

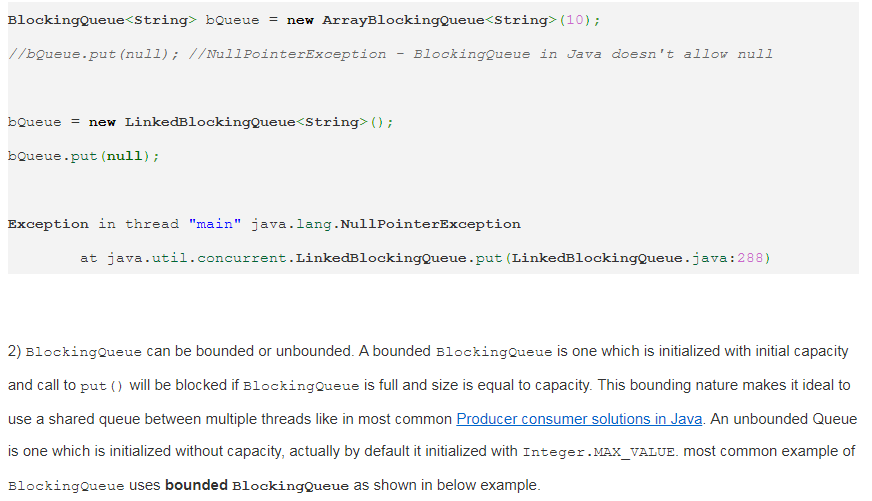
# **BlockingQueue:**A blocking queue is a queue that blocks when you try to dequeue from it and the queue is empty, or if you try to enqueue items to it and the queue is already full. A thread trying to dequeue from an empty queue is blocked until some other thread inserts an item into the queue. A thread trying to enqueue an item in a full queue is blocked until some other thread makes space in the queue, either by dequeuing one or more items or clearing the queue completely.

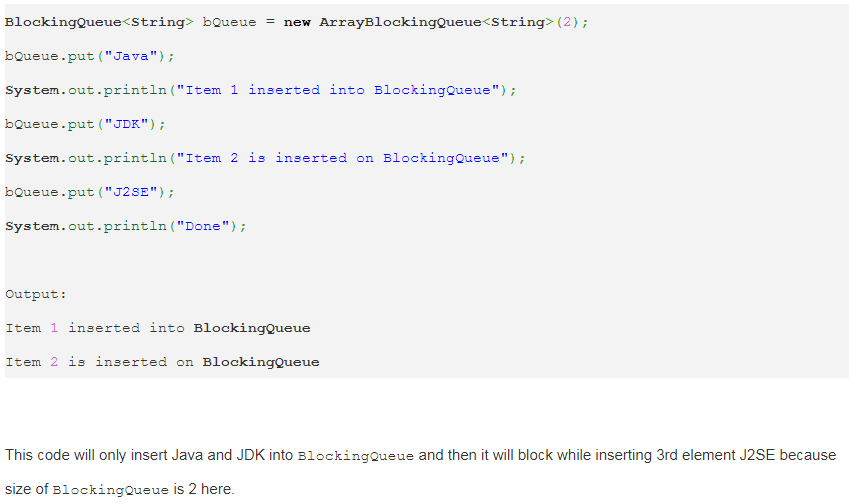
The java.util.concurrent package contains a set of synchronized Queue interfaces and classes. Blocking Queue extends Queue with operations that wait for the queue to become nonempty when retrieving an element and for space to become available in the queue when storing an element.

Here is a diagram showing two threads cooperating via a blocking queue:

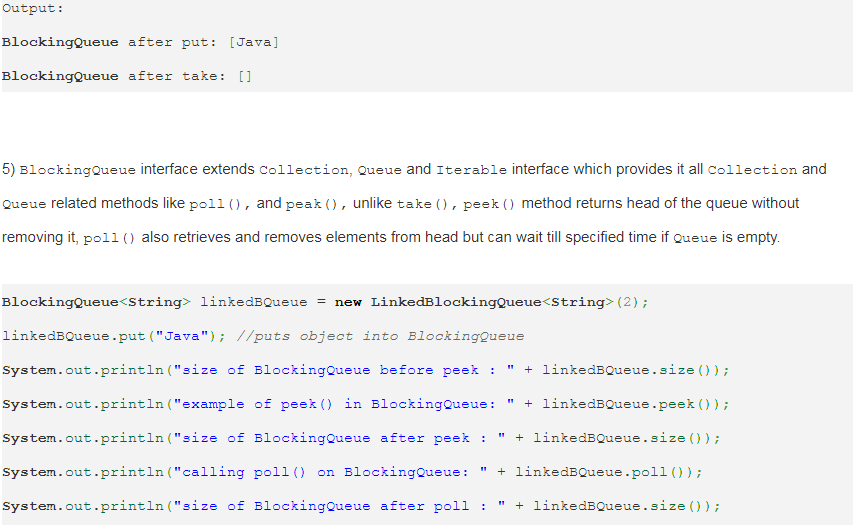
|  |
| --- |
| A BlockingQueue with one thread putting into it, and another thread taking from it. |
| **A BlockingQueue with one thread putting into it, and another thread taking from it.** |

