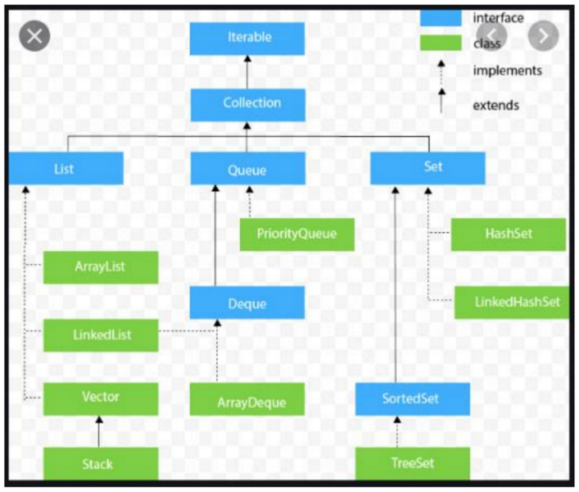
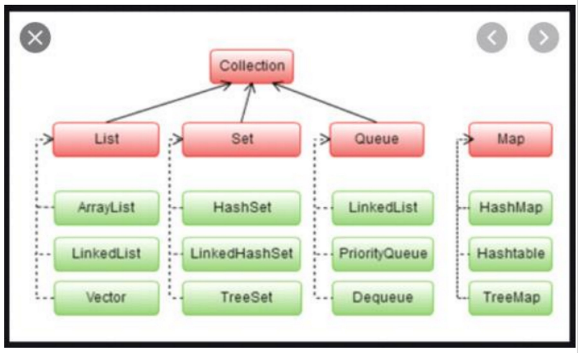
Collection Notes





Disadvantage of Array:

- Whenever we use array, it cannot grow the size dynamically
- While storing the data in the array, index to be provided by the developer
- We cannot instantiate the array without size. To overcome these disadvantages Collection framework is introduced in Java.

Collection Framework:

- Parent in the collection framework is Iterable.
- Collection framework is used to store the group of objects together
- Collection is an interface.
- Collections is class
- **List Interface**: This List interface has many abstract methods. If we try to implement this interface we need to override or provide the logic for all the abstract method. Since we don't know this, Java creates have created some class called Array List, Vector which implements List interface.

Array List:

- ArrayList extends AbstractList which implements List interface If you print any collection it will print values present in the collection because it has overridden the toString() method Where as Array cannot print
- Internal Data Structure for Array List is Object[]
- Array List accepts Duplicates
- Default capacity of AL is 10
- What is the data structure used by ArrayList is Array (Object Array)
- There is a variable present in Arraylist class and it creates the Array list with this

capacity when we create the object or when we call the constructor

- Array List allows duplicates
- Array list is not synchronized.
- Array List is Ordered

- **What happens inside add method or What is the internal implementation of add method ?**
  - AL internally uses the Object array (Object[]) First it checks for the capacity If is reached then it will increment the capacity by 50% on old capacity Later it does the copy of the elements using Arrays.copy of method
  - If the capacity is not reached then directly it will add the elements
- **What internally happens when remove(2) is called or what is the internal implementation of remove method ?**
  - The argument that the remove method accepts is index. . First AL checks whether the index is present by validating the passed index is negative or greater than or equal t array size or not, if not present then throws Array index out of bound exception if not get the element present in the index and then it removes from that position and re arrange the AL and returns the removed element
  - During the remove method, left shift operation will be performed. Internally it uses System.arraycopy method to copy the data and perform the left shift operation
  - Since the left shift operation is performed, to remove the empty space we should reduce the index in remove method

- **How to convert Array to AL ?**
  - Using iteration or Arrays.asList(); After performing the Arrays.asList we cannot add any new element because it is not creating the AL. To solve this List l= new ArrayList(Arrays.asList()); This statement will call the single arguement constructor of AL which accepts collection object

- **How to create Custom ArrayList ?**
  - Create the class and create the constructor. Since the internal Data structure of array list is Object[], create the instance of Object[] and then initialize it in the constructor
  - Have the variables for default value, index
  - Define the array, because AL internally uses Array List
  - Create the Default constructor and initialize the array

  Create Add method
  - In Add method check the capacity by comparing index and the default capacity or length of Object[]
  - If index reached to default capacity then increment the size by 50% and do array copy
  - Else Add the element and increment the index.
  - Create another class and instead of using the AL use custom arraylist

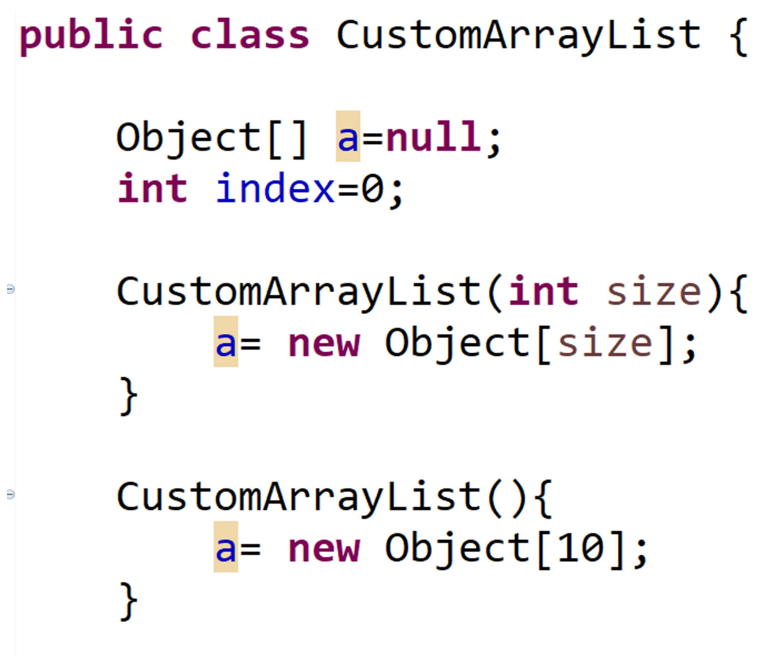
Create the remove method:

