Collections:

The Java collections framework is a set of classes and interfaces that implement commonly reusable collection data structures.

The Collection is a framework that provides an architecture to store and manipulate the group of objects.

Java Collection means a single unit of objects. Java Collection framework provides many interfaces Set, List, Queue, Deque and classes are ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet.

It is like a container which we can store and manipulate the data in a efficient manner that represents a single unit of objects.

ArrayList:

ArrayList class uses a dynamic array for storing the elements. It is like an array, but there is no size limit. We can add or remove elements anytime. So, it is much more flexible than the traditional array.

```
class Name{
        static String name;
        static ArrayList<String> Name=new ArrayList<>();
        public static void getName(int getName){
    System.out.println(Name.get(getName));
  }
        public void setEmployeeName(String Name){
                name=Name;
        }
        public static void deleteName(int deleteName){
                Name.remove(deleteName);
        }
        public static int validEmployee(String validName){
                return Name.indexOf(validName);
        }
        public static void updateEmployee(String oldName,String newName) {
                Name.set(Name.indexOf(oldName), newName);
        }
        public static boolean EmployeeName(String Name){
                if(Pattern.matches("[a-zA-Z ]{3,40}",Name)){
                        int word=Name.charAt(0);
                        for(int index=1;index<Name.length();index++){
                                if(Name.charAt(index)!=word){
                                        return true;
                                }
                        }
                }
```

```
                return false;
        }
        public static void employeeName(){
                Scanner scanner=new Scanner(System.in);
                while(true){
                        System.out.println("Employee Name:");
                        String EmployeeName=scanner.nextLine();
                        if(EmployeeName(EmployeeName)){
                                Name.add(EmployeeName);
                                break;
                        }
                        else{
                                System.out.println("The Employee name should contain only
alphabets and do not include special characters or numeric values.");
                        }
                }
        }
}
```

Linked list:

Linked List class uses a doubly linked list to store the elements.
It provides a linked-list data structure.
It inherits the Abstract List class and implements List and Deque interfaces.

```
class EmailId extends Name{
        static String emailId;
   static LinkedList<String> emailid=new LinkedList<>();
   public static void getEmailid(int getEmail){
       System.out.println(emailid.get(getEmail));
   }
   public void setEmployeeEmailid(String setEmailid){
                emailId=setEmailid;
        }
   public static void deleteEmailid(int deleteEmail){
      emailid.remove(deleteEmail);
   }
        public static boolean Emailid(String email)
        {
                String emailRegex = "^[a-zA-Z_+&-]+(?:\\."+"[a-zA-Z_+&-]+)*@" +"(?:[a-zA-Z-
]+\\.)+[a-z" +"A-Z]{2,7}$";
                Pattern pattern = Pattern.compile(emailRegex);
                if (email == null)
                        return false;
                return pattern.matcher(email).matches();
        }
        public static int validEmailid(String oldId){
           return emailid.indexOf(oldId);
        }
        public static void updateEmailid(String oldId,String newId){
           emailid.set(emailid.indexOf(oldId),newId);
        }
```

```
        public static void emailId()
        {
           Scanner scanner=new Scanner(System.in);
           while(true){
              System.out.println("Enter the email Id: ");
                    String email =scanner.next();
                    if (Emailid(email)){
                       if(emailid.indexOf(email)==-1){
                       emailid.add(email);
                             break;
                       }
                       else{
                          System.out.println("Same id should not repeated.Enter different id:");
                       }
                    }
                    else
                             System.out.println("Enter the valid email id like
username@domainname.com and Domain name should contain only alphabets.");
              }
        }
}
```

HashSet

It used to create a collection that uses a hash table for storage

Hashing is the mechanism by which hashset stores the element

It is used if we want to access elements randomly or store a list of items which cannot contain duplicate values

Example

```
Import java.util.*

Class Book{

Int id;

String name,author,publisher;

Int quantity

Public Book(int id ,String name,String author,String publisher,int quantity){

this.id=id;

this.name=name;

this.author=author;

this.publisher=publisher;

this.quantity=quantity;

}
```

}

Public class demo{

Public static void main(String[] args){

HashSet<Book> set=new Hashset<Book>

Book b1=new Book(101,"Let us C","Pavithra");

Book b2=new Book(102,"Data Communications & Networking")

Set.add(b1);

Set.add(b2);

For(Book b:set){

System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);

LinkedHashSet

```
import java.util.*;
public class demo{


    public static void main(String args[])
    {

        LinkedHashSet<String, String> lhm
            = new LinkedHashSet<String, String>();

        lhm.put("one");
        lhm.put("two");
        lhm.put("four");

        System.out.println(lhm);



                System.out.println("Getting value for key 'one': "
                           + lhm.get("one"));


        System.out.println("Size of the map: "
                           + lhm.size());

        System.out.println("Is map empty? "
                           + lhm.isEmpty());

        System.out.println("Contains key 'two'? "
                           + lhm.containsKey("two"));

        System.out.println(
```

```
        System.out.println("delete element 'one': "
                           + lhm.remove("one"));

        System.out.println("Mappings of LinkedHashMap : "
                           + lhm);
    }
}
```

TreeSet

```
import java.util.*;

Class demo{


        public static void main(String[] args)
        {

                Set<String> ts1 = new TreeSet<>();



                ts1.add("A");
                ts1.add("B");
                ts1.add("C");
                 ts1.add("C");
)               System.out.println(ts1);
        }
}
```

Queue:
Queue interface orders the element in FIFO(First In First Out) manner.
In FIFO, first element is removed first and last element is removed at last.
Priority queue:
The Priority Queue class provides the facility of using queue.
But it does not orders the elements in FIFO manner.
It inherits Abstract Queue class.

```
import java.util.Queue;
import java.util.PriorityQueue;
class Main {
   public static void main(String[] args) {
      Queue<Integer> numbers = new PriorityQueue<>();
      numbers.offer(5);
      numbers.offer(1);
      numbers.offer(2);
      System.out.println("Queue: " + numbers);
      int accessedNumber = numbers.peek();
      System.out.println("Accessed Element: " + accessedNumber);
      int removedNumber = numbers.poll();
      System.out.println("Removed Element: " + removedNumber);
      System.out.println("Updated Queue: " + numbers);
   }
}
```

Deque:

Deque Interface is a linear collection that supports element insertion and removal at both ends.
It has two types:
Array Deque
Linked list

Array Deque:

The ArrayDeque class provides the facility of using deque and resizable-array. It inherits AbstractCollection class and implements the Deque interface.

```
import java.util.Deque;
import java.util.ArrayDeque;
class Main {
   public static void main(String[] args) {
      Deque<Integer> numbers = new ArrayDeque<>();
      numbers.offer(1);
      numbers.offerLast(2);
      numbers.offerFirst(3);
      System.out.println("Deque: " + numbers);
      int firstElement = numbers.peekFirst();
      System.out.println("First Element: " + firstElement);
      int lastElement = numbers.peekLast();
      System.out.println("Last Element: " + lastElement);
      int removedNumber1 = numbers.pollFirst();
      System.out.println("Removed First Element: " + removedNumber1);
      int removedNumber2 = numbers.pollLast();
      System.out.println("Removed Last Element: " + removedNumber2);
      System.out.println("Updated Deque: " + numbers);
   }
}
```