Unit II

Java Programming Concepts

Reflection

- What is Reflection :
  - Reflection is a very useful approach to deal with the Java class at runtime, it can be use to load the Java class, call its methods or analysis the class at runtime.
  - Reflection is a API that contains different classes for above purpose.
  - Reflection API comes in java.lang package.

- What is use of run time class loading or class method calling?
  - For example there is web application in which there are two separate pages one for add student information and second is for add subject information.
  - Here when user submits student form then at server side dynamically student object should get created and if subject form get submitted then subject object should get created.
  - So which object should get created is decided at run time so we have to load class run time because only after loading class we can create it's object.

- What programmer can do using reflection
  - 1. We can load class at runtime.
  - 2. can create object at runtime.
  - 3. can get list of all public / private methods of class

    and also we can run thoes methods at runtime.
  - 4. can get list of all public / private variables and also we can set or get it's value at runtime.

- How to load class ?
  - Using forName() method of Class (Class is class in Java like Object class) class.
  - forName() method return Object of Class class.
  - E.g

```
class Student
{
        // Class Methods and Variables
}

class Test
{
    public static void main(String args[])
    {
      Class c=Class.forName("Simple");
      System.out.println(c.getName());
    }
}
```

Output: Student

- How to create instance/object and call public method?
  - The newInstance() method of Class class and Constructor class is used to create a new instance of the class.
  - E.g

```
class Student
{
      Public void message(){ s.o.p("Hello I am Student....")}
}

class Test
{
    public static void main(String args[])
    {
      Class c=Class.forName("Simple");
      Student s=(Student)c.newInstance(); // instance created
      s.message();  // method called
    }
}
```

Output: Hello I am Student....

- How to call private method?
  - By the help of java.lang.Class class and java.lang.reflect.Method class, we can call private method from any other class.
  - getDeclaredMethod(String methodname,Class[] parameterTypes) function of Class class returns Method object. On Method of call setAccessible(true) function and then call invoke() function that call method.
  - E.g

```
class Student
{
      private void message(){ s.o.p("Hello I am Private Student....")}
}
class Test
{
    public static void main(String args[])
    {
      Class c = Class.forName("Student");
      Object o= c.newInstance();
      Method m =c.getDeclaredMethod("message", null);
      m.setAccessible(true);
      m.invoke(o, null);
    }
}
```

Output: Hello I am Private Student....

Boxing and Unboxing

- In Java for each primative class there is a wrapper class.
- What is use of wrapper class?
  - For example for int there is an Integer wrapper class.
  - Oftenly we need to do different processing on int like conversion of int to float,binary,hexdecimal or string.
  - Integer wrapper class provides all this functionaliy in form of in built function. We just have to use it
  - E.g int i =3;

```
Integer i = new Integer(3);
float f = i.floatValue()     // returns float value of intereg.

String s = Integer.toBinaryString(3); // return binary representation of 3 in string format.
```

- What is boxing ?
  - Boxing means convert primative variable in corresponding wrapper class.
  - And this is conversion done by JVM itself.
  - e.g

```
 class Test
{
    Public void msg(Integer n)   // Here automatically boxing done.
     {    s.o.p.(“Value is”+n)
     }

    public static void main(String args[])
    {
        Test t =new Test();
        Int i =10;
        t.msg(i);   // here we are passing int as a parameter. In method
                    // defination we have mentioned Integer arrgument.
    }
}
```

Output: Value is 10

- What is unboxing ?
  - Boxing means convert wrapper object variable in corresponding primative variable.
  - And this is conversion done by JVM itself.
  - e.g

```
 class Test
{
    Public void msg(int n)   // Here automatically unboxing done.
     {    s.o.p.("Value is"+n)
     }

    public static void main(String args[])
    {
        Test t =new Test();
        Integer i =10;
        t.msg(i);   // here we are passing Integer object as a parameter.
                    //In method defination we have mentioned int
                    //arrgument
    }
}
```

