Date → always Cw
→ Args Constr
· getTime() → will record

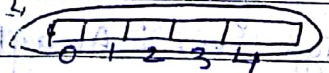
→ finding first Non repeating character in a string

→ String is a final class, so we cannot override any of the string class method because for overriding inheritance is mandatory and final class cannot be inherited

Arrays

→ Arrays are nothing but contigeous memory location wherein we can store homogenious (similar type) elements which will share the comman name
→ If we are using arrays, processing will be faster.
→ There are two types of arrays
① primitive arrays
② Object arrays or derieved arrays
① Primitive Array :- Array which are created using primitive data types are called as primitive array
→ ⓐ In primitive arrays we can store primitive data
② Derived Arrays/Object Arrays :- Arrays of derieved types are called as derieved/object arrays
→ In derieved/Object arrays we can store object's address

→ General syntax (Declaration and initialization)
type[] refvar = new type[size]; int
→ Size can be either int type data or byte type data or short type data
→ Arrays size cannot be long, double and float type.
→ type can either a primitive (any of 8) data type or java type
eg int[] arr = new int[5]



→ If you want to store any element in the array then you should use the following syntax
refvar[index] = value;
eg- arr[0] = 10;

→ length is a variable of special class arr

eg1
```
psvm(String[] arr)
{ int[] arr = new int[5];
  arr[0] = 10;
  arr[1] = 20;
  arr[2] = 30;
  arr[3] = 40;
  arr[4] = 50;
// arr[5] = 60; //array index out of bound exception
  sop("The length of array is :: "+arr.length);
  for(int i=0; i< arr.length; i++)
  { sop(arr[i]); }
```

output

```
The length is :: 5
10
20
30
40
50
```

eg2
```
psvm(String[] args)
{ double[] darr = new double[5];
  darr[0] = 10.5;
  darr[1] = 10.4;
  darr[2] = 10.3;
  darr[3] = 90.2;
  darr[4] = 90.1;
  sopln(darr.length);
  for(double d : darr)
  { sopln(d); }
```

output

```
5
10.5
10.4
10.3
90.2
90.1
```

→ Enhanced for loop was introduced in JDK 1.5.

→ Enhanced for loop is used to fetch the elements stored in a arrays or collection, without any condition

→ General syntax for enhanced for loop

```
for(ArrayType var : ArrayRefVar)
{ sopln(var);
    or use var's value for some purpose!
}
```

→ If you have certain conditions or if you want to fetch the partial elements from an array or collection then we can use Regular for loop

→ If you have no condition to fetch the elements of a array or to perform some operation on all the contents of an array then it is best to use Enhanced for loop

→ Enhanced for loop is also called as for each loop as each & every element will be iterated without fail

→ Enhanced for loop is used for array and collection (dynamic array)

Q. find largest number and its index

```
PSVM(String[] args)
{ int[] arr = {90,190,900,200,300};
  int largest = arr[0];
  int largestIndex = 0;
  for(int i=0; i< arr.length ; i++)
  { if( largest < arr[i])
    { largest = arr[i];
      largestIndex = i;
    } }
SOPln("largest Number is"+ largest + "at Index"+largestIndex)
```

Q. find smallest number and its index

```
int smallest = arr[0];
int smallestIndex = 0;
for(int i=0; i< arr.length ; i++)
{ if( smallest > arr[i])
  { smallest = arr[i];
    smallestIndex = i; }
SOP("smallest Number :"+smallest+"@Index"+
                                smallestIndex);
}
```

Sorting the array in ascending order

```
PSVM (String[] args)
{ int[] arr = {50, 40, 20, 30, 10};
  for(int i=0; i< arr.length ; i++)
  { for (int j=i+1; j< arr.length; j++)
    { if( arr[i] > arr[j] )
      { int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp; }
    }
  }
  for (int a : arr)
  { Sopln(a); }
}
```

Sorting the array in descending order

```
for(int i=0 ; i< arr.length; i++)
{ for (int j=i+1; j< arr.length; j++)
  { if( arr[i] < arr[j] )
    { int temp = arr[i];
      arr[i] = arr[j];
      arr[j] = temp ;
    }
  }
  for( int a ; arr)
  { Sopln(a);
```

→ Sort() static function is available in java.util.Arrays

BinarySearch " " " " " " " " .

```
PSVM (String[] args)
{ double[] darr = {10.3, 10.2, 10.5, 10.1, 10.4};
  Arrays.sort(darr);
  for( double d : darr)
  { Sopln(d); }
  Sopln (Arrays.BinarySearch (darr, 10.1));
  Sopln (Arrays.BinarySearch (darr, 10.3));
  Sopln ( Arrays.BinarySearch( darr, 90.9));
}
```

10.1
10.2
10.3
10.4
10.5

0
2
-6

```
PSVM (String[] args)
{ int h=5;
  int[] arr = new int[5] / 10/2 / 'z'
  for (int a ; arr)
  { Sopln (a); }
};
```

0 0 0 0 0

→ If we have created an array and if we have not stored any elements then every location of that array will have default value based on the type of the array

→ Different Declaration of Array

```
int[] arr = new int[size];
int[] arr = {10,20,30,40,50};
int[] arr = new int[]{10,20,30,50,40};
int[] arr = new int[size]
int[] arr = new int[size]
```

Interview

```
PSVM (String[] args)
{ final int[] arr = {10,20};
  Sopln(arr[0]);
  arr[0] = 90;
  Sopln(arr[0]);
  Sopln("Done");
}
```

10
90
Done

```
final int a = 10
a=90 X
```

```
final A a1 = new A();
```

a1 = new A() X

final int[] arr = {10,20}

arr = new int[3] X

Concept of final for reference variable.

