1)write a query second height salary in emp table.

Ans-Consider below simple table:

Name Salary

---------------

abc 100000

bcd 1000000

efg 40000

ghi 500000

How to find the employee whose salary is second highest. For example, in above table, “ghi” has the second highest salary as 500000.

Below is simple query to find the employee whose salary is highest.

SELECT name, MAX(salary) as salary FROM employee

We can nest the above query to find the second largest salary.

SELECT name, MAX(salary) AS salary

FROM employee

WHERE salary < (SELECT MAX(salary)

FROM employee);

There are other ways also as suggested by RajnishKrJha.

SELECT name, MAX(salary) AS salary

FROM employee

WHERE salary IN

(SELECT salary FROM employee MINUS SELECT MAX(salary)

FROM employee);

SELECT name, MAX(salary) AS salary

FROM employee

WHERE salary (SELECT MAX(salary)

FROM employee);

One way as suggested by **Arka Poddar**.  
IN SQL Server using Common Table Expression or **CTE**, we can find the second highest salary:

WITH T AS

(

SELECT \*

DENSE\_RANK() OVER (ORDER BY Salary Desc) AS Rnk

FROM Employees

)

SELECT Name

FROM T

WHERE Rnk=2;

**How to find the third largest salary?**  
Simple, we can do one more nesting.

SELECT name, MAX(salary) AS salary

FROM employee

WHERE salary < (SELECT MAX(salary)

FROM employee

WHERE salary < (SELECT MAX(salary)

FROM employee)

);

Note that instead of nesting for second, third largest salary, we can find nth salary using general query like in mysql:

select salary from employee order by salary desc limit n-1,1;

Or

Select name,salary from employee A where n-1 = (Select count(1) from employee B where B.salary>A.salary)

2) write a query Highest Salary in each department

Ans-select \* from (select ename,deptno,sal,hiredate,dense\_rank() over(partition by deptno order by sal desc,hiredate asc) rn from emp)  
where rn=1;

3)write a program repeated charter in a string

Ans-

|  |
| --- |
| // Java program to find the first  // repeated character in a string  import java.util.\*;    class Main  {      // This function prints the first repeated      // character in str[]      static char firstRepeating(char str[])      {          // Creates an empty hashset          HashSet<Character> h = new HashSet<>();            // Traverse the input array from left to right          for (int i=0; i<=str.length-1; i++)          {              char c = str[i];                // If element is already in hash set, update x              // and then break              if (h.contains(c))                  return c;                else // Else add element to hash set                  h.add(c);          }            return '\0';      }        // Driver method to test above method      public static void main (String[] args)      {          String str = "geeksforgeeks";          char[] arr = str.toCharArray();          System.out.println(firstRepeating(arr));      }  } |

**Output:**

e

4)write a program first character occurrence in a string.

Ans-

|  |
| --- |
| class NoOfOccurenceOfCharacters {      static final int MAX\_CHAR = 256;        static void getOccuringChar(String str)      {          // Create an array of size 256 i.e. ASCII\_SIZE          int count[] = new int[MAX\_CHAR];            int len = str.length();            // Initialize count array index          for (int i = 0; i < len; i++)              count[str.charAt(i)]++;            // Create an array of given String size          char ch[] = new char[str.length()];          for (int i = 0; i < len; i++) {              ch[i] = str.charAt(i);              int find = 0;              for (int j = 0; j <= i; j++) {                    // If any matches found                  if (str.charAt(i) == ch[j])                      find++;              }                if (find == 1)                  System.out.println("Number of Occurrence of " +                   str.charAt(i) + " is:" + count[str.charAt(i)]);          }      }      public static void main(String[] args)      {          Scanner sc = new Scanner(System.in);          String str = "geeksforgeeks";          getOccuringChar(str);      }  } |

Output:

Number of Occurrence of g is:2

Number of Occurrence of e is:4

Number of Occurrence of k is:2

Number of Occurrence of s is:2

Number of Occurrence of f is:1

Number of Occurrence of o is:1

Number of Occurrence of r is:1

5)difference between arraylist and linked list

Ans-**ArrayList:-**

* ArrayList uses a dynamic array.
* ArrayList is not efficient for manipulation because a lot of shifting is required.
* ArrayList is better to store and fetch data.

**LinkedList:-**

* LinkedList uses doubly linked list.
* LinkedList is efficient for manipulation.
* LinkedList is better to manipulate data
* Arraylist
* Manipulation with ArrayList is **slow** because it internally uses array. If any element is removed from the array, all the bits are shifted in memory.
* ArrayList is **better for storing and accessing** data.
* linked list
* Manipulation with LinkedList is **faster** than ArrayList because it uses doubly linked list so no bit shifting is required in memory.
* LinkedList is **better for manipulating** data.

Two popular lists in Java are:

1. [ArrayList](https://www.geeksforgeeks.org/arraylist-in-java/):-Implemented with the concept of dynamic array.

ArrayList<Type> arrL = new ArrayList<Type>();

Here Type is the data type of elements in ArrayList

to be created

2. [LinkedList](http://geeksquiz.com/linked-list-in-java/):-Implemented with the concept of doubly linked list.

LinkedList<Type> linkL = new LinkedList<Type>();

Here Type is the data type of elements in LinkedList

to be created

Comparision between [ArrayList](https://www.geeksforgeeks.org/arraylist-in-java/)and [LinkedList](http://geeksquiz.com/linked-list-in-java/):-

1. Insertions are easy and fast in LinkedList as compared to ArrayList because there is no  
   risk of resizing array and copying content to new array if array gets full which makes  
   adding into ArrayList of O(n) in worst case, while adding is O(1) operation in LinkedList  
   in Java. ArrayList also needs to be update its index if you insert something anywhere except  
   at the end of array.
2. Removal also better in LinkedList than ArrayList due to same reasons as insertion.
3. LinkedList has more memory overhead than ArrayList because in ArrayList each index only  
   holds actual object (data) but in case of LinkedList each node holds both data and address  
   of next and previous node.
4. Both LinkedList and ArrayList require O(n) time to find if an element is present or not. However we can do Binary Search on ArrayList if it is sorted and therefore can search in O(Log n) time.