- Check if the indextobedeleted should not be negative and should not be greater than or equal to array size.
- If yes throw the exception.
- Find the number of shift operations (no of Shift operations =Array.lengh-indextoBedeleted-1)
- If the no of shift operations>0 then using the System.arrayCopy method copy the data and do shift operation

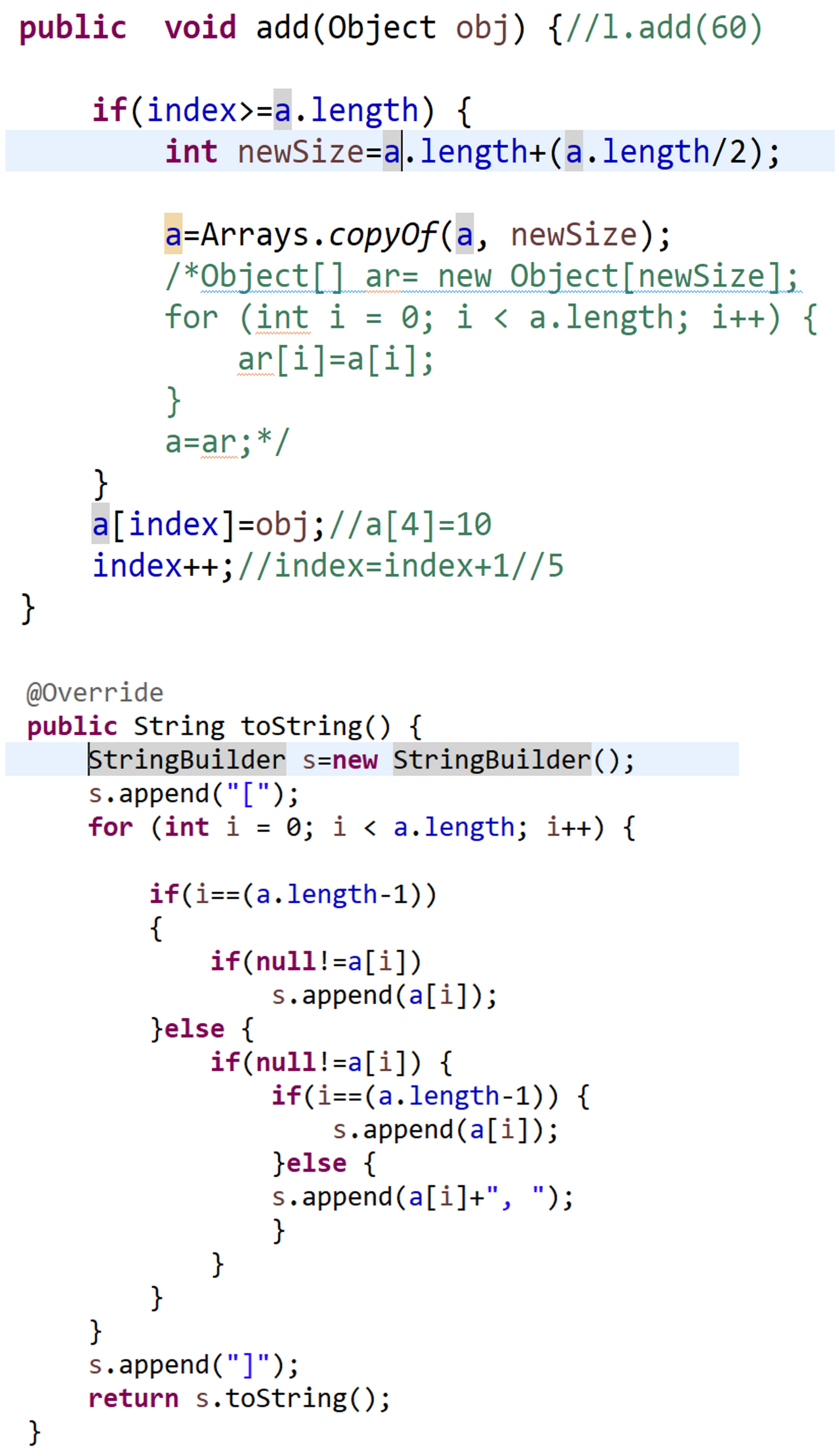
Create the Get Method:

- Check if the index passed to the get method should not be negative and should not be greater than or equal to array size.
- If yes throw the exception.
- If not get the data from internal DS Object[] and return the data

Custom ArrayList code snippet



```java
public class CustomArrayList {

    Object[] a=null;
    int index=0;

    CustomArrayList(int size){
        a= new Object[size];
    }

    CustomArrayList(){
        a= new Object[10];
    }
```



```java
public  void add(Object obj) {//l.add(60)

    if(index>=a.length) {
        int newSize=a.length+(a.length/2);

        a=Arrays.copyOf(a, newSize);
        /*Object[] ar= new Object[newSize];
        for (int i = 0; i < a.length; i++) {
            ar[i]=a[i];
        }
        a=ar;*/
    }
    a[index]=obj;//a[4]=10
    index++;//index=index+1//5
}

 @Override
 public String toString() {
     StringBuilder s=new StringBuilder();
     s.append("[");
     for (int i = 0; i < a.length; i++) {

         if(i==(a.length-1))
         {
             if(null!=a[i])
                 s.append(a[i]);
         }else {
             if(null!=a[i]) {
                 if(i==(a.length-1)) {
                     s.append(a[i]);
                 }else {
                 s.append(a[i]+", ");
                 }
             }
         }
     }
     s.append("]");
     return s.toString();
 }
```

Vector

- In AL increase capacity by 50 % of the existing capacity where Vector will increment the capacity by 100%
- Vector is synchronized and AL is not synchronized.
- Vector is also Ordered and allows duplicates