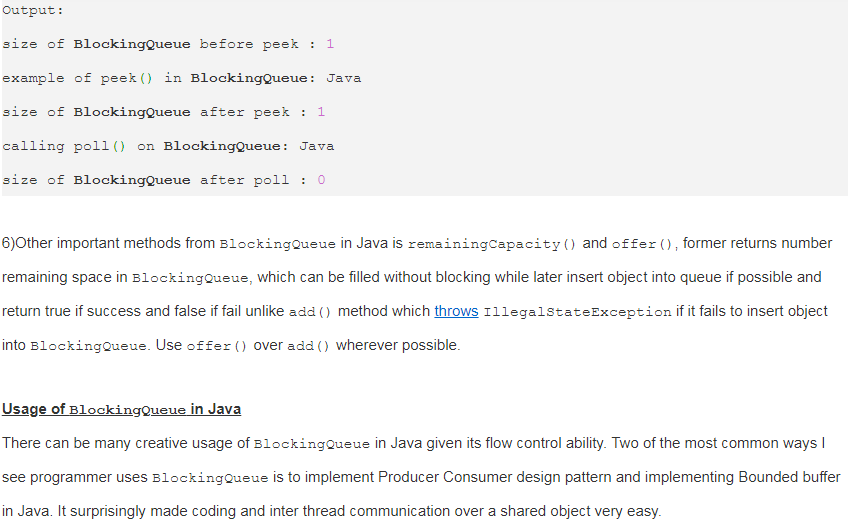
Following list of points about BlockingQueue in Java will help to learn and understand more about it:

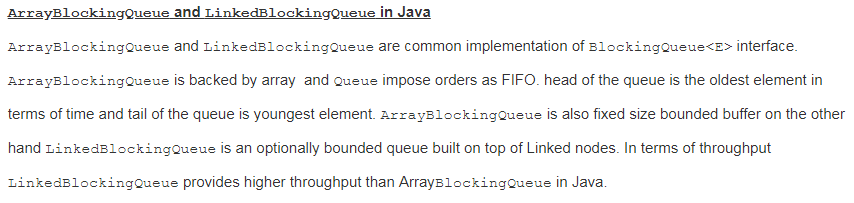
1. BlockingQueue in Java doesn't allow null elements, various implementation of BlockingQueue like ArrayBlockingQueue, LinkedBlockingQueue throws [NullPointerException](http://javarevisited.blogspot.sg/2012/06/common-cause-of-javalangnullpointerexce.html) when you try to add null on queue.  
   

