**Collection** is a framework in Java where we can manipulate the group of Objects. We can perform operations such as Sorting, Searching, Deleting and Inserting objects.

**Interfaces: - Set, List, Queue, Dqueue etc.**

**Classes: - ArrayList, Vector, LinkedList, HashTable, HashSet etc.**

**Iterable Interface**

**Represent Interfaces**

**implements**

**extends**

**SortedSet Interface**

**implements**

**implements**

**implements**

**DQueue Interface**

**implements**

**Set Interface**

**Queue Interface**

**List Interface**

**Collection Interface**

**Structure of Collections**

**ArrayList: -** In ArrayList we can create and manipulate dynamic array, where we can store “n” no. of. Elements.

1. It can contain duplicate values.
2. It maintains insertion order.
3. It is not synchronized.
4. It allows random access to fetch an element.

**ArrayList ar = new ArrayList ();**

**Ar.add (10); Ar.add (20); Ar.add (30);**

**System.out.println (“Size Is: “+ar.size()); // Will return 3**

**//To fetch an elements from ArrayList**

**For(int i=0 i<ar.size (); i++)**

**{**

**System.out.println (ar.get(i)); //Will print 10, 20, 30**

**}**

**We can use Generics also to create an ArrayList: -**

**ArrayList<Integer> ar = new ArrayList<Integer> (); //Will create only Integer ArrayList**

**ArrayList<String> ar = new ArrayList<String> (); //Will create only String ArrayList**

**ArrayList<E> ar = new ArrayList<E> (); //Will create and store any type of data in ArrayList**

**ArrayList Example: -**

Public class **Employee**

**{**

**String name;**

**Int age;**

**String dept;**

**Public Employee (String name, int age, String dept)**

**{**

**This.name = name;**

**This. Int = int;**

**This.dept = dept;**

**}**

**}**

**Employee e1 = new Employee (“Akshay”, 1, “Testing”);**

**Employee e2 = new Employee (“Pradip”, 2, “Developer”);**

**Employee e3 = new Employee (“Shete”, 3, “BA”);**

**ArrayList<Employee> a1 = new ArrayList<Employee> ();**

**A1.add (e1); a1.add (e2); a1.add (e3); //Add Employee Objects into ArrayList**

**//Using Iterator we can traverse each object in ArrayList**

**Iterator<Employee> it = a1.iterator ();**

**While (it.hasNext ())**

**{**

**Employee e = it.next ();**

**System.out.println €;**

**}**

**Few Methods of ArrayList are: -**

**retainAll ();**

**removeAll ();**

**addAll ();**

**LinkedList: -** It is a class in Collection which implements 2 interfaces i.e. **List and DQueue**.

**LinkedList<String> ll = new LinkedList<String> ();**

**ll.add (“Akshay”); ll.add(“Pradip”); ll.add(“Shete”);**

**//To print LinkedList**

**System.out.println (ll);**

**//OR**

**For (String s : ll)**

**{**

**System.out.println(s);**

**}**

**Few Methods of LinkedList are: -**

**addFirst ();**

**addLast ();**

**get ();**

**set (index, element);**

**getFirst ();**

**getLast ();**

**removeFirst ();**

**removeLast ();**

**Map Interface: -** It stores value in **KEY- > VALUE** format. **Map<Key,Value> OR Map<K,V> KEY** should be always unique.

**Examples of Map:** - 1) To store Error Code 2) To store ZIP Code 3) To store config/environmental variables.

**Methods of Map:** - put (), get (key), putAll (), containsKey (key), remove (key), entrySet () etc.

**extends**

**MAP Interface**

**SortedMap Interface**

**implements**

**implements**

**extends**

**SortedMap Interface**

**HashMap:** - It stores Object/Elements in KEY, VALUE pair. It implements Map interface. It contains Unique elements. HashMap is not synchronized. It maintains no order. It can allow only 1 **null** key but many **null** values.

**Public static void main (String[] args)**

**{**

**HashMap<Integer, String> hm = new HashMap<Integer, String> ();**

**Hm.put (1, “Akshay); hm.put (2, “Pradip”); hm.put (3, “Shete”);**

**//To print HashMap**

**For (Entry<Integer, String> m : hm.entrySet ())**

**{**

**System.out.println (m.getKey () +” “+ m.getValue ());**

**}**

**//OR**

**System.out.println (hm);**

**HashTable: -** It is synchronized i.e. it is thread safe. It also stores Objects/Elements in **KEY-VALUE** pair. It implements Map interface. It stores key in the form of HashCode. **KEY-VALUE** pair must be unique. No **null** values in **KEY-VALUE** pair is acceptable/allowed.