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Java program to demonstrate difference between ArrayList and  // LinkedList.  import java.util.ArrayList;  import java.util.LinkedList;    public class ArrayListLinkedListExample  {      public static void main(String[] args)      {          ArrayList<String> arrlistobj = new ArrayList<String>();          arrlistobj.add("0. Practice.GeeksforGeeks.org");          arrlistobj.add("1. Quiz.GeeksforGeeks.org");          arrlistobj.add("2. Code.GeeksforGeeks.org");          arrlistobj.remove(1);  // Remove value at index 2          System.out.println("ArrayList object output :" +  arrlistobj);            // Checking if an element is present.          if (arrlistobj.contains("2. Code.GeeksforGeeks.org"))              System.out.println("Found");          else              System.out.println("Not found");              LinkedList llobj = new LinkedList();          llobj.add("0. Practice.GeeksforGeeks.org");          llobj.add("1. Quiz.GeeksforGeeks.org");          llobj.add("2. Code.GeeksforGeeks.org");          llobj.remove("1. Quiz.GeeksforGeeks.org");          System.out.println("LinkedList object output :" + llobj);            // Checking if an element is present.          if (llobj.contains("2. Code.GeeksforGeeks.org"))              System.out.println("Found");          else              System.out.println("Not found");      }  } |

**Output:**

ArrayList object output :[0. Practice.GeeksforGeeks.org, 2. Code.GeeksforGeeks.org]

Found

LinkedList object output :[0. Practice.GeeksforGeeks.org, 2. Code.GeeksforGeeks.org]

Found

6)difference between comparable and comparator

Ans-**Comparable interface:** Class whose objects to be sorted must implement this interface.In this,we have to implement compareTo(Object) method.

For example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | public class Country implements Comparable{         @Override      public int compareTo(Object arg0) {          Country country=(Country) arg0;          return (this.countryId < country.countryId ) ? -1: (this.countryId > country.countryId ) ? 1:0 ;  }} |

If any class implements comparable inteface then collection of that object can be sorted automatically using Collection.sort() or Arrays.sort().Object will be sort on the basis of compareTo method in that class.

Objects which implement Comparable in java can be used as keys in a SortedMap like TreeMap or SortedSet like TreeSet without implementing any other interface.

**Comparator interface:** Class whose objects to be sorted do not need to implement this interface.Some third class can implement this interface to sort.e.g.CountrySortByIdComparator class can implement Comparator interface to sort collection of country object by id. For example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | public class CountrySortByIdComparator implements Comparator<Country>{        @Override      public int compare(Country country1, Country country2) {            return (country1.getCountryId() < country2.getCountryId() ) ? -1: (country1.getCountryId() > country2.getCountryId() ) ? 1:0 ;      }    } |

Using Comparator interface,we can write different sorting based on different attributes of objects to be sorted.You can use anonymous comparator to compare at particular line of code. For example:

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21 | Country indiaCountry=new Country(1, 'India');          Country chinaCountry=new Country(4, 'China');          Country nepalCountry=new Country(3, 'Nepal');          Country bhutanCountry=new Country(2, 'Bhutan');            List<Country> listOfCountries = new ArrayList<Country>();          listOfCountries.add(indiaCountry);          listOfCountries.add(chinaCountry);          listOfCountries.add(nepalCountry);          listOfCountries.add(bhutanCountry);    //Sort by countryName               Collections.sort(listOfCountries,new Comparator<Country>() {                   @Override                 public int compare(Country o1, Country o2) {                       return o1.getCountryName().compareTo(o2.getCountryName());                 }             }); |
|  |  |

**Comparator vs Comparable**

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter** | **Comparable** | **Comparator** | |
| Sorting logic | Sorting logic must be in same class whose objects are being sorted. Hence this is called natural ordering of objects | Sorting logic is in separate class. Hence we can write different sorting based on different attributes of objects to be sorted. E.g. Sorting using id,name etc. | |
| Implementation | Class whose objects to be sorted must implement this interface.e.g Country class needs to implement comparable to collection of country object by id | Class whose objects to be sorted do not  need to implement this interface.Some other class can implement this interface. E.g.-CountrySortByIdComparator class can implement Comparator interface to sort collection of country object by id | |
| Sorting method | int compareTo(Object o1) This method compares this object with o1 object and returns a integer.Its value has following meaning 1. positive – this object is greater than o1 2. zero – this object equals to o1 3. negative – this object is less than o1 | int compare(Object o1,Object o2) This method compares o1 and o2 objects. and returns a integer.Its value has following meaning. 1. positive – o1 is greater than o2 2. zero – o1 equals to o2 3. negative – o1 is less than o1 | |
| Calling method | Collections.sort(List) Here objects will be sorted on the basis of CompareTo method | Collections.sort(List, Comparator) Here objects will be sorted on the basis of Compare method in Comparator | |
| Package | Java.lang.Comparable | Java.util.Comparator |

**Java code:**  
**For Comparable:** We will create class country having attribute id and name.This class will implement Comparable interface and implement CompareTo method to sort collection of country object by id.

**1. Country.java**

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | package org.arpit.javapostsforlearning;  //If this.cuntryId < country.countryId:then compare method will return -1  //If this.countryId > country.countryId:then compare method will return 1  //If this.countryId==country.countryId:then compare method will return 0  public class Country implements Comparable{      int countryId;      String countryName;        public Country(int countryId, String countryName) {          super();          this.countryId = countryId;          this.countryName = countryName;      }        @Override      public int compareTo(Object arg0) {          Country country=(Country) arg0;          return (this.countryId < country.countryId ) ? -1: (this.countryId > country.countryId ) ? 1:0 ;      }        public int getCountryId() {          return countryId;      }        public void setCountryId(int countryId) {          this.countryId = countryId;      }        public String getCountryName() {          return countryName;      }        public void setCountryName(String countryName) {          this.countryName = countryName;      }    } |