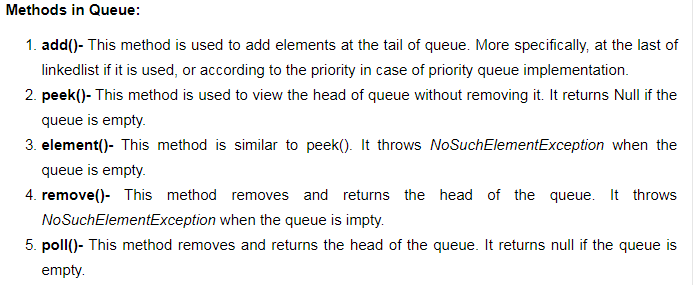




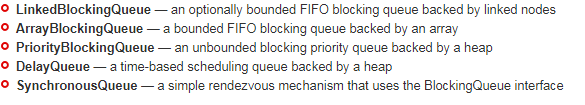


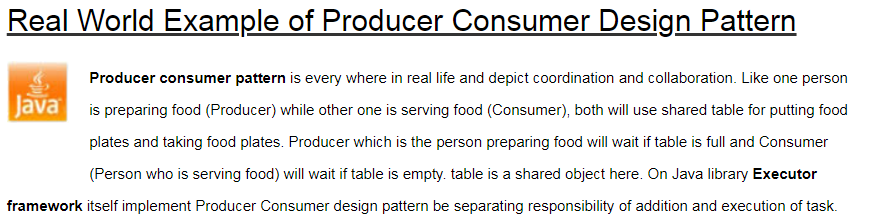




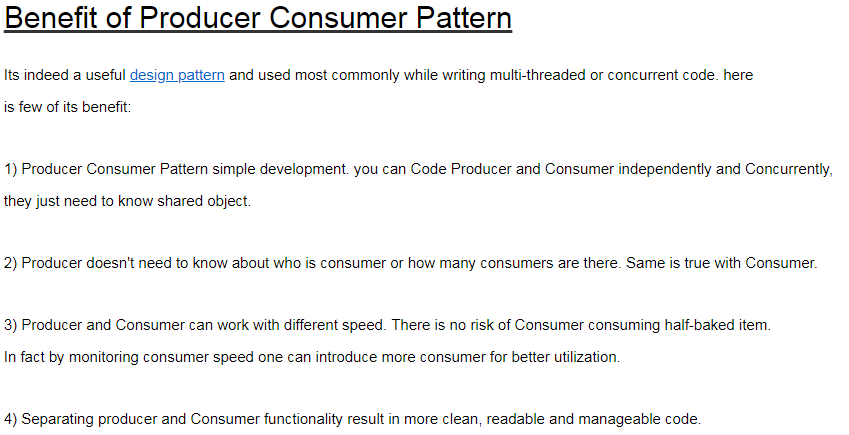


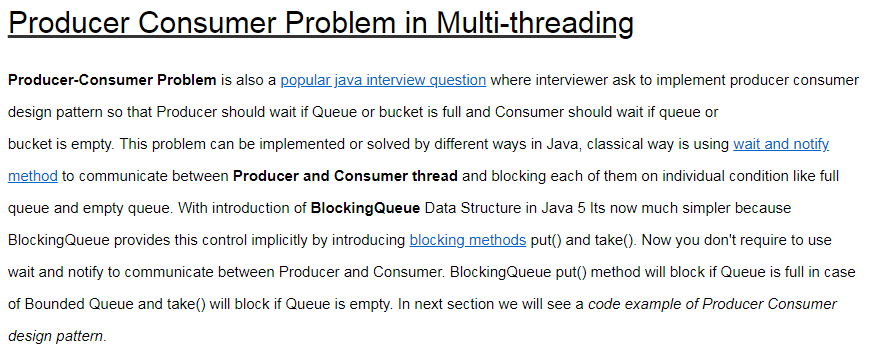
This interface is implemented by the following classes:

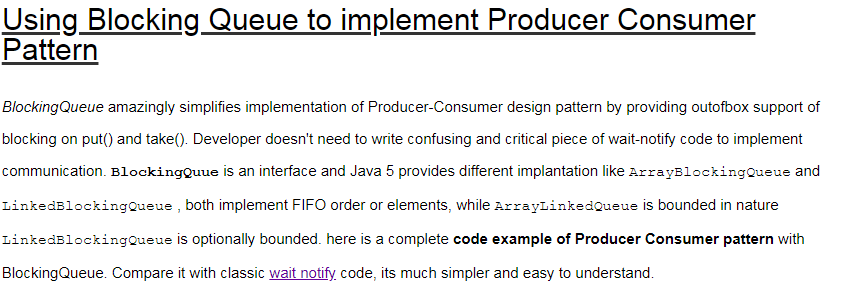


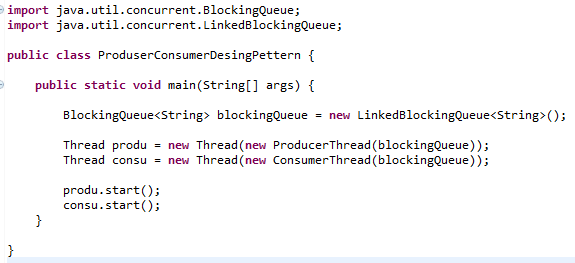


**Problem**  
To make sure that the producer won’t try to add data into the buffer if it’s full and that the consumer won’t try to remove data from an empty buffer.

