1. What is @SpringBootApplication

The @SpringBootApplication is a combination of three annotations @Configuration (used for Java-based configuration), @ComponentScan (used for component scanning), and @EnableAutoConfiguration (used to enable auto-configuration in [Spring Boot](http://javarevisited.blogspot.sg/2018/01/how-to-learn-spring-core-spring-mvc-boot-security-framework.html)).

Java

Spring

Spring boot

Java 8

Database

Procedures

1. Comparator ache he vs Comparable khrab he: Done
2. Session Factory vs Session: Done
3. Hash table/HashMap internals
4. Servlets Life cycle: Done
5. Servlet config vs context: Done
6. Jsp implicit objects
7. Spring AOP
8. Model is a memory use to sending data from controller to ui.
9. Collection & Collections
10. Why Equals and Hash code.
11. Second Highest Salary
12. Do you have idea about multithreading?
13. Difference between error and exception
14. What is the term thread safe?
15. How to automate spring boot build creation using maven or Jenkins
16. difference between stream and collection api.
17. what is predicates
18. different ways of creating the objects.
19. what is reflection api.
20. Why java doesn't support multiple inheritance, and what is the solution.
21. difference between interface and abstract class.
22. How can we achive 100% abstraction, which option developer has to prefer.
23. Compile time polymorphism and run time polymorphism.
24. can we override our main method.
25. if two classes have main method which one jvm will choose.
26. **which package imported by default by java- java.lang**
27. what are the access modifiers in java and scope of it.
28. transient keyword
29. how to deserialize our java object
30. what is serialization and deserialization / serializable and DE serializable
31. what is marker interface, and present in java name some of them
32. did u use try with resource?
33. how HashMap internally works
34. null as HashMap key
35. custom object as a hashmap key / custom class as a hashmap key
36. how to avoid concurrent modification exception
37. difference between fale pass and fale safe iterator
38. difference between exception and error
39. try without catch
40. in which condition finally block will not get executed - Besides a System.exit(), the finally block will not run if the JVM crashes for some reason (e.g. infinite loop in your try block).
41. how to avoid null pointer exception
42. difference between ClassNotFoundException and ClassDefNotFoundError
43. how to create custom runtime exception
44. what is stackOverflow Error
45. String is a class in java
46. String constant pool in java
47. how to compare two string objects
48. which is the final class String, StringBuffer, StringBuilder: all are final only
49. difference between above three things.
50. benefits of spring mvc: pending
51. whether spring beans are thread safe or not: no it is not.
52. what is thread safe.
53. what is ApplicationContext.
54. what is Autowiring and difference ways of Autowire.
55. what happens if we run spring boot application as java application: proper termination with JMX not Terminate properly in the case of java running it will destroy directly – Process. Destroy ()
56. what is spring boot dependency management
57. process of connecting to database using spring boot jdbc
58. meaning of transaction and how to manage the transaction.
59. Autoconfiguration concept of spring boot and
60. how to disable autoconfiguration in spring boot
61. What is JDK JRE and JVM
62. Without JDK can we run our java programs
63. Concurrent HashMap
64. HashMap logical program: to find the first max occurrence of a number, used LinkedHashMap
65. What is method reference concept, how it works.
66. Fail pass Fail safe
67. Concurrent HashMap
68. Equals and hash code contract.
69. List out some marker interfaces.
70. Multithreading questions.
71. What is iterator and how to use iterator.
72. Interface vs Abstract class.
73. Shallow and deep cloning
74. Controller vs Rest Controller
75. What is Exception ? provide keywords related to exception handling?
76. What is Checked and Unchecked exception?
77. Difference b/w throw and throws?
78. What is Exception Narowing and Widening?
79. What are rules we should follow for exception while overriding a method?
80. Different maven commad
81. Stream and parallal stream
82. What is static keyword, what is use where to use, and what are benefits of it.
83. Why main is static method.
84. Why do we need abstract classes.
85. When we use arraylist and when we use linkedList
86. Why we prefer Spring boot over spring
87. 200 vs 201(success res with some fields) vs 204(success res with empty body) status code
88. Post vs PUT vs GET
89. Advantages and disadvantages of Lambda expression: Debug may be the problem
90. How Microservices actually works
91. How you ensure your code quality.

Session vs SessionFactory

First of all, asking the difference between these interfaces doesn't make any sense. It seems like asking the difference between car manufacturing plant and car. A manufacturing plant is a place where the cars will be produced. Similarly, SessionFactory is an instance which will create Session objects.

### Sessionfactory:

* It is one instance per datasource/database.
* It is thread safe.
* It is a heavy weight object, because it maintains datasources, mappings, hibernate configuration information’s etc.
* Sessionfactory will create and manage the sessions.
* If you have 5 datasources/databases, then you must create 5 session factory instances.
* sessionfactory is an immutable object and it will be created as singleton while the server initializes itself.

### Session:

* It is one instance per client/thread/one transaction.
* It is not thread safe.
* It is light weight.
* sessions will be opened using sessionfactory.openSession() and some database operations will be done finally session will be closed using session.close().

Servlet context vs config

The ServletConfig parameters are specified for a particular servlet and are unknown to other servlets. It is used for intializing purposes.

The ServletContext parameters are specified for an entire application outside of any particular servlet and are available to all the servlets within that application. It is application scoped and thus globally accessible across the pages.

# Servlet Life Cycle method

There are three life cycle methods of a Servlet :

* init()
* service()
* destroy()

Let’s look at each of these methods in details:

1. **init() method**: The **Servlet.init()** method is called by the Servlet container to indicate that this Servlet instance is instantiated successfully and is about to put into service.

2. **service() method**: The **service()** method of the Servlet is invoked to inform the Servlet about the client requests.

* This method uses **ServletRequest** object to collect the data requested by the client.
* This method uses **ServletResponse** object to generate the output content.

1. **destroy() method**: The **destroy()** method runs only once during the lifetime of a Servlet and signals the end of the Servlet instance.

# JSP IMPLICIT OBJECTS

|  |  |
| --- | --- |
| **S.No.** | **Object & Description** |
| 1 | **request**  This is the **HttpServletRequest** object associated with the request. |
| 2 | **response**  This is the **HttpServletResponse** object associated with the response to the client. |
| 3 | **out**  This is the **PrintWriter** object used to send output to the client. |
| 4 | **session**  This is the **HttpSession** object associated with the request. |
| 5 | **application**  This is the **ServletContext** object associated with the application context. |
| 6 | **config**  This is the **ServletConfig** object associated with the page. |
| 7 | **pageContext**  This encapsulates use of server-specific features like higher performance **JspWriters**. |
| 8 | **page**  This is simply a synonym for **this**, and is used to call the methods defined by the translated servlet class. |
| 9 | **Exception**  The **Exception** object allows the exception data to be accessed by designated JSP. |

How to find nth highest salary in sql

SELECT name, MAX(salary) AS salary FROM employee WHERE salary < (SELECT MAX(salary) FROM employee);

Spring AOP Module:

<https://github.com/in28minutes/spring-master-class/tree/master/03-spring-aop>

Why Reflection API.  
 100 % abstraction.

Class level modifiers.

