**Java Important Questions**

**Core Java**

1. Strings in Java
2. StringBuffer
3. StringBuilder
4. String Pool
5. Integers
6. Integer Cache
7. Static Classes
8. Scope of variables
9. Continuous Integration, Continuous Delivery, Continuous Deployment
10. Heap vs Stack memory
11. Coupling in OOP
12. RMI
13. Serial & Throughput garbage collector
14. Volatile and static variables in Java
15. Synchronization
16. Serialization
17. Design Patterns in Java
18. SoftReference and WeakReference in Java
19. Double brace initialization in Java
20. Cloneable Interface and object cloning
21. Marker Interface
22. Virtual Function

**Collections in Java**

1. Fail Fast and Fail Safe iterators
2. ArrayList vs Vector.
3. Internal working of ArrayList
4. LinkedList, Queue, Stack, Priority Queue
5. Set, HashSet, LinkedHashSet and its internal working
6. Map, TreeMap, HashMap, LinkedHashMap and its internal working

**Miscellaneous Questions**

1. Is it possible to make class “static”?
2. Difference b/w equals, == and ===
3. Examples where finally block will not executed in Java.
4. Different ways of creating object in Java  
   *--> Using new keyword*

*--> Using new instance*

*--> Using clone() method*

*--> Using deserialization*

*--> Using the newInstance() method of the Constructor class*

1. How can you avoid serialization in the child class if the base class is implementing the Serializable interface?
2. Can we change the scope of the overridden method in the subclass?
3. Can we modify the throws clause of the superclass method while overriding it in the subclass?
4. String, StringBuilder and StringBuffer

| **Feature** | **String** | **StringBuilder** | **StringBuffer** |
| --- | --- | --- | --- |
| **Introduction** | Introduced in JDK 1.0 | Introduced in JDK 1.5 | Introduced in JDK 1.0 |
| **Mutability** | Immutable | Mutable | Mutable |
| **Thread Safety** | Thread Safe | Not Thread Safe | Thread Safe |
| **Memory Efficiency** | High | Efficient | Less Efficient |
| **Performance** | High(No-Synchronization) | High(No-Synchronization) | Low(Due to Synchronization) |
| **Usage** | This is used when we want immutability. | This is used when Thread safety is not required. | This is used when Thread safety is required. |

1. Integer Cache
2. **Multithreading**
   1. **Java Memory Model**

JMM defines how threads in a multithreaded program interact with memory and each other.

JMM Concepts:

*Happens-Before Relationship:* If one operation happens before the other, the effect of the first operation should be visible to the other.

*Volatile Variables:* These variables ensure a happens-before relationship for read-and-write operations on that variable. When a thread writes to a variable, it shares its local memory with the main memory, ensuring other threads will see the updated value when they read it. Volatile variables are used for simple synchronization scenarios but not for complex ones.

*Memory Visibility:* This property ensures that changes made by one thread to the shared resource should be visible to others. Happens-Before relationship and synchronization mechanisms like locks and monitors ensure memory visibility.