HashMap

- HashMap is a class which implements from Map interface
- **Important note is Map doesn't implement from Collection Interface**

➢ **What is the internal data structure of hash Map ?**
  - Node[]

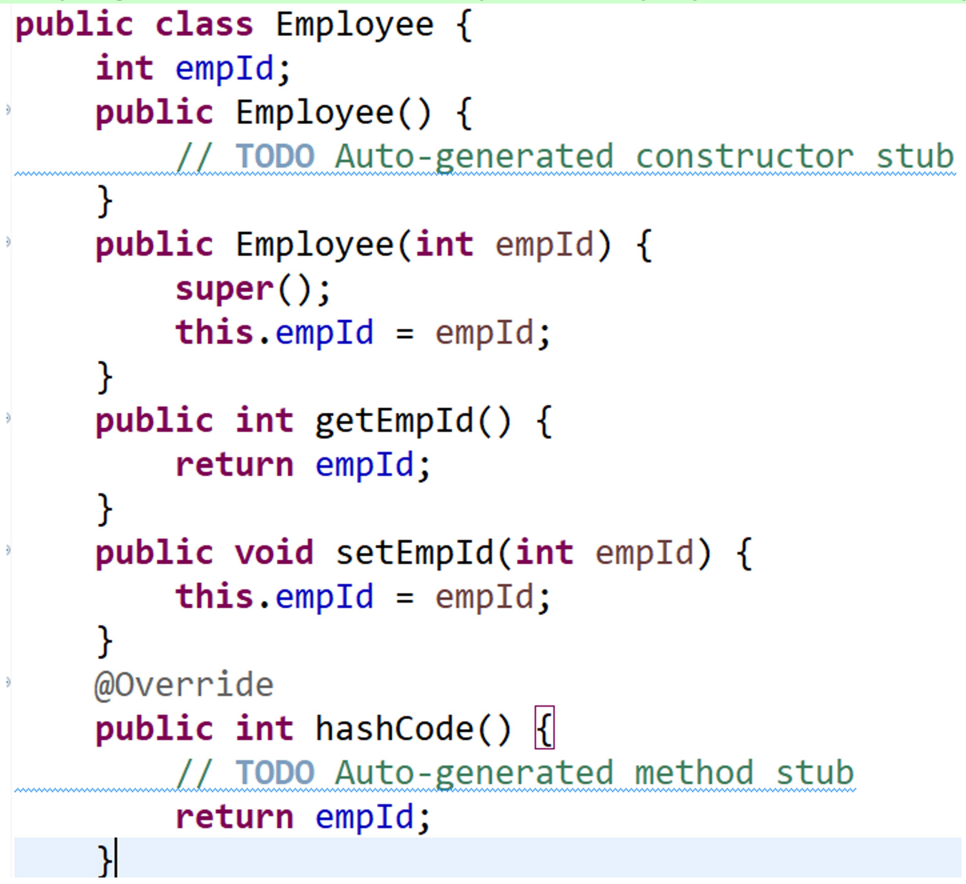
➢ **How does the hash map works internally ? Or what is the implementation of put method ?**
  - Put method accepts 2 arguments, first object is called as key object and second one is called as Value object. Since we cannot store 2 values, internally hash map has class called Node.
  - First find out hash of the key
  - Then divide it by the 16 buckets which gives the index
  - Now at this index store the element in the form of Node by calling the internal Node class constructor
  - Node will contain hash, key , value and address of next node
  - If it happens that index of 2 keys are same then it is referred as collision
  - Example: Hashmap.put("A", 64)
    Hashmap.put("B", 64)
    Here in this example, Keys are different but when we calculate the hashcode hashcode for A is 65
    Hash code for B is 66, these both will result index as 4, this is called as collision
  - In case collision hash map follows linked list approach
  - At index first node will contain the address next node
  - Hash Map doesn't allow duplicates keys,

➢ **How does the hash map identify that the key provided by the developer is duplicate ?**
  - Hash map internally uses has code and equals method, Using these method it identifies that the key is duplicate. If the hash map key is user defined object like Employee or Department then we need to override the equals and hashcode method to tell hashmap that derive the duplicate check based on the implementation of these has code and equals method

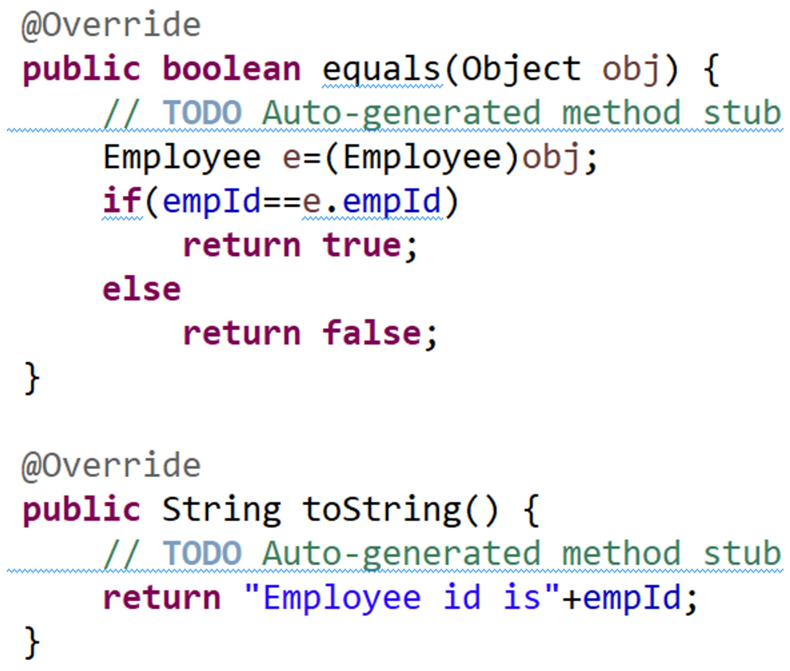
➢ **What is the internal implementation of get method of Hash Map ?**
  - first system finds the hash of the key then find out index by dividing the hash /16
  - Now it compares the hash at the index.
  - If there are multiple nodes present then it checks hash of each and every node until it is matched Once it is matched it checks the key.
  - If key also matches the retrieves the value and returns it