**select e.empName, count(\*) from employee e group by e.empName**

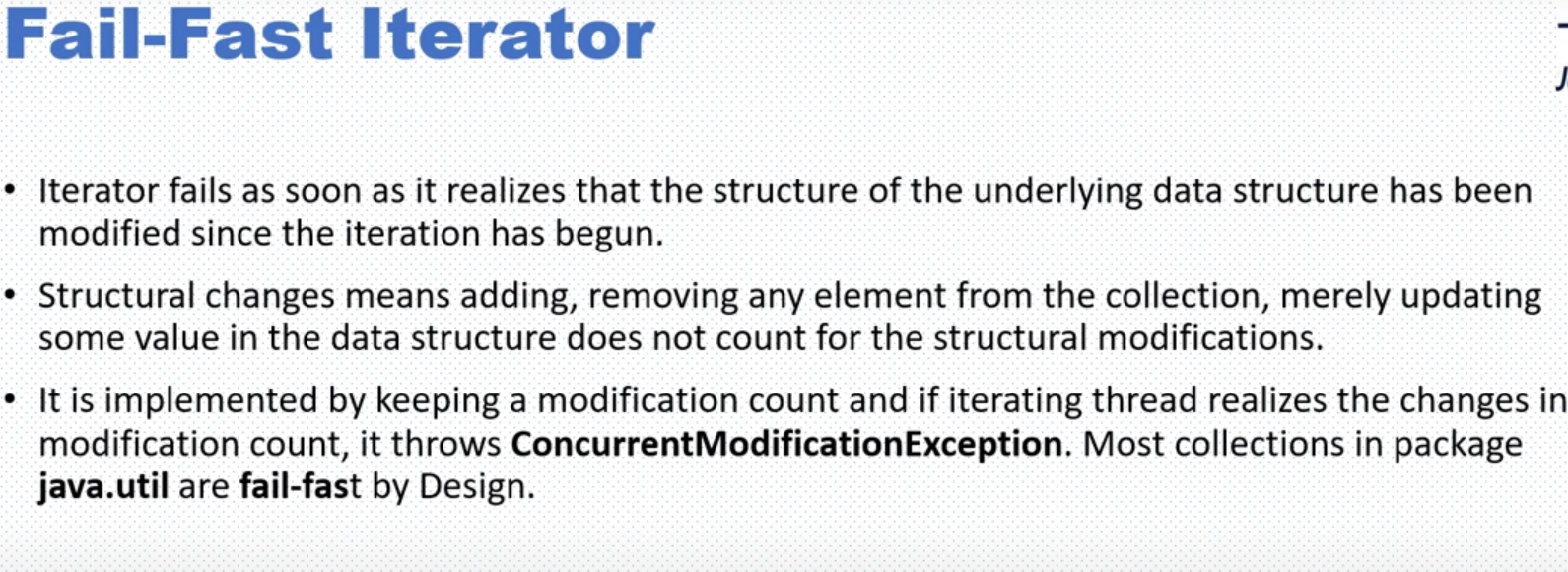
**select e.empName, s.salary from employee left join Salary s on s.empId=e.empId**

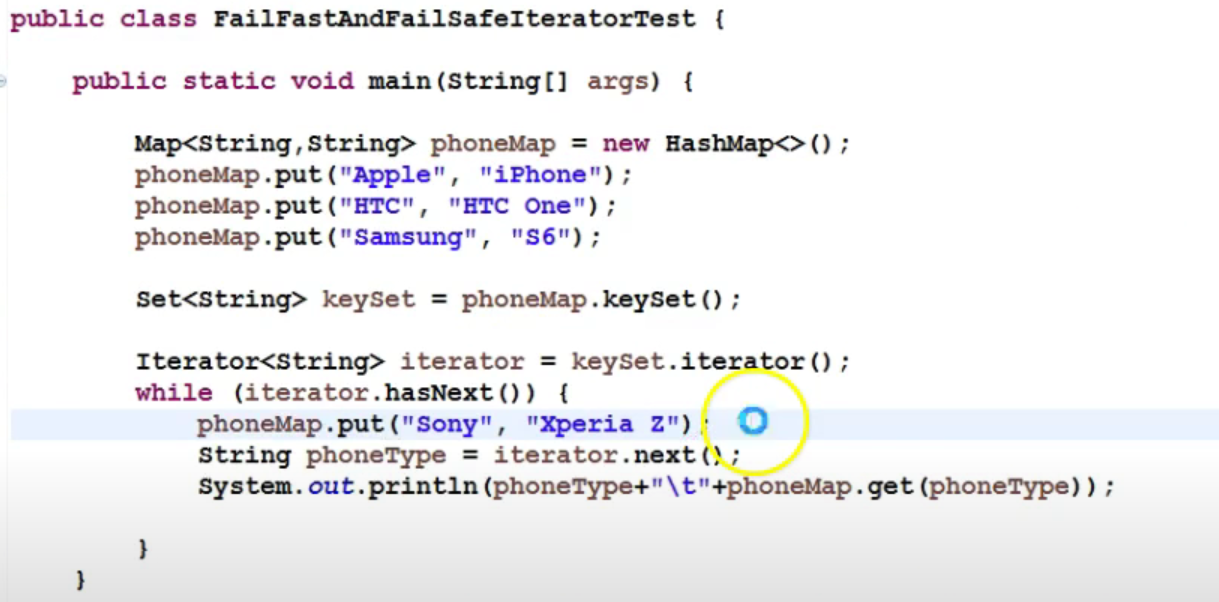
**where s.salary > 50000**

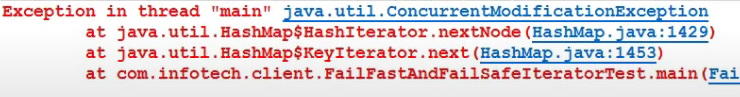
**==========================================================**

