Day 21

Dictionary<K,V>

- It is abstract class declared in java.util package.
- It is super class of Hashtable.
- It is used to store data in key/value pair format.
- It is not a part of collection framework
- It is introduced in jdk 1.0
- Methods:

1. public abstract boolean isEmpty()
2. public abstract V put(K key, V value)
3. public abstract int size()
4. public abstract V get(Object key)
5. public abstract V remove(Object key)
6. public abstract Enumeration keys()
7. public abstract Enumeration elements()

- Implementation of Dictionary is Obsolete.

Map<K,V>

- It is part of collection framework but it doesn't extend Collection interface.
- This interface takes the place of the Dictionary class, which was a totally abstract class rather than an interface.
- HashMap, Hashtable, TreeMap etc are Map collection's.
- Map collection stores data in key/value pair format.
- In map we can not insert duplicate keys but we can insert duplicate values.
- It is introduced in jdk 1.2
- Map.Entry<K,V> is nested interface of Map<K,V>.
- Following are abstract methods of Map.Entry interface.
    1. K getKey()
    2. V getValue()
    3. V setValue(V value)
- Abstract Methods of Map<K,V>

1. boolean isEmpty()
2. V put(K key, V value)
3. void putAll(Map<? extends K,? extends V> m)
4. int size()
5. boolean containsKey(Object key)
6. boolean containsValue(Object value)
7. V get(Object key)
8. V remove(Object key)
9. void clear()

10. Set keySet()
11. Collection values()
12. Set<Map.Entry<K,V>> entrySet()

- An instance, whose type implements Map.Entry<K,V> interface is called enrty instance.

```
class Pair<K,V> implements Entry<K, V>
{
    private K key;
    private V value;
    @Override
    public K getKey()
    {
        return this.key;
    }
    @Override
    public V getValue()
    {
        return this.value;
    }
    @Override
    public V setValue(V value)
    {
        this.value = value;
        return this.value;
    }
}
```

```
class Program
{
    public static void main(String[] args)
    {
        Entry<Integer, String> e = new Pair<>();
        //Here pair instance is entry instance
    }
}
```

- Map is collection of entries where each entry contains key/value pair.

Hashtable

- It is Map<K,V> collection which extends Dictionary class.
- It can not contain duplicate keys but it can contain duplicate values.
- In hashtable, Key and value can not be null.
- It is synchronized collection.
- It is introduced in jdk 1.0
- In Hashtable, if we want to use instance non final type as key then it should override equals and hashCode method.

HashMap<K,V>

- It is map collection
- It's implementation is based on Hashtable.
- It can not contain duplicate keys but it can contain duplicate values.
- In HashMap, key and value can be null.
- It is unsynchronized collection. Using Collections.synchronizedMap() method, we can make it synchronized.

```
Map m = Collections.synchronizedMap(new HashMap(...));
```

- It is introduced in jdk 1.2.
- Instantiation

```
Map<Integer, String> map = new HashMap<>( );
```

- Note : In HashMap, if we want to use element of non final type as a key then it should override equals() and hashCode() method.

LinkedHashMap<K,V>

- It is sub class of HashMap<K,V> class
- Its implementation is based on LinkedList and Hashtable.
- It is Map collection hence it can not contain duplicate keys but it can contain duplicate values.
- In LinkedHashMap, key and value can be null.
- It is unsynchronized collection. Using Collections.synchronizedMap() method we can make it synchronized.

```
Map m = Collections.synchronizedMap(new LinkedHashMap(...));
```

- LinkedHashMap maintains order of entries according to the key.
- Instantiation:

```
Map<Integer, String> map = new LinkedHashMap<>();
```

- It is introduced in jdk 1.4

TreeMap

- It is map collection.
- It can not contain duplicate keys but it can contain duplicate values.
- It TreeMap, key not be null but value can be null.

