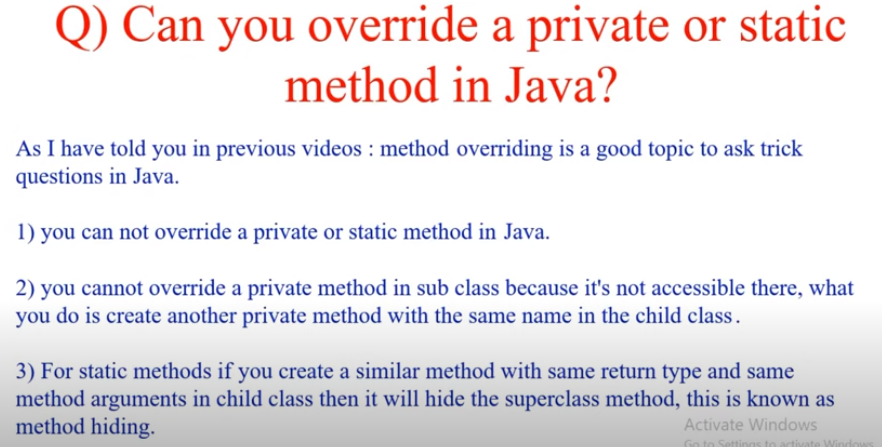




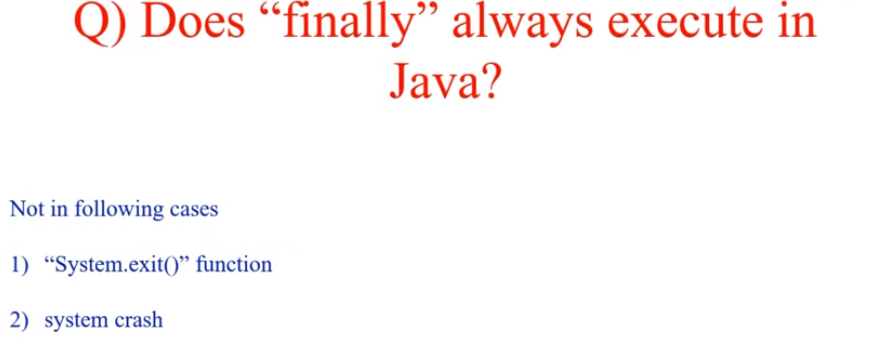


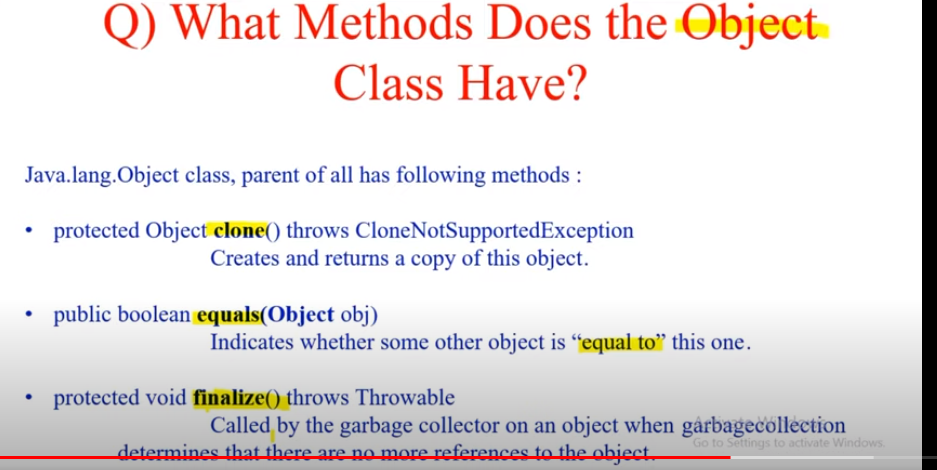


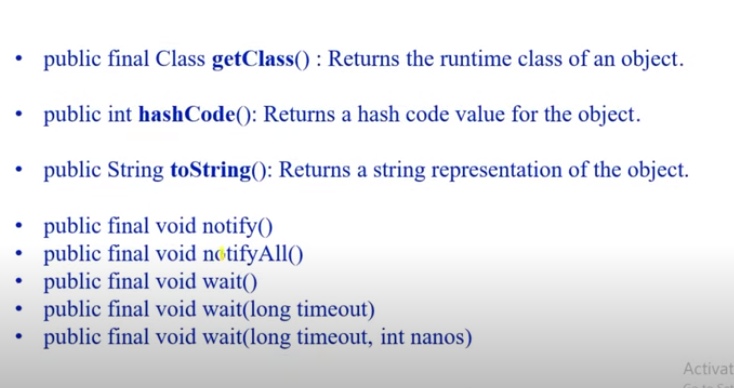


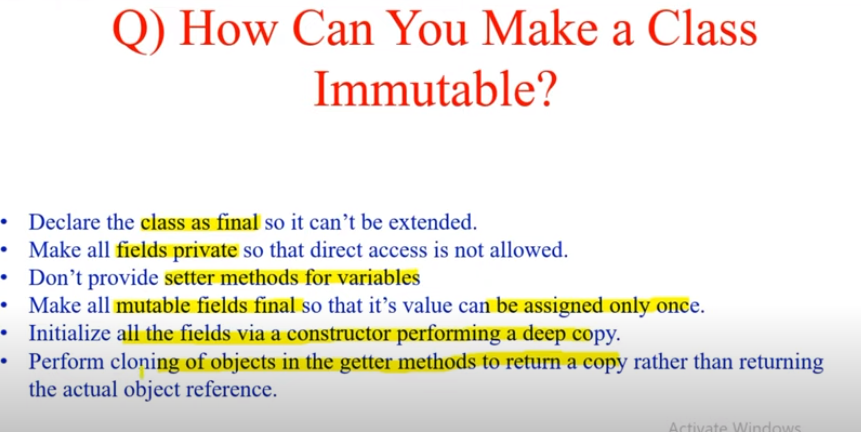


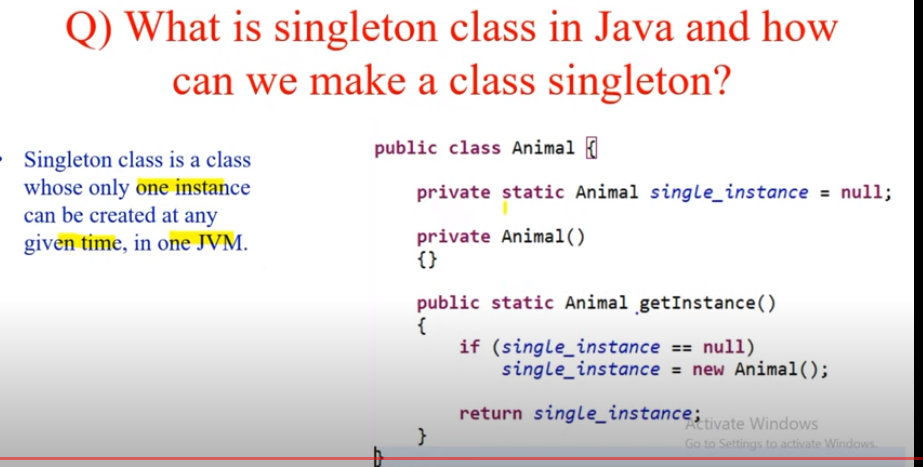
* You can call the static method with class name as well as object name.











First level and second level cache in Hibernate?

Difference bwn Arrya List and Linked List?

Difference between HashMap and Hash Table?

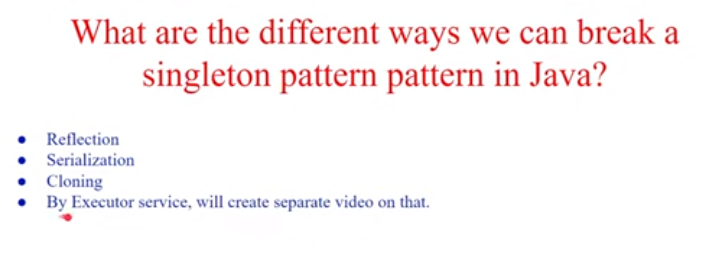
Java Garbage Collection?How it works?

