Let’s see the concurrent modification exception scenario with an example.

```
package com.journaldev.ConcurrentModificationException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

public class ConcurrentModificationExceptionExample {
        public static void main(String args[]) {
                List<String> myList = new ArrayList<String>();
                myList.add("1");
                myList.add("2");
                myList.add("3");
                myList.add("4");
                myList.add("5");
                Iterator<String> it = myList.iterator();
                while (it.hasNext()) {
                        String value = it.next();
                        System.out.println("List Value:" + value);
                        if (value.equals("3"))
                                myList.remove(value);
                }
                Map<String, String> myMap = new HashMap<String, String>();
                myMap.put("1", "1");
                myMap.put("2", "2");
                myMap.put("3", "3");
                Iterator<String> it1 = myMap.keySet().iterator();
                while (it1.hasNext()) {
                        String key = it1.next();
                        System.out.println("Map Value:" + myMap.get(key));
                        if (key.equals("2")) {
                                myMap.put("1", "4");
                                // myMap.put("4", "4");
                        }
                }

        }
}
```

Above program will throw java.util.ConcurrentModificationException when executed
The following methods can be used to avoid ConcurrentModificationException:

1. You can convert the list to an array and then iterate on the array. This approach works well for small or medium size list but if the list is large then it will affect the performance a lot.
2. You can lock the list while iterating by putting it in a synchronized block. This approach is not recommended because it will cease the benefits of multithreading.
3. If you are using JDK1.5 or higher then you can use **ConcurrentHashMap** and **CopyOnWriteArrayList** classes. This is the recommended approach to avoid concurrent modification exception.
4. You can use the iterator remove() function to remove the object from underlying collection object. But in this case, you can remove the same object and not any other object from the list.

Let's run an example using Concurrent Collection classes.

```
package com.journaldev.ConcurrentModificationException;

import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.OnWriteArrayList;

public class AvoidConcurrentModificationException {

public static void main(String[] args) {

        List<String> myList = new OnWriteArrayList<String>();

        myList.add("1");
        myList.add("2");
        myList.add("3");
        myList.add("4");
        myList.add("5");

        Iterator<String> it = myList.iterator();
        while (it.hasNext()) {
                String value = it.next();
                System.out.println("List Value:" + value);
                if (value.equals("3")) {
                        myList.remove("4");
                        myList.add("6");
                        myList.add("7");
                }
        }
```

```
                System.out.println("List Size:" + myList.size());

                Map<String, String> myMap = new ConcurrentHashMap<String, String>();
                myMap.put("1", "1");
                myMap.put("2", "2");
                myMap.put("3", "3");

                Iterator<String> it1 = myMap.keySet().iterator();
                while (it1.hasNext()) {
                        String key = it1.next();
                        System.out.println("Map Value:" + myMap.get(key));
                        if (key.equals("1")) {
                                myMap.remove("3");
                                myMap.put("4", "4");
                                myMap.put("5", "5");
                        }
                }

                System.out.println("Map Size:" + myMap.size());
        }

}
```

The output of the above program is shown below. You can see that there is no ConcurrentModificationException being thrown by the program

```
List Value:1
List Value:2
List Value:3
List Value:4
List Value:5
List Size:6
Map Value:1
Map Value:2
Map Value:4
Map Value:5
Map Size:4
```

From the above example it’s clear that:

1. Concurrent Collection classes can be modified safely, they will not throw ConcurrentModificationException.
2. In case of CopyOnWriteArrayList, iterator doesn’t accommodate the changes in the list and works on the original list.
3. In case of ConcurrentHashMap, the behaviour is not always the same.

4. Use for loop to avoid java.util.ConcurrentModificationException

If you are working on single-threaded environment and want your code to take care of the extra added objects in the list then you can do so using for loop rather than an Iterator.

```
    for(int i = 0; i<myList.size(); i++){
            System.out.println(myList.get(i));
            if(myList.get(i).equals("3")){
            myList.remove(i);
            i--;
            myList.add("6");
    }
}
```