**Q) What are fail pass and fail-safe iterators**

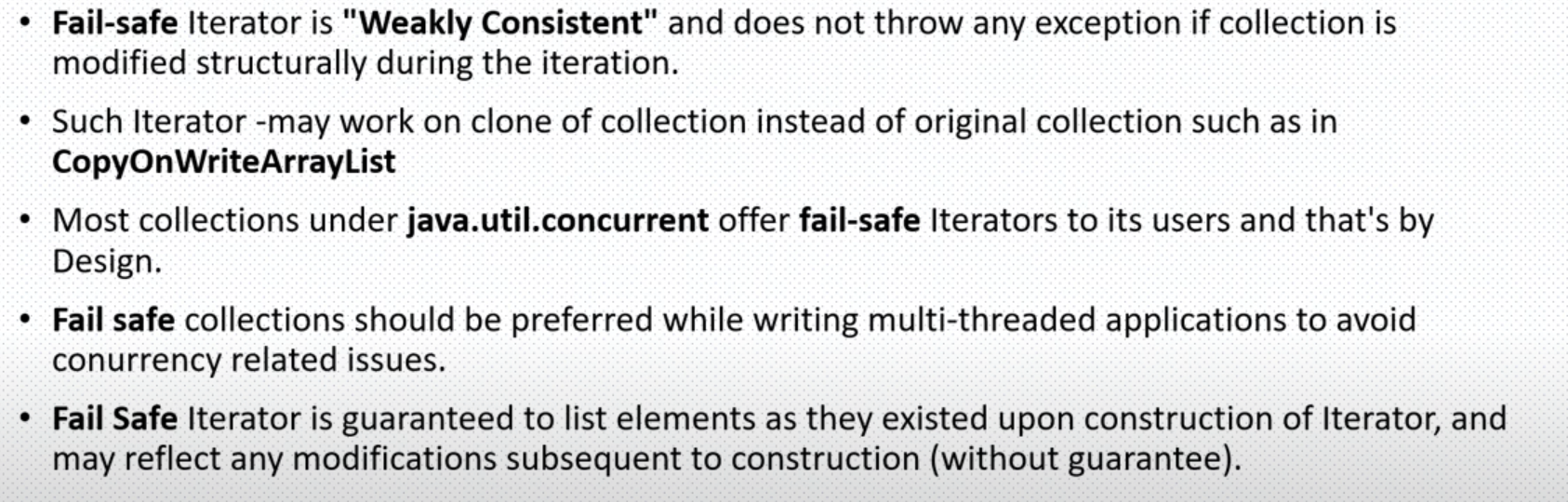
**A)**

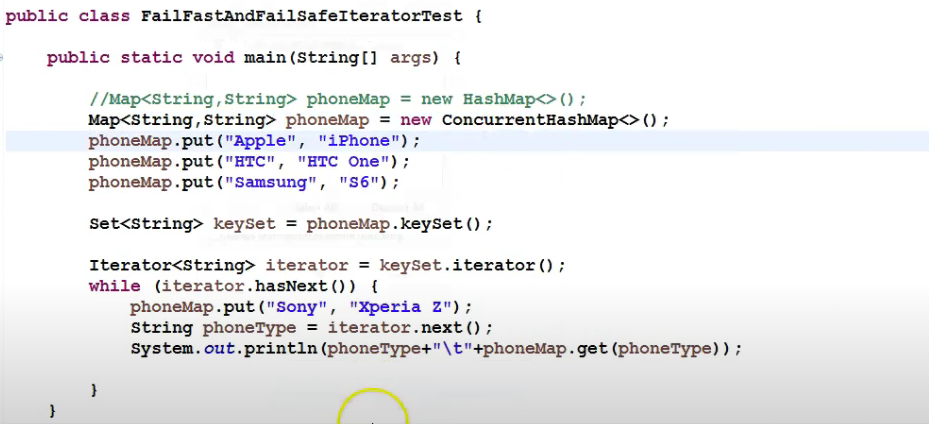


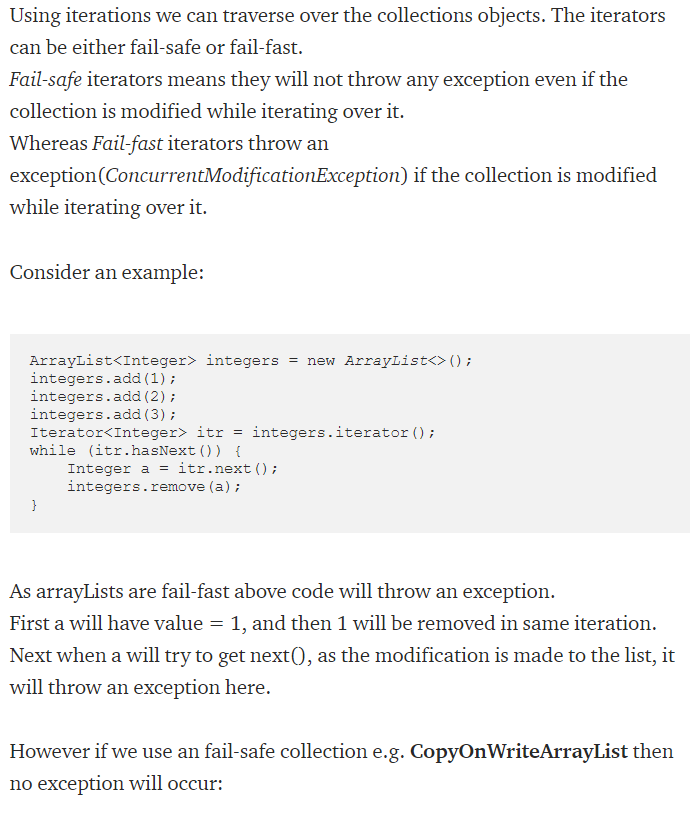


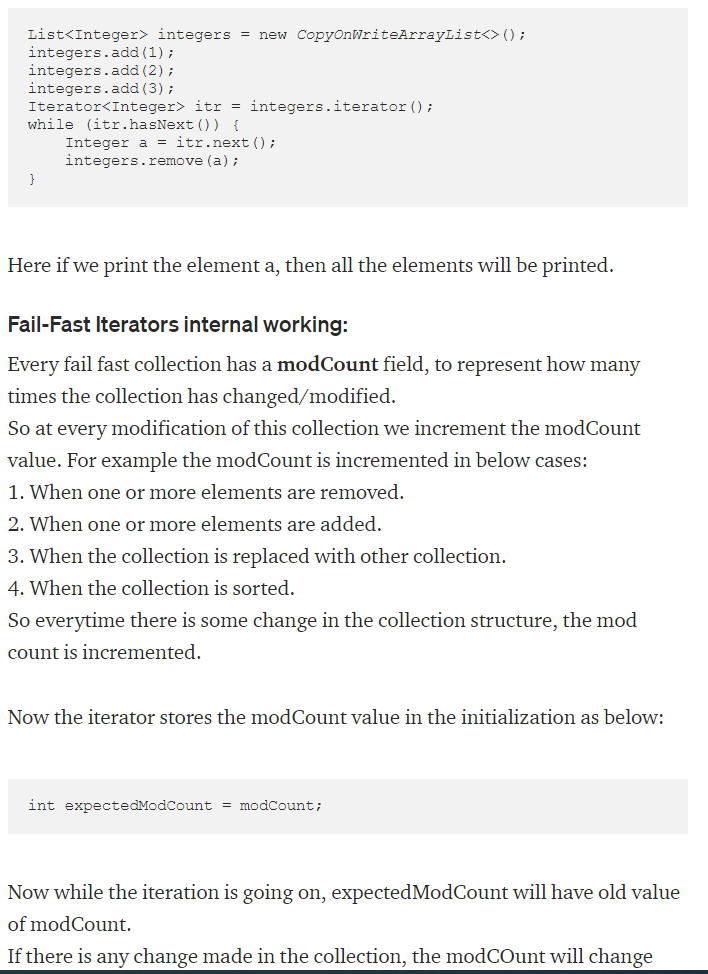


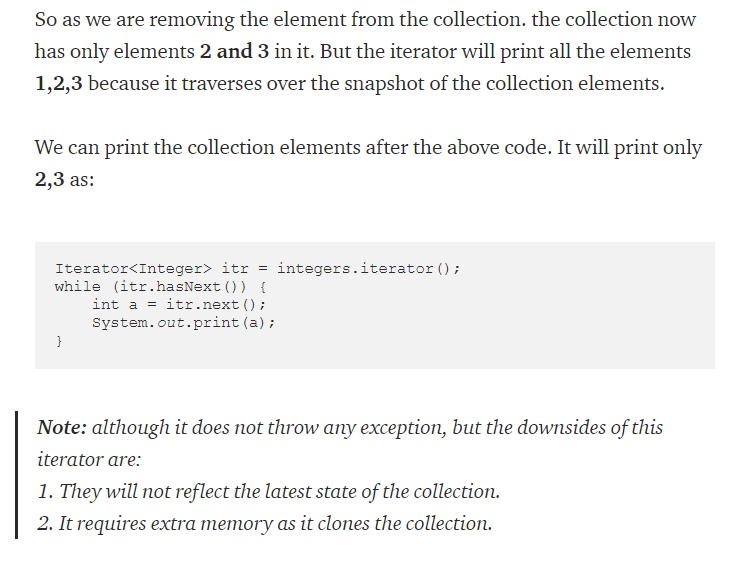
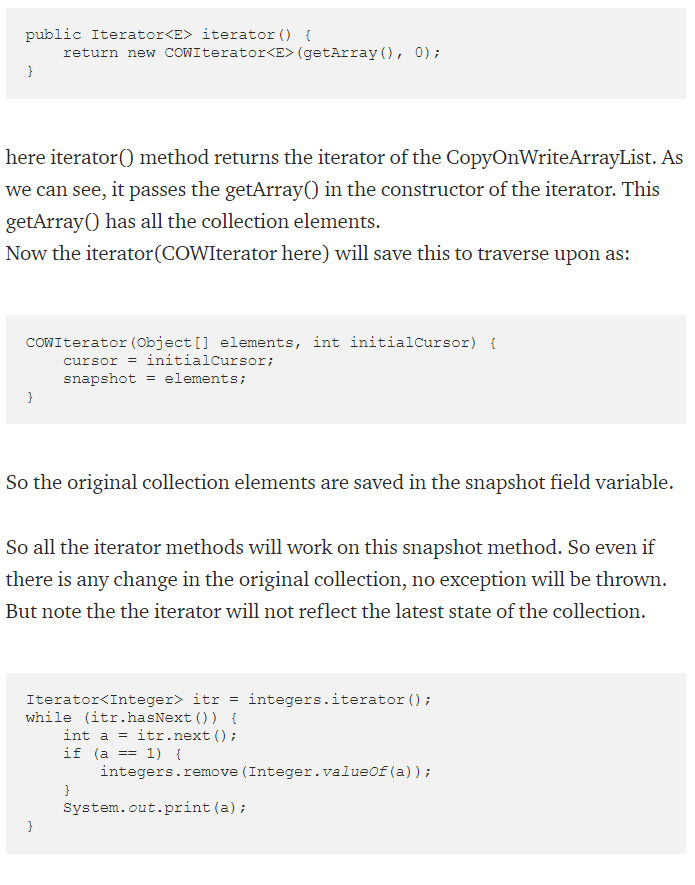
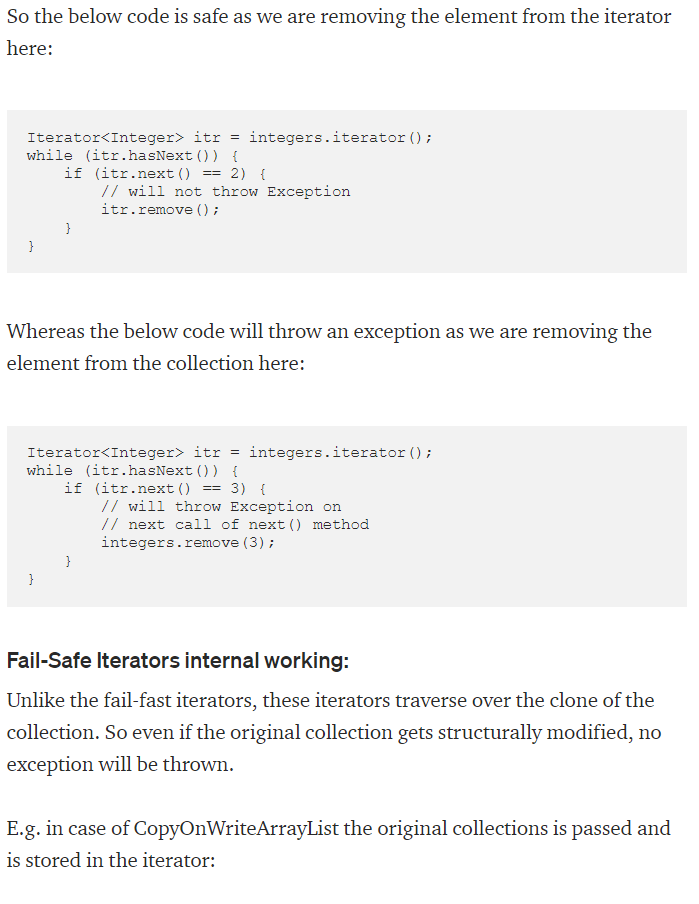
=========

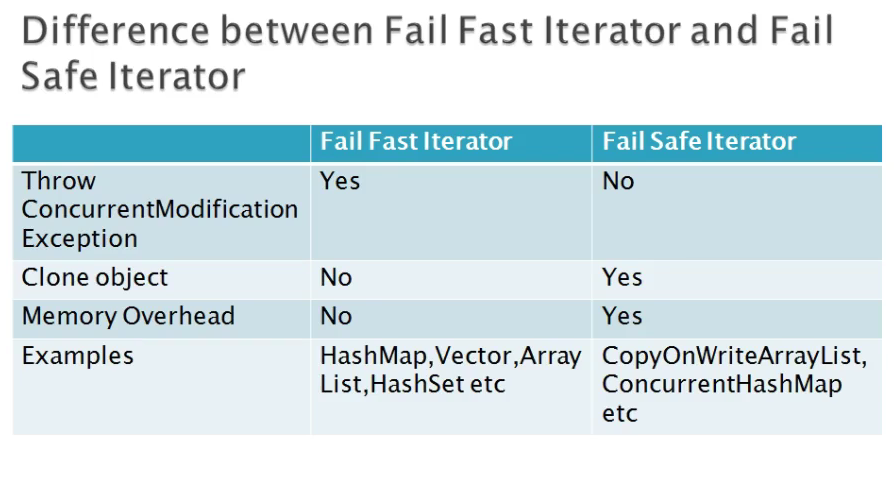












**==========================================================**

**Q) Interface vs Abstract class.**

**Type of methods:** Interface can have only abstract, default and static methods. Abstract class can have abstract and non-abstract methods and don’t have default methods.