- Implementation of TreeMap is based on Red-Black Tree.
- It maintains entries in sorted form according to the key.
- It is unsynchronized collection. Using Collections.synchronizedSortedMap() method, we can make it synchronized.

```
SortedMap m = Collections.synchronizedSortedMap(new TreeMap(...));
```

- Instantiation:

```
Map<Integer, String> map = new TreeMap<>();
```

- It is introduced in jdk 1.2
- Note : In TreeMap, if we want to use element of non final type as a key then it should implement Comparable interface.

Types of Iterator

1. Fail Fast Iterator
2. Fail Safe Iterator(Not Fail Fast)

Fail Fast Iterator

- During traversing, using collection reference, if we try to modify state of collection and if iterator do not allows us to do the same then such iterator is called "Fail Fast" Iterator. In this case JVM throws ConcurrentModificationException.

```
Vector<Integer> v = new Vector<>();
v.add(10);
v.add(20);
v.add(30);
v.add(40);
v.add(50);

Integer element = null;
Iterator<Integer> itr = v.iterator();
while( itr.hasNext())
{
    element = itr.next();
    System.out.print(element+"  ");
    if( element == 50 )
        v.add(60); //ConcurrentModificationException
}
```

Fail safe Iterator

- During traversing, if itrator allows us to do changes in underlying collection then such iterator is called fail safe iterator.

```
Vector<Integer> v = new Vector<>();
v.add(10);
v.add(20);
v.add(30);
v.add(40);
v.add(50);

Integer element = null;
Enumeration<Integer> e = v.elements();
while( e.hasMoreElements())
{
    element = e.nextElement();
    System.out.print(element+" ");
    if( element == 50 )
        v.add(60); //OK
}
```

File IO

- File is container that is used to store record permanantly on HDD.
- Stream is an abstraction(object) that is used to produce(write) and consume(read) information from source to destination.
- If we want to do file handling we should use types declared in java.io package.
- Interfaces

1. Closeable
2. Flushable
3. FilenameFilter
4. DataInput
5. DataOutput
6. ObjectInput
7. ObjectOutput
8. Serializable

- Classes

1. Console
2. File
3. InputStream
4. OutputStream
5. FileInputStream
6. FileOutputStream
7. BufferedInputStream
8. BufferedOutputStream
9. DataInputStream

10. DataOutputStream
11. ObjectInputStream
12. ObjectOutputStream
13. PrintStream
14. Reader
15. Writer
16. FileReader
17. FileWriter
18. BufferedReader
19. BufferedWriter
20. InputStreamReader
21. OutputStreamWriter
22. PrintWriter

Binary File

- e.g .class, .obj, .mp3, .jpg etc.
- If we read binary file then we must use specific program.
- Binary require less processing hence it is faster than text file.
- It doesnt save data in human readable format.
- In java, InputStream, OutputStream and their sub classes are used to manipulate binary file.

Text File

- e.g .java, .txt, .rtf, .doc, .xml etc.
- We can read text file using any text editor.
- Text file require more processing hence it is slower than binary file
- It can save data in human readable format
- Reader, Writer and their sub classes are used to manipulate text file.

java.io.File

- It is a java class whose instance represent operating System file, directory or drive.
- Use:
    1. To create empty file / empty directory
    2. To read metadata of OS File, Directory and Drive.

implements Serializable

```
class Employee implements Serializable
{
    private String name;
    private int empid;
    private float salary;
    public Employee(String name, int empid, float salary) {
        super();
        this.name = name;
        this.empid = empid;
```

```
        this.salary = salary;
    }
    @Override
    public String toString() {
        return String.format("%d%s%f", empid,name,salary);
    }


}
try( ObjectOutputStream outputStream=new ObjectOutputStream(new
BufferedOutputStream(new FileOutputStream(new File(pathname))));)

try( ObjectInputStream inputStream=new ObjectInputStream(new
BufferedInputStream(new FileInputStream(new File(pathname))));)
```