➢ **What is the performance issue in Hash Map ?**
  - After calculating the index it may happens that multiple indexes may get stored at a specific index.
  - If we try to retrieve the value then it has to compare with all the nodes present at that index. This is termed as performance issue in Hash Map.
  - To get rid of this B-tree implementation is used from Java 1.8 onwards

- **Write a program to not to allow duplicate employees into hashmap ?**

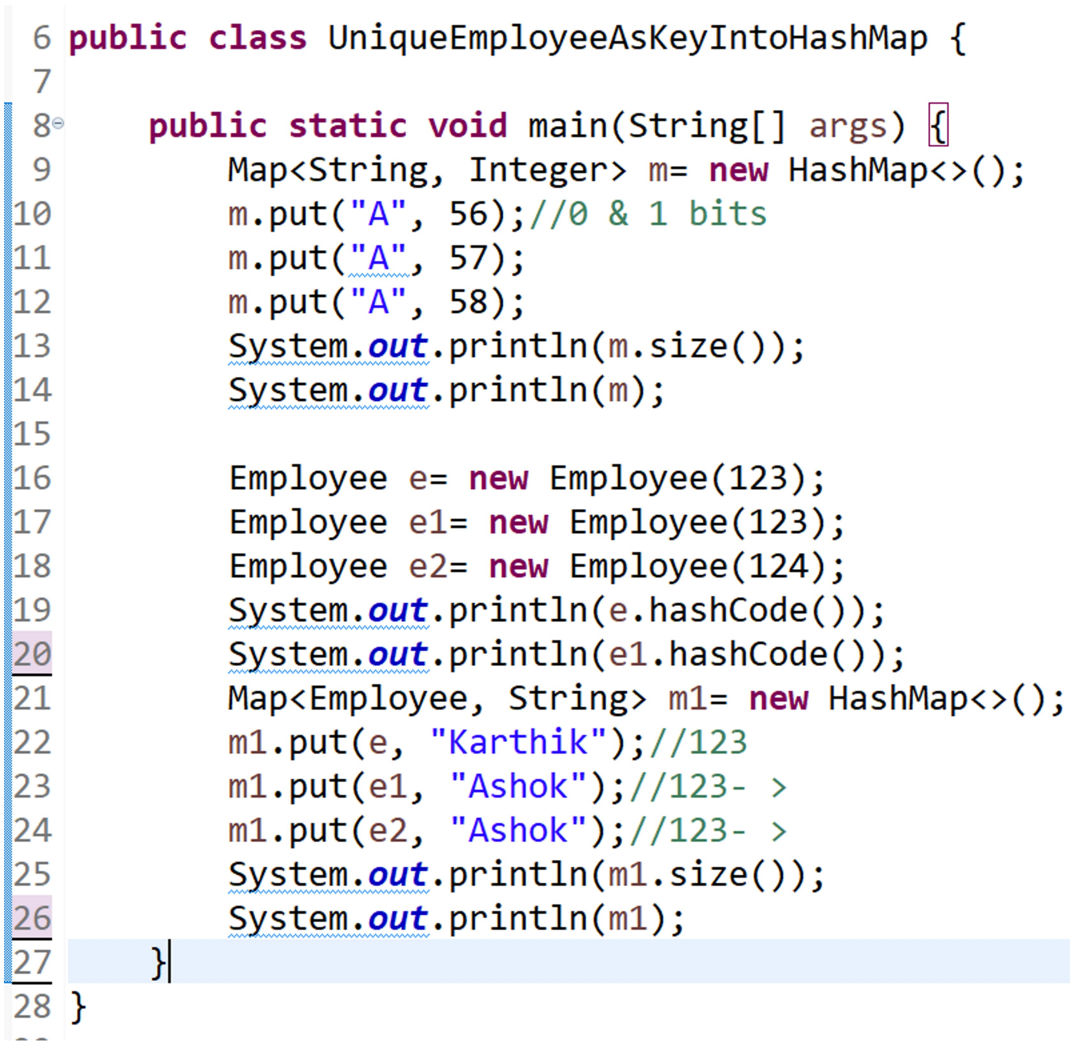


```java
public class Employee {
    int empId;
    public Employee() {
        // TODO Auto-generated constructor stub
    }
    public Employee(int empId) {
        super();
        this.empId = empId;
    }
    public int getEmpId() {
        return empId;
    }
    public void setEmpId(int empId) {
        this.empId = empId;
    }
    @Override
    public int hashCode() {
        // TODO Auto-generated method stub
        return empId;
    }
```



```java
@Override
public boolean equals(Object obj) {
    // TODO Auto-generated method stub
    Employee e=(Employee)obj;
    if(empId==e.empId)
        return true;
    else
        return false;
}

@Override
public String toString() {
    // TODO Auto-generated method stub
    return "Employee id is"+empId;
}
```

Override the hashcode and equals method like above and in the main class, when ever we call hash map put method system calls the hashcode and equals method and determines if it is duplicate then value will be overridden



```
 6 public class UniqueEmployeeAsKeyIntoHashMap {
 7
 8     public static void main(String[] args) {
 9         Map<String, Integer> m= new HashMap<>();
10         m.put("A", 56);//0 & 1 bits
11         m.put("A", 57);
12         m.put("A", 58);
13         System.out.println(m.size());
14         System.out.println(m);
15
16         Employee e= new Employee(123);
17         Employee e1= new Employee(123);
18         Employee e2= new Employee(124);
19         System.out.println(e.hashCode());
20         System.out.println(e1.hashCode());
21         Map<Employee, String> m1= new HashMap<>();
22         m1.put(e, "Karthik");//123
23         m1.put(e1, "Ashok");//123- >
24         m1.put(e2, "Ashok");//123- >
25         System.out.println(m1.size());
26         System.out.println(m1);
27     }
28 }
```

In the above program, output from line number 13 is size is 1 and 14 is [A,58] The key object is String object. In string class hash code and equals method is overridden and hence if we give same key value is over ridden Similarly, Above we implemented hash code and equals method in Employee class. If 2 employees has same emo id then it is considered as duplicate the same we mentioned in hash code and equals method so If you look at the above line number 25 output is 2 as e and e1 are 2 different object but the employee id passed to the object is same and as per hash code and equals implementation these are considered as duplicates.

