Legacy Classes

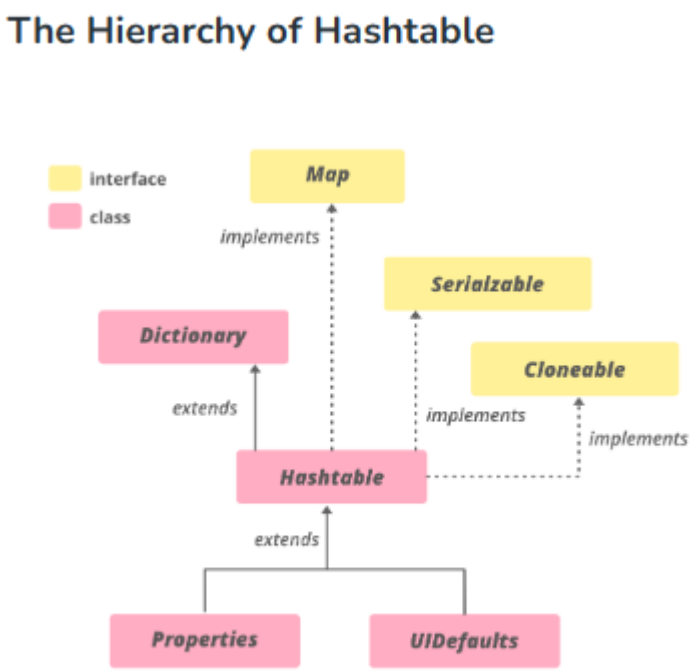
```
Java collection framework was introduced in java 1.2, the older version of java classes are called Legacy
classes.
```

In this context, legacy means "should not be used anymore in new code". But you can if you want. And as they are not deprecated yet, they will be there for now at least.

All the legacy classes are synchronized. The **java.util** package defines the following **legacy** classes:

1. HashTable
2. Stack
3. Dictionary
4. Properties
5. Vector

HashTable



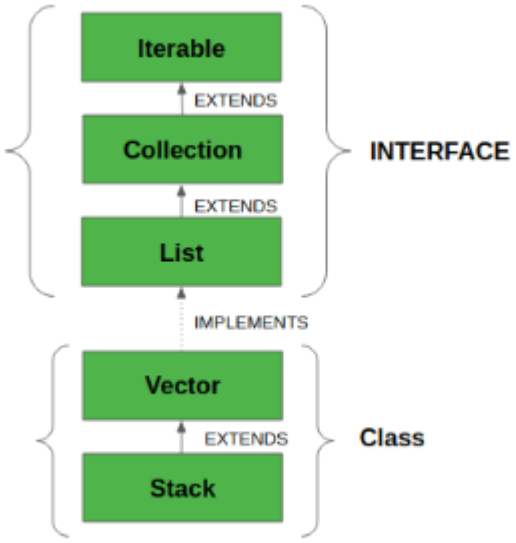
The Hashtable class is similar to HashMap. It also contains the data into key/value pairs. It doesn't allow to enter any null key and value because it is synchronized.

```
import java.util.Hashtable;
import java.util.Iterator;
import java.util.Map;

public class HashTableTest {

    public static void main(String[] args) {
        Map<Integer, String> map = new Hashtable<>();

        //to add values
        map.put(1, "one");
        map.put(1, "ONE");//override the value
        map.put(2, "two");

        //to check the key contains in map
        boolean hasKeyPresent = map.containsKey(1);
        boolean hasKeyPresent2 = map.containsKey(3);

        //remove entry from map
        map.remove(2);

        //using for loop
        for (Map.Entry<Integer, String> m : map.entrySet()) {
            System.out.println("key is: " + m.getKey());
            System.out.println("value is: " + m.getValue());
        }


        //using iterator
        Iterator<Map.Entry<Integer, String>> iterator = map.entrySet().iterator();
        while (iterator.hasNext()) {
            Map.Entry<Integer, String> next = iterator.next();
            System.out.println("key is: " + next.getKey());
            System.out.println("value is: " + next.getValue());
        }
    }
}
```