**HashTable hm = new HashTable ();**

**Hm.put (1, “Akshay); hm.put (2, “Pradip”); hm.put (3, “Shete”);**

**//To print HashTable used Enumeration interface**

**Enumeration e = hm.elements ();**

**While (e.hasMoreElements ())**

**{**

**System.out.println (e.nextElement ());**

**}**

**// OR**

**Set s = hm.entrySet ();**

**System.out.println (s);**

**Set interface does not allow duplicate elements. Set cannot be accessed using indexes.**

**TreeSet always add elements in sorted form and dosen’t allow null values. It implements SortedSet interface.**

**TreeMap never allows null as Key. If KEY is duplicate it overwrites respected value.**

**Uses of Collection: -**

1. **Ready to use Classes**
2. **Better program speed**
3. **Reduced programming efforts**

**Java Additional Points: -** A regular expression is a sequence of characters that forms a search pattern. It is mainly used in pattern matching for searching and editing.

**[]** -> [xyz] -> Matches x | y | z

[a-z] -> Matches from a-z

**.** -> b.t -> Matches anything like bat, bbt, b8t etc.

**[^]** -> [^xyz] -> Matches anything except x, y, z

**()** -> Ma(nn|tt)er -> Matches either Manner or Matter

\**n** -> [(a-z)]\n -> Matches dd, hh etc.

**?** -> Zero/One -> Ba?it matches Bait, Bit etc.

**Star** (\*) -> Zero or More -> 10\*1 matches 11,101,1001 etc.

**+** -> One or More -> 10+1 matches 101, 1001, 100001 etc.

**{m}** -> Exactly ‘m’ times -> 10{4}1 -> 100001

**{m,n}** -> Atleast ‘m’ times and Atmost ‘n’ times -> 10{3,5}1 matches 10001, 1000001 etc.

**{m,}** -> Atleast ‘m’ times and maximum any times -> 10{2,} -> 100, 10000000000000001 etc.

**\w** -> Alphanumeric Characters and Underscore -> [A-Za-z0-9\_]

**\W** -> Non-Word Characters -> [^A-Za-z0-9\_]

**\d** -> Digits -> [0-9]

**\D** -> Non-Digits -> [^0-9]

**\s** -> White Space Characters -> [\t\n\f\r]

**\S** -> Non-White Space Characters -> [^\t\n\f\r]

Using with **\\ for ex. [\\w]**

**Matches: -** Check whether sequence matches or not.

**Compile (String regex)**: - returns the instance of regex.

**Pattern p = Pattern.compile (“\\d{3}-\\d{3}-\\d{4}”);**

**Matcher m = p.matcher(mno); //Where String mno = 123-123-1234**

**Return m.matches (); //It will return either true or false**

**For name pattern would be: - ([A-Za-z]+\\s?)+**

An annotation is a metadata that provides information about program and is not a part of program itself.

A stream is a sequence of data that flows from source to destination.

JDBC makes it very easy to connect to DB and perform database related operations. It consists various interfaces and classes.

**A driver act as an instance which will act like a translator between 2 parties.**

1. **Load the driver**
2. **Make connection**
3. **Send SQL queries**
4. **Process the result**

**For Ex: -**

**Class.forName (oracle.jdbc.driver.OracleDriver); //Load The Driver**

**Connection conn = DriverManager.getConnection(jdbc:oracle:thin:@hostname:port/dbname,username,password); //Make Connection**

**Statement stmt = conn.createStatement(); //Send SQL Queries**

**ResultSet rs = stmt.executeQuery(“SQL Query”); //Process The Result**

**OR**

**PreparedStatement psmt = conn.prepareStatement(“SQL Query”); //PrepareStatements are pre-compiled SQL Queries. It is an interface.**

**ResultSet rs = psmt.executeQuery();**

**PreparedStatement psmt = conn.prepareStatement(“insert into student values (?,?,?,?)”);**

**Psmt.setInt(1,100); psmt.setString(2,”Akshay”); etc.**

**Int n = psmt.executeUpdate ();**

**Generics** are used to create Classes, Interfaces, Methods in which type of Object on which they operate is specified as a parameter. The used of Generics: -

1. Type checking is done at compile time.
2. Eliminate need of casting.

**E (Elements), T (Type), K (Key), V (Values), N (Number) etc.**