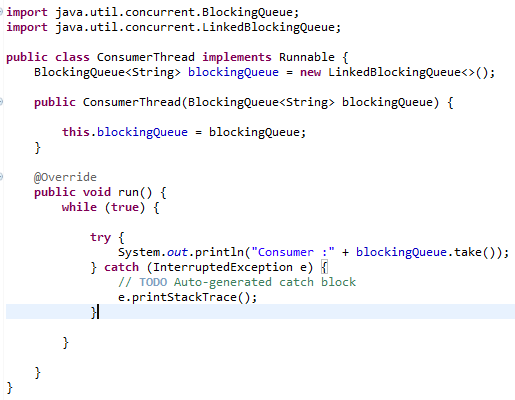


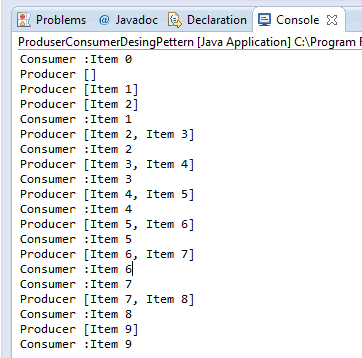










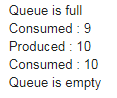


You see Producer Thread  produced number and Consumer thread consumes it in FIFO order because blocking queue allows elements to be accessed in FIFO.