**Final Variables:** Variables declared in a Java interface are by default final. An abstract class may contain non-final variables.

**Type of variables:** Abstract class can have final, non-final, static and non-static variables. Interface has only **static and final variables**.

**Implementation:** Abstract class can provide the implementation of interface. Interface can’t provide the implementation of abstract class.

**::::::::::::::::Inheritance vs Abstraction:** A Java interface can be implemented using keyword “implements” and abstract class can be extended using keyword “extends”.

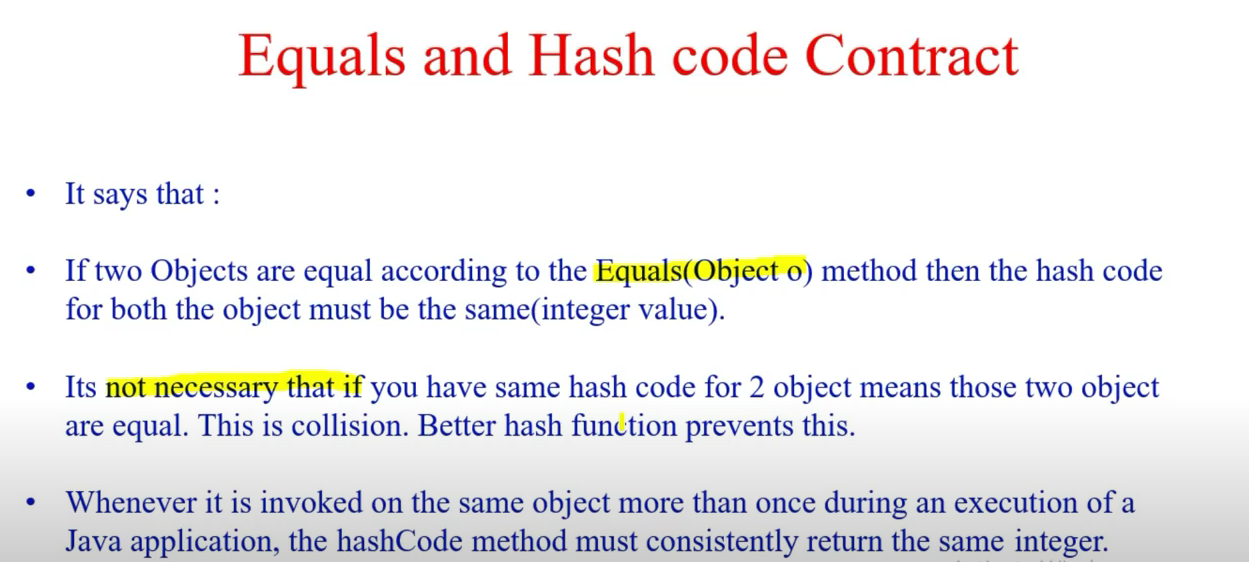
**Multiple implementation:** An interface can extend another Java interface only, an abstract class can extend another Java class and implement multiple Java interfaces.

