Today topics of discussion
---
1. Summary of Collection
2. Map and its internal working
3. Map and its implementation classes
4. Comparable and Comparator Interface

Map
===
=> It is not a child interface of Collection.
=> If we want to represent group of Objects as key-value pair then we need to go for Map.
=> Both keys and values are Objects only
=> Duplicate keys are not allowed but values are allowed.
=> Key-value pair is called as "Entry".

refer: MapHierarchy.png

Map methods
---
1. It contains 12 methods which is common for all the implementation Map Objects
a. Object put(Object key,Object value) // To add key,value pair
b. void putAll(Map m) // To add another map
c. Object get(Object key) // To get the value based on key
d. Object remove(Object key) //To remove an entry based on key
e. boolean containsKey(Object key) //Check whether it contains key or not
f. boolean containsValue(Object value)//Check whether it contains value or not
g. boolean isEmpty() //To check wheter the Map is empty or not
h. int size() //To get the size of a Map
i. void clear() //To remove all Entry from a map

views of a Map
j.Set keySet() //Convert the key's of Map into Set for reading purpose
k.Collection values() //Convert the values of Map into Collection for reading purpose
l.Set entrySet() // Convert whole Entry of Map into Set for reading purpose.

Entry(I)
---
1. Each key-value pair is called Entry.
2. Without existence of Map,there can't be existence of Entry Object.
3. Interface entry is defined inside Map interface.

```
interface Map{
      interface Entry{
            Object getKey();  //To get the key using Map.Entry Object
            Object getValue();//To get the value using Map.Entry Object
            Object setValue(Object newValue);//To update the value Using Map.Entry Object
      }
}
```

HashMap

Underlying DataStructure: Hashtable
insertion order : not preserved
duplicate keys : not allowed

```
 duplicate values        : allowed
 Heterogenous objects    : allowed
 null insertion          : for keys allowed only once,but for values can be any no.
 implementation interface: Serializable,Cloneable.

Difference b/w HashMap(c) and Hashtable(c)
==========================================
 HashMap  => All the methods are not synchronized.
 Hashtable => All the methods are synchronzied.

 HashMap   => At a time multiple threads can operate on a Object, so it is not
ThreadSafe.
 Hashtable  => At a time only one Thread can operate on a Object, so it is
ThreadSafe.

 HashMap   => Pefromance is high.
 Hashtable  => Performance is low.

 HashMap   => null is allowed for both keys and values.
 Hashtable  => null is not allowed for both keys and values,it would result in
NullPointerException.

 HashMap   => Introduced in 1.2v
 Hashtable => Introduced in 1.0v

Constructors
============
 1. HashMap hm=new HashMap()
            //default capacity => 16, loadfactor => 0.75(upon increase of data by
75% automatically

       size of HashMap will be doubled)

 2. Hashmap hm=new HashMap(int capacity);

 3. HashMap hm=new HashMap(int capacity,float fillration);

 4. HashMap hm=new HashMap(Map m);

eg#1.
eg#1.
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            HashMap hm = new HashMap();
            hm.put(10,"sachin");
            hm.put(7,"dhoni");
            hm.put(18,"kohli");
            hm.put(45,"rohith");
            System.out.println(hm);//hm.toString() will be called

            Set s = hm.keySet();//To get the keys from Map
            System.out.println(s);
            System.out.println(s.getClass().getName());

            System.out.println();
```

```
            Collection c = hm.values();//To get the values from Map
            System.out.println(c);
            System.out.println(c.getClass().getName());

            System.out.println();

            Set mapData = hm.entrySet();//To get the K,V from Map as Set
            System.out.println(mapData);
            System.out.println(mapData.getClass().getName());

            System.out.println();
            Iterator itr = mapData.iterator();
            while(itr.hasNext()){
                  //Object is return type of next(), i am converting to Map.Entry
Object to call methods of Entry interface
                  // The data of Map.Entry is stored as object
                  Map.Entry data =(Map.Entry)itr.next();
                  System.out.println(data.getClass().getName());
                  System.out.println(data.getKey() + ": " + data.getValue());
                  if (data.getKey().equals(10))
                  {
                        data.setValue("SRT");
                  }
            }
            System.out.println();
            System.out.println(hm);
      }
}
```

```
LinkedHashMap
=============
 => It is the child class of HashMap.
 => It is same as HashMap,but with the following difference

       HashMap             => underlying datastructure is hashtable.
       LinkedHashMap => underlying datastructure is LinkedList + hashtable.

       HashMap             => insertion order not preserved.
       LinkedHashMap => insertion order preserved.

       HashMap             => introduced in 1.2v
       LinkedHashMap => introduced in 1.4v


eg#1.
import java.util.*;

class Test
{
      public static void main(String[] args)
      {
            LinkedHashMap hm = new LinkedHashMap();
            hm.put(10,"sachin");
            hm.put(7,"dhoni");
            hm.put(18,"kohli");
            hm.put(45,"rohith");
            System.out.println(hm);//hm.toString() will be called

            System.out.println();
```

```
            Set s = hm.keySet();//To get the keys from Map
            System.out.println(s);
            System.out.println(s.getClass().getName());

            System.out.println();

            Collection c = hm.values();//To get the values from Map
            System.out.println(c);
            System.out.println(c.getClass().getName());

            System.out.println();

            Set mapData = hm.entrySet();//To get the K,V from Map as Set
            System.out.println(mapData);
            System.out.println(mapData.getClass().getName());

            System.out.println();
            Iterator itr = mapData.iterator();
            while(itr.hasNext()){
                  Map.Entry data =(Map.Entry)itr.next();
                  System.out.println(data.getKey() + ": " + data.getValue());
                  if (data.getKey().equals(10))
                  {
                        data.setValue("SRT");
                  }
            }
            System.out.println();
            System.out.println(hm);
      }
}



import java.util.*;

class Test
{
      public static void main(String[] args)
      {
            HashMap h = new HashMap();

            //Creating a key
            Integer i1= new Integer(10);
            Integer i2= new Integer(10);

            //Adding the data to HashMap
            h.put(i1,"sachin");
            h.put(i2,"Messi");

            System.out.println(h);//{10=Messi}

      }
}
```

