29 July 2025 09:07

Search the collection for an element = whether the element is present in the list

2 possibilities
1. We find the element in the list
2. The element is not in the list

ArrayList = compare each and every element with search element
To conclude that element is not in list
for n elements compare n = O(n)

TreeSet = Binary tree to store the elements
Every comparison eliminates half the subtree
O(logn) = to conclude that the element is not in the list

HashSet = A good hashcode might give us the conclusion that element is not in the list
In just 1 comparison!!!

Red , green , blue, magenta = values to insert in hashset

```
hashcode Hash_function(element)
       {
            return Length(element)
           //Return 1; //WORST
       }
```

hashcode	buckets
1	
2	
3	red
4	blue
5	green
6	
7	magenta
8	
9	

Search whether white is in the list

hashcode = length(white) =5
Comparing the value at bucket 5 we can conclude

Object class = super class of class
Hashcode() = returns a unique hashcode for each object
They return the reference- address

Iterator interface = to traverse any collection

Iterator interface
- hasNext = check if there is a next element
- next = fetch the next element from the collection
- remove = remove current element

```
Iterator<?> iterator = collection.iterator();
```

Interface referencename = collection.API_that_returns_obj_IteratorImpl

Streams to traverse list
The list is treated as a Stream of elements ----
One element flows to the function at a time
It can be processed

Sorting a collection ----
Collections.sort()

TreeSet - sorts the elements in ascending order using InORDER traversal

Comparable gives us the default SORT ordering !!!!

```
 sort( obj )    ob j=  new DescRoll()    OR  obj =  (s1,s2)->{….}
 {
           For(int …)
             for( …. )
                 if( obj.compare(s1,s2) )
                         Swap
                  Else
                    not swap

 }
```

Map = store each element as a PAIR [key -value]

Map = interface , does not extend from Collection
 HashMap = the hashcode is generated based on the KEY
 TreeMap = the comparison is done based on KEY

Put
Get

JDBC = Java Database Connectivity

 Standalone programs =   single process program
 Multi tier application =
    Client process< ===================> Server process
     Java program (JVM) < ==============> DB process (non java program)
       Data comes in the Java program - use JAVA programming to deal with data

Manage external libraries and classpaths in a better way !!
     JAR file = Java **Archive** file [visualize it as a zip file - folders and subfolders and files]

    JAR = packaging that is used to pack java packages and classes and deliver !!!
      JAR must be available to my program !!!
           My program executor should search for the jar in FS
                classpath=.;loc1;loc2;loc3

Build tasks are offloaded to Build tools = MAVEN Build tool
      MAVEN has a configuration file = pom.xml
       will have list of dependencies = which jars are needed
      MAVEN = repository = STOREHOUSE of jars
       --- download the jars , put them in classpath and make them available to my code

```
      MAVEN  PROJECT STRUCTURE  ---
              main
                 --java
                    ---  study
                           XYZ.class
```

    ESTABLISH a connection with the database
   step 1 = Want a driver [code that will translate from JAVA to DB format]
          GO to mvn repository
            Get the dependency of MYSQL driver
   step2 = URL to connect to the database
    step3 = Get the connection

 JDBC URL   =   "protocol:typeofdriver://IP:port/db"
            =   "jdbc:mysql://localhost:3306/alpha"

      JEE = Java Enterprise Endition
      JDBC = **Standard** provided in the JEE for all DB connectivity

      Lots of interfaces of JDBC
         We are coding to interface =   **java.sql**
            We are not coding to IMPLEMENTATION
      With every database driver different packages, classnames !!!

JVM ---->Query---->DRIVER -----> MysqlDB - query runs on serverside

rs.next() --- move the cursor
To next row of resultset
It also returns the boolean
whether next row present

-->rs

id	Name	Dob	City
12	Ppp		
13	Sfjj		

PreparedStatement = interface java.sql
GOOD for queries that have values coming from variables

☑ String sql = "insert into student values ("+var1+",'"+var2+"','"+var3+.....)
TEDIOUS SYNTAX

With prepared statement -
String sql = "insert into student values (?,?,?,?,?)";

Q marks are read from L to R 1,2,3,4,5

call pstmt setters to set the variables

Statement	PreparedStatement
con.createStatement()	pstmt =con.prepareStatement(SQL)

Stmt is not bound to a single query
 	
Stmt.exeuteUpdate(SQL)
Stmt.executeQuer(SQL)
	Pstmt is bound to a single sql
	
	pstmt.setInt(1, var)
	pstmt.setString(2,_)
	
	pstmt.executeUpdate();

Statement queries compiled at query firing time	Precompiled , so quicker than the statement query
	? Syntax is convenient when variables are used

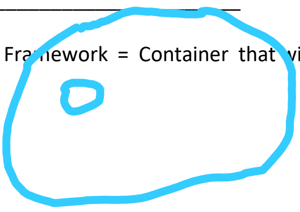
____

Junit = Library used for Unit Testing .

- Test a component - it satisfies the exepected Test cases

Test case = test whether the use case is working
____

Test Framework = Container that will hold the component and test it



Container communication using
Annotations !!!

Annotatation --- Sticker , Tags
Annotations can be applied to different Targets
methods, property, parameter of methods, Type-class
they are in regular packages , with .class file

Test Framework = Test Engine
Contain the class to be tested

When some container will INTROSPECT the class
it will find the annotation
it will do something about it

@Override = COMPILER looks at this and ensures that f1 signature matches super class f1
void f1()
{
}

@FunctionalInterface = COMPILER looks at this and ensures that interface has exact 1 abstract method
interface Test
{
}

____
When u see annotations
3 players
1. Annotation - package

____
When u see annotations
3 players
1. Annotation - package
2. Target where it is applied
3. Container that will find the annotation and do something about it

____

DAOService ========> Entity===> populated from db

Service ===> Dep1 =====>Dep2
           ===> Dep3

Unit Testing = the dependecies come from other components
  To test my component I will use the STUB /MOCK /DUMMY of the dependency component

____

Java Backend !!

Container Based Framework !
       Components = BEANS

POJO = Plain Old Java Object
                Properties , constructors, getters and setters , toString

BEAN = Managed classes, Managed POJOs
I write the class
Spring Container Manages the LIFECYCLE of the class-instance
LIFECYCLE = objects are created [how many to create]
                  dependencies are injected
                  call back methods are invoked

____

Write a Bean and instruct the container about it .

instruct the container about it   = Spring Configuration !!!!

How to do the Spring Configuration ?
1. XML
2. Using Java Configuration classes
3. Direct Annotations

____

GET a template of the Spring Project
        Spring Initializer Project

  Download a spring initializr project = spring.io

____

Context   = Spring Container IoC   = **Inversion of Control**

By default the Spring Context creates a Bean Object on startup
        Eager Initialization
        Using Singleton Factory to create the bean
                Only one object of the bean will be created for that container instance
                @Lazy(value=true) = the singleton instance is not created till DEMAND

We instruct the container to use a Prototype factory = One bean instance per request
@Scope(value = "prototype")
____

Dependency Injection

1. Dependency = property of the bean class
2. Injection = Setting the property of the bean class

HAS-A relation that we will have in the bean
  class Student
  {

```
        String name;     Student is dependent on String
        MyDate  dob;   Student is dependent of MyDate
   }
```

Bean is a managed class !!
 Object creation is done by the container !!
  The dependencies are injected by the container !!!!
        Through setters
        Through constructors
        Through properties