**Accessibility of Data Members:** Members of a Java interface are public by default. A Java abstract class can have class members like private, protected, etc.

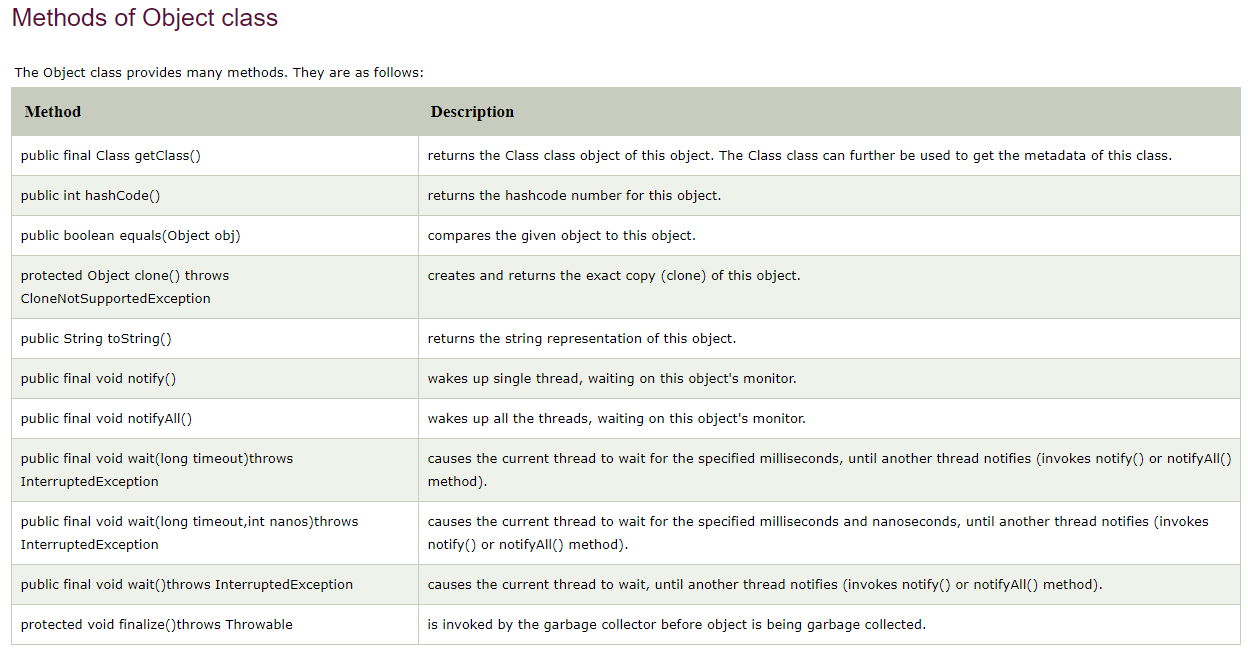
We can create constructor in abstract class: for initializing it’s not static data members through it’s subclass constructor, but not in interface.

**==========================================================**

**Q) Equals and hash code contract.**



**==========================================================**



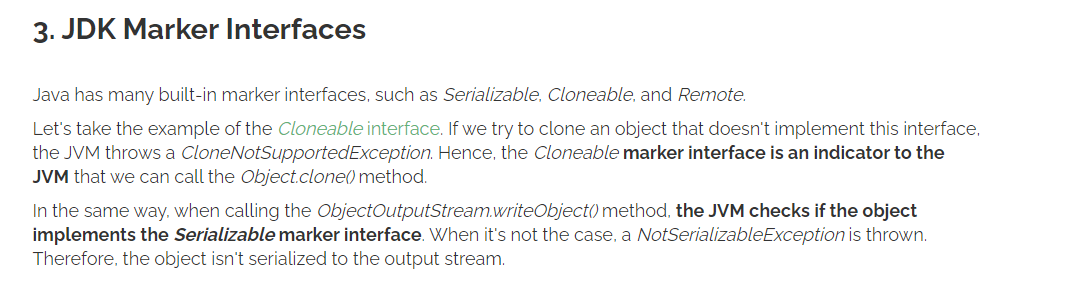
==============

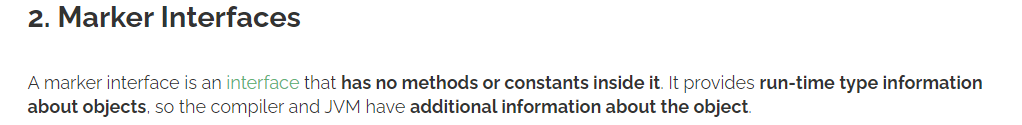
=============================== List out some marker interfaces. ========================

Without methods

============

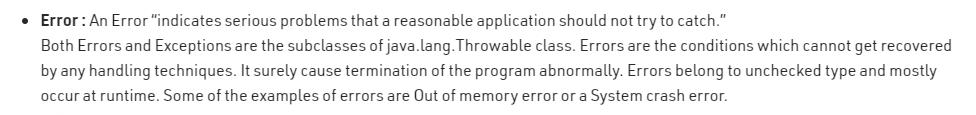
* java. lang. Cloneable **Interface** : This **interface** is used to mark **the** cloning operation. ...
* java. io. Serializable **Interface** : ...
* java. util. EventListener : ...
* java. rmi. Remote :

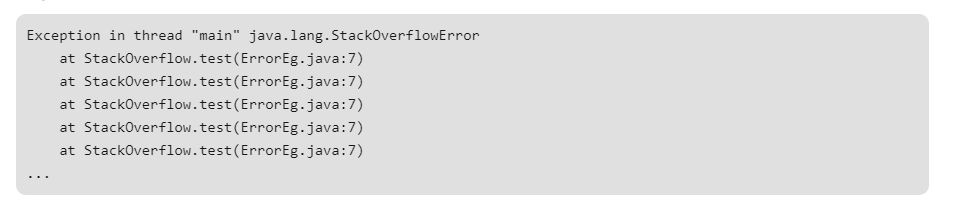


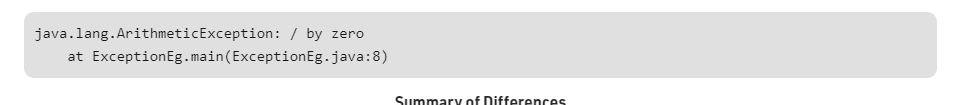
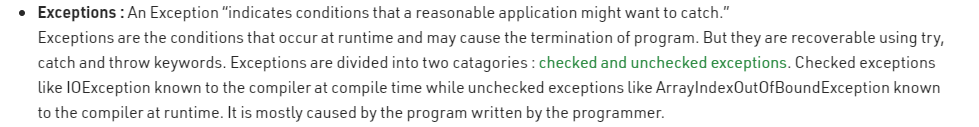


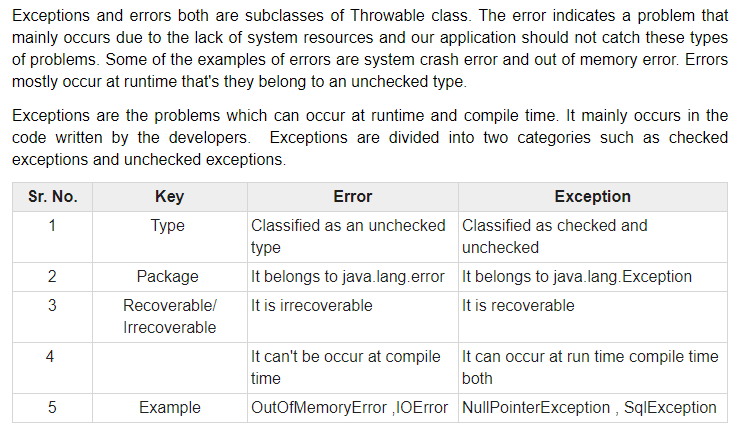
=============================================================