Output: Value is 10

serialization and deserialization

- What is serialization ?
  - Serialization is a process of converting an object into a sequence of bytes which can be persisted (stored) to a disk or database or can be sent through streams.
  - It means we can store current state of object.
  - Current state of object means it's instance variables values.
  - Java provides Serializable API encapsulated under java.io package for serializing and deserializing objects which include,

    java.io.serializable

    java.io.Externalizable

    ObjectInputStream

    and ObjectOutputStream etc.
  - To make object serializable class must implements Serializable interface

serialization and deserialization

- What is deserialization ?
  - The reverse process of creating object from sequence of bytes is called deserialization.
  - It means we can regain object from its stored state.
  - Regained/Restored object we can use further in program like any other object.
  - Example on nex page.

```
import java.io.*;
class studentinfo implements Serializable
{
      String name;
      int rid;
      static String contact;
      studentinfo(string n, int r, string c)
      {
          this.name = n;
          this.rid = r;
          this.contact = c;
      }
}

class Test
{
   public static void main(String[] args)
   {
     try
     {
        Studentinfo si = new studentinfo("Abhi", 104, "110044");
        FileOutputStream fos = new FileOutputStream("student.ser"); // file in which object get
                                                                    // stored
        Objectoutputstream oos = new ObjectOutputStream(fos);
        oos.writeObject(si);
        oos.close();
        fos.close();
     }catch (Exception e){ e. printStackTrace(); }
    }
}
```

serialization and deserialization

- What is deserialization ?
  - The reverse process of creating object from sequence of bytes is called deserialization.
  - It means we can regain object from its stored state.
  - Regained/Restored object we can use further in program like any other object.
  - Example on nex page.

  - In Serialization transient varibale can not serialized, means it's status can not be saved or stored.

```
import java.io * ;
class DeserializationTest
{
 public static void main(String[] args)
 {
  studentinfo si=null ;
  try
  {
   FileInputStream fis = new FileInputStream("student.ser");
   ObjectOutputStream ois = new ObjectOutputStream(fis);
   si = (studentinfo)ois.readObject();
  }
  catch (Exception e)
   { e.printStackTrace(); }
  System.out.println(si.name);
  System.out. println(si.rid);
  System.out.println(si.contact);
 }
}
```

Output:

```
Abhi
104
null
```

Java Utility classes : StringTokenizer, Observable

- In Java there are some classes which provides extra functionality for processing on data like strings etc.. This classes are called utility classes
- Utility classes ecapsulated in java.util package
- What is StringTokenizer ?
  - It is a utility class provided in Java.
  - The string tokenizer class allows an application to break a string into tokens.
  - Following are the ways to create object of StringTokenizer class

Constructor	Use
StringTokenizer(String str)	creates StringTokenizer with specified string which we want to be tokenized.
StringTokenizer(String str, String delim)	creates StringTokenizer with specified string and delimeter.

Methods	Use
boolean hasMoreTokens()	checks if there is more tokens available.
String nextToken()	returns the next token from the StringTokenizer object.

```
import java.util.StringTokenizer;

 public class Simple{
  public static void main(String args[]){
    StringTokenizer st = new StringTokenizer("my name is khan"," ");
      while (st.hasMoreTokens()) {
          System.out.println(st.nextToken());
      }
    }
 }
```

Output :

- What is Observable utility class?
  - If we want to have a class which is monitored by other classes in the our application then we say that we want the class to be observable. I.e. there might be some changes in its state which we would want to broadcast to the rest of the program.
  - Means changes in state of object can be monitoe by other classes or program.