IdentityHashMap
==========

It is same as HashMap, with the following differences

a. In case of HashMap,jvm will use equals() to check whether the keys are

duplicated or not.
      equals() => meant for ContentComparison.
b. In case of IdentityHashMap,jvm wil use == operator to identify whether the keys are duplicated.
    or not.
                        refer: HashMap vs IdentityHashMap .png

Note:
Garbage collector actions

```
import java.util.*;
class Test
{
      public static void main(String[] args) throws Exception
      {
            Employee e = new Employee();
                  ;;;;;;;;
                  ;;;;;;;;
                  ;;;;;;;;

            e = null;//Garbage object
            System.gc();//Informing JVM to active GC thread to clean garbage object
            Thread.sleep(5000);
      }
}
class Employee
{
      @Override
      public void finalize(){
            System.out.println("Cleaning the object");
      }
}
```

WeakHashMap
==========
 It is exactly same as HashMap, with the following differences.
1. HashMap will always dominate Garbage Collector, that is if the Object is a part of HashMap
   and if the Object is Garbage Object, still Garbage Collector won't remove that Object from
   heap since it is a part of HashMap. HashMap dominates GarbageCollector.
2. Garbage Collector will dominate WeakHashMap, that is if the Object is part of WeakHashMap and
   if that Object is Garbage Object, then immediately Garbage Collector will remove that Object
   from heap even though it is a part of WeakHashMap, so we say Garbage Collector dominates
   "WeakHashMap".