**2.ComparatorMain.java**

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | package org.arpit.javapostsforlearning;    import java.util.ArrayList;  import java.util.Collections;  import java.util.List;    public class ComparatorMain {        /\*\*       \* @author Arpit Mandliya       \*/      public static void main(String[] args) {           Country indiaCountry=new Country(1, 'India');           Country chinaCountry=new Country(4, 'China');           Country nepalCountry=new Country(3, 'Nepal');           Country bhutanCountry=new Country(2, 'Bhutan');                List<Country> listOfCountries = new ArrayList<Country>();              listOfCountries.add(indiaCountry);              listOfCountries.add(chinaCountry);              listOfCountries.add(nepalCountry);              listOfCountries.add(bhutanCountry);                System.out.println('Before Sort  : ');              for (int i = 0; i < listOfCountries.size(); i++) {                  Country country=(Country) listOfCountries.get(i);                  System.out.println('Country Id: '+country.getCountryId()+'||'+'Country name: '+country.getCountryName());              }              Collections.sort(listOfCountries);                System.out.println('After Sort  : ');              for (int i = 0; i < listOfCountries.size(); i++) {                  Country country=(Country) listOfCountries.get(i);                  System.out.println('Country Id: '+country.getCountryId()+'|| '+'Country name: '+country.getCountryName());              }      }    } |

**Output:**

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10 | Before Sort  :  Country Id: 1||Country name: India  Country Id: 4||Country name: China  Country Id: 3||Country name: Nepal  Country Id: 2||Country name: Bhutan  After Sort  :  Country Id: 1|| Country name: India  Country Id: 2|| Country name: Bhutan  Country Id: 3|| Country name: Nepal  Country Id: 4|| Country name: China |

**For Comparator:** We will create class country having attribute id and name and will create another class CountrySortByIdComparator which will implement Comparator interface and implement compare method to sort collection of country object by id and we will also see how to use anonymous comparator.

**1.Country.java**

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | package org.arpit.javapostsforlearning;    public class Country{      int countryId;      String countryName;        public Country(int countryId, String countryName) {          super();          this.countryId = countryId;          this.countryName = countryName;      }        public int getCountryId() {          return countryId;      }        public void setCountryId(int countryId) {          this.countryId = countryId;      }        public String getCountryName() {          return countryName;      }        public void setCountryName(String countryName) {          this.countryName = countryName;      }    } |

**2.CountrySortbyIdComparator.java**

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15 | package org.arpit.javapostsforlearning;    import java.util.Comparator;  //If country1.getCountryId()<country2.getCountryId():then compare method will return -1  //If country1.getCountryId()>country2.getCountryId():then compare method will return 1  //If country1.getCountryId()==country2.getCountryId():then compare method will return 0   public class CountrySortByIdComparator implements Comparator<Country>{        @Override      public int compare(Country country1, Country country2) {            return (country1.getCountryId() < country2.getCountryId() ) ? -1: (country1.getCountryId() > country2.getCountryId() ) ? 1:0 ;      }    } |

**3.ComparatorMain.java**

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54 | package org.arpit.javapostsforlearning;    import java.util.ArrayList;  import java.util.Collections;  import java.util.Comparator;  import java.util.List;    public class ComparatorMain {        /\*\*       \* @author Arpit Mandliya       \*/      public static void main(String[] args) {           Country indiaCountry=new Country(1, 'India');           Country chinaCountry=new Country(4, 'China');           Country nepalCountry=new Country(3, 'Nepal');           Country bhutanCountry=new Country(2, 'Bhutan');                List<Country> listOfCountries = new ArrayList<Country>();              listOfCountries.add(indiaCountry);              listOfCountries.add(chinaCountry);              listOfCountries.add(nepalCountry);              listOfCountries.add(bhutanCountry);                System.out.println('Before Sort by id : ');              for (int i = 0; i < listOfCountries.size(); i++) {                  Country country=(Country) listOfCountries.get(i);                  System.out.println('Country Id: '+country.getCountryId()+'||'+'Country name: '+country.getCountryName());              }              Collections.sort(listOfCountries,new CountrySortByIdComparator());                System.out.println('After Sort by id: ');              for (int i = 0; i < listOfCountries.size(); i++) {                  Country country=(Country) listOfCountries.get(i);                  System.out.println('Country Id: '+country.getCountryId()+'|| '+'Country name: '+country.getCountryName());              }                //Sort by countryName              Collections.sort(listOfCountries,new Comparator<Country>() {                    @Override                  public int compare(Country o1, Country o2) {                      return o1.getCountryName().compareTo(o2.getCountryName());                  }              });                System.out.println('After Sort by name: ');              for (int i = 0; i < listOfCountries.size(); i++) {                  Country country=(Country) listOfCountries.get(i);                  System.out.println('Country Id: '+country.getCountryId()+'|| '+'Country name: '+country.getCountryName());              }      }    } |

**Output:**

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15 | Before Sort by id :  Country Id: 1||Country name: India  Country Id: 4||Country name: China  Country Id: 3||Country name: Nepal  Country Id: 2||Country name: Bhutan  After Sort by id:  Country Id: 1|| Country name: India  Country Id: 2|| Country name: Bhutan  Country Id: 3|| Country name: Nepal  Country Id: 4|| Country name: China  After Sort by name:  Country Id: 2|| Country name: Bhutan  Country Id: 4|| Country name: China  Country Id: 1|| Country name: India  Country Id: 3|| Country name: Nepal |

7) how to sort without using collection sorting method

Ans-ArrayList < Integer > arraylist = new ArrayList < Integer > ();

arraylist.add(10010);

arraylist.add(5);

arraylist.add(4);

arraylist.add(2);

for (int i = 0; i < arraylist.size(); i++) {

for (int j = arraylist.size() - 1; j > i; j--) {

if (arraylist.get(i) > arraylist.get(j)) {

int tmp = arraylist.get(i);

arraylist.get(i) = arraylist.get(i);

arraylist.get(j) = tmp;

}

}

}

for (int i: arraylist) {

System.out.println(i);

}

It is giving error while swapping, The LHS should be variable. I understand it. Set method works here but I do not want to use. Is there a way to do it without using set method? Help is really appreciated.

Or

arraylist.get(i)= arraylist.get(i);

arraylist.get(j) =tmp;

You can't assign a value to a method call. As the compiler told you, the left hand side of an assignment must be a variable.