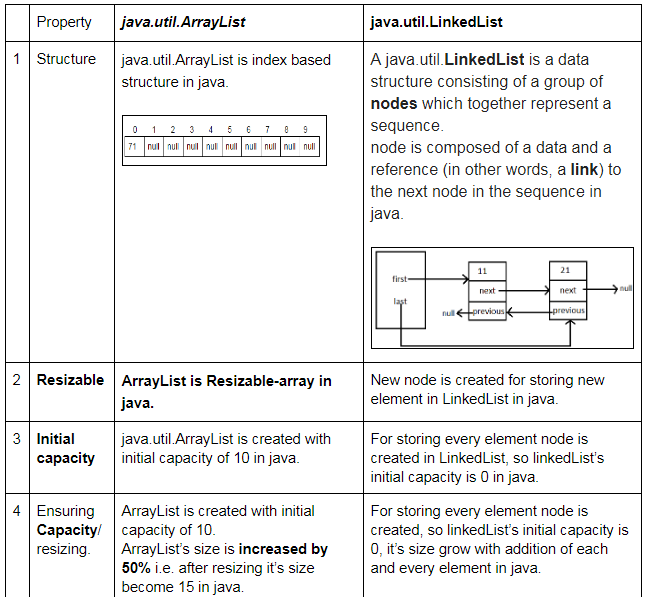


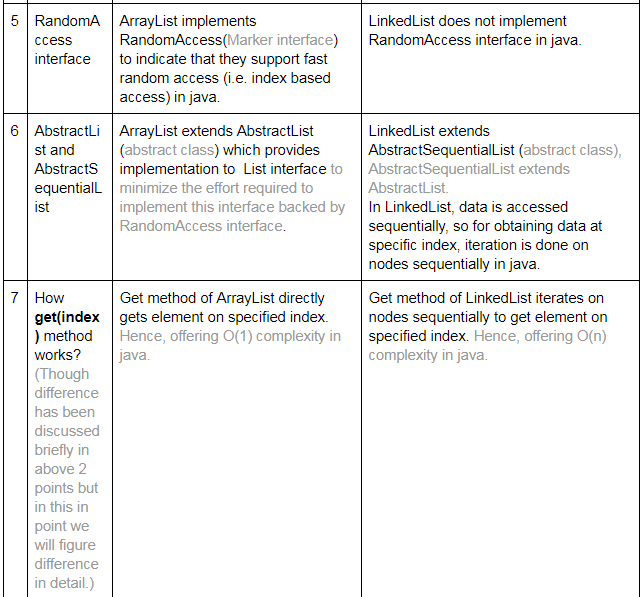


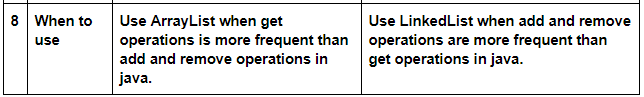


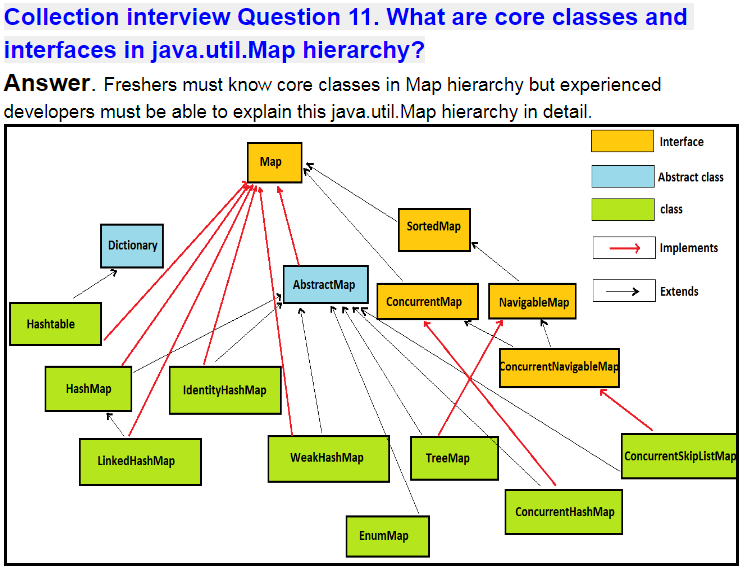


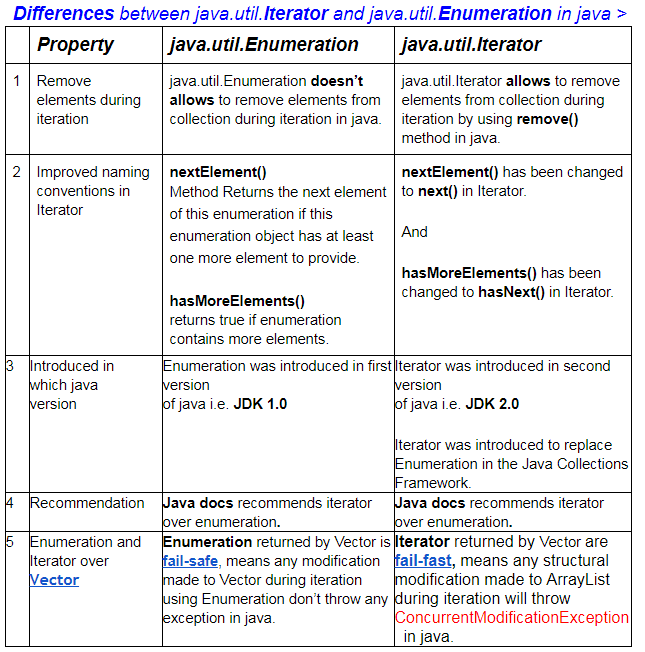
# What are differences between ArrayList and LinkedList in java?