→ toString() method for char array is override to give the concatenation of char.

```
PSVM (String[] args)
{ int [] arr = new int[5];
  Sopln(arr);
double[] darr = new double[5];
  Sopln(darr);
Char[] carr = new Char[5]
  Sopln(carr);
  Carr[0] = 'A';
  Carr[1] = 'P';
  Carr[2] = 'E';
}} Sopln(carr);
```

[I@ —
[D@ —
[C
A
APE

Derived Array / Object Arrays :-

```
class A {}
Class Tester
{ PSVM (String[] args)
    A[] ref = { new A(), new A(), new A(), new A() }
    Sopln("The length of ref is" + ref.length);
    For (A a1 : ref)
  {} { Sopln(a1); }
```

```
class B{ int i;
         B(int i)
          {this.i = i;}
       Public String toString()
       { return "i value = "+i;}

Public Class Tester2
{ PSVM(String[] args)
  { B[] ref = { new B(10), new B(20), new B(30)};
      for (B b1 : ref)
  }{ } Sopln(b1);{
}
```

i value = 10
i value = 20
i value = 30
i value = 40
i value = 50

```
psvm (String[] args)
{ String[] ref = { new String("Hi"), "Hello", "Welcome", "Namaste"
                                          new String("Hey") ;
   for (String s : ref)                  | Hi
    { sopln(s) ; }                       | Hello
}                                        | Welcome
                                         | Namaste
                                         | Hey
```

```
psvm (String[] args)
{ String rv = "Qspiders is the best institute in the universe";
   String[] ref = rv.split(" ");         | Qspiders
   for (String s : ref)                  | is
    { sopln(s); }                        | the
   sopln(ref.length)                     | best
}                                        | institute
                                         | in
                                         | the
                                         | universe
                                         | 8
```

Q What is the no. of words in a given sentence

```
PSVM(String[] args)
{ String rv = "Ram+is+my+Hero";
   String[] ref = rv.split("\\+");       | Ram
   for (String s : ref)                  | is
{ { sopln(s); }                          | my
                                         | Hero
```

Regex Concept

Input String Impossible is Nothing

Output String Nothing is Impossible

```
psvm (String[] args)
{ String rv1 = "Impossible is Nothing";
   String rv2 = "" // empty String
   String[] ref = rv.split(" ");
   rv2 = ref[2] + " " + ref[1] + " " + ref[0]
```

Calculate the count of character in a given string

rv.length()

```
char[] carr = rv.toCharArray()
sopln(carr.length)
```

Boxing

```
PSVM (String[] args)
{ Integer rv1 = new Integer(90); // boxing
  Sopln(rv1);
  Double rv2 = new Double(90.5); //boxing
{ Sop(rv2);
```

90 — string form

90.5

Sopln(rv1) is implicit call to rv1.toString(). here toString method is overridden to return the value based on object value. Return type is string

```
PSVM(String[] args)
{ Integer rv1 = new Integer(10);
  int a = rv1.intValue();
  Double rv2 = new Double(10);
  double d = rv2.doubleValue();
  Character rv3 = new Character('z');
  Char ch = rv3.charValue();
  Boolean rv4 = new Boolean(true);
  boolean b = rv4.booleanValue();
  Sopln(a + "\n" + b + "\n" + c + "\n" + d);
}
```