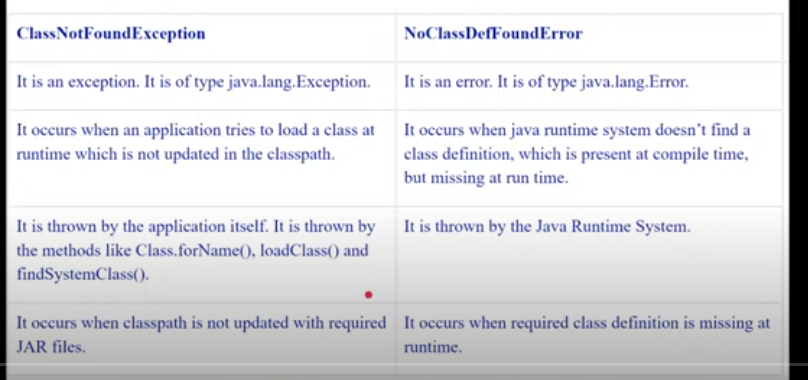
What is serialization? Why do we need it?

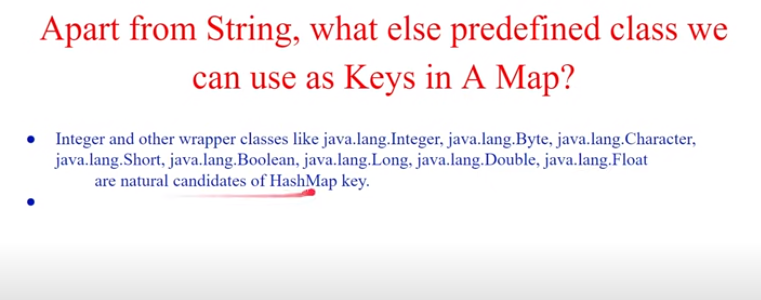
What is weak hash map?

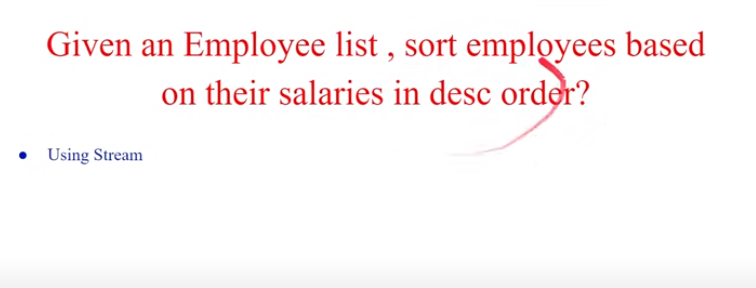
What are functional interfaces?

If synchronized map is there, then why we are introducing concurrent hash map?