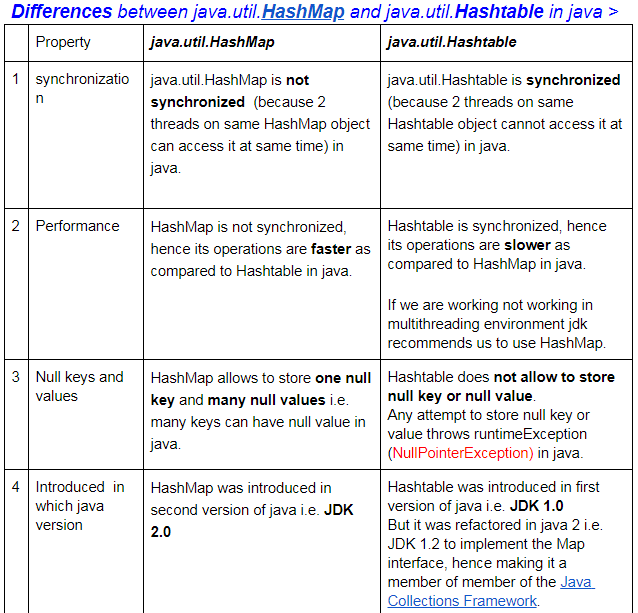


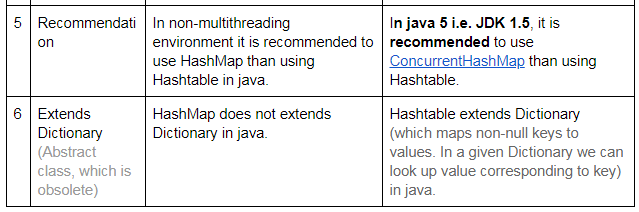


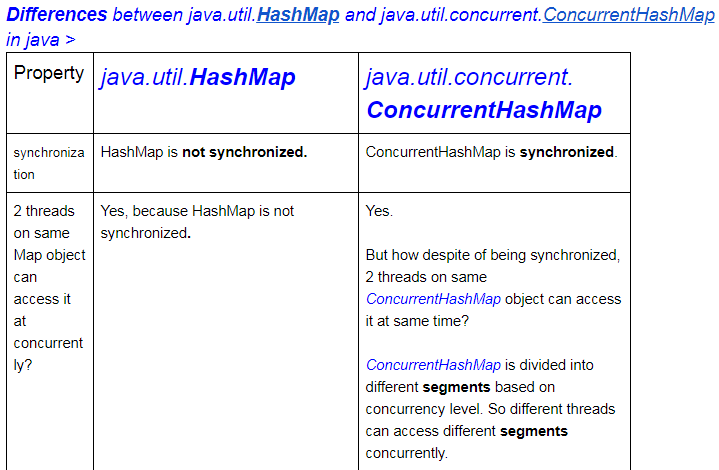


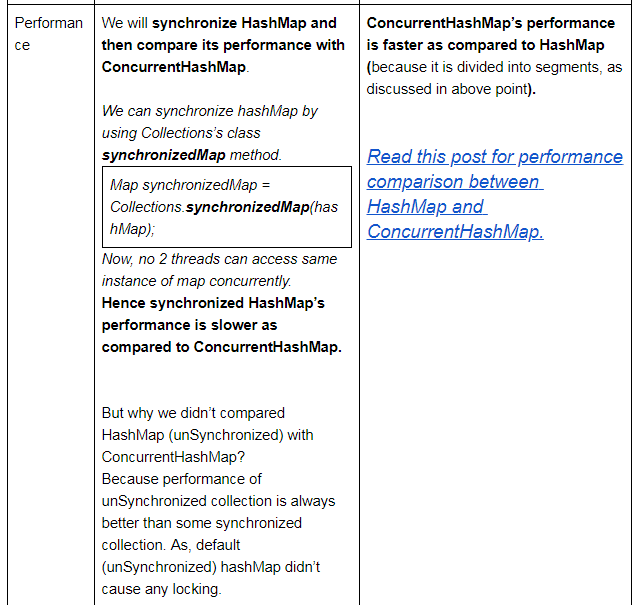


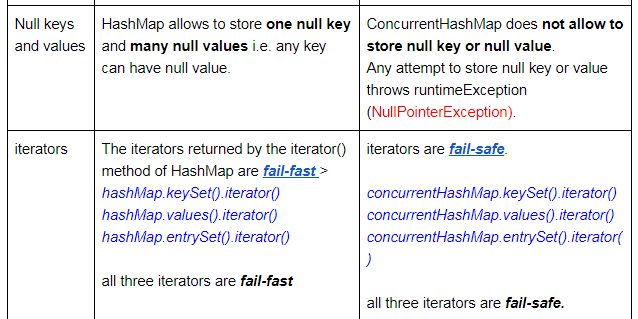


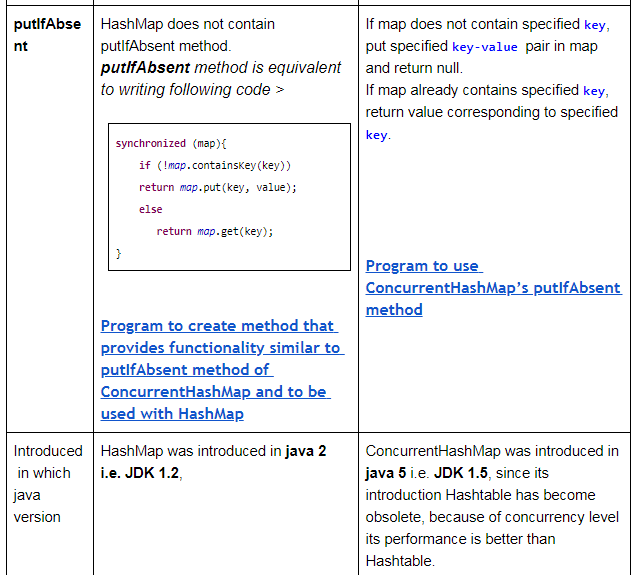


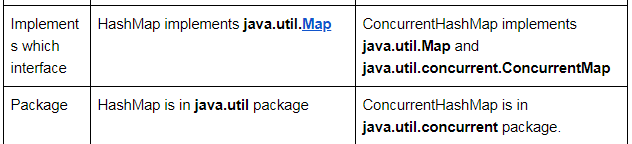


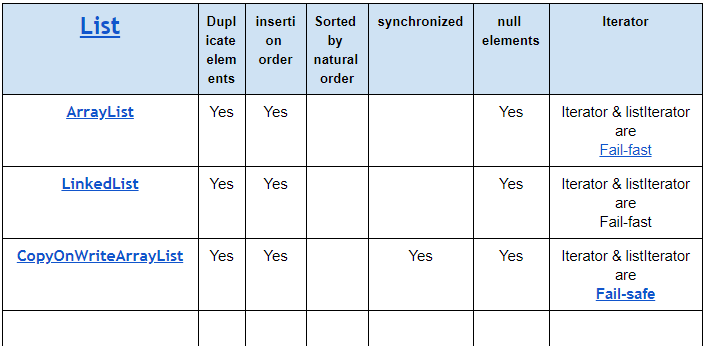


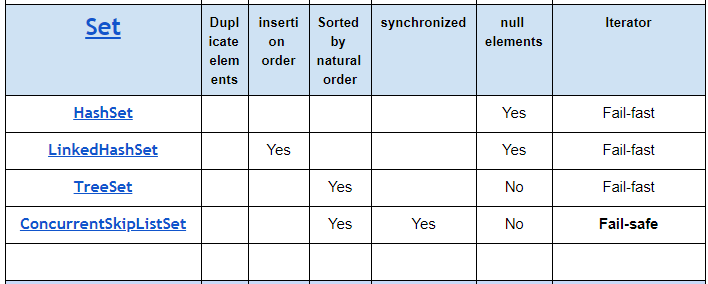


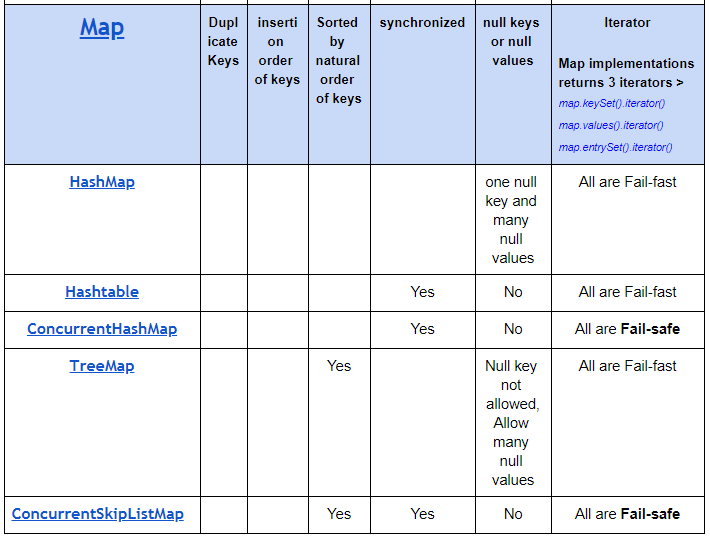


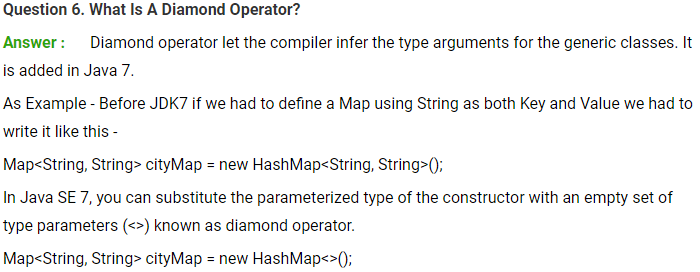


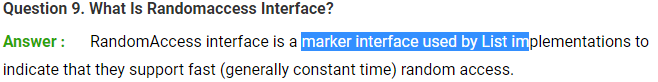


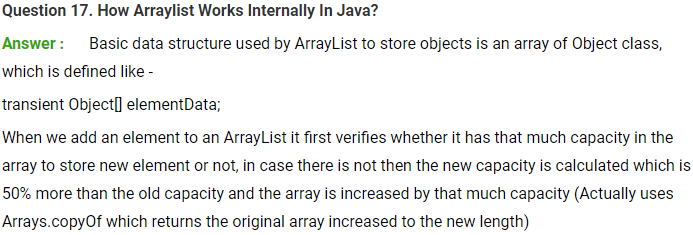


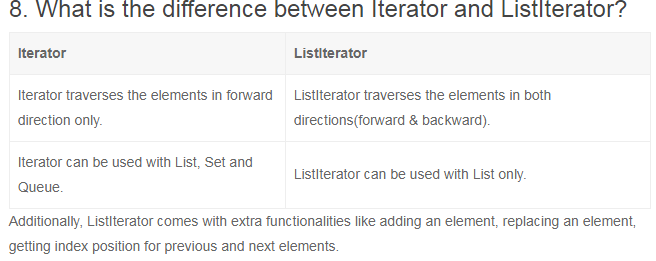












## Recap

|  |  |
| --- | --- |
| **ClassNotFoundException** | **NoClassDefFoundError** |
| It is an exception. It is of type java.lang.Exception. | It is an error. It is of type java.lang.Error. |
| It occurs when an application tries to load a class at run time which is not updated in the classpath. | It occurs when java runtime system doesn’t find a class definition, which is present at compile time, but missing at run time. |
| It is thrown by the application itself. It is thrown by the methods like Class.forName(), loadClass() and findSystemClass(). | It is thrown by the Java Runtime System. |
| It occurs when classpath is not updated with required JAR files. | It occurs when required class definition is missing at runtime. |

Serialization in java :

**Marker interface:**

Marker interface in Java. It is an empty interface (no field or methods). Examples ofmarker interface are Serializable, Clonnable and Remote interface. ... A class that implements the Cloneable interface indicates that it is legal for clone() method to make a field-for-field copy of instances of that class.

