Collections API

P116

1) Collection API is set of java classes and Interfaces which is used to store and manage heterogeneous data.

2) In a collection each element is of Object type.

3) Every element is stored as an Object form

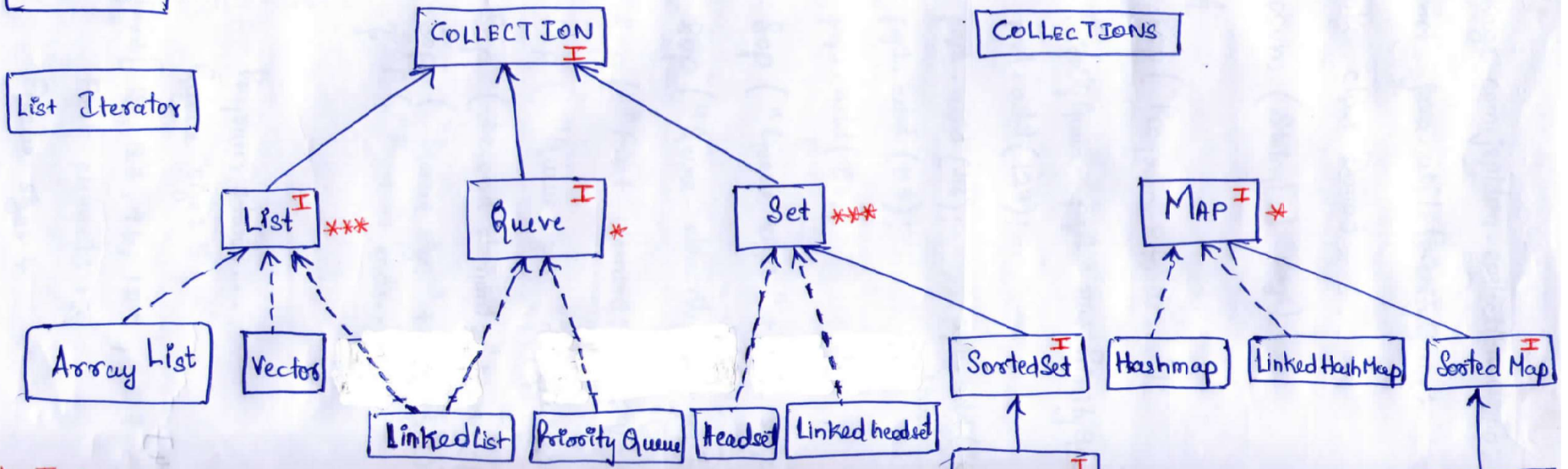
```
1) package com.jspiders.collections.demo;
   import java.util.ArrayList;
   public class Demo1
   {
     psvm (String[] args)
     {
       Sop ("Program starts...");
       ArrayList list1 = new ArrayList();
       list1.add("test"); //upcast
       list1.add (123); //auto boxing and auto upcast
       list1.add (12.34); // auto boxing and auto upcast
       list1.add (123);
       list1.add (null);
       Sop ("List size: " + list1.size());
       Sop (list1);
       Sop (list1.get(2));
       Sop ("List elements are");
       for (int i=0 ; i<list1.size() ; i++)
       {
         Sop(i + " element is " + list1.get(i));
       }
       Sop ("----------");
       Sop ("List elements using for-each");
       for (Object obj : list1)
       {
         Sop ("element is "+obj);
       }
       Sop ("Program ends...");
     }
   }
```

O/P :-

```
Program starts...
List size: 5
[test, 123, 12.34, 123, null]
12.34
List elements are
0 elements is test
1 elements is 123
2 element is 12.34
3 element is 123
4 element is null
----------
List elements using for-each
----------
Program ends...
```

Iterator *****

List Iterator



List - Type

* List Is A type of Collection
* List element are stored in index based.
* List is auto Indexed.
* List allows duplicate elements.
* List allows null value
* List element are referred with index. ex: Question paper

Queue - Type

* Queue Is - A type of Collection.
* Queue elements are stored without index.
* Queue implements FIFO
* Queue allows duplicate elements
* Queue doesn't allow null value.

Ex:- Ticket Queue.

Set

* Set Is-A type of Collection.
* Set elements are stored without index.
* Set doesn't allow duplicate elements.
* Set allows null.

Ex:

Map

* Map elements are stored in key - value pair.
* key is Object type & unique.
* Value is object type, duplicate allowed
* Value is referred using key.