Use set method :

arraylist.set(i,arraylist.get(j));

arraylist.set(j,tmp);

Is there a way to do it without using set method?

No. Unless you wish to convert your ArrayList to an array, sort the array, and update the ArrayList with the sorted array.

8) difference between Association, Composition and Aggregation

Ans-**Association**

Association is relation between two separate classes which establishes through their Objects. Association can be one-to-one, one-to-many, many-to-one, many-to-many.  
In Object-Oriented programming, an Object communicates to other Object to use functionality and services provided by that object. **Composition** and **Aggregation** are the two forms of association.  


filter\_none

edit

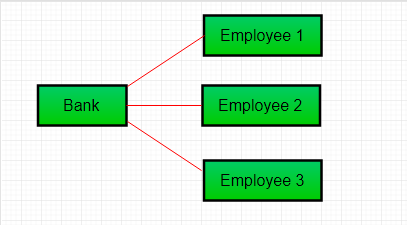
play\_arrow

brightness\_4

|  |
| --- |
| // Java program to illustrate the  // concept of Association  import java.io.\*;    // class bank  class Bank  {      private String name;        // bank name      Bank(String name)      {          this.name = name;      }        public String getBankName()      {          return this.name;      }  }    // employee class  class Employee  {      private String name;        // employee name      Employee(String name)      {          this.name = name;      }        public String getEmployeeName()      {          return this.name;      }  }    // Association between both the  // classes in main method  class Association  {      public static void main (String[] args)      {          Bank bank = new Bank("Axis");          Employee emp = new Employee("Neha");            System.out.println(emp.getEmployeeName() +                 " is employee of " + bank.getBankName());      }  } |

Output:

Neha is employee of Axis

In above example two separate classes Bank and Employee are associated through their Objects. Bank can have many employees, So it is a one-to-many relationship.  
[](https://cdncontribute.geeksforgeeks.org/wp-content/uploads/Aggre.png)

**Aggregation**

It is a special form of Association where:

* It represents **Has-A** relationship.
* It is a **unidirectional association** i.e. a one way relationship. For example, department can have students but vice versa is not possible and thus unidirectional in nature.
* In Aggregation,**both the entries can survive individually** which means ending one entity will not effect the other entity

filter\_none

edit

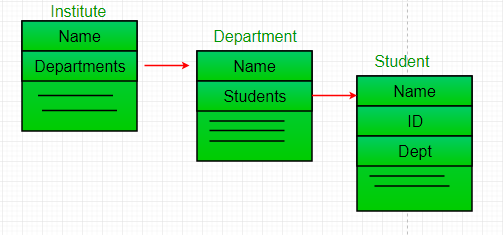
play\_arrow

brightness\_4

|  |
| --- |
| // Java program to illustrate  //the concept of Aggregation.  import java.io.\*;  import java.util.\*;    // student class  class Student  {      String name;      int id ;      String dept;        Student(String name, int id, String dept)      {            this.name = name;          this.id = id;          this.dept = dept;        }  }    /\* Department class contains list of student  Objects. It is associated with student  class through its Object(s). \*/  class Department  {        String name;      private List<Student> students;      Department(String name, List<Student> students)      {            this.name = name;          this.students = students;        }        public List<Student> getStudents()      {          return students;      }  }    /\* Institute class contains list of Department  Objects. It is asoociated with Department  class through its Object(s).\*/  class Institute  {        String instituteName;      private List<Department> departments;        Institute(String instituteName, List<Department> departments)      {          this.instituteName = instituteName;          this.departments = departments;      }        // count total students of all departments      // in a given institute      public int getTotalStudentsInInstitute()      {          int noOfStudents = 0;          List<Student> students;          for(Department dept : departments)          {              students = dept.getStudents();              for(Student s : students)              {                  noOfStudents++;              }          }          return noOfStudents;      }    }    // main method  class GFG  {      public static void main (String[] args)      {          Student s1 = new Student("Mia", 1, "CSE");          Student s2 = new Student("Priya", 2, "CSE");          Student s3 = new Student("John", 1, "EE");          Student s4 = new Student("Rahul", 2, "EE");            // making a List of          // CSE Students.          List <Student> cse\_students = new ArrayList<Student>();          cse\_students.add(s1);          cse\_students.add(s2);            // making a List of          // EE Students          List <Student> ee\_students = new ArrayList<Student>();          ee\_students.add(s3);          ee\_students.add(s4);            Department CSE = new Department("CSE", cse\_students);          Department EE = new Department("EE", ee\_students);            List <Department> departments = new ArrayList<Department>();          departments.add(CSE);          departments.add(EE);            // creating an instance of Institute.          Institute institute = new Institute("BITS", departments);            System.out.print("Total students in institute: ");          System.out.print(institute.getTotalStudentsInInstitute());      }  } |

Output:

Total students in institute: 4

In this example, there is an Institute which has no. of departments like CSE, EE. Every department has no. of students. So, we make a Institute class which has a reference to Object or no. of Objects (i.e. List of Objects) of the Department class. That means Institute class is associated with Department class through its Object(s). And Department class has also a reference to Object or Objects (i.e. List of Objects) of Student class means it is associated with Student class through its Object(s).  
It represents a **Has-A** relationship.  
[](https://www.geeksforgeeks.org/cdncontribute.geeksforgeeks.org/wp-content/uploads/Reference.png)

**When do we use Aggregation ??**  
Code reuse is best achieved by aggregation.

**Composition**

Composition is a restricted form of Aggregation in which two entities are highly dependent on each other.

* It represents **part-of** relationship.
* In composition, both the entities are dependent on each other.
* When there is a composition between two entities, the composed object **cannot exist**without the other entity.

Lets take example of**Library**.