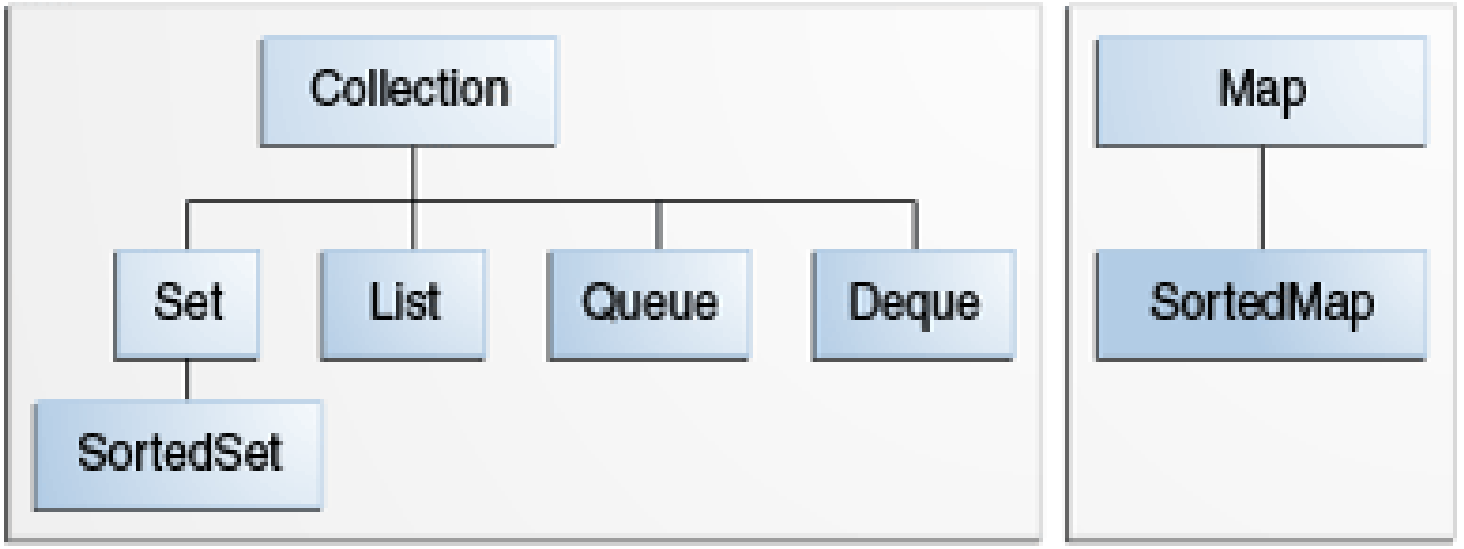
    There are

    1. Observable class
    2. Observer interface
  - The class who want to be observed must extend Observable class
  - And class who want to observe must implement Observer interface

Java Collection Framework

- A collection — sometimes called a container — is simply an object that groups multiple elements into a single unit. Collections are used to store, retrieve, manipulate, and aggregate data.
- What Is a Collections Framework?
  - A collections framework is a unified architecture for representing and manipulating collections. All collections frameworks contain the following:
    - Interfaces: Interfaces allow collections to be manipulated independently of the details of their representation.
    - Implementations(class): These are the concrete implementations (class) of the collection interfaces. In essence, they are reusable data structures.
    - Algorithms(functions): These are the methods that perform useful computations, such as searching and sorting,

- Benefits of the Java Collections Framework
  - Reduces programming effort
  - Increases program speed and quality
  - Reduces effort to learn and to use new APIs
  - Fosters software reuse
- Core Interfaces in Collection Framework



- The List Collection
  - A List is an ordered Collection (sometimes called a sequence). Lists may contain duplicate elements. In addition to the operations inherited from Collection, the List interface includes operations for the following:
    - Positional access — manipulates elements based on their numerical position in the list. This includes methods such as get, set, add, addAll, and remove.
    - Search — searches for a specified object in the list and returns its numerical position. Search methods include indexOf and lastIndexOf.
    - Iteration — extends Iterator semantics to take advantage of the list's sequential nature. The listIterator methods provide this behavior.

- List Collection Types or Concreate Classes
  - ArrayList
  - Vector (Thread Safe)
  - LinkedList (Uses two pointer to traverse in both direction)
  - Stack
- How to create collection?
  - ArrayList a= new ArrayList();
  - Vector v = new Vector();
  - LinkedList l = new LinkedList();
- Methods
  - Insert : a.add(obj);   // obj is object want to added
  - Get : a.get(i);   // i is index
  - Rmove : a.remove(obj); // obj is want to be removed;

- The Set Collection
  - A Set is a Collection that cannot contain duplicate elements. Set is not a ordered collection.
- Set Collection Types or Concreate Classes
  - HashSet
  - TreeSet (Uses binary tree to stored. Elements are in sorted format)
  - LinkedHashSet (Uses two pointer to traverse in both direction)
- How to create collection?
  - HashSet h= new HashSet();
  - TreeSet t = new TreeSet();
  - LinkedHashSet l = new LinkedHashSet();
- Methods
  - Insert : a.add(obj);   // obj is object want to added
  - Get : To get objects we have to use iterator.
  - Rmove : a.remove(obj); // obj is want to be removed;

- The Map Collection
  - A Map is an object that maps keys to values. A map cannot contain duplicate keys.
  - Each key can map to at most one value.
  - Map can contain duplicate objects but object must have unique key.
- Map Collection Types or Concreate Classes
  - HashMap
  - TreeMap (Uses binary tree to stored. Elements are in sorted format)
  - LinkedHashMap (Uses two pointer to traverse in both direction)
- How to create collection?
  - HashMap h= new HashMap();
  - TreeMap t = new TreeMap();
  - LinkedHashMap l = new LinkedHashMap();
- Methods
  - Insert : h.put(key,obj);   // key – value pair
  - Get : To get objects we have to use iterator. There are i.getKey() and i.getValue() methods where i is iterator refference.
  - Rmove : h.remove(obj); // obj is want to be removed;

- The HashTable Collection
  - A Hashtable is an array of list.Each list is known as a bucket.The position of bucket is identified by calling the hashcode() method.A Hashtable contains values based on the key. It implements the Map interface and extends Dictionary class.
  - It contains only unique elements.
  - It may have not have any null key or value.
  - It is synchronized (Thread safe)
  - It is a map.