Ex: locker and key

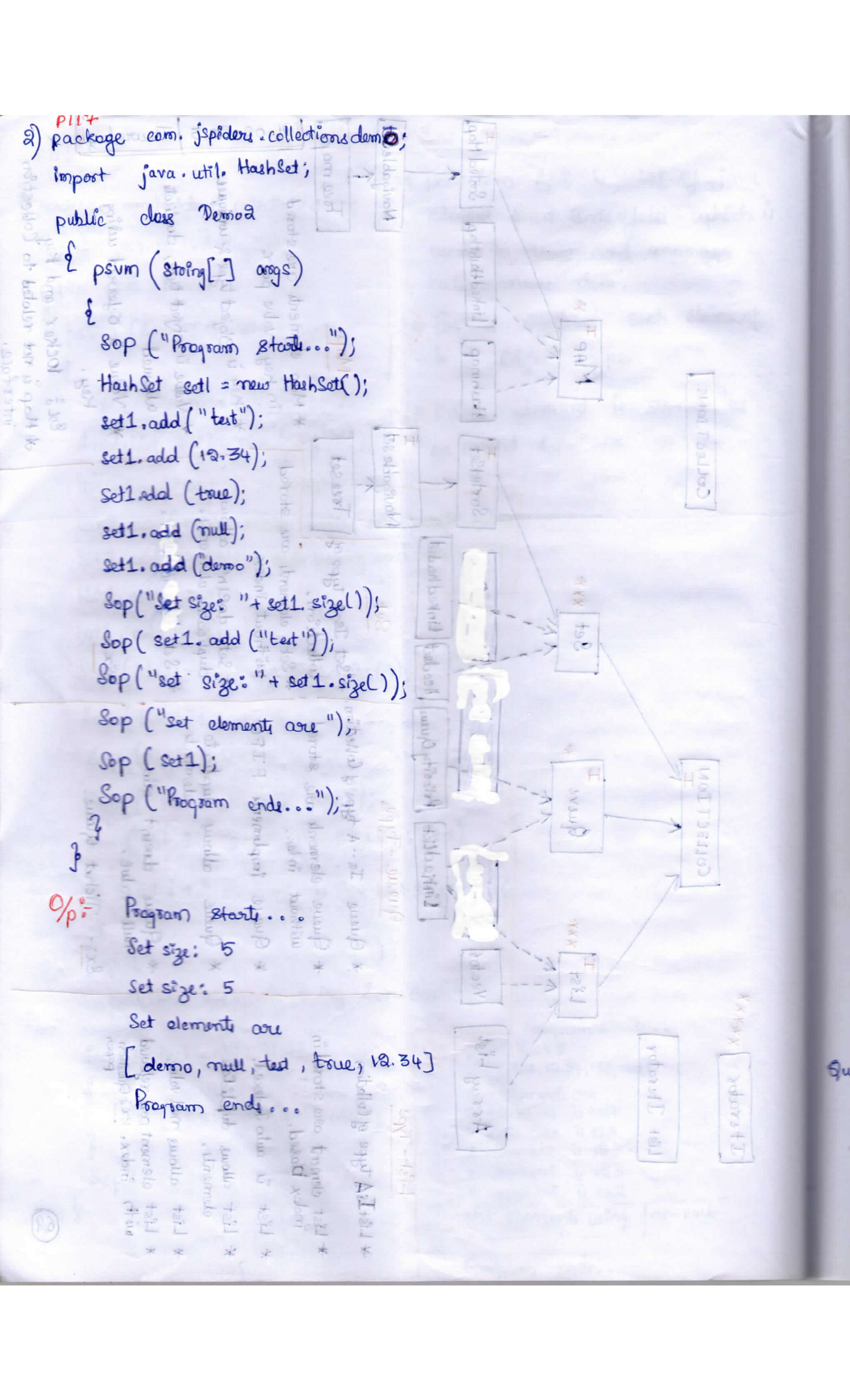
* Map is not related to Collection interface.

21-02-2013 Thursday

P117

2)
```
package com.jspiders.collectionsdemo;
import java.util.HashSet;
public class Demo2
{
  psvm (String[] args)
  {
    Sop ("Program starts...");
    HashSet set1 = new HashSet();
    set1.add("test");
    set1.add (12.34);
    Set1.add (true);
    set1.add (null);
    set1.add ("demo");
    Sop("Set size: " + set1.size());
    Sop( set1.add ("test"));
    Sop("set size:" + set1.size());
    Sop ("set elements are ");
    Sop (set1);
    Sop ("Program ends...");
  }
}
```

O/p:-
```
Program starts...
Set size: 5
Set size: 5
Set elements are
[demo, null, test, true, 12.34]
Program ends...
```