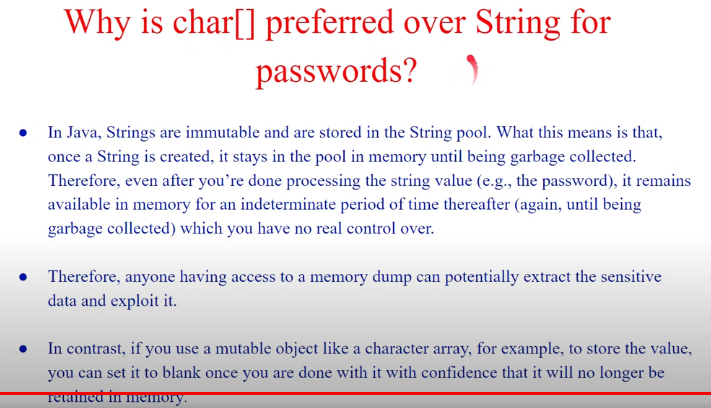






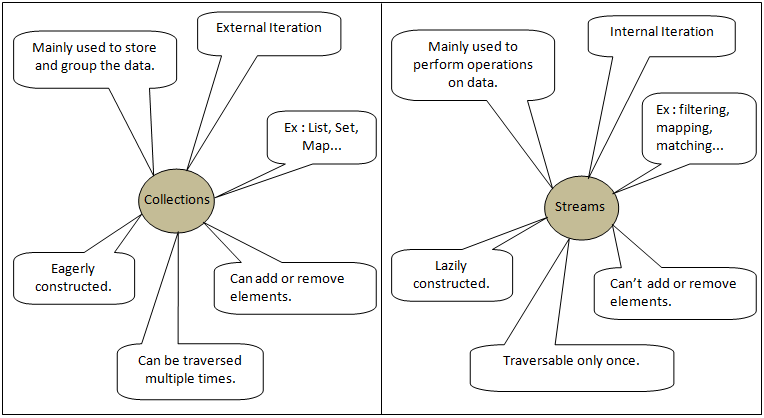






**Difference Between Collections Vs Streams In Java :**

|  |  |
| --- | --- |
| **Collections** | **Streams** |
| Collections are mainly used to store and group the data. | Streams are mainly used to perform operations on data. |
| You can add or remove elements from collections. | You can’t add or remove elements from streams. |
| Collections have to be iterated externally. | Streams are internally iterated. |
| Collections can be traversed multiple times. | Streams are traversable only once. |
| Collections are eagerly constructed. | Streams are lazily constructed. |
| Ex : List, Set, Map… | Ex : filtering, mapping, matching… |

[](https://i0.wp.com/javaconceptoftheday.com/wp-content/uploads/2019/04/CollectionsVsStreamsInJava.png?ssl=1)

# Difference between Comparable and Comparator

|  |  |
| --- | --- |
| **Comparable** | **Comparator** |
| 1) Comparable provides a **single sorting sequence**. In other words, we can sort the collection on the basis of a single element such as id, name, and price. | The Comparator provides **multiple sorting sequences**. In other words, we can sort the collection on the basis of multiple elements such as id, name, and price etc. |
| 2) Comparable **affects the original class**, i.e., the actual class is modified. | Comparator **doesn't affect the original class**, i.e., the actual class is not modified. |
| 3) Comparable provides **compareTo() method** to sort elements. | Comparator provides **compare() method** to sort elements. |
| 4) Comparable is present in **java.lang** package. | A Comparator is present in the **java.util** package. |
| 5) We can sort the list elements of Comparable type by **Collections.sort(List)** method. | We can sort the list elements of Comparator type by **Collections.sort(List, Comparator)** method. |

**Collection vs Collections:**

| **Collection** | **Collections** |
| --- | --- |
| It is an interface. | It is a utility class. |
| It is used to represent a group of individual objects as a single unit. | It defines several utility methods that are used to operate on collection. |
| The Collection is an interface that contains a static method since java8. The Interface can also contain abstract and default methods. | It contains only static methods. |

| Sr.No | Compile Time Polymorphism | Run time Polymorphism |
| --- | --- | --- |
| 1 | In Compile time Polymorphism, the call is resolved by the compiler. | In Run time Polymorphism, the call is not resolved by the compiler. |
| 2 | It is also known as Static binding, Early binding and overloading as well. | It is also known as Dynamic binding, Late binding and overriding as well. |
| 3 | Method overloading is the compile-time polymorphism where more than one methods share the same name with different parameters or signature and different return type. | Method overriding is the runtime polymorphism having same method with same parameters or signature, but associated in different classes. |
| 4 | It is achieved by function overloading and operator overloading. | It is achieved by virtual functions and pointers. |
| 5 | It provides fast execution because the method that needs to be executed is known early at the compile time. | It provides slow execution as compare to early binding because the method that needs to be executed is known at the runtime. |
| 6 | Compile time polymorphism is less flexible as all things execute at compile time. | Run time polymorphism is more flexible as all things execute at run time |