public interface Serializable

{

// nothing here

}

**Serialization:**

**It is a mechanism to write state of an object into byte stream.**

**To perform serialization we have to implement Serializable interface.**

**It is a Marker Interface. Which definition we define above.**

It is mainly used in Hibernate, RMI, JPA, EJB and JMS technologies.

## Java Serialization with Inheritance (IS-A Relationship)

If a class implements serializable then all its sub classes will also be serializable. Let's see the example given below:

**import** java.io.Serializable;

**class** Person **implements** Serializable{

**int** id;

String name;

Person(**int** id, String name) {

**this**.id = id;

**this**.name = name;

}

}

**class** Student **extends** Person{

String course;

**int** fee;

**public** Student(**int** id, String name, String course, **int** fee) {

**super**(id,name);

**this**.course=course;

**this**.fee=fee;

}

}

Now you can serialize the Student class object that extends the Person class which is Serializable. Parent class properties are inherited to subclasses so if parent class is Serializable, subclass would also be.

## Java Serialization with Aggregation (HAS-A Relationship)

If a class has a reference to another class, all the references must be Serializable otherwise serialization process will not be performed. In such case, *NotSerializableException* is thrown at runtime.

**class** Address{

String addressLine,city,state;

**public** Address(String addressLine, String city, String state) {

**this**.addressLine=addressLine;

**this**.city=city;

**this**.state=state;

}