Queue

3)
```
package com.jspiders.collectionsdemo;

import java.util.PriorityQueue;

public class Demo3
{ psvm (String[] args)
  {
    Sop("Program starts...");

    PriorityQueue pq1 = new PriorityQueue();

    pq1.add(123);

    pq1.add(46);   /* pq1.add("test"); Because in priority queue second element
                      should be same type as head element */
    pq1.add(23);

    pq1.add(87);

    Sop("Queue size: " + pq1.size());

    Sop("Queue elements: " + pq1);

    Sop("First element: " + pq1.poll());

    Sop("Queue size: " + pq1.size());

    Sop("Second element: " + pq1.poll());

    Sop("Queue size: " + pq1.size());

    Sop("Program ends...");
  }
}
```

O/p:
```
Program starts...
Queue size: 5
Queue elements: [12, 23, 46, 123, 87]   // sorts internally
First element: 12
Queue size: 4
Second element: 23
Queue size: 3
Program ends...
```

4*) In PriorityQueue whenever an element is added, it compares and sorts with the existing element.

5*) PriorityQueue is autosorted queue.

6*) When we display element in PriorityQueue, the elements are displayed in random order.

java.lang.
exception: ClassCastException

7*) poll() in queue removes head element. Whenever poll() method gets executed the queue size gets reduced by 1.

8*) Peek() returns head element when peek() method gets executed, there is no change in queue size.

P119

4)

```
package com.jspiders.collectionsdemo;
import java.util.PriorityQueue;
public class Demo3
{
  psvm (String[] args)
  {
    Sop ("Program starts...");
    PriorityQueue pq1 = new PriorityQueue();
    pq1.add(123);
    pq1.add(46);
    pq1.add(23);
    pq1.add(12);
    pq1.add(87);
    Sop ("Queue size: " + pq1.size());
    Sop ("Queue elements: " + pq1);
    Sop ("Head element is: " + pq1.peek());
    Sop ("Queue size: " + pq1.size());
    Sop ("Remove Head element: " + pq1.poll());
    Sop ("Queue size: " + pq1.size());
    Sop ("Program ends...");
  }
}
```

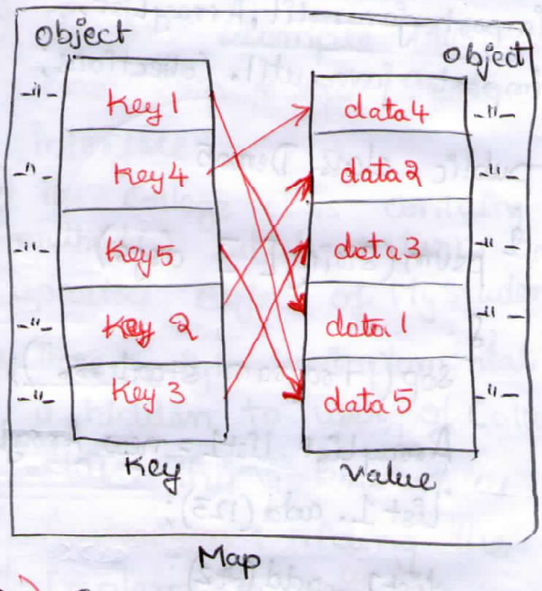
O/p:

```
Program starts...
Queue size: 5
Queue elements: [12, 23, 46, 123, 87]
```

MAP

22-02-2013 | Friday

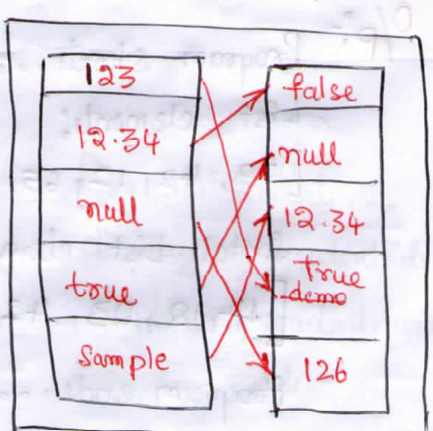
5)
```
package com.jspiders.collectionsdemo;
import java.util.Hashmap;
public class Demo4
{
    psvm (String[] args)
    {
        Sop ("Program starts...");
        HashMap map1 = new HashMap
        map1.put (123,"demo");
        map1.put (true, null);
        map1.put ("Sample", 12.34);
        map1.put (12.34, false);
        map1.put (null, 126);
        map1.put (123, true);
        Sop ("Map elements: " + map1);
        Sop ("Value is " + map1.get (null));
        Sop ("Key are: " + map1.keySet());
        Sop ("Values are: " + map1.values());
        Sop ("Program ends...");
    }
}
```