**Difference between error and exception**



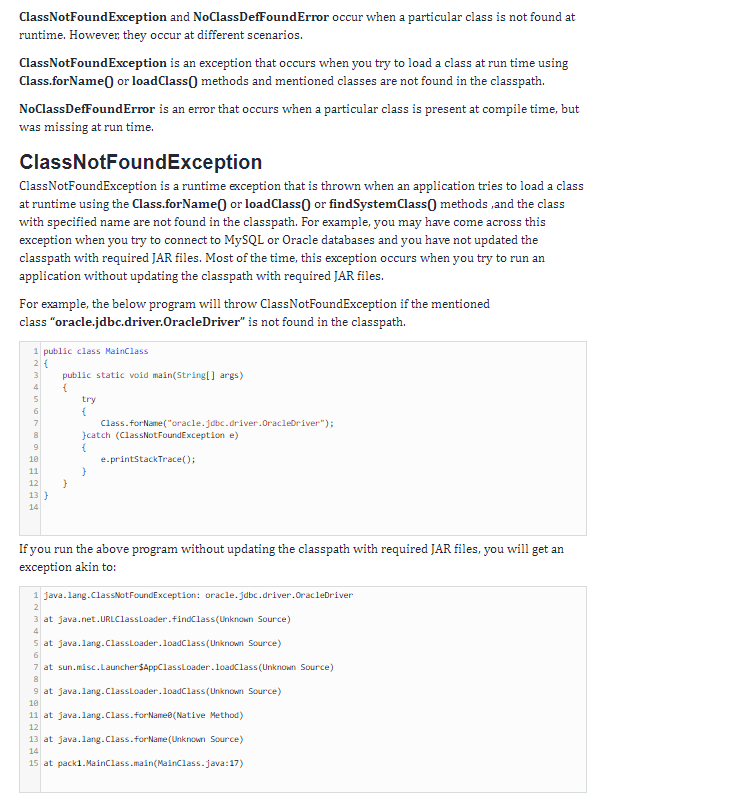


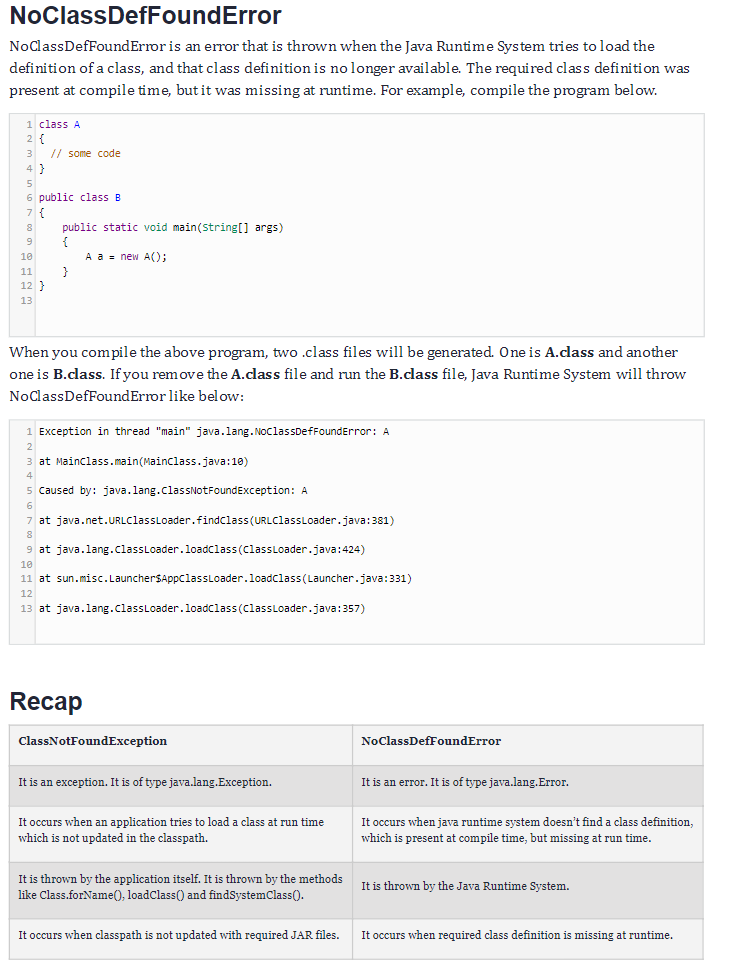




=====================================================================================