Stack

LIFO – last in first out data structure

Stack class extends Vector class, which follows the LIFO(LAST IN FIRST OUT) principal for its elements. The stack implementation has only one default constructor, i.e., Stack().




```
package com.hdfc.collections;

import java.util.Stack;

public class StackTest {
    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();
        System.out.println(stack.isEmpty());//true

        stack.push(1);//insert
        stack.push(2);//insert
        stack.push(3);//insert
        stack.push(4);//insert
        stack.push(5);//insert

        System.out.println(stack.isEmpty());//false
        System.out.println(stack.contains(5));//to check if present?
        System.out.println(stack.contains(6));//to check if present?

        System.out.println(stack.peek());//to check
        System.out.println(stack.peek());//to check

        //System.out.println(stack.pop());//5
        //System.out.println(stack.pop());
        //System.out.println(stack.pop());
        //System.out.println(stack.pop());
        //System.out.println(stack.pop());//1
        //System.out.println(stack.pop());//.EmptyStackException

    }
}
```

Vector

Vector is a special type of ArrayList that defines a dynamic array. ArrayList is not synchronized while **vector** is synchronized. The vector class has several legacy methods that are not present in the collection framework. Vector implements Iterable after the release of JDK 5 that defines the vector is fully compatible with collections, and vector elements can be iterated by the for-each loop

```
package com.hdfc.collections;

import java.util.Arrays;
import java.util.List;
import java.util.Vector;

public class VectorTest {
    public static void main(String[] args) {

        //same as ArrayList
        Vector<Integer> vector = new Vector<>();
        vector.add(1);
        vector.add(1);
        vector.add(1);
        vector.add(1);

        vector.clear(); //remvoe all entries from vector
        vector.contains(2);//false
        vector.forEach(e -> System.out.println(e)); //java-8


        // to convert Collection to Array
        Object[] objects = vector.toArray();

        //To convert Arrays to Collection
        Integer[] array = new Integer[]{1,2};
        List<Integer> integers = Arrays.asList(array);

    }
}
```

Properties

Properties class extends Hashtable class to maintain the list of values. The list has both the key and the value of type string. The **Property** class has the following two constructors:

```
package com.hdfc.collections;

import java.util.Properties;

public class PropertiesTest {
    public static void main(String[] args) {
        Properties p = new Properties();
        p.put("key","value");
        p.put("key1","value2");

        Properties  p2 = new Properties(p);
        p2.get("key");
        p2.isEmpty();
        p2.contains("key-2");
    }
}
```

Dictionary

The **Dictionary** class operates much like Map and represents the **key/value** storage repository. The **Dictionary class** is an abstract class that stores the data into the key/value pair. We can define the dictionary as a list of key/value pairs.

Dictionary is an abstract class, superclass of Hashtable. You should not use Dictionary as it is obsolete. As for Hashtable, the advantage it had over other maps such as HashMap was thread safety, but with the introduction of ConcurrentHashMap since Java 1.5, there is no real reason to use it any longer - see javadoc
As of the Java 2 platform v1.2, this class was retrofitted to implement the Map interface, making it a member of the Java Collections Framework. Unlike the new collection implementations, Hashtable is synchronized. If a thread-safe implementation is not needed, it is recommended to use HashMap in place of Hashtable. If a thread-safe highly-concurrent implementation is desired, then it is recommended to use ConcurrentHashMap in place of Hashtable.
In summary: Don't use Dictionary or Hashtable, unless you really have to for compatibility reasons, use either HashMap if you don't need thread safety, or ConcurrentHashMap if your map is used in a concurrent environment.

Dictionary is an abstract base class of Hashtable. Both are still in JDK for backwards compatibility with old code. We are expected to use HashMap and other implementations of Map interface introduced in Java 1.2.

```
package com.hdfc.collections;

import java.util.Dictionary;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.Map;

public class HashTableTest {

    public static void main(String[] args) {

        //Dictionary<Integer, String> map = new Dictionary<>();//abstract class cannot initialize

        Dictionary<Integer, String> map = new Hashtable<>();

        //to add values
        map.put(1, "one");
        map.put(1, "ONE");//override the value
        map.put(2, "two");


        //remove entry from map
        map.remove(2);
    }
}
```