9*) Each map type classes provides two methods
* keySet()
* values()

10*) keySet() method returns set of keys, whereas values() method returns the collection of values

O/P:-
```
Program starts...
Map elements: {null=126, true=null, 123=true, 12.34=false, Sample=12.34}
Value is 126
Key are: [null, true, 123, 12.34, Sample]
Values are: [126, null, true, false, 12.34]
Program ends...
```



P121

6) package com.jspiders.collectionsdemo;

```
import java.util.ArrayList;
import java.util.Collections;

public class Demo5
{
  psvm(String[] args)
  {
    sop("Program starts...");
    ArrayList list1 = new ArrayList();
    list1.add(123);
    list1.add(42);
    list1.add(12);
    list1.add(65);
    list1.add(4);
    sop("List elements:");
    sop(list1);
    Collections.sort(list1);
    sop("Sorted List elements:");
    sop(list1);
    sop("Program ends...");
  }
}
```

O/P:-

```
Program starts...
List elements:
[23, 42, 12, 65, 4]
Sorted List elements:
[4, 12, 23, 42, 65]
Program ends...
```

Collections Class

11) Collections Class contains several methods which can be operated on any type of collection.

Ex:- sort() is a static method which is used to sort the elements of Collection. This method stores the sorted elements back in the same Collection type.

P122 | 100% Abstraction | (to make s/w more extensible)

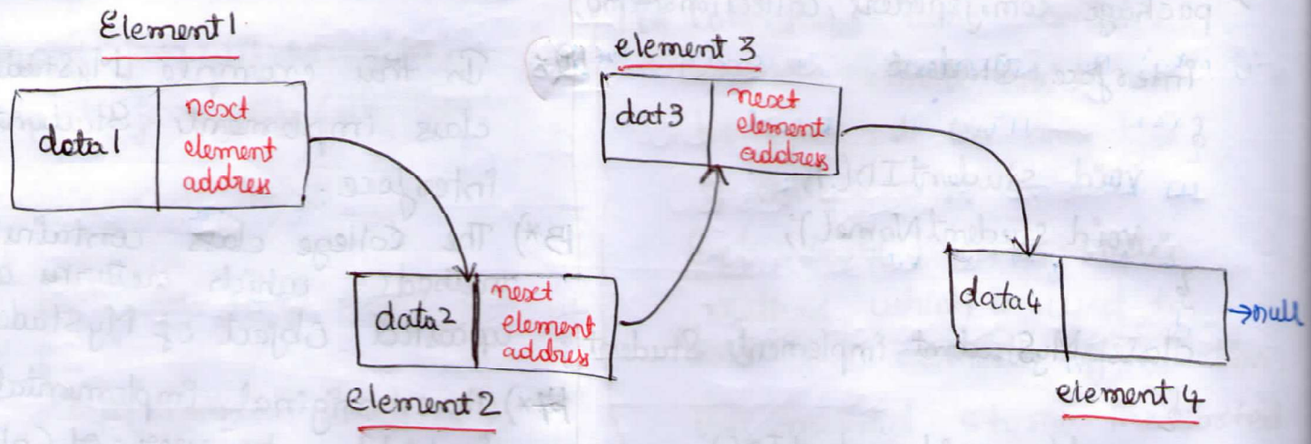
7)
```
package com.jspiders.collectionsdemo;
Interface Student
{
    void studentID();
    void studentName();
}
class MyStudent implements Student
{
    public void studentID()
    {
        Sop ("displaying student id...");
    }
    public void studentName()
    {
        Sop("displaying student name...");
    }
}
class College
{
    Student studentInfo()
    {
        return new MyStudent();
    }
}
public class Demo6
{
    psvm(String[] args)
    {
        Sop("Program starts...");
        College bms = new College();
        Student st1 = bms.studentInfo();
        st1.studentID();
        st1.studentName();
        Sop("Program ends...");
    }
}
```

12*) In this example MyStudent class implements Student interface.

13*) The College class contains one method() which returns an upcasted object of MyStudent

14*) Here original implementation is hidden to user of College class. This is known as Abstraction: "hiding the implementation". It can be achieved by using interface and overriding the interface method in a class and upcasting the class object to interface type.