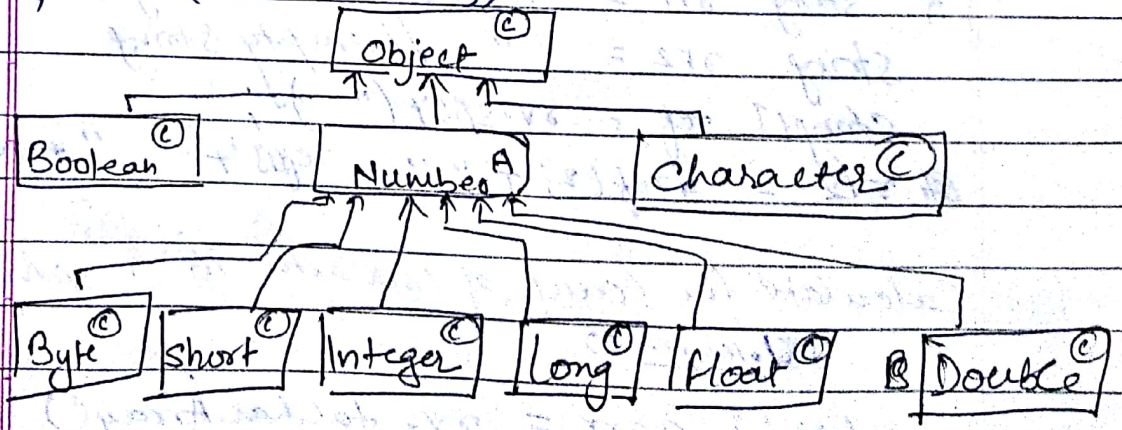
10 → primitive value

10.5

z

true

```
PSVM (String[] args)
{ Integer rv1 = 10; //auto-boxing
  int a = rv1; //auto-unboxing
  Boolean rv2 = true // auto boxing
  boolean b = rv2; // auto-unboxing
{ Sopln(a + "\n" + b);
```



Wrapper class Hierarchy @ java.lang

Boxing

→ Converting the primitive data into an object is called as Boxing operation

or wrapping up the primitive data into an object by using wrapper classes is called as Boxing operation

→ To perform this boxing operation we have wrapper classes corresponding to primitive data types

Unboxing

→ Converting the primitive data which is stored as an object into its primitive data types

or Getting the primitive data which is wrapped into an object by using value() method of wrapper class is called as unboxing operation

→ Both boxing and unboxing is automatic which is also called as auto-boxing & auto-~~auto~~unboxing

→ All the wrapper classes which support boxing and unboxing operation, are available in built-in package java.lang

→ To use all wrapper classes in any package or any other package there is no need to import java.lang pack

→ All the wrapper classes are sub-class to object class directly or indirectly

→ All the number related wrapper classes are subclass to a class called Number

→ Number is an abstract class which is subclass to object class directly

→ All the wrapper classes are concrete, final and immutable.

Q Is java 100% object oriented programming language

→ No, java is not pure object oriented programming because java supports primitive type

→ If in a programming language, if everything is dealt through object and if it doesnot support primitive type then we can say that it is 100% object oriented

Advantage/uses of (wrappers) boxing and unboxing

→ If you want to develop generic method which can receive any input, then we can use boxing and unboxing

ex- public boolean equals(Object obj)
{ }

```
class A
{ }
Public class Tester
{ static void test(Object obj)
{ Sopln("Input Recieved");
  Sopln(obj);
PSVM(String[] args)
{ test(new A());          → Auto Upcasting
  test(new String("hello")); → Auto Upcasting
  test("Hey");            → Auto Upcasting
  test(new Integer(90));  → Auto upcasting → to Integer wrapper class
  test(10);               → Auto boxing then Auto upcasting → Double
  test(90.5);             → Auto boxing then Auto upcasting
  test('A');              → Auto boxing then Auto upcasting → Character
  test(true);             → Auto boxing then Auto upcasting → Boolean
```

Generic Method

```
PSVM(String[] args)
{ int a = Integer.parseInt("100");
  Sopln(a);
  String s1 = "900";
  String s2 = "100";
  Sopln(s1+s2);
  int b = Integer.parseInt(s1);
  int c = Integer.parseInt(s2);
  Sopln(b+c);
}
```

