29th July 2023

Collection framework

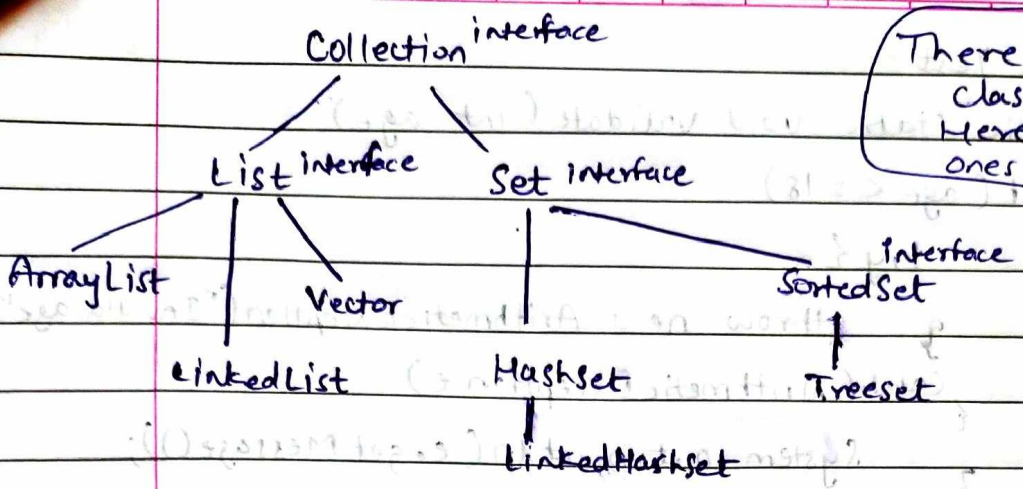
Array disadvantages:
(i) fixed size
(ii) manual shifting

- Collection represents group of objects
- Framework represents inbuilt interface, class & methods which helps to manipulate Collection.
- It provides all advantages of array. It also overcomes disadvantages of array. It is internal implementation of data structure

List - ordered

Set - unordered

Collection



There are more classes. Here only important ones are shown.

List - It stores element index wise.

List	Set
(i) stores indexwise element	(i) Set does not maintain index
(ii) Duplicate element is allowed in List	(ii) Duplicate object is not allowed.
(iii) Insertion order is preserved in List	(iii) Insertion order is not preserved in set.

Generic <>

```
import java.util.ArrayList;
import java.util.List;
public class Test {
    public static void main(String[] args)
    {
        List<Integer> al = new ArrayList<>();
        al.add(20);
        al.add(50);
        al.add(20);
        al.add(70);
        System.out.println(al);
        int x = al.get(1);
        System.out.println(x);
} }
```

hasNext() - पुढे element आहे की नाही इतकेच सांगते

next() - moves to next element

```
import java.util.Iterator;
import java.util.ArrayList;
import java.util.List;
public class Test {
    p.s.v.main(String[] args)
    {   List<Integer> al= new ArrayList<>();
        al.add(20);
        al.add(50);
        Iterator<Integer> itr = al.iterator();
        while(itr.hasNext())
        {
            int x = itr.next();
            System.out.println(x);
        }
    }
}
```

```
import java.util.Iterator;
import java.util.ArrayList;
import java.util.List;
public class Test
{
    p.s.v.m(---)
    {   List<String> stList = new ArrayList<>();
        stList.add("BMW");
        stList.add("VW");
        stList.add("Tata");
        Iterator<String> itr = stList.iterator();
        while (itr.hasNext())
        {
            System.out.println(itr.next());
        }
    }
}
```

```
public class Test
{
    p.s.v.m.(String[] args)
    {
        List<Student> al = new ArrayList<>();
        Student s1 = new Student();
        s1.setRollno(1);
        s1.setName("aaa");
        Student s2 = new student();
        s2.setRollno(2);
        s2.setName("bb");

        al.add(s1);
        al.add(s2);

        System.out.println(al);  // addresses of s1 & s2 will be printed
        Iterator<Student> it = al.iterator();
        while(it.hasNext())
        {
            Student st = it.next();
            System.out.println(st.getRollno() + " " + st.getName());
        }
    }
}
```

Student.java

```
private int rollno;
private String name;
public int getRollno()
{
    return rollno;
}
public String getName()
{
    return name;
}
public void setRollno(int x)
{
    this.rollno = x;
}
public void setName(String s)
{
    this.name = s;
}
```

```
p.s.v.m.(String[] args)
{
    List<Student> al = new ArrayList<>();
    Student s1 = new Student();
    s1.setRollno(1);
    s1.setName("aaa");
    al.add(s1);
    s1 = new Student();
    s1.setRollno(2);
    s1.setName("bbb");
    al.add(s1);
```