}

**import** java.io.Serializable;

**public** **class** Student **implements** Serializable{

**int** id;

String name;

Address address;//HAS-A

**public** Student(**int** id, String name) {

**this**.id = id;

**this**.name = name;

}

}

Since Address is not Serializable, you can not serialize the instance of Student class.

#### Note: All the objects within an object must be Serializable.

## Java Serialization with the static data member

If there is any static data member in a class, it will not be serialized because static is the part of class not object.

**class** Employee **implements** Serializable{

**int** id;

String name;

**static** String company="SSS IT Pvt Ltd";//it won't be serialized

**public** Student(**int** id, String name) {

**this**.id = id;

**this**.name = name;

}

}

## Java Serialization with array or collection

Rule: In case of array or collection, all the objects of array or collection must be serializable. If any object is not serialiizable, serialization will be failed.

## Externalizable in java

The Externalizable interface provides the facility of writing the state of an object into a byte stream in compress format. It is not a marker interface.

Externalization serves the purpose of custom Serialization, where we can decide what to store in stream.

The Externalizable interface provides two methods:

* **public void writeExternal(ObjectOutput out) throws IOException**
* **public void readExternal(ObjectInput in) throws IOException**

## Java Transient Keyword

If you don't want to serialize any data member of a class, you can mark it as transient.

Visit next page for more details.