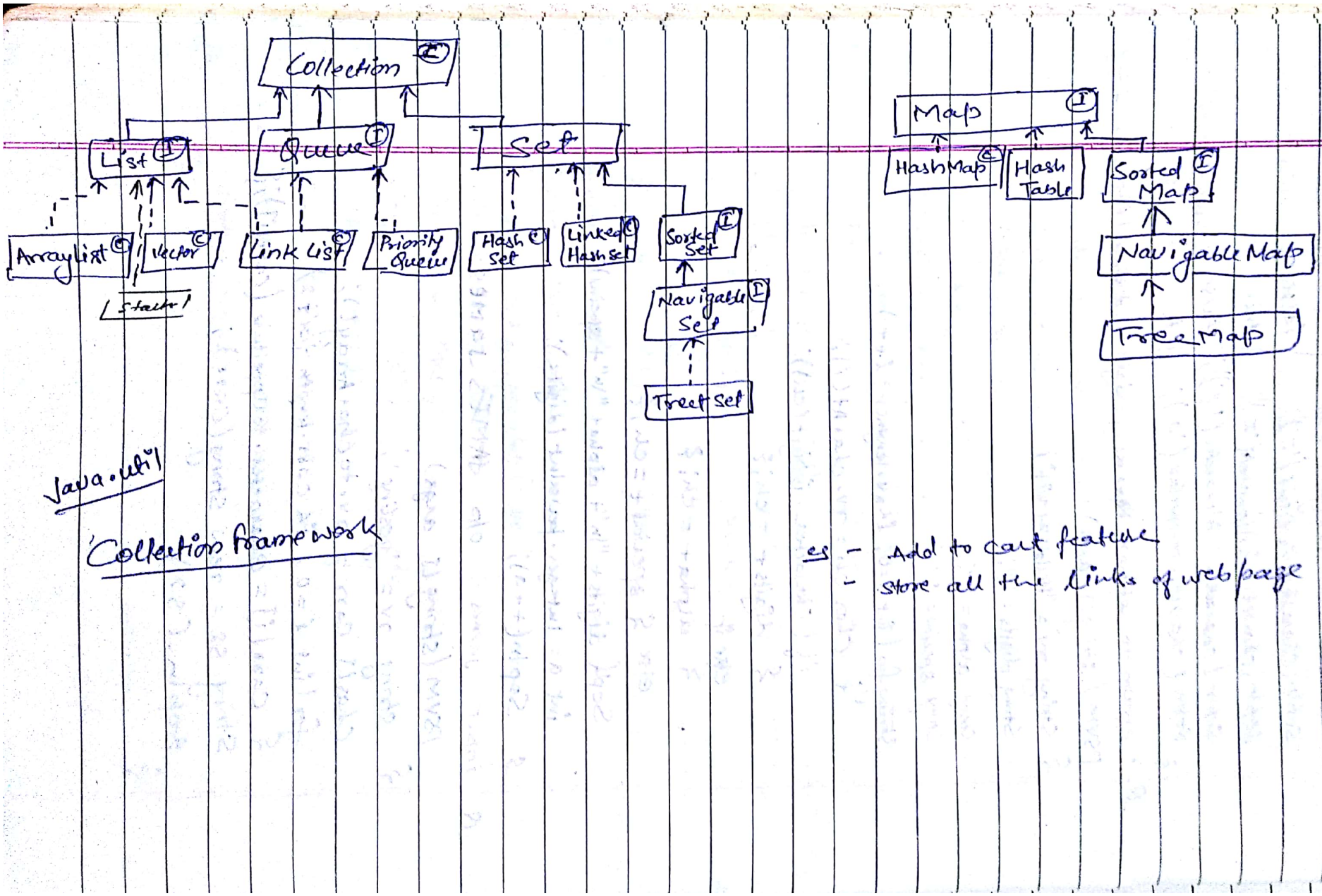
```
PSVM (String[] args)
{ Sopln (Character.isDigit('9'));        | true
  Sopln (Character.isDigit('z'));        | false
  Sopln (Character.isAlphabetic('z'));   | true
  Sopln (Character.isAlphabetic('$'));   | false
  Sopln (Character.toUpperCase('a');     | A
}
```

Q. Program to separate Alphabets, digit and special char: in String

```
PSVM (String[] args)
{ String rv = "Hello1234+?!";
  String digits = "";
  String alpha = "";
  String special = "";
  String for (int i=0; i<rv.length(); i++)
    { Char ch = rv.charAt(i);
      if (Character.isDigit(ch))
      { digits+ = ch; }
      else if
      { alpha+ = ch; }
      else { special+ = ch; }
  Sopln( digits+ "\n"+ alpha+ "\n" + special);
  int a = Integer.parseInt(digits);
{ Sopln(++a);
```

Q input james o/p ~~JAMES~~ JAMES

```
PSVM (String[] args)
{ String rv = "gosling";
  Char[] carr = rv.toCharArray();
  for (int i = 0; i< carr.length ;i=i+2)
  { carr[i] = Character.toUpperCase (carr[i]); }
  String s2 = new String(carr);
  Sopln (s2)
}
```



Java.util

Collection framework

eg - Add to cart feature
- store all the links of web page

java.util package

Arraylist features

1. dynamic
2. heterogenous
3. Indexed
4. duplicate
5. null

10	10.5	'A'	"Hey"	true	null	10	collectionstopic.A@
0	1	2	3	4	5	6	7

add(index,element) | get(index)
add(element) | remove(index) | remove(element)
size() | clear()
list1.addAll(list2) | protected Method list.removeRange(2,6)
list2.addAll(2,list3)
list1.removeAll(list2) | Collections.sort(list) | Collections.reverse(list)
list2.retainAll(list3) | Collections.BinarySearch(list,8)

```
class A {
PSVM(String[] args)
{ Arraylist list = new Arraylist();
  list.add(10); list.add(10.5); list.add('A'); list.add("Hey");
  list.add(true); list.add(10); list.add(new A());
  list.add(null)
  Sopln(list)
  Sopln("size is :"+ list.size());
  for(int i=0; i<list.size(); i++)
  { Sopln(list.get(i)); }
}
```

[10, 10.5, A, Hey, true, null, 10, collectionstopic.A@...]
Size is:: 8
10
10.5
Hey
true
null
10
collectionstopic.A@1d555ad

```
PSVM(String[] args)
{ Arraylist list = new Arraylist();
  list.add(10); list.add(20); list.add(30);
  Sopln(list);
  list.add("Hi");
  Sopln(list);
  list.add(1,90);
  Sopln(list);
  list.add(2,"Hey");
  Sopln(list);
}
```

[10, 20, 30]
[10, 20, 30, Hi]
[10, 90, 20, 30, Hi]
[10, 90, Hey, 20, 30, Hi]