Reusing s1 to point to different objects.

```
Iterator< Student > it = al. Iterator();
while ( it. hasNext())
{
    System.out. Student st = it.next();
      System.out.println(st.getRollno());
      System.out.println (st.getName());
 }
}
```

```
public class Test1
{
  p.s.v. main(String[] args)
  {
    List<Student> al = new ArrayList<>();
    Scanner sc = new Scanner (System.in);
    Sysout. (" Enter no. of students to add");
    int n = sc.nextInt();
    for(int i=1; i<=n; i++)
    {
      Student s1 = new Student();
      Sysout. (" Enter rollno");
      s1. setRollno(sc.nextInt());
      Sysout. (" Enter name");
      s1. setName ( sc. next());
      al. add(s1);
    }
Iterator<Student> itr = al. iterator();
while(itr.hasNext())
{
  Student st = itr.next();
  Sysout. ( st.getRollno());
  Sysout. ( st. getName());
 }
}
}
```

My code :-

```
public class Test
{
    p.s.v.m.(String[] args)
    {
        List<Employee> empList = new ArrayList<>();
        Employee e1 = new Employee();
        e1.setEmpid(12345);
        e1.setName("XYZ");
        empList.add(e1);
        e1.setEmpid(67890);
        e1.setName("ABC");
        empList.add(e1);
        System.out.println(empList);
        Iterator<Employee> it = empList.iterator();
        while(it.hasNext())
        {   Employee emp = it.next();
            System.out.println(emp.getEmpid() + " : " +
                               emp.getName());
        }
    }
}
```

Output :-

```
[Employee [empid=67890, name=ABC],
Employee [empid=67890, name=ABC]
]

67890 : ABC
67890 : ABC
```

```
int n = sc.nextInt();
for(int i=1; i<=n; i++)
{
}
```

if it would have been `i<=sc.nextInt();`, code will ask for input in every iteration

```
while(it.hasNext())
{
    Student s1 = it.next();
    // NoSuchElementException - it.next().getId();
}
```

30th July 2023

List Inside List

```
List<String> mh = new ArrayList();
mh.add("Pune");
mh.add("Nashik");        Sysout.(mh);
List<String> gj = new ArrayList();
gj.add("Surat");
gj.add("Gandhinagar");   Sysout.(gj);
List<List<String>> india = new ArrayList();
india.add(mh);
India.add(gj);
System.out.println(india);
//[[Pune,Nashik], [Surat, Gandhinagar]]
Iterator<List<String>> itr = india.iterator();
while(itr.hasNext())
{
    List<String> state = itr.next();
    Iterator<String> itr2 = state.iterator();
    while(itr2.hasNext())
    {
        String city = itr2.next();
        Sysout.(city);
    }
}
```

<> optional

```
List<String> st = new ArrayList<>();
```

M T W T F S S | Page No.: | Date: | YOUVA

Java Generics

compile-time checking with generics | reduces work of tester
class cast exception w/o generics
No type-casting needed with generics

1. This feature is added in JDK 1.5 version.
2. It provides type safety. It overcomes class cast exception problem.
3. No need to typecast.
4. It reduces tester's efforts

Without generics.

```
List l = new ArrayList();
l.add(10);
l.add(30);
l.add("aaa");
l.add(40);
int x = (int) l.get(0);
String s = (String) l.get(1); ← Runtime exception - Class cast exception
```

With generics

```
List<Integer> l = new ArrayList<>();
l.add(10);
l.add(30);
// compile-time error- l.add("aaa");
int x = l.get(0);
// error:- String s = l.get(1);
```

```
List<String> l = new ArrayList<>();
l.add("aaa");
l.add("bbb");
l.add("ccc");
for(String s : l)
{
}
Sysout.(s);
```

for each = enhanced for loop

Lambda expression

```
l.forEach((x) -> System.out.println(x));
```

↑ We are providing implementation of method

Method reference

```
l.forEach(System.out::println);
```

Functional programming

```
List<List<String>> India = new ArrayList();
india.forEach((al) -> al.forEach((city) -> System.out.println(city)));
```

OR

```
india.forEach((al) -> al.forEach(System.out::println));
```

Set

```
Set<Integer> s = new HashSet<>();
s.add(10);
s.add(5);
s.add(10);
s.add(30);
System.out.println(s);
```

10 will appear once, 5 once & 30 once

Duplicates won't be stored & order won't be presened

```
Iterator<Integer> itr = s.iterator();
while(itr.hasNext())
{
    int x = itr.next();
    System.out.println(x);
}
```

```
for(int x : s)
{
    System.out.println(x);
}
```

Q. How to remove duplicate elements from list ?

```
List<String> mh = new ArrayList();
mh.add("Pune"); mh.add("Mumbai");
```

Remove duplicates w/o order

```
Set<String> s = new HashSet(mh);
```

Remove duplicates to preserve order

```
Set<String> s = new LinkedHashSet<>(mh);
```

To preserve order

```
Set<String> s = new LinkedHashSet<>(ArrayListName);
```

Q. How to sort list elements?

→ Collections.sort(arraylistName);

↑ utility method

Remove duplicates + Sort elements in arraylist

Set<String> s = new Treeset<String>(arrayListName);