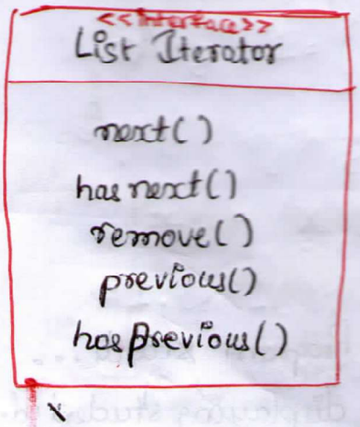
O/P:-
```
Program starts...
displaying student id...
displaying student name...
Program ends.
```



Any Collection type (Ex- LINKED List)

20) The List Iterator should be used only for List Type Collection because List Type is developed by using doubly linked concept.

* Only for List Type:



Iterator:

14*) Collection API provides an iterator to iterate each element of any Collection type.

15*) Iterator is an Interface which provides 3 methods().

15.1*) next() - this method iterates to next element and returns the element. If there is no next element then exception will be thrown NoSuchElementException.

15.2*) hasNext() - this method checks whether next element exists or not. If next element is present it returns true otherwise it returns false.

15.3*) remove() - this method removes the element from the collection. When one element is removed, the prev element will point to the next element of removed element.

ITERATOR

25-02-2013 | MONDAY

P123

8) package com.spiders.collectionsdemo;
import java.util.HashSet;
import java.util.Iterator;
public class Demo7
{
psvm (String[] args)
{
Sop ("Program starts...");
HashSet set1 = new HashSet();
set1.add(12);
set1.add("demo");
set1.add(12.21);
set1.add("true");
set1.add(null);
Sop("Set elements are "+set1);
Iterator itr1 = set1.iterator();
while (itr1.hasNext())
{
Sop ("element is "+itr1.next());
}
Sop ("Program ends");
}
}

O/P:-
Program starts...
Set elements are[null, demo, 12.21, true, 12]
element is null
element is demo
element is 12.21
element is true
element is 12
Program ends...

<< interface >>
Iterator
next()
hasNext()
remove()

Unknown
override all methods

16*) Using Iterator object, we can iterate only once the element of collection.

17*) Iterator object can be used for List type, Queue type, Set type of collection.

List Iterator :-

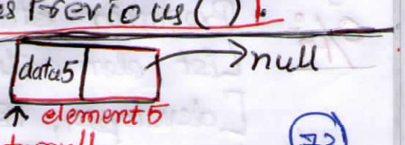
18*) Java Collection also provides List Iterator which can used only for List type.

19*) This Iterator provides additional two methods,

/* Sop("Set element again"+set1);
while (itr1.hasNext())
{
Sop("element is "+itr1.next());
} */

19.1*) To iterate in previous directional (or) element by using previous().

19.2*) To check previous element existence by using hasPrevious().



(Last element)
∵ Iterator is pointing to null.

/* Set elements again[null, demo, 12.21, true, 12] */

P124

9) package com.jspiders.collectionsdemo;

import java.util.ArrayList;

class X
{
  void test1()
  {
    Sop("running test1() of class X");
  }
}

public class Demo9
{
  psvm(String[] args)
  {
    Sop("Program starts...");
    ArrayList list1 = new ArrayList();
    list1.add("developer");
    list1.add(12);
    list1.add(new X());
    Sop("List1 elements are:");
    Sop(list1);
    String s1 = (String) list1.get(0);
    Sop("length of s1:" + s1.length());
    X x1 = (X) list1.get(2);
    x1.test1();
    Sop("Program ends...");
  }
}

O/P:
Program starts...
List elements are:
[developer, 12, com.jspiders.collectionsdemo.X@ad3ba4]
length of s1 = 9
running test1() of class X
Program ends...

*) Whenever elements of collection are read it will be in Object type, it should be downcasted to respective class type to access the subclass members.

GENERICS

P125

10) package com.jspiders.collectionsdemo;
import java.util.ArrayList;
public class Demo10
{ psvm (String[] args)
{ sop("Program starts..");
ArrayList<String> list1 = new ArrayList<String>();
list1.add("Selenium");
list1.add("Developer");
list1.add("Tester");
Sop("Elements : "+list1);
Sop(list1.get(0).toUpperCase());
Sop(list1.get(1).toUpperCase());
Sop(list1.get(2).toUpperCase());
Sop("Program ends...");
}
}

Program starts...
Elements: [Selenium, Developer, Tester]
SELENIUM
DEVELOPER
TESTER
Program ends...

22*) Whenever all elements type of collections is same, then we go for generics

23*) If a collection is declaration with generics we cannot store elements apart from generic type.

24*) If collection is defined with generic then no need to downcast explicitly each element, java does automatic downcast (implicitly).

Ex: An Arraylist of String type stores
ArrayList<String> list1;
only String objects in Arraylist. When elements are read automatically [downcasted/casted to String type.