eg#1.
```
import java.util.*;

class Test
{
      public static void main(String[] args) throws Exception
      {
            HashMap hm = new HashMap();
            Temp t= new Temp();
            hm.put(t,"shri");
```

```
            System.out.println(hm);//{temp=shri}

            t= null;//Making eligible for Garbage Collection
            System.gc();//Triggering garbage collector thread to clean 't'
            Thread.sleep(5000);

            System.out.println(hm);//{temp=shri}

      }
}

class Temp
{
      @Override
      public String toString(){
                  return "temp";
      }

      @Override
      public void finalize(){
            System.out.println("cleaning temp object");
      }
}
eg#2.
import java.util.*;

class Test
{
      public static void main(String[] args) throws Exception
      {
            WeakHashMap hm = new WeakHashMap();
            Temp t= new Temp();
            hm.put(t,"shri");
            System.out.println(hm);//{temp=shri}

            t= null;//Making eligible for Garbage Collection
            System.gc();//Triggering garbage collector thread to clean 't'
            Thread.sleep(5000);

            System.out.println(hm);//{}

      }
}
class Temp
{
      @Override
      public String toString(){
                  return "temp";
      }

      @Override
      public void finalize(){
            System.out.println("cleaning temp object");
      }
}
```

Hashtable:
=> The Underlying Data Structure for Hashtable is Hashtable Only.
=> Duplicate Keys are Not Allowed. But Values can be Duplicated.

=> Insertion Order is Not Preserved and it is Based on Hashcode of the Keys.
=> Heterogeneous Objects are Allowed for Both Keys and Values.
=> null Insertion is Not Possible for Both Key and Values. Otherwise we will get Runtime
Exception Saying NullPointerException.
=> It implements Serializable and Cloneable,but not RandomAccess.
=> Every Method Present in Hashtable is Synchronized and Hence Hashtable Object is Thread
Safe.

Constructors:
1) Hashtable h = new Hashtable();
Creates an Empty Hashtable Object with Default Initial Capacity 11 and
Default Fill Ratio 0.75.
2) Hashtable h = new Hashtable(intinitialcapacity);
3) Hashtable h = new Hashtable(intinitialcapacity, float fillRatio);
4) Hashtable h = new Hashtable(Map m);

eg#1.

```
import java.util.*;

class Test
{
	public static void main(String[] args)
	{
		Hashtable hm = new Hashtable();//Default capacity is 11
		hm.put(new Temp(5),"A");
		hm.put(new Temp(2),"B");
		hm.put(new Temp(6),"C");
		hm.put(new Temp(15),"D");
		hm.put(new Temp(23),"E");
		hm.put(new Temp(16),"f");

		System.out.println(hm);
	}
}
class Temp
{
	int i;
	Temp(int i){
		this.i=i;
	}
	public int hashCode(){
		return i;
	}

	public String toString(){
		return i+" " ;
	}
}
```

Note;

```
public class Object{
	public native int hashCode();//Code is not from java language it will binded during runtime

	@Override
	public String toString(){
			return getClass().getName()+ "@" +
```

```
Integer.toHexString(hashCode());
	}
}

eg#1.
class Test{
	@Override
	public int hashcode(){
		return 10;
	}
}
Test t1= new Test();//Test@A
Test t2= new Test();//Test@A

class Student{
	int rollNo;
	Student(int rollNo){
		this.rollNo = rollNo;
	}
	@Override
	public int hashCode(){
		return rollNo;
	}
}
Student std1= new Student(10);//Student@A
Student std2= new Student(100);//Student@64

hashCode() method :
1. For every object jvm will generate a unique number which is nothing but
hashCode.
2. Jvm will using hashCode while saving objects into hashing related data
structures like   HashSet, HashMap, and Hashtable etc.
3. If the objects are stored according to hashCode searching will become very
efficient
   (The most powerful search algorithm is hashing which will work based on
hashCode).
4. If we didn't override hashCode() method then Object class hashCode() method will
be executed   which generates hashCode based on address of the object but it
doesn't mean hashCode represents    address of the object.
5. Based on our programming requirement we can override hashCode() method to
generate our own   hashcode.
6. Overriding hashCode() method is said to be proper if and only if for every
object we have to generate a unique number as hashcode for every object.

 public native int hashCode()
	=> It generates the hashCode based on the address of the Object.

 public String toString(){
	return getClass().getName() + "@" + Integer.toHexString(hashCode());
 }
 here getClass().getName() =>
classname@hexa_decimal_String_representation_of_hashCode

Example1:
class Student {
	public int hashCode() {
		return 100;
	}
}
```

It is improper way of overriding hashCode() method because for every object we are generating same
hashcode.

