- JVM guarantees that synchronized code will be executed by only one thread at a time.
- JAVA keyword **synchronized** is used to create synchronized code and internally it uses locks on Object or Class to make sure only one thread is executing the synchronized code.
- I mean Java synchronization works on locking and unlocking of the resource, so no thread enters into synchronized code.
- We can use synchronized keyword in two ways, one is to make a complete method synchronized and other way is to create synchronized block.

32. How do you sort an object that you created?

- Sort it will be able to sort.
- Also, I can store my objects into a TreeSet or TreeMap → Ex: NEXT PAGE
- Java provides a number of ways to sort a list.
  - COMPARABLE - COMPARATOR interfaces can be used for sorting. In these cases, We should override the compareTo method.
- Another way is List interface sort method which can use a comparator. With this method we can sort ascending or descending.

```
users.sort(Comparator.comparing(User::getUserID));
```

- If we don't want to modify the original list, but return a new sorted list; then we can use the sorted() method from the Stream interface…

```
List<User> sortedUsers = users.stream()
  .sorted(Comparator.comparing(User::getUserID))
  .collect(Collectors.toList());
```

33. Difference between Hashtable and HashMap in Java?

There are several differences between HashMap and Hashtable in Java:

- Hashtable is synchronized, whereas HashMap is not. This makes HashMap better for non-threaded applications, as unsynchronized Objects typically perform better than synchronized ones.
- Hashtable does not allow null keys or values. HashMap allows one null key and any number of null values.
- For example; one of HashMap's subclasses is LinkedHashMap, so in the event that you'd want predictable iteration order (which is insertion order by default), you could easily swap out the HashMap for a LinkedHashMap. This wouldn't be as easy if you were using Hashtable.

If synchronization is not an issue for me, I' prefer using HashMap. If it becomes an issue, then I prefer Collections.synchronizedMap() or ConcurrentHashMap.

- Both Hashtable and HashMap implements Map interface and both are Key and Value.
- HashMap is not thread-safe while Hashtable is a thread-safe collection.
- Second important difference is performance since HashMap is not synchronized.
  It performed better than Hashtable. → Collections.synchronizedMap(…Map…);

34. How would you handle Exception?

I would use try-catch-finally approach to handle the Exception

- 1- I would put my code that might generate an exception inside a try-catch block. With try-catch block I can rethrow an exception or try to perform my recovery steps. Also, If needed I can use multi or Union Catch blocks
- 2- I can also use throws keyword. BUT it does mean that anyone that calls my method now needs to handle it too!
- 3- Another way is AutoCloseable: When we place references that are AutoCloseable in the try declaration, then we don't need to close the resource ourselves. We can still use a finally block, though, to do any other kind of cleanup we want. try-with