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Java program to illustrate  // the concept of Composition  import java.io.\*;  import java.util.\*;    // class book  class Book  {        public String title;      public String author;        Book(String title, String author)      {            this.title = title;          this.author = author;      }  }    // Libary class contains  // list of books.  class Library  {        // reference to refer to list of books.      private final List<Book> books;        Library (List<Book> books)      {          this.books = books;      }        public List<Book> getTotalBooksInLibrary(){           return books;      }    }    // main method  class GFG  {      public static void main (String[] args)      {            // Creating the Objects of Book class.          Book b1 = new Book("EffectiveJ Java", "Joshua Bloch");          Book b2 = new Book("Thinking in Java", "Bruce Eckel");          Book b3 = new Book("Java: The Complete Reference", "Herbert Schildt");            // Creating the list which contains the          // no. of books.          List<Book> books = new ArrayList<Book>();          books.add(b1);          books.add(b2);          books.add(b3);            Library library = new Library(books);            List<Book> bks = library.getTotalBooksInLibrary();          for(Book bk : bks){                System.out.println("Title : " + bk.title + " and "              +" Author : " + bk.author);          }      }  } |

Output

Title : EffectiveJ Java and Author : Joshua Bloch

Title : Thinking in Java and Author : Bruce Eckel

Title : Java: The Complete Reference and Author : Herbert Schildt

In above example a library can have no. of **books** on same or different subjects. So, If Library gets destroyed then All books within that particular library will be destroyed. i.e. book can not exist without library. That’s why it is composition.

**Aggregation vs Composition**

1. **Dependency:** Aggregation implies a relationship where the child **can exist independently** of the parent. For example, Bank and Employee, delete the Bank and the Employee still exist. whereas Composition implies a relationship where the child **cannot exist independent** of the parent. Example: Human and heart, heart don’t exist separate to a Human
2. **Type of Relationship:** Aggregation relation is **“has-a”** and composition is **“part-of”**relation.
3. **Type of association:**Composition is a **strong** Association whereas Aggregation is a **weak** Association.

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Java program to illustrate the  // difference between Aggregation  // Composition.    import java.io.\*;    // Engine class which will  // be used by car. so 'Car'  // class will have a field  // of Engine type.  class Engine  {      // starting an engine.      public void work()      {            System.out.println("Engine of car has been started ");        }    }    // Engine class  final class Car  {        // For a car to move,      // it need to have a engine.      private final Engine engine; // Composition      //private Engine engine;     // Aggregation        Car(Engine engine)      {          this.engine = engine;      }        // car start moving by starting engine      public void move()      {            //if(engine != null)          {              engine.work();              System.out.println("Car is moving ");          }      }  }    class GFG  {      public static void main (String[] args)      {            // making an engine by creating          // an instance of Engine class.          Engine engine = new Engine();            // Making a car with engine.          // so we are passing a engine          // instance as an argument while          // creating instace of Car.          Car car = new Car(engine);          car.move();        }  } |

Output:

Engine of car has been started

Car is moving

In case of aggregation, the Car also performs its functions through an Engine. but the Engine is not always an internal part of the Car. An engine can be swapped out or even can be removed from the car. That’ why we make The Engine type field non-final.

9)hashset internal flow.

Ans-**HashSet** uses HashMap internally to store it’s objects. Whenever you create a HashSet object, one **HashMap** object associated with it is also created. This HashMap object is used to store the elements you enter in the HashSet. The elements you add into HashSet are stored as **keys** of this HashMap object. The value associated with those keys will be a **constant**.

Every constructor of HashSet class internally creates one HashMap object. You can check this in the source code of HashSet class in JDK installation directory. Below is the some sample code of the constructors of HashSet class.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | private transient HashMap<E,Object> map;    //Constructor - 1    public HashSet()  {          map = new HashMap<>();          //Creating internally backing HashMap object  }    //Constructor - 2    public HashSet(Collection<? extends E> c)  {          map = new HashMap<>(Math.max((int) (c.size()/.75f) + 1, 16));     //Creating internally backing HashMap object          addAll(c);  }    //Constructor - 3    public HashSet(int initialCapacity, float loadFactor)  {          map = new HashMap<>(initialCapacity, loadFactor);        //Creating internally backing HashMap object  }    //Constructor - 4    public HashSet(int initialCapacity)  {          map = new HashMap<>(initialCapacity);          //Creating internally backing HashMap object  } |

You can notice that each and every constructor internally creates one new HashMap object.

## How HashSet Works Internally In Java?

Whenever you insert an element into HashSet using **add()** method, it actually creates an entry in the internally backing HashMap object with element you have specified as it’s key and constant called “**PRESENT**” as it’s value. This “PRESENT” is defined in the HashSet class as below.

|  |  |
| --- | --- |
| 1  2 | // Dummy value to associate with an Object in the backing Map  private static final Object PRESENT = new Object(); |

Let’s have a look at add() method of HashSet class.

|  |  |
| --- | --- |
| 1  2  3  4 | public boolean add(E e)  {          return map.put(e, PRESENT)==null;  } |

You can notice that, add() method of HashSet class internally calls put() method of backing HashMap object by passing the element you have specified as a key and constant “PRESENT” as it’s value.

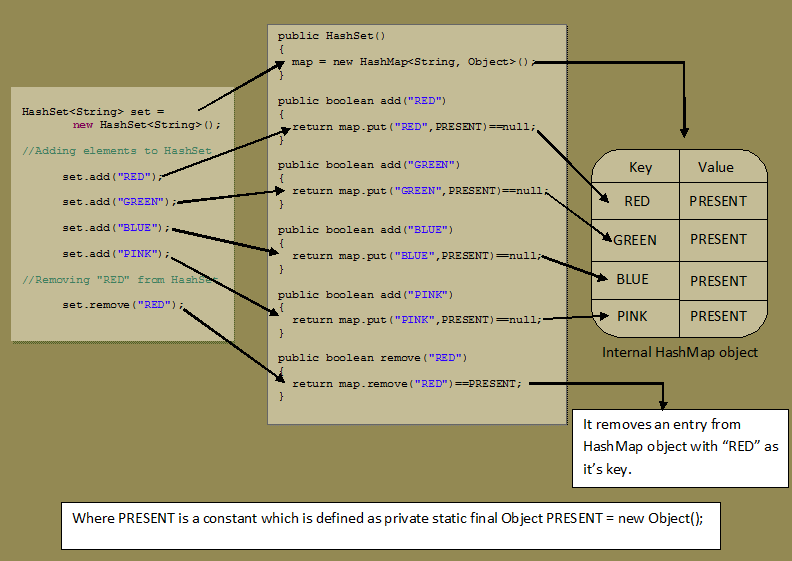
**remove()** method also works in the same manner.

|  |  |
| --- | --- |
| 1  2  3  4 | public boolean remove(Object o)  {          return map.remove(o)==PRESENT;  } |

Let’s see one example of HashSet and how it maintains HashMap internally.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | public class HashSetExample  {      public static void main(String[] args)      {          //Creating One HashSet object            HashSet<String> set = new HashSet<String>();            //Adding elements to HashSet            set.add("RED");            set.add("GREEN");            set.add("BLUE");            set.add("PINK");            //Removing "RED" from HashSet            set.remove("RED");      }  } |

See the below picture how above program works internally. You can observe that internal HashMap object contains elements of HashSet as keys and constant “PRESENT” as their value.



