### Class loader and explain the types of class loader

Class loaders are the part of the Java Runtime Environment that dynamically loads Java classes into the Java virtual machine. It is responsible for locating libraries, reading there content and loading the classes contained within the libraries. When JVM is started three class loaders are used  
  
1. **Bootstrap class loader**  
  
2. **Extensions class loader**  
  
3. **System class loader**  
**Bootstrap class loader** loads the core java libraries. It is written in native code. The bootstrap class loader is responsible for loading key java classes like java.lang.Object and other runtime code into memory. The runtime classes are packaged inside jre/lib/rt.jar file.   
  
**Extensions class loader** loads the code in the extension directories. It is implemented by ExtClassLoader class.  
  
**System class loader** the code found on the java.class.path which map to the system class path variables. It is implemented by AppClassLoader class. All user classes by default are load by the system class loader.

**ArrayList and LinkedList:**

ArrayList and LinkedList. Which of the two List implementations you use depends on

your specific needs. If you need to support random access, without inserting or removing elements from any place other than the end, then ArrayList offers the optimal collection.If, however, you need to frequently add and remove elements from the middle of the list and only access the list elements sequentially, then LinkedList offers the better implementation.

**Java ArrayList and ConcurrentModificationException:**

ArrayList is one of the basic implementations of List interface and

it’s part of Java Collections Framework. We can use iterator to traverse through ArrayList elements.

Let’s check a sample program of ArrayList.

ConcurrentListExample.java

package com;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import java.util.concurrent.CopyOnWriteArrayList;

public class **ConcurrentListExample** {

public static void main(String[] args) {

List<String> list = new ArrayList<>();

list.add("1");

list.add("2");

list.add("3");

list.add("4");

list.add("5");

// get the iterator

Iterator<String> it = list.iterator();

//manipulate list while iterating

while(it.hasNext()){

System.out.println("list is:"+list);

String str = it.next();

System.out.println(str);

**if(str.equals("2"))list.remove("5"); // throw ConcurrentModificationException**

//below code don't throw ConcurrentModificationException

//because it doesn't change modCount variable of list

if(str.equals("4")) list.set(1, "4");

}

}

}

When we run above program, we get java.util.ConcurrentModificationException as soon as the ArrayList is modified.

It happens because ArrayList iterator is fail-fast by design. What it means is that once the iterator is created,

if the **ArrayList** is modified, it throws **ConcurrentModificationException**.