Inserting is not good in Arraylist as it will shuffle the elements, and say if 1000 elements are there in Arraylist then 1000 shuffling will be there. Not good

```
PSVM (String[] args)
{ ArrayList list1 = new Arraylist();
  List1.add(10); list1.add(20); list1.add(30); list1.add(40);
  Arraylist list2 = new Arraylist();
  list2.add(50); list2.add(60); list2.add(70);
  Arraylist list3 = new Arraylist();
  list3.add(80); list3.add(90); list3.add(100);
```

Sopln(list1);	[10, 20, 30, 40]
Sopln(list2);	[50, 60, 70] :
Sopln(list3);	[80, 90, 100]
Sopln(=====);	==========
list1.addAll(list2)	
Sopln(list1);	[10, 20, 30, 40, 50, 60, 70]
list2.addAll(1, list3)	
Sopln(list2);	[50, 80, 90, 100, 60, 70]

```
}}
```

```
PSVM (String[] args)
{ Arraylist list = new Arraylist();
  list.add("Hi"); list.add(90.5); list.add(true); list.add("Hey");
```

Sopln(list);	[Hi, 90.5, true, Hey]
list.remove(0);	
Sop(list)	[90.5, true, Hey]
list.clear();	
Sopln(list);	[]

```
}}
```

```
PSVM (String[] args)
{ ArrayList list = new ArrayList();
  list.add(80.5); list.add(90.5); list.add(100.5);
```

Sopln(list)	[80.5, 90.5, 100.5]
Sopln(list.remove(80.5));	true
Sopln(list);	[90.5, 100.5]
Sopln("=======");	=======
Sopln(list.remove(0));	90.5
Sopln(list)	

```
}}
```

→ There are two versions of remove() method which will remove the element

① remove(index)

→ this method will remove the indexed element and after removing the indexed element, it will return the element, which was removed

② remove(element)

This method will remove the given element and after removing the element it will return a boolean value true and if the given element is not removed then it will return the boolean value false

```
PSVM(String[] args)
{ ArrayList list1 = new ArrayList();
list1.add(10); list1.add(20); list1.add(30); list1.add(40); list1.add(50);
ArrayList list2 = new ArrayList();
list2.add(30); list2.add(40); list2.add(50); list2.add(60); list2.add(70); list2.add(80);
ArrayList list3 = new ArrayList();
list3.add(60); list3.add(70); ----- list3.add(100);
```

Sopln(list1);	[10, 20, 30, 40, 50]
Sopln(list2);	[30, 40, 50, 60, 70, 80]
Sopln(list3);	[60, 70, 80, 90, 100]
Sopln("======");	==========
list1.removeAll(list2);	[10, 20]
Sopln(list1); Sopln("======");	==========
list2.retainAll(list3);	
Sopln(list2);	[60, 70, 80]

```
}
class Tester extends ArrayList
{ PSVM(String[] args)
{ Tester list = new Tester()
list.add(10); list.add(20) ---------- list.add(80)
```

Sopln(list)	[10, 20, 30, 40, 50, 60, 70, 80]
list.removeRange(2,6)	
Sopln(list	[10, 20, 70, 80]

```
}
```

inclusive — exclusive

like in substring(startIndex, endIndex)

```
PSVM (String[] args)
{ ArrayList list = new ArrayList()
  list.add("8spider") "Jspider" "Ospider" "Nspider" "Hspider"
  Sopln(list) [8spider, Jspider, Ospider, Nspider, Hspider]
  Collections.sort(list)
  Sopln(list)                    [Hspider, Jspider, Nspider, Ospider, 8spider]
  Sopln(Collection.BinarySearch(list, "8spider")   1
                                (list, "Nspider");  2
                                (list, "pider");   -5
```

```
PSVM (String[] args)
{ ArrayList list = new ArrayList();
  list.add("Mango"); list.add("Grapes"); list.add(90); list.add("Apple");
  Sopln(list)                  [Mango, Grapes, 90, Apple]
  Collection.sort(list)        RTE or exception
  Sopln(list)
```

```
PSVM (String[] args)
{ ArrayList list = new ArrayList();
  list.add(10.1); list.add(10.2) ------ ---- list.add(10.5);
  Sopln(list);                 [10.1, 10.2, 10.3, 10.4, 10.5]
  Collections.reverse(list);
  Sopln(list);                 [10.5, 10.4, 10.3, 10.2, 10.1]
```

Collections API or framework

→ Collection is framework, which is having lot of classes, interfaces and many built in functions which will serve for comman purpose

→ In collections, size will be dynamic

→ In collections, we can store heterogeneous element

Note → Collections is also called as dynamic array

** In collections, all the elements will be stored as an object

→ All the collections related classes are available in java.util package

→ Collection is categorized into four types

1. List
2. Queue
3. Set
4. Map

→ In collections framework there is a root interface collection and concrete class called collections.