- **Why does the hashmap doesn't allow duplicate keys ?**
  - HashMap internally calls equals and hashcode method due to which if the user provides the duplicate key then previous value is overridden by current value

- **What is meant by loading factor in Hash Map ?**
  - The Load factor is a measure that decides when to increase the HashMap capacity to maintain the get() and put() operation complexity of O(1).
  - The default load factor of HashMap is 0.75f (75% of the map size)
  - We insert the first element, the current load factor will be 1/16 = 0.0625. Check is 0.0625 > 0.75 ? The answer is No, therefore we don't increase the capacity.
  - Next we insert the second element, the current load factor will be 2/16 = 0.125. Check is 0.125 > 0.75 ? The answer is No, therefore we don't increase the capacity.
  - Similarly, for 3rd element, load factor = 3/16 = 0.1875 is not greater than 0.75, No change in the capacity.
  - 4th element, load factor = 4/16 = 0.25 is not greater than 0.75, No change in the capacity.
  - 5th element, load factor = 5/16 = 0.3125 is not greater than 0.75, No change in the capacity.

- 6th element, load factor = 6/16 = 0.375 is not greater than 0.75, No change in the capacity.
- 7th element, load factor = 7/16 = 0.4375 is not greater than 0.75, No change in the capacity.
- 8th element, load factor = 8/16 = 0.5 is not greater than 0.75, No change in the capacity.
- 9th element, load factor = 9/16 = 0.5625 is not greater than 0.75, No change in the capacity.
- 10th element, load factor = 10/16 = 0.625 is not greater than 0.75, No change in the capacity.
- 11th element, load factor = 11/16 = 0.6875 is not greater than 0.75, No change in the capacity.
- 12th element, load factor = 12/16 = 0.75 is equal to 0.75, still No change in the capacity.
- 13th element, load factor = 13/16 = 0.8125 is greater than 0.75, at the insertion of the 13th element we double the capacity. Now the capacity is 32.

Hash Set

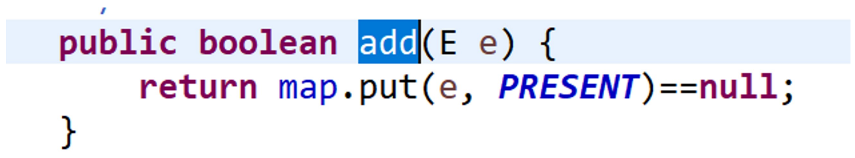
- Hash set is a class in Java and it implements from Set interface.
- Set internally implements from collection interface

➢ **What is the internal data structure of Hash Set ?**

- Hash Set internally uses the Hash Map structure
- Where the map gets initialized in (hashset constructor)
- Where the map gets initialized in (hashset constructor)
- Default capacity is 16 and loading factor 0.75

➢ **When the set is using the map internally, map needs key and value but in hash set when we use the add method to insert the element we pass only 1 object, will that be treated as Key or value ?**

- What ever we passed to the add method of hash set will be taken as key and value is (OBJECT)--you can find final Object PRESENT=new Object();



```
public boolean add(E e) {
    return map.put(e, PRESENT)==null;
}
```

➢ **why set doesn't allow duplicates ?**

- because it follows map structure and map doesn't allow duplicates
- Hashset or hashmap doesn’t guarantee the order Where linked hashmap and linked hash set guarantee the insertion order why because it uses double linked list approach and each node contain the address of next hence insertion order is preserved

➢ **Why Hash Set doesn't have get method ?**

- The purpose of get method is to retrieve the value present at that index in case of array list
- The purpose of get method is to retrieve the value for the passed key in case of HashMap
- Incase of hash set internal data structure is Hash Map and value for all the elements of Hash set is Object of Object class. So if we have the get method what ever we pass we get same value which is object of object class and we can get this object by creating it hence there is no get method in hash set.

➢ **What is meant by Synchronization ?**

- Array List, Hash Map and Hash set are not synchronized. What exactly synchronized means. If 2 threads are trying to perform some operation on the same object then while 1 threading working other thread should go into wait mode.
- ArrayList is not synchronized means multiple threads can work on same array list object at a time which may lead to inconsistent output where as vector is synchronized so when multiple threads are working on same vector object, out put is consistent.

➢ **What is Comparable, comparator and what are the differences between them ?**

- In Java, the Comparator and Comparable interfaces are both used for sorting and ordering objects, but they differ in their implementations and purposes.
- Collection.sort by default calls the comparable interface compare to method unless we provide the comparator to sort method

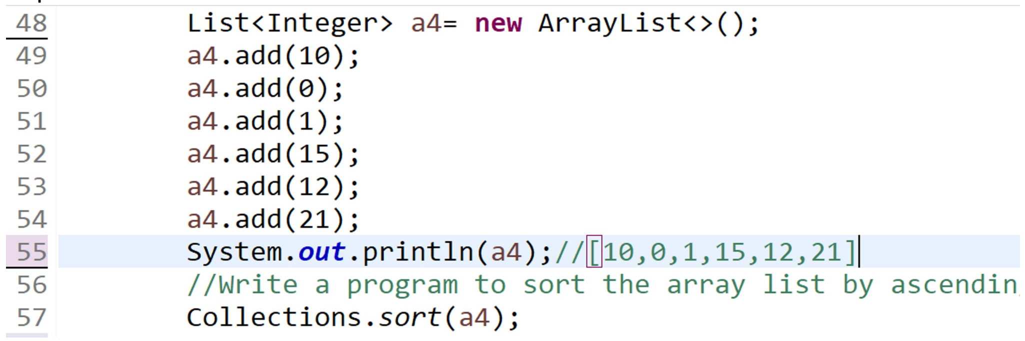
Comparable	**Comparator**
It is present in Java.lang package	It is present in Java.util package
It has abstract method **compareTo** which takes 1 arg	It has the abstract method **compare** which takes 2 args
By default all the wrapper classes implements comparable interface	User or developer should write the comparator to have specific logic

What is iterator ?

