## **List.of** vs  **Arrays.asList**

1. List.of can be best used when data set is less and unchanged, while Arrays.asList can be used best in case of large and dynamic data set.
2. List.of take very less overhead space because it has field-based implementation and consume less heap space, both in terms of fixed overhead and on a per-element basis. while Arrays.asList take more overhead space because while initialization it creates more objects in heap.
3. Collection returned by List.of is immutable and hence thread-safe while Collection returned by Arrays.asList is mutable and not thread safe. (Immutable collection instances generally consume much less memory than their mutable counterparts.)
4. List.of doesn't allow null elements while Arrays.asList allows null elements.

Other way of creating object

Class.forName(“com.nt.Employee”).newInstance();

1. In try block if I place a return statement will finally gets exceuted or not?-yes it will executed