HashTable example

```
Hashtable<Integer,String> hm=new  Hashtable<Integer,String>();
// Here Integer & String in angle braket is Key and Value types.
 hm.put(100,"Amit");   // Here 100 is key and Amit is value
 hm.put(102,"Ravi");
 hm.put(101,"Vijay");
 hm.put(103,"Rahul");
 HashSet hs = hm.entrySet();
 iterator i = hs.iterator();
 while(i.hastNext())
 {
     Map m = (Map) i.next();
    System.out.println(m.getKey()+" "+m.getValue());
 }
}
```

- The Dictionary Collection (Same like Map but functions names are different)
  - Dictionary is an abstract class that represents a key/value storage repository and operates much like Map.
  - Given a key and value, you can store the value in a Dictionary object. Once the value is stored, you can retrieve it by using its key. Thus, like a map, a dictionary can be thought of as a list of key/value pairs.
  - Enumeration elements( ): Returns an enumeration of the values contained in the dictionary.
  - Object get(Object key) : Returns the object that contains the value associated with key. If key is not in the dictionary, a null object is returned.
  - boolean isEmpty( ) :Returns true if the dictionary is empty, and returns false if it contains at least one key.
  - Enumeration keys( ) :Returns an enumeration of the keys contained in the dictionary.
  - Object put(Object key, Object value) :Inserts a key and its value into the dictionary. Returns null if key is not already in the dictionary; returns the previous value associated with key if key is already in the dictionary.
  - Object remove(Object key):Removes key and its value. Returns the value associated with key. If key is not in the dictionary, a null is returned.
  - int size( ):Returns the number of entries in the dictionary.

- The Stack Collection (List Type)
  - Stack is a subclass of Vector that implements a standard last-in, first-out stack.
  - Stack only defines the default constructor, which creates an empty stack. Stack includes all the methods defined by Vector, and adds several of its own.
  - Apart from the methods inherited from its parent class Vector, Stack defines following methods:
    - **boolean empty()** : Tests if this stack is empty. Returns true if the stack is empty, and returns false if the stack contains elements.
    - **Object peek( )** : Returns the element on the top of the stack, but does not remove it.
    - **Object pop( )** : Returns the element on the top of the stack, removing it in the process.
    - **Object push(Object element) :**Pushes element onto the stack. element is also returned.
    - **int search(Object element)** : Searches for element in the stack. If found, its offset from the top of the stack is returned. Otherwise, .1 is returned.

- Followings are the common methods in all collection
  - **Int size()** : returns number of elements present in collection
  - **boolean isEmpty()** : Returns true if this list contains no elements.
  - **boolean contains(Object o)** : Returns true if this list contains the specified element. (Not in Map, HashTable , Dictionary)
    - In Map there are
      - containsKey() : return true if key is present in map
      - containsValue() : return true if value is present in map
  - **Object[] toArray()** : Returns an array containing all of the elements in this list in proper sequence (from first to last element). (Not in Map, HashTable, Dictionary)
  - **Clear()** : deletes all elements in collection

- Iterator
  - Iterator is in collection framework
  - Iterator enables you to cycle through a collection, obtaining or removing elements.
  - In general, to use an iterator to cycle through the contents of a collection, follow these steps:
    - Obtain an iterator to the start of the collection by calling the collection's iterator( ) method.
    - Set up a loop that makes a call to hasNext( ). Have the loop iterate as long as hasNext( ) returns true.
    - Within the loop, obtain each element by calling next( ).

- The Methods Declared by Iterator
  - boolean hasNext( ) : Returns true if there are more elements. Otherwise, returns false.
  - Object next( ) : Returns the next element. Throws NoSuchElementException if there is not a next element.
  - void remove( ) : Removes the current element. Throws IllegalStateException if an attempt is made to call remove( ) that is not preceded by a call to next( ).