- We know that Iterable is the parent in the collection framework. Iterable has an abstract method called Iterator<T> iterator(); This iterator method return type is Iterator
- Iterator is an interface in java
- List interface in directly extends interable so the classes (Array List, Vector) should override this abstarct method iterator.
  In Java, an Iterator is an interface that provides a way to traverse and access elements of a collection in a sequential manner. It is part of the Java Collections Framework and is used to iterate over elements of various collection classes like List, Set, Queue, and Map.
  The Iterator interface provides three main methods:
  boolean hasNext(): This method returns true if there are more elements to iterate over, and false otherwise. It is commonly used in a loop to check if there are more elements before retrieving the next element.
  E next(): This method returns the next element in the collection. It moves the iterator forward to the next element and returns that element. If there are no more elements, it throws a NoSuchElementException.
  void remove(): This method removes the last element returned by the next() method from the underlying collection. It is an optional operation, and not all collections support it. If the remove() method is called without a preceding call to next(), or if it is called multiple times after a single call to next(), it will throw an IllegalStateException.

➢ What are the differences between iterator and List iterator ?

➢ Write a program to sort the array list by ascending

- Using collections.sort method we can sort the array List.
- Other technique is to use any of the sorting technique. Like merge sort, quick sort etc.
- Collections.sort method will sort the data by ascending order
- Sort method takes the List object as parameter so we can pass Array List, or Vector or any class that implements the List interface
- 

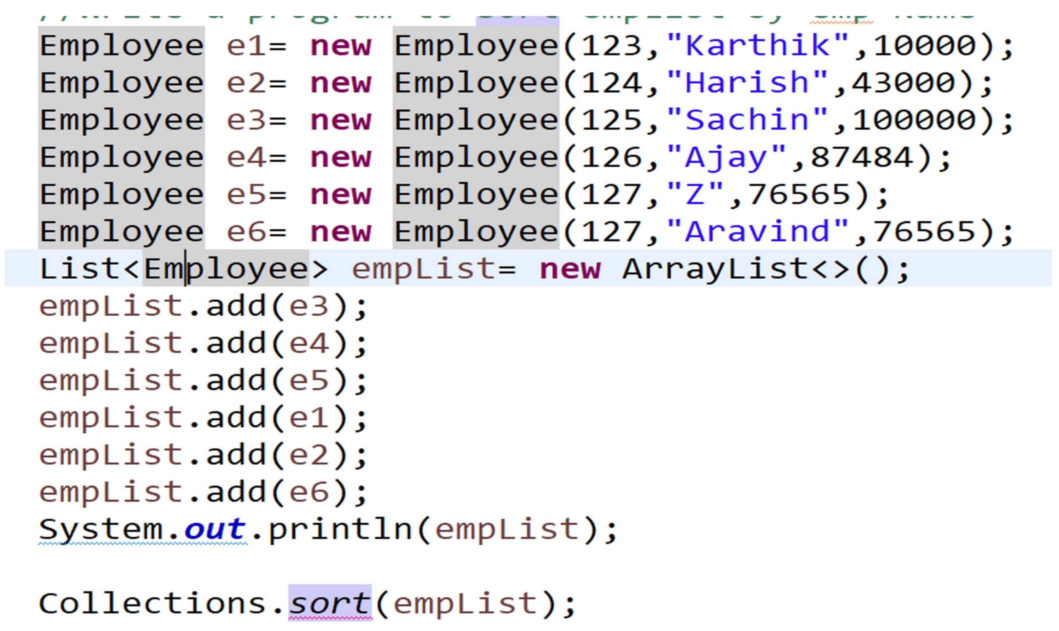
```java
        List<Integer> a4= new ArrayList<>();
        a4.add(10);
        a4.add(0);
        a4.add(1);
        a4.add(15);
        a4.add(12);
        a4.add(21);
        System.out.println(a4);//[10,0,1,15,12,21]
        //Write a program to sort the array list by ascendin
        Collections.sort(a4);
```

- In the above example, array List contains the integers so when Collections.sort method is called, Sort method checks that what type of data List is holding. Here in the above it is storing the data as Integer
- Integer class by default implements comparable interface so it has overridden the compareTo

method. By default implementation of compareTo implementation of Integer wrapper class is ascending order.