In the same manner, all methods of HashSet class process internally backing HashMap object to get the desired result. If you know how HashMap works, it will be easy for you to understand how HashSet works. You go through the source code of HashSet class once, you will get a clear picture about how HashSet works internally in Java.

10)difference between fail fast and failsafe

## Ans-In this article, we’ll introduce the concept of Fail-Fast and Fail-Safe Iterators.**Fail-Fast systems abort operation as-fast-as-possible exposing failures immediately and stopping the whole operation.**

Whereas, **Fail-Safe systems don’t abort an operation in the case of a failure. Such systems try to avoid raising failures as much as possible.**

**2. Fail-Fast**Iterators

Fail-fast iterators in Java don’t play along when the underlying collection gets modified.

Collections maintain an internal counter called modCount. Each time an item is added or removed from the Collection, this counter gets incremented.

When iterating, on each next() call, the current value of modCount gets compared with the initial value. If there’s a mismatch, it throws ConcurrentModificationException which aborts the entire operation.

**Default iterators for Collections from java.util package such as ArrayList, HashMap, etc. are Fail-Fast.**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | ArrayList<Integer> numbers = // ...    Iterator<Integer> iterator = numbers.iterator();  while (iterator.hasNext()) {      Integer number = iterator.next();      numbers.add(50);  } |

In the code snippet above, the ConcurrentModificationException gets thrown at the beginning of a next iteration cycle after the modification was performed.

The Fail-Fast behavior isn’t guaranteed to happen in all scenarios as it’s impossible to predict behavior in case of concurrent modifications.**These iterators throw ConcurrentModificationException on a best effort basis**.

If during iteration over a Collection, **an item is removed using Iterator‘s remove() method, that’s entirely safe and doesn’t throw an exception**.

However, if the Collection‘s remove() method is used for removing an element, it throws an exception:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | ArrayList<Integer> numbers = // ...    Iterator<Integer> iterator = numbers.iterator();  while (iterator.hasNext()) {      if (iterator.next() == 30) {          iterator.remove(); // ok!      }  }    iterator = numbers.iterator();  while (iterator.hasNext()) {      if (iterator.next() == 40) {          numbers.remove(2); // exception      }  } |

## **3. Fail-Safe Iterators**

Fail-Safe iterators favor lack of failures over the inconvenience of exception handling.

Those iterators create a clone of the actual Collection and iterate over it. If any modification happens after the iterator is created, the copy still remains untouched. Hence, these Iterators continue looping over the Collection even if it’s modified.

However, it’s important to remember that there’s no such thing as a truly Fail-Safe iterator. The correct term is Weakly Consistent.

That means, **if a** **Collection is modified while being iterated over, what the Iterator sees is weakly guaranteed**. This behavior may be different for different Collections and is documented in Javadocs of each such Collection.

The Fail-Safe Iterators have a few disadvantages, though. One disadvantage is that the **Iterator isn’t guaranteed to return updated data from the Collection**, as it’s working on the clone instead of the actual Collection.

Another disadvantage is the overhead of creating a copy of the Collection, both regarding time and memory.

**Iterators on Collections from java.util.concurrent package such as ConcurrentHashMap, CopyOnWriteArrayList, etc. are Fail-Safe in nature.**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | ConcurrentHashMap<String, Integer> map = new ConcurrentHashMap<>();    map.put("First", 10);  map.put("Second", 20);  map.put("Third", 30);  map.put("Fourth", 40);    Iterator<String> iterator = map.keySet().iterator();    while (iterator.hasNext()) {      String key = iterator.next();      map.put("Fifth", 50);  } |

In the code snippet above, we’re using Fail-Safe Iterator. Hence, even though a new element is added to the Collection during the iteration, it doesn’t throw an exception.

The default iteratorfor the ConcurrentHashMap is weakly consistent. This means that this Iterator can tolerate concurrent modification, traverses elements as they existed when Iterator was constructed and may (but isn’t guaranteed to) reflect modifications to the Collection after the construction of the Iterator.

Hence, in the code snippet above, the iteration loops five times, which means**it does detect the newly added element to the Collection.**

11)how to create datasource in hibernate.

Ans-I'm creating SessionFactory and I have my datasource as object in code where I'm creating SessionFactory, but i cannot set datasource to Hibernate Configuration object. So how can I set my datasource to my SessionFactory?

Configuration configuration = new Configuration();

Properties properties = new Properties();

properties.setProperty("hibernate.dialect", "org.hibernate.dialect.MySQLInnoDBDialect");

configuration.setProperties(properties);

configuration.setProperty("packagesToScan", "com.my.app");

SessionFactory sessionFactory = configuration.configure().buildSessionFactory();

12).how create multiple data source

Ans- [**Spring Boot**](https://projects.spring.io/spring-boot/) with [**Spring Data**](http://projects.spring.io/spring-data/) makes it easy to access a database through so called Repositories. But what if you want to access **multiple databases** maybe even with different Database Management Systems?

Luckily Spring provides a way of doing this.