Example2:

```
class Student {
      int rollno;
      public int hashCode() {
            return rollno;
      }
}
```

It is proper way of overriding hashcode() method because for every object we are generating a different hashcode.

toString() method vs hashCode() method

==========

eg#1.

```
class Test{
      int i;
      Test(int i){
            this.i=i;
      }
      public static void main(String[] args){
            Test t1=new Test(10);
            Test t2=new Test(100);
            System.out.println(t1);//Test@....
            System.out.println(t2);//Test@....
      }
}
```

Object==>toString() called.
Object==>hashCode() called.
In this caseObject class toString( )method got executed which is internally calls Object class hashCode( ) method.

eg#2.

```
class Test{
      int i;
      Test(int i){
            this.i=i;
      }
      public int hashCode(){
            return i;
      }
      public static void main(String[] args){
            Test t1=new Test(10);
            Test t2=new Test(100);
            System.out.println(t1);//Test@A
            System.out.println(t2);//Test@64
      }
}
```

Object==>toString() called.
 Test ==>hashCode() called.
In this case Object class toString( ) method got executed which is internally calls Test
class hashCode( ) method.

eg#3.

```
class Test{
      int i;
      Test(int i){
            this.i=i;
      }
      public int hashCode(){
            return i;
      }
      public String toString(){
            return i+"";
      }
      public static void main(String[] args){
            Test t1=new Test(10);
            Test t2=new Test(100);
            System.out.println(t1);//10
            System.out.println(t2);//100
      }
  }
```

Output:
10
100
In this case Test class toString() method got executed and hashCode() wont be executed.

Note :
1. if we are giving opportunity to Object class toString() method it internally calls hashCode() method. But if we are overriding toString() method it may not call hashCode() method.
2. We can use toString() method while printing object references and we can use hashCode() method while saving objects into HashSet or Hashtable or HashMap

Properties:
=> It is the Child Class of Hashtable.
=> In Our Program if anything which Changes Frequently (Like Database User Name, Password, Database URLs Etc)
     Never Recommended to Hard Code in Java Program.
=> Because for Every Change in Source File we have to Recompile, Rebuild and Redeploying
     Application and Sometimes Server Restart Also Required, which Creates Business Impact to the Client.
=> To Overcome this Problem we have to Configure Such Type of Propertiesin Properties File.
=> The Main Advantage in this Approach is if a there is a Change in Properties File, to Reflect that Change Just Redeployment is Enough,
      which won't Create any Business Impact.
=> We can Use Properties Object to Hold Properties which are coming from Properties File.

Constructor:
      Properties p = new Properties();

1) public String getProperty(String pname);
      To Get the Value associated with specified Property.
2) public String setProperty(String pname, String pvalue);
      To Set a New Property.
3) public Enumeration propertyNames();It Returns All Property Names.
4) public void load(InputStream is);

```
      To Load Properties from Properties File into Java Properties Object.
5) public void store(OutputStreamos, String comment);
      To Store Properties from Java Properties Object into Properties File


eg#1
import java.util.*;
import java.io.*;

class Test
{
      public static void main(String[] args)throws Exception
      {
            Properties  p = new Properties();//properties object is created

            //Creating  a FileInputStream to read the data from a file called
"database.properties"
            FileInputStream fis = new FileInputStream("database.properties");

            //Data loaded into properties object throug fis
            p.load(fis);

            System.out.println(p);
            System.out.println();

            System.out.println("URL IS      :: "+p.getProperty("url"));
            System.out.println("USERNAME IS :: "+p.getProperty("username"));
            System.out.println("PASSWORD IS :: "+p.getProperty("password"));

            p.setProperty("iNeuron","NavinReddy");//Setting a new property

            FileOutputStream fos = new FileOutputStream("database.properties");
            p.store(fos,"MAP operation got concluded");//Added like a comment

      }
}
abc.properties
------------------
#MAP operation got concluded
#Sun Oct 16 12:47:16 IST 2022
password=root123
url=jdbc\:mysql\:///abc
iNeuron=NavinReddy
username=root
```