S.NO	Tips
1	Establish whether you have a memory problem.
2	**Reduce the number of temporary objects being used, especially in loops.**
3	**Avoid creating temporary objects within frequently called methods.**
4	**Presize collection objects.**
5	**Reuse objects where possible**.
6	**Empty collection objects before reusing them. (Do not shrink them unless they are very large.)**
7	Use custom conversion methods for converting between data types (especially strings and streams) to reduce the number of temporary objects.
8	Define methods that accept reusable objects to be filled in with data, rather than methods that return objects holding that data. (Or you can return immutable objects.)
9	Create only the number of objects a class logically needs (if that is a small number of objects).
10	Replace strings and other objects with integer constants. Compare these integers by identity.
11	Use primitive data types instead of objects as instance variables.
12	**Avoid creating an object that is only for accessing a method.**
13	Flatten objects to reduce the number of nested objects.
14	Preallocate storage for large collections of objects by mapping the instance variables into multiple arrays.
15	**Use StringBuilder rather than the string concatenation operator (+).**
16	Use methods that alter objects directly without making copies.
17	Create or use specific classes that handle primitive data types rather than wrapping the primitive data types.
18	Consider using a ThreadLocal to provide threaded access to singletons with state.
19	Use the final modifier on instance-variable definitions to create immutable internally accessible objects.
20	Reduce object-creation bottlenecks by targeting the object-creation process.
21	Keep constructors simple and inheritance hierarchies shallow.
22	**Avoid initializing instance variables more than once.**
23	Use the clone() method to avoid calling any constructors.
24	Clone arrays if that makes their creation faster.
25	Create copies of simple arrays faster by initializing them; create copies of complex arrays faster by cloning them.
26	Eliminate object-creation bottlenecks by moving object creation to an alternative time.
27	Create objects early, when there is spare time in the application, and hold those objects until required.
28	**Use lazy initialization when there are objects or variables that may never be used, or when you need to distribute the load of creating objects.**
29	Use lazy initialization only when there is a defined merit in the design, or when identifying a bottleneck which is alleviated using lazy initialization.