→ Common features of List

1. Size of list is dynamic
2. We can heterogenous elements
3. It is a indexed type of collections
4. We can store duplicates
5. We can store null
6. As it is indexed type of collection, we can get the elements based on index

→ List is an interface and it is inheriting from another interface collection

→ There are 3 classes implementing List interface features

1. ArrayList
2. Vector
3. Linked List

→ Q. When we should choose List type of collection

→ Whenever we need to store element based on index and if we want to maintain insertion order, then we will choose List type of collection

ArrayList → memory allocation will be contiguous, so, while developing if there is more searching operation, then it will be faster if you choose Arraylist

② LinkedList :- memory allocation will be scattered, while developing if insertion operation more then we should choose Linked list

③ Vector - To achieve thread safety we should use ~~thread~~ vector

Queue

If we want to maintain the general queue then we should choose Queue type collection

eg- movie ticket

FIFO → first In and first out

Features of Queue

① Size is Dynamic
② We can store heterogenous elements*
③ It is not indexed type of collection
④ We can store duplicates
⑤ We cannot store null
⑥ As it is not a index type of collection we cannot get elements based on index
we should use two queue method
peak()
pole()
⑦ Queue is an Interface, which is inheriting from another Interface collection
⑧ Two classes Inherit from Queue
① Priority Queue
② Linked List

*

PSVM (String[] args)
{ Priority_Queue queue = new PriorityQueue();
queue.add(50); queue.add(40) ... 30 20 10

Sopln(queue);	[10, 20, 30, 40, 50]
Sopln(queue.size())	5
Sopln(queue.poll)	10
Sopln(queue)	[20, 30, 40, 50]
Sopln(queue.poll)	20
Sopln(queue)	[30, 40, 50]
	30
	[40, 50]
Sopln(queue)	40
Sopln(queue.poll)	[50]
Sopln(queue)	50
Sopln(queue.poll)	null
Sopln(queue)	[]

}
order can be anything while displaying but logically it will be sorted.

PSVM(String[] arg)
{ PriorityQueue queue = new PriorityQueue();
queue.add("Frankin") ... "Edwin" Sobhin Bharat Ashwin
int size = queue.size(); // size of queue change on poop
for(int i=0; i< size; i++)
{ //Sopln(queue.peak())
Sopln(queue.poll());
}

Ashwin
Bharat
Sobhin
Edwin
Frankin

PSVM(String[] args)
{ PriorityQueue queue = new PriorityQueue();
queue.add('E')...
queue.add('D')
.add(90) / null 'D'
.add('3')
.add('A')
Sopln(queue);

RunTime Error
i.e exception

SET

→ Whenever we need to store unique elements then we should use set type of collection

→ Features of set

① Generic
② We can store heterogenous element
③ It is not a indexed type of collections
④ Cannot store duplicates (it ignores)
⑤ Null is allowed
⑥ As it is not a indexed type of collection we cannot get the element based on index. we should use iterator method

→ set is an interface which is inheriting from collection interface

→ There are three class implementing set

① Hashset
② Linked Hashset
③ Treeset

```
PSVM(String[] args)
{ Hashset set = new Hashset();
 set.add(10);
 set.add(10.5);
 set.add(true);
 set.add(null); // null is allowed
 set.add('A');
 set.add(10); // Duplicates are ignored // No error
 set.add(10.5);
 Sopln(set); [null, A, true, 10, 10.5,]
 Sopln("size is::" + set.size()] size is::5
}
}
```

// LinkedHashSet class maintains the insertion order by avoiding duplicates

```
PSVM(String[] args)
{ LinkedHashSet set = new LinkedHashSet();
// set.add(10);
  set.add(10.5);
  set.add(true);
  set.add(null);
  set.add('A');
  set.add(10);
  set.add(10.5);
  Sopln(set);        [10, 10.5, true, null, A]
  Sopln(set.size()); 5
```

TreeSet (11)

Auto sorting

only similar value like Queue

```
PSVM(String[] args)
{ TreeSet set = new TreeSet();
  set.add(50); set.add(40);      30  20  10
  Sopln(set.size()); } 5
  Sopln(set)         } [10, 20, 30, 40]
```

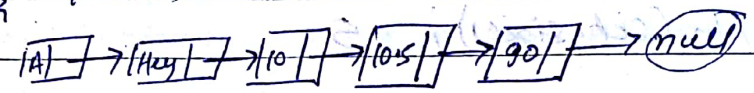
How to know if element is added or discarded (in case of duplicates)

```
PSVM(String[] args)
{ HashSet set = new HashSet();
  Sopln(set.add(10));     true
  Sopln(set.add(10));     false
  Sopln(set.add(90));     true
  Sopln(set.add(100));    true
  Sopln(set.add(90));     false
  Sopln(set);             [100, 10, 90]
{
```

→ How to get the element in set

→ every element will be storing in form of data structure

```
PSVM(String[] args)
{ HashSet set = new HashSet();
  set.add(10); set.add(10.5); set.add('A'); set.add("Hey"); set.add(90);
  sop(set)                                          [A, Hey, 10, 10.5, 90]
  Iterator itr = set.iterator();                    A
  while(itr.hasNext())                              Hey
  { sop(itr.next());                                10
  {   itr.remove();                                 10.5
  { sop(set)                                        90
                                                    []
```