# **ClassNotFoundException vs. NoClassDefFoundError**

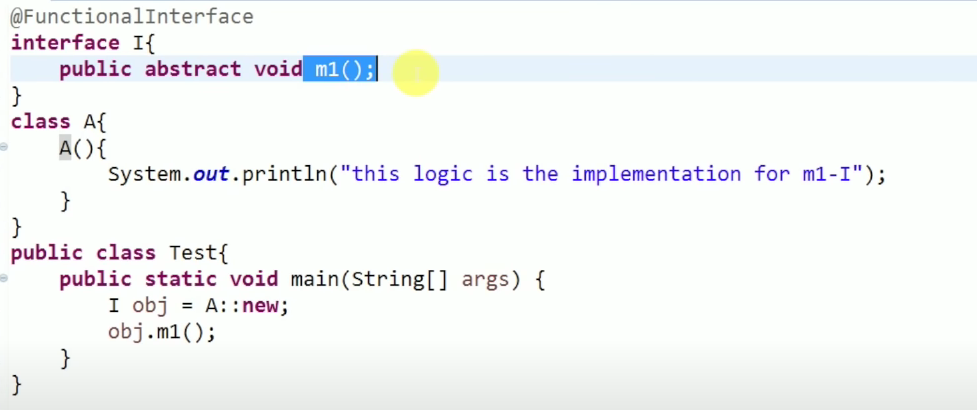




**how to disable autoconfiguration in spring boot**

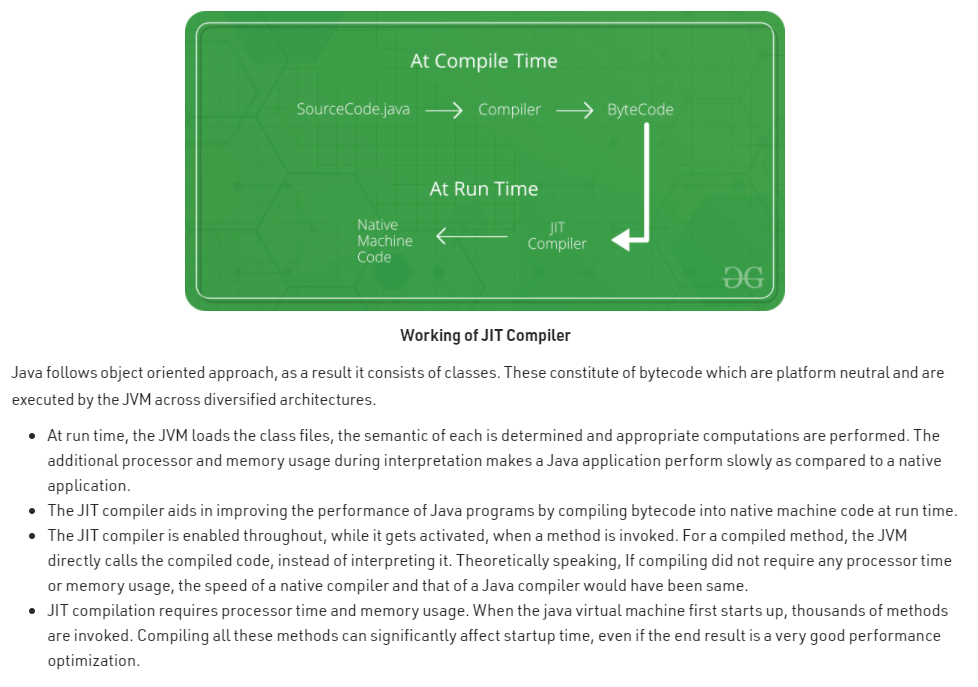


# Constructor Method Reference

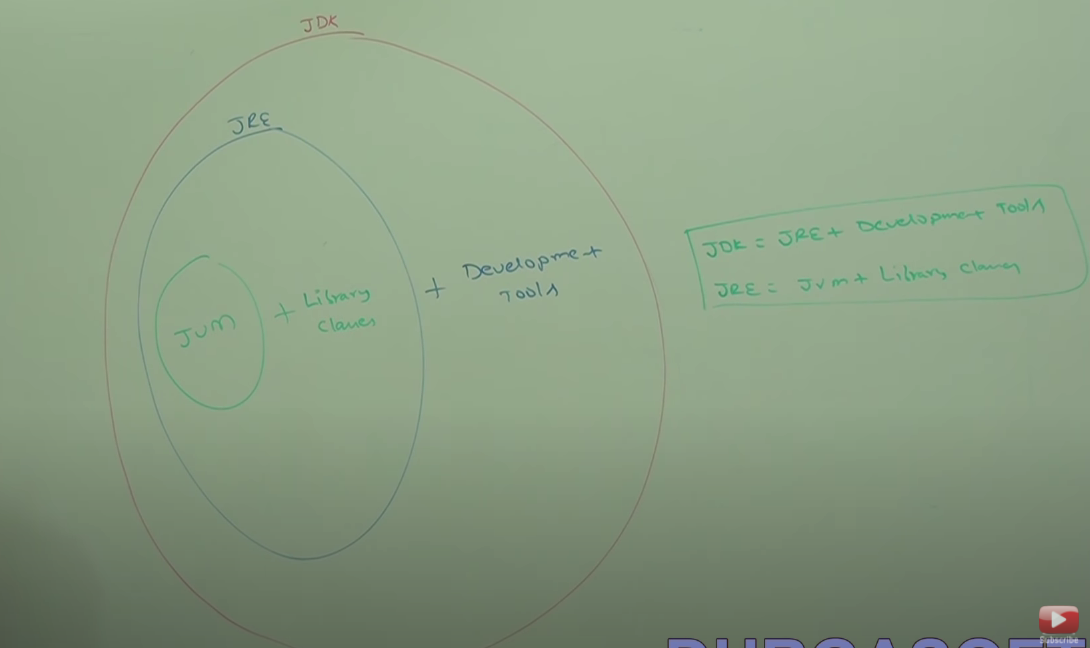


=====================================================================================

JIT COMPILER

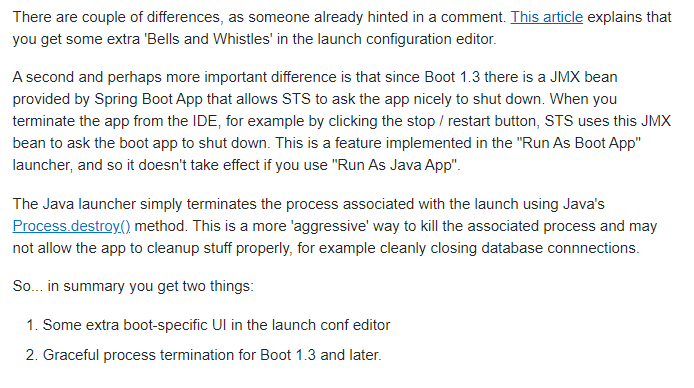


**JDK JRE JVM**

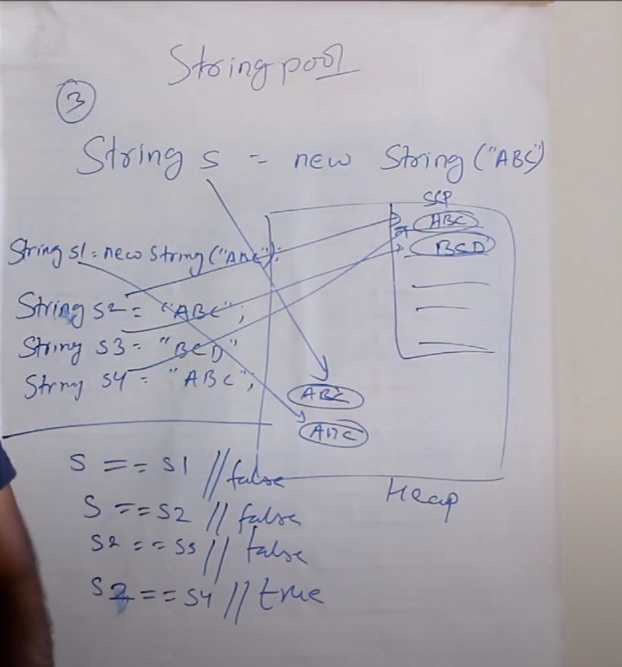


=====================================================================================

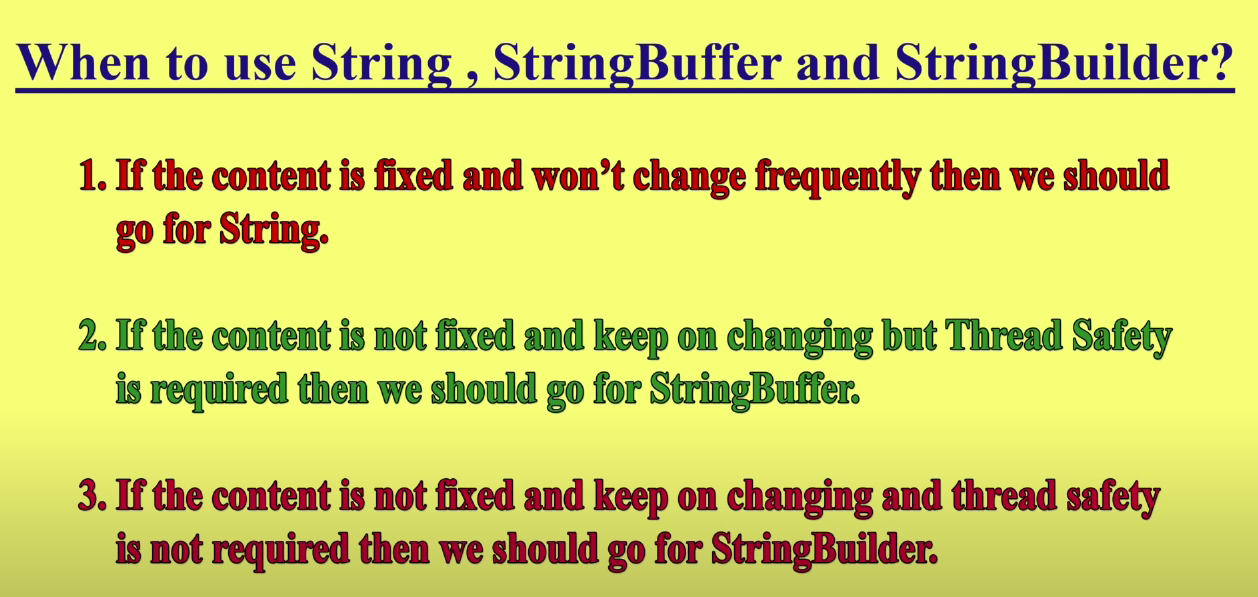
# [Run As: Spring Boot App and Run As: Java Application?](https://stackoverflow.com/questions/37490144/is-there-a-difference-between-run-as-spring-boot-app-and-run-as-java-applicati)