I provided an example project with two PostgreSQL datasources on GitHub: <https://github.com/jahe/spring-boot-multiple-datasources>

#### 1. Add an additional datasource configuration to your application.properties

# Oracle DB - "foo"  
spring.datasource.url=jdbc:oracle:thin:@//db-server-foo:1521/FOO  
spring.datasource.username=fooadmin  
spring.datasource.password=foo123spring.datasource.driver-class-name=oracle.jdbc.OracleDriver

# PostgreSQL DB - "bar"  
bar.datasource.url=jdbc:postgresql://db-server-bar:5432/bar  
bar.datasource.username=baradmin  
bar.datasource.password=bar123bar.datasource.driver-class-name=org.postgresql.Driver

#### 2. Set the SQL Dialect to “default” in your application.properties to let Spring autodetect the different SQL Dialects of each datasource

spring.jpa.database=default

#### 3. Create a Java Package for each datasource with two nested Packages “domain” and “repo”

src/main/java  
- com.foobar  
 - foo  
 - domain  
 - repo  
 - bar  
 - domain  
 - repo

#### 4. Create a Configuration Class for the Oracle database “foo” named “FooDbConfig.java”

package com.foobar;  
   
@Configuration  
@EnableTransactionManagement  
@EnableJpaRepositories(  
 entityManagerFactoryRef = "entityManagerFactory",  
 basePackages = { "com.foobar.foo.repo" }  
)  
public class FooDbConfig {  
   
 @Primary  
 @Bean(name = "dataSource")  
 @ConfigurationProperties(prefix = "spring.datasource")  
 public DataSource dataSource() {  
 return DataSourceBuilder.create().build();  
 }  
   
 @Primary  
 @Bean(name = "entityManagerFactory")  
 public LocalContainerEntityManagerFactoryBean   
 entityManagerFactory(  
 EntityManagerFactoryBuilder builder,  
 @Qualifier("dataSource") DataSource dataSource  
 ) {  
 return builder  
 .dataSource(dataSource)  
 .packages("com.foobar.foo.domain")  
 .persistenceUnit("foo")  
 .build();  
 }  
   
 @Primary  
 @Bean(name = "transactionManager")  
 public PlatformTransactionManager transactionManager(  
 @Qualifier("entityManagerFactory") EntityManagerFactory   
 entityManagerFactory  
 ) {  
 return new JpaTransactionManager(entityManagerFactory);  
 }  
}

#### 5. Create a Configuration Class for the PostgreSQL database “bar” named “BarDbConfig.java”

package com.foobar;  
  
@Configuration  
@EnableTransactionManagement  
@EnableJpaRepositories(  
 entityManagerFactoryRef = "barEntityManagerFactory",  
 transactionManagerRef = "barTransactionManager",  
 basePackages = { "com.foobar.bar.repo" }  
)  
public class BarDbConfig {  
   
 @Bean(name = "barDataSource")  
 @ConfigurationProperties(prefix = "bar.datasource")  
 public DataSource dataSource() {  
 return DataSourceBuilder.create().build();  
 }  
   
 @Bean(name = "barEntityManagerFactory")  
 public LocalContainerEntityManagerFactoryBean   
 barEntityManagerFactory(  
 EntityManagerFactoryBuilder builder,  
 @Qualifier("barDataSource") DataSource dataSource  
 ) {  
 returnbuilder  
 .dataSource(dataSource)  
 .packages("com.foobar.bar.domain")  
 .persistenceUnit("bar")  
 .build();  
 }

@Bean(name = "barTransactionManager")  
 public PlatformTransactionManager barTransactionManager(  
 @Qualifier("barEntityManagerFactory") EntityManagerFactory  
 barEntityManagerFactory  
 ) {  
 return new JpaTransactionManager(barEntityManagerFactory);  
 }  
}

#### 6. Create an Entity “Foo.java” for the Oracle database “foo”

package com.foobar.foo.domain;  
   
@Entity  
@Table(name = "FOO")  
public class Foo {  
   
 @Id  
 @GeneratedValue  
 @Column(name = "ID")  
 private Long id;  
   
 @Column(name = "FOO")  
 private String foo;  
   
 Foo(String foo) {  
 this.foo = foo;  
 }  
   
 Foo() {  
 // Default constructor needed by JPA  
 }  
}

#### 7. Create a Repository “FooRepository.java” for the Oracle database “foo”

package com.foobar.foo.repo;  
   
@Repository  
public interface FooRepository extends JpaRepository<Foo, Long> {  
   
 Foo findById(Long id);   
   
}

#### 8. Create an Entity “Bar.java” for the PostgreSQL database “bar”

package com.foobar.bar.domain;  
   
@Entity  
@Table(name = "BAR")  
public class Bar {  
   
 @Id  
 @GeneratedValue  
 @Column(name = "ID")  
 private Long id;  
   
 @Column(name = "BAR")  
 private String bar;  
   
 Bar(String bar) {  
 this.bar = bar;  
 }  
   
 Bar() {  
 // Default constructor needed by JPA  
 }  
}

#### 9. Create a Repository “BarRepository.java” for the PostgreSQL database “bar”

package com.foobar.bar.repo;  
   
@Repository  
public interface BarRepository extends JpaRepository<Bar, Long> {  
   
 Bar findById(Long id);   
   
}

#### 10. Create the Spring Boot Main Class “Application.java”

package com.foobar;  
   
@SpringBootApplication  
public class Application {  
   
 public static void main(String[] args) {  
 SpringApplication.run(Application.class, args);  
 }  
   
}

#### 11. Use the Repositories in a REST Controller (or somewhere else)

package com.foobar;  
   
@RestController  
public class FooBarController {  
   
 private final FooRepository fooRepo;  
 private final BarRepository barRepo;  
   
 @Autowired  
 FooBarController(FooRepository fooRepo, BarRepository barRepo) {  
 this.fooRepo = fooRepo;  
 this.barRepo = barRepo;  
 }  
   
 @RequestMapping("/foobar/{id}")  
 public String fooBar(@PathVariable("id") Long id) {  
 Foo foo = fooRepo.findById(id);  
 Bar bar = barRepo.findById(id);  
   
 return foo.getFoo() + " " + bar.getBar();   
 }  
}

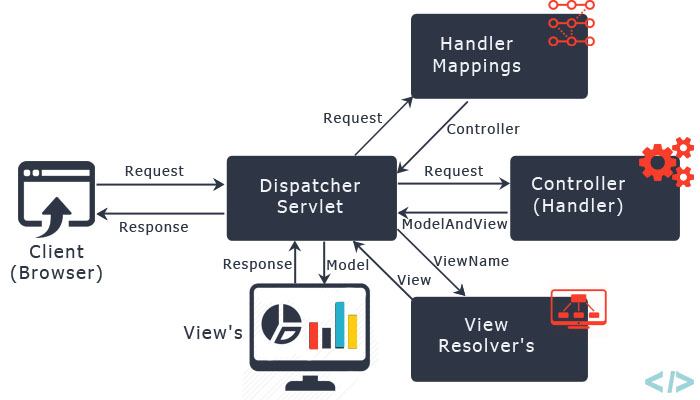
13)explain spring mvc

Ans- Spring MVC is mostly used with Spring for any web application development. It is very powerful and nice layered architecture for flow and configuration. It is very flexible to integrate with other web frameworks like struts.