End null and element null is different

hasNext() → returns boolean

it returns true if pointed to element else ~~false~~ returns

next()
- ↳ return element
- ↳ Push hasNext()

remove() - remove element returned by next()

remove() method cannot be used alone [Runtime Error]

remove() method should be used along with next();

using one iterator object we can iterate only one

```
PSVM(String[] args)
{ HashSet set = new HashSet()
set.add("Mumbai"); ... "Chennai" "Bangalore" "Hyderabad"
sop(set)                                            [Mumbai, Chennai, Bangalore, Hyderabad]
Iterator itr = set.iterator();                      [Mumbai]
while(itr.hasNext())                                Chennai
{ sop(itr.next()) }                                 Bangalore
Iterator itr1 = set.iterator();                     Hyderabad
while(itr1.hasNext())                               = = = = -
{ sop(itr1.next()); }                               Mumbai Bangalore Chennai Hy
                                                    Mumbai
                                                    Bangalore
                                                    Chennai
                                                    Hyderabad
```

Iterator itr = set.iterator();

Full abstraction

① interface Iterator
{
boolean hasNext();
object next();
void remove();
}

② class unknown implements Iterator
Anonymous
{ Public boolean hasNext()
{ if pointing to any Element return true
if pointing to null (end) return false
public object next()
{ store value pointed by hasNext();
Push (increament) hasNext() to next element
return the value returned by hasNext()
}
public remove()
{ remove the element pointed returned by next() method from collection
}

③ Anonymous av = new Anonymous();
→ (hasNext()
next()
remove())

4. Iterator itr = new Unknown();
→ hasNext()
→ next()
→ remove()

5. Iterator itr = set.iterator()
→ hasNext()
→ next()
→ remove()

→ iterator() method is a non-static method of Hashset class which will return the Iterator (interface) type object in which hasNext(), next() and remove() method will be available with some implementation

→ It is auto-upcasting, because we are storing it Iterator type object in iterator reference variable itr.

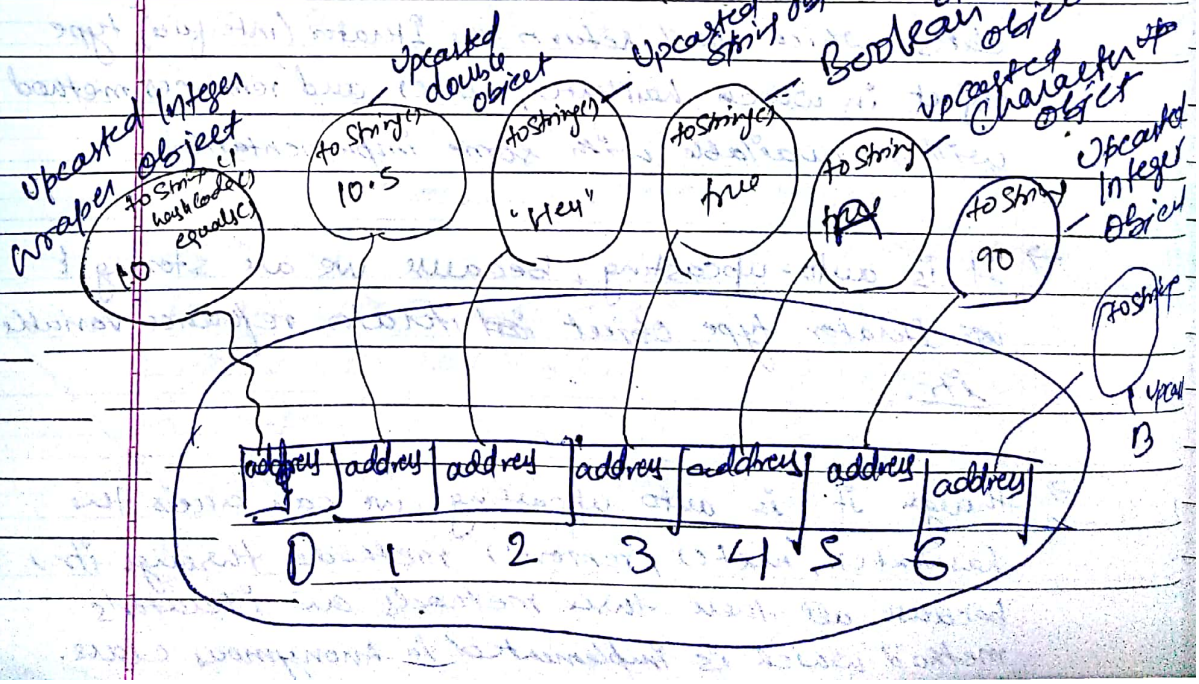
→ Though it is auto upcasting, we can access the hasNext(), next(), remove() methods through itr, because all these three methods are iterator's method which is implemented in Anonymous class.