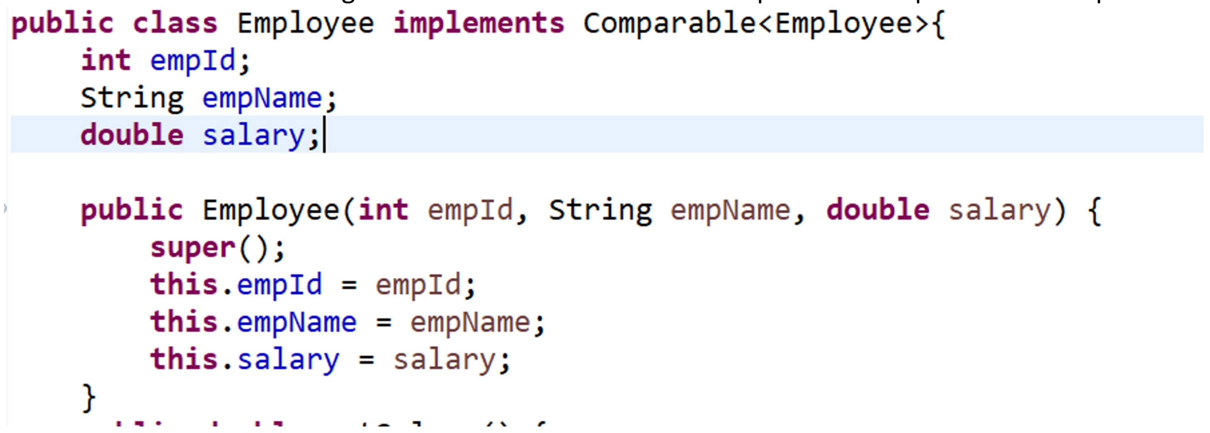
- So the working of sort method is that it check the generics supplied to the List should implement comparable or comparator. If not it will throw the compilation error.
- All the wrapper class implements the comparable interface. The default implementation of all the wrapper class compareTo method follows natural sorting order.

- **Sorting the Array List which contains the employee Object by Emp Id (or) Write a program to sort the employee by emp id**

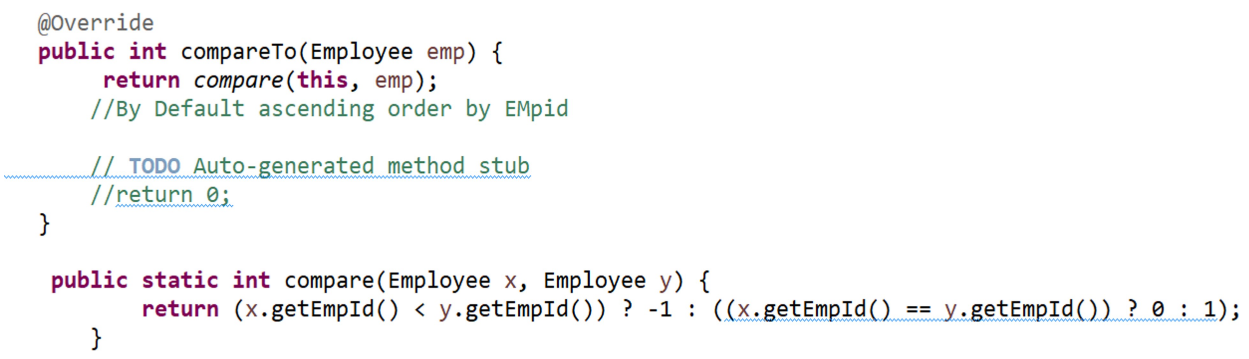


```
Employee e1= new Employee(123,"Karthik",10000);
Employee e2= new Employee(124,"Harish",43000);
Employee e3= new Employee(125,"Sachin",100000);
Employee e4= new Employee(126,"Ajay",87484);
Employee e5= new Employee(127,"Z",76565);
Employee e6= new Employee(127,"Aravind",76565);
List<Employee> empList= new ArrayList<>();
empList.add(e3);
empList.add(e4);
empList.add(e5);
empList.add(e1);
empList.add(e2);
empList.add(e6);
System.out.println(empList);

Collections.sort(empList);
```

In the above program, we can see the compilation error at sort method. Why the sort method shows the compilation error is because, when we use sort method, sort method accepts List interface so the generics used in the sort should implement comparable or comparator

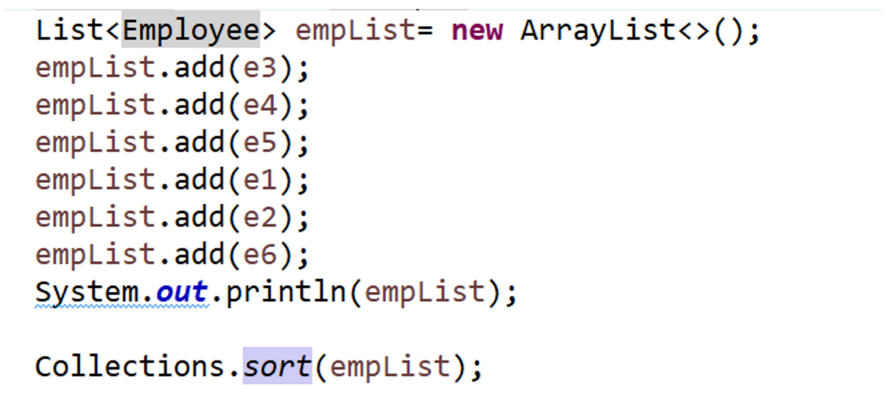


```
public class Employee implements Comparable<Employee>{
    int empId;
    String empName;
    double salary;

    public Employee(int empId, String empName, double salary) {
        super();
        this.empId = empId;
        this.empName = empName;
        this.salary = salary;
    }
```



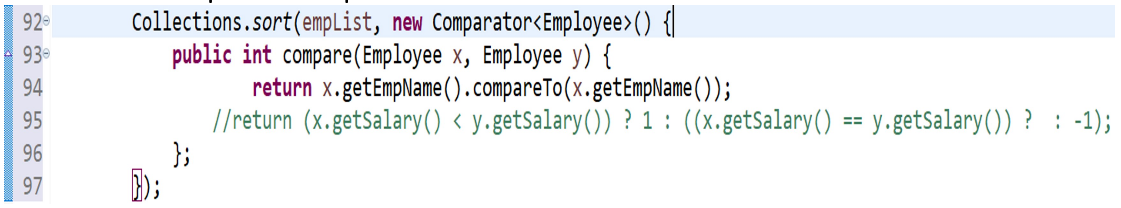
```
@Override
public int compareTo(Employee emp) {
     return compare(this, emp);
    //By Default ascending order by EMpid

    // TODO Auto-generated method stub
    //return 0;
}

 public static int compare(Employee x, Employee y) {
        return (x.getEmpId() < y.getEmpId()) ? -1 : ((x.getEmpId() == y.getEmpId()) ? 0 : 1);
    }
```

Look at above program, Since the generics is Employee, we have implemented the comparable interface and overridden compareTo method. The logic of compareTo method is that order by emp id so when the above program executes it sorts the employees by emp id



```
List<Employee> empList= new ArrayList<>();
empList.add(e3);
empList.add(e4);
empList.add(e5);
empList.add(e1);
empList.add(e2);
empList.add(e6);
System.out.println(empList);

Collections.sort(empList);
```

Compilation error is resolved.