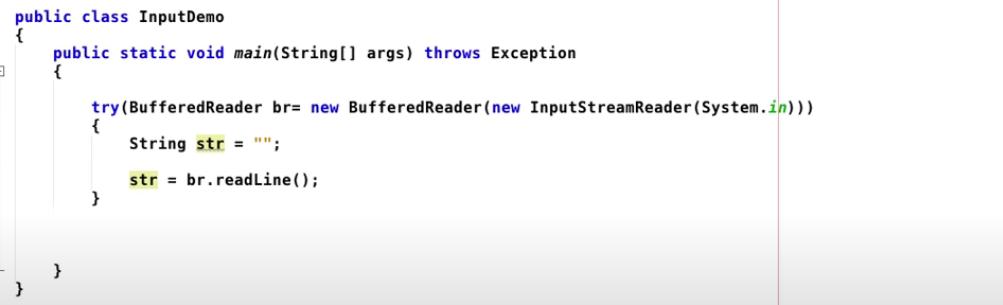
**String contant pool area**



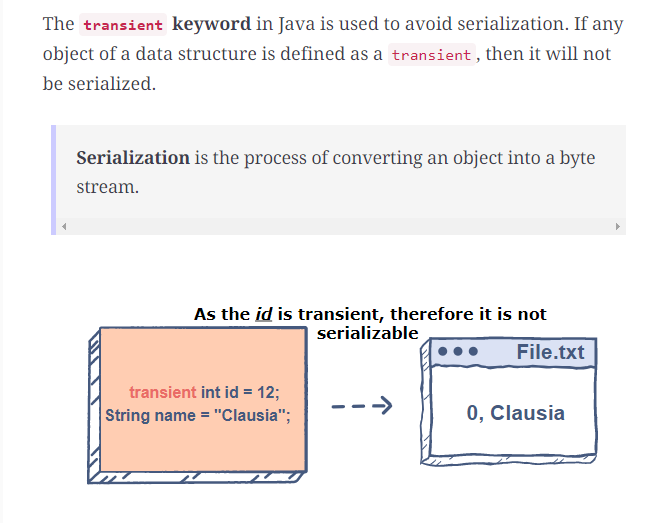
**string vs stringbuffer vs stringbuilder**



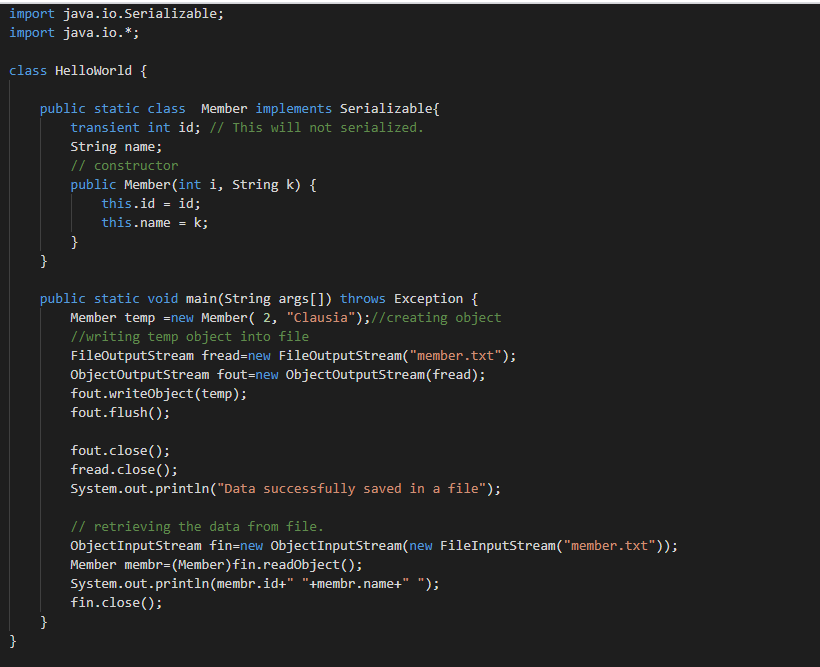
# **Try with Resources**



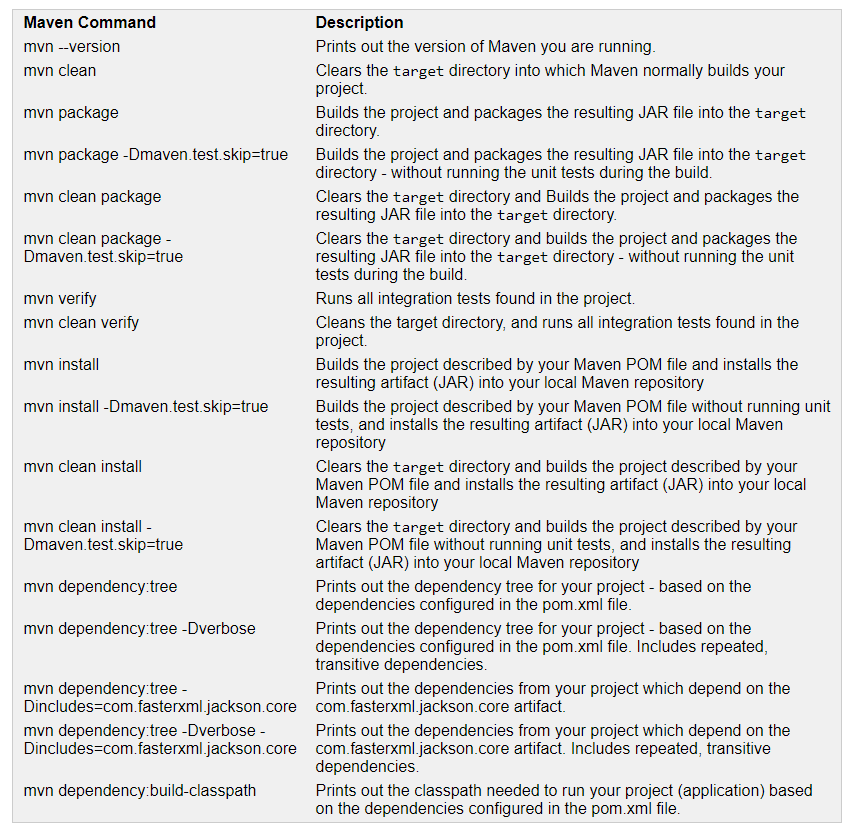
# what-is-the-transient-keyword-in-java

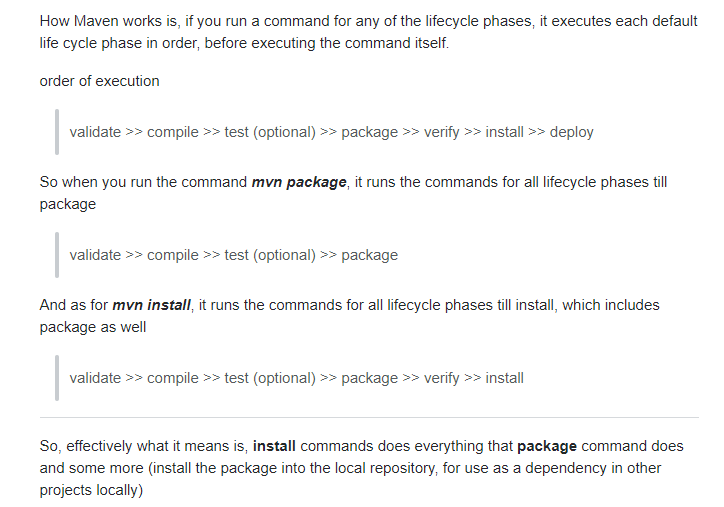






**maven-commands**





Compile time polymorphism and run time polymorphism.

In overloading, method resolution always takes care by the compiler based on reference type, runtime object is dummy in method overloading.

Compile-time polymorphism. Early binding. Static polymorphism.

In overriding, method resolution always takes care by the JVM based on run type object.

Runtime Polymorphism, Dynamic Polymorphism, Late Binding