In case of upcasting we can access the overridden method of interface, and we will get the latest implementation

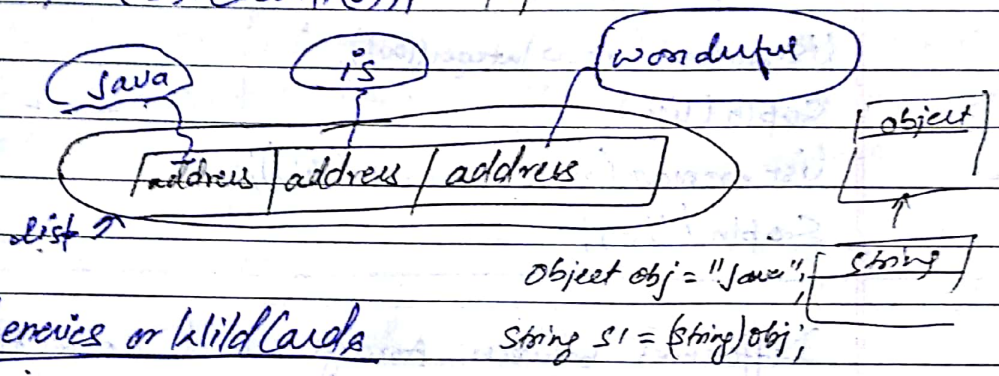
In collection all the elements will be stored as objects

```
class B
{ @override
    public String toString()
    { return "Gone"; }
}
public class Tester22
{ psvm (String[] args)
  { ArrayList a = new ArrayList();
    list.add(10); // Auto-Boxing and Auto-Upcasting
    list.add(10.5);  "   "   "   "
          ("Hey");  Auto-Upcasting
          (true);  Auto Boxing & Auto upcasting
          ('A');   "   "   "   "
          (90);    "   "   "   "
    List add(new B())  Auto upcasting
    Sopln(list.get(6));  Gone
     "   (list get(5))   90
                4        A
                3        true
                2        Hey
                1        10.5
                0        10
```

add / generic method



```
PSVM (String[] args)
{ ArrayList list = new ArrayList()
  list.add("Java"); list.add("is"); list.add("wonderful");
// sopln(list.get(0).toLowerCase()); // → CTE
// sopln(list.get(0).length()); // → CTE
// sopln(list.get(0).charAt()); // CTE
// Due to upcasting all the subclass (String class) features are hidden
String s1 = (String) list.get(0);   // Down casting to String type
Sopln(s1.toUpperCase());            JAVA
String s2 = (String) list.get(1);   // Down casting to String type
Sopln(s2.length());                 2
String s3 = (String) list.get(2);   // Down casting to String type
Sopln(s3.charAt(6));                f
}
```



Generics or WildCards

```
class WebElements
{ void test()
  { sopln("test()"); }
public class Tester21
{ public static void main(String[] args)
{ ArrayList<String> list1 = new ArrayList<String>();
  list.add("Java");
  list.add("is");
  list.add("Awesome");
Sopln(list1.get(0).toUpperCase());      JAVA
sopln(list1.get(0).length());           4
Sopln(list1.get(0).charAt(6));          e
ArrayList<Integer> list2 = new ArrayList<Integer>();
    list.add(10);
    list.add(20);
    list.add(30);
  Sopln
ArrayList<WebElements> list3 = new ArrayList<WebElements>
        list3.add(new WebElements);
```

Remove () method Overloading

```
remove () (int index)
{ search index
  if (not found
  Runtime Exception
  if found
  remove
{
```

```
PSVM (String[] args)
{ Arraylist list = new Arraylist();
  list.add (100); list.add ('A'); list.add ("hey"); list.add (90.5);
  Sopln(list)
// list.remove (100);
// list.remove ('A');
list.remove (new Integer(100));
Sopln (list);
list.remove (new character('A'));
Sopln (list);
{
```

Differences between Arrays and collection

①	size is fixed	size is dynamic
②	Can store only homogenous element	can store hetegenous element
③	less built in functionalities	More built in functionalities
④	We can store both primitive data and object	we can store only objects
⑤	Data structures are not used	Data structure is used
⑥	performance is good & faster	performance is low

Differences b/w Arrays and Collection

①	Can store heterogenous element*	can store heterogenous element
②	Can store duplicates	Cannot store duplicates
③	Cannot store null	can store null
④	Two classes inherit from Queue ① priority Queue ② Linked list	Three classes implements - Hashset - Linked Hashset - Treeset
⑤	Peek & poll method	Iterator

Date
Page

difference b/w ~~Array~~List and ~~Hash~~ Map

→ It is collection sub interface	Its is a separate implementation
→ List stores the element value alone and internally maintains the index for each element	Map stores key value pair Each value must be associated with key in Map
→ Allow duplicate elements	It doesnot allow duplicate element for key . for value it accepts duplicate element
→ List can have any number of null element	map allows one null key and number of null values
→ Its indexbased . Element can be retrieved using get() method using index	Its not a index based Element can be retrieved using key value

Difference b/w ~~Arraylist and~~ set And Map

→ Set implementation comes under collection Interface	Map implementation is separate
→ Set stores only Element	Map stores data in form of key value pair
→ add() method is used to add elements in set	put method is used to add element in map
→ Duplicate is not allowed in set	Duplicate is not allowed in map for key but Value can be duplicate
→ Set is slower than map	Map is faster than set because unique key is used to access object