- **Sorting the Array List which contains the employee Object by Emp Name (or) Write a program to sort the emplist by employee name**
    - Since the compareTo method in Employee class follows the sort by emp Id, to solve the above question of sort the employees by name we should not change the implementation of compareTo method because, if we change then where ever sort method is used that implementation will get change, to overcome this we should use separate comparator
    - 

```
        Collections.sort(empList, new Comparator<Employee>() {
            public int compare(Employee x, Employee y) {
                return x.getEmpName().compareTo(x.getEmpName());
              //return (x.getSalary() < y.getSalary()) ? 1 : ((x.getSalary() == y.getSalary()) ?  : -1);
            };
        });
```

In the above code comparator is created and using the overloaded sort method specific comparator is passed so that it sorts the empList by name. If we want to sort the employees by salary then the commented line in the above program will work.

- **What is Concurrent Modification Exception ?**
    - While iterating over the collection and performing the next method, if the user is trying to perform any structural modification then concurrent modification exception is thrown.
    - Map, Set and List whenever we try to iterate and trying add the element then these collection will throw the concurrent modification exception.
      1. l.add(10);
      l.add(25);
      Iterator it=l.iterator();
      While(it.hasNext())
      {
       Sysout(it.next());
      l.add(45)
      }
       Here in the above program, whenever we add the element to collection internally java uses the **modcount** variable and it gets incremented whenever we try to perform the add or remove operation so when we try to call next() method (look at next() method in ArrayList or any other collection) it checks for the earlier mod count and expected mod count, if both are not matching it throws the concurrent modification exception
      To get rid of this error we go for Concurrent hash map, If it is a array list then we go for Copy on write

array list
If it is a set then we go for copy on write array set

- **CopyOnWriteArrayList**
- In order to have the concurrency in the list we go for Copy on write ArrayList
- It has the lock in add and remove etc
- It uses the reentrant lock It doesn't throw concurrent modiofication exception because the iteration will happen on new array and not on the original array
- The design pattern followed in this is iterative design pattern (For Iteration separate collection, adding the data separate collection is used)

Linked List

- The major difference between Array list and Linked list regards to their structure.
- Arrays list are index based data structure where each element associated with an index.
- On the other hand, Linked list relies on references where each node consists of the data and the references to the previous and next element
- Basically, an array is a set of similar data objects stored in sequential memory locations under a common heading or a variable name.
- While a linked list is a data structure which contains a sequence of the elements where each element is linked to its next element.
- There are two fields in an element of linked list. One is Data field, and other is link field, Data field contains the actual value to be stored and processed. Furthermore, the link field holds the address of the next data item in the linked list

Difference between Array List and Linked List

Array List	**Linked List**
This class uses a dynamic array to store the elements in it. With the introduction of generics, this class supports the storage of all types of objects.	This class uses a link (Address of next node) store the elements in it. Similar to the ArrayList, this class also supports the storage of all types of objects.
Manipulating ArrayList takes more time due to the internal implementation. Whenever we remove an element, internally, the array is traversed and the memory bits are shifted.	Manipulating Linked List takes less time compared to ArrayList because, there is no concept of shifting the memory bits. The list is traversed and the reference link is changed
This class works better when the application demands storing the data and accessing it. Adding data at the end is easy, removing or adding data in middle is time consuming which involves left shift operation	This class works better when the application demands manipulation of the stored data. Example: deleting or adding in middle or last etc

- If at all we have to create the user defined linked list or custom linked list then we should use the static class to define the data and the next node address like below

```
class CustomNode{
    Object data;
    CustomNode next;
    int customNodeIndex;
    CustomNode(Object data,int in){
        this.data=data;
        this.customNodeIndex=in;
    }
}
```

- How to Add or insert element to custom linked list:
  - In order to perform the insert first create the object to Node class by passing the value (it is like calling the single argument constructor) then take the head node and check if it is null , for the first

    time it will be null so assign the above created node object
    - For the next element onwards to insert the node, iterate the Node till you reach the null and then assign the element (i.e -> when you get node.next is null that means there is no next element since the next is null and it is not pointing to any other node)
    - (because in Linked list every node will contain the address of next node, when the node.next is null means there is no next node and this is the right place to insert the new node)
- How to Count the Nodes ?
    - In order to count the number of nodes, Take a variable and initialize it to 0, and increment it until you get node.next is null. (because in Linked list every node will contain the address of next node, when the node.next is null means there is no next node)
- How to get the element in Custom Linked List ?
    - Start the logic from first node, and compare the index that is passed in get method with the index stored in the Linked List. By taking the next node repeat the logic until you get the value same. When the node.next is null and still if we didn't find the element means, the object passed in the get method is not present in the Linked list
- How to delete the Node To delete the node ?
    - user has to provide which node to be deleted, i.e to the delete method user will mentioned which object he wants to delete Iterate over the linked link till you reach Node.next ==null then find it, each time when this logic is performed, take the value of previous node and keep it in temp variable and when the user entered node matches with the node to be deleted then change the reference like. Previous.next = n.next
- How to add the node before Head ? Please look at the program
- How to add the node in middle or at a specific position ? Please look at the program
- How to remove the node which is at head or middle or end ? Please look at the program