In Spring Web MVC, DispatcherServlet class works as the front controller. It is responsible to manage the flow of the spring mvc application.

The @Controller annotation is used to mark the class as the controller in Spring 3.

The @RequestMapping annotation is used to map the request url. It is applied on the method.

Spring MVC Architecture

## **Spring MVC Execution Flow**

Step 1: First request will be received by DispatcherServlet.

Step 2: DispatcherServlet will take the help of HandlerMapping and get to know the Controller class name associated with the given request.

Step 3: So request transfer to the Controller, and then controller will process the request by executing appropriate methods and returns ModelAndView object (contains Model data and View name) back to the DispatcherServlet.

Step 4: Now DispatcherServlet send the model object to the ViewResolver to get the actual view page.

Step 5: Finally DispatcherServlet will pass the Model object to the View page to display the result.

Now Let us understand Spring MVC flow with hello world application Example.

First of all, we create Dynamic web project in eclipse and following are the required files for the same:

* JavaController.java
* welcomePage.jsp
* web.xml
* welcome-servlet.xml
* index.jsp

###### **JavaController.java**

JavaController.java

Java



|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | package com.codeNuclear;    import javax.servlet.http.HttpServletRequest;    import org.springframework.stereotype.Controller;  import org.springframework.ui.ModelMap;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.servlet.ModelAndView;      @Controller  public class JavaController  {  @RequestMapping("/codeNuclear")  public ModelAndView helloWorld(ModelMap model, HttpServletRequest request)  {       String message =  "Welcome to codeNuclear.com Spring MVC Sessions";     message += "<br>You Did it....!";       System.out.println(message);         model.addAttribute("message", message);       return new ModelAndView("welcomePage", "welcomeMessage", message);  }//ModelAndView closed  } |

###### **welcomePage.jsp**

welcomePage.jsp

Java



|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11 | <%@ page language="java" contentType="text/html; charset=ISO-8859-1"      pageEncoding="ISO-8859-1"%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <body>    <font face="verdana" size="2">       ${welcomeMessage}    </font>  </body>  </html> |

###### **web.xml**

web.xml

XHTML



|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | <?xml version="1.0" encoding="UTF-8"?>  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd" id="WebApp\_ID" version="2.5">    <display-name>SpringMvcExample</display-name>      <servlet>    <servlet-name>welcome</servlet-name>       <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>       <load-on-startup>1</load-on-startup>    </servlet>      <servlet-mapping>       <servlet-name>welcome</servlet-name>       <url-pattern>/</url-pattern>    </servlet-mapping>    <welcome-file-list>      <welcome-file>/</welcome-file>    </welcome-file-list>  </web-app> |

###### **welcome-servlet.xml**

servlet.xml

XHTML



|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:context="http://www.springframework.org/schema/context"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="  http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans-3.0.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context-3.0.xsd">    <context:component-scan base-package="com.codeNuclear" />    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">  <property name="prefix" value="/jsp/" />  <property name="suffix" value=".jsp" />  </bean>    </beans> |

###### **index.jsp**

index.jsp

Java



|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | <%@ page language="java" contentType="text/html; charset=ISO-8859-1"      pageEncoding="ISO-8859-1"%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <title>codeNuclear.com Spring MVC 3.x</title>  </head>  <body>    <font size="2px" face="verdana">    Welcome...     <a href="codeNuclear.html"><br> Click here to check the output.</a>  </font>    </body>  </html> |

## **Practical Execution Flow**

Step 1)Run the application, then index.jsp file will be executed then click on the link given (I have given <a href=”codeNuclear.html”>Click here to check the output.</a>).

Step 2)Once you click on that link, container will check the URL pattern at web.xml and passes the request to the DispatcherServlet.

Step 3)DispatcherServlet then passes that request to our controller class.

Step 4)DispatcherServlet verifies this ‘codeNuclear’ name with the string in @RequestMapping(“-“) in our controller class if same it will executes the ModelAndView method, which gives ModelAndView object as return type.

Java



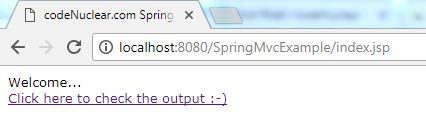
|  |  |
| --- | --- |
| 0  1  2 | return new ModelAndView("welcomePage", "welcomeMessage", message); |

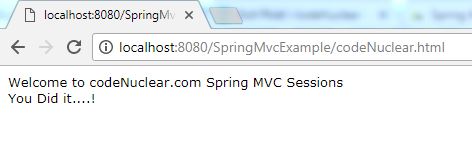
Means first argument is ‘View’ page name [ Where we are sending our result ], second, third arguments are key,values.

Step 5)So DispatcherServlet search for the name welcomePage in /jsp folder with extension .jsp , once the file was opened you can access the data by using the key welcomeMessage [2nd parameter in ModelAndView object].

Check welcomePage.jsp and printing the result by calling the key ${welcomeMessage} from welcomePage.jsp.

Output





14) how to give security in spring mvc

Ans- <https://docs.spring.io/spring-security/site/docs/current/guides/html5/hellomvc-javaconfig.html>

15)what is dependency injection

Ans-Dependency injection is basically providing the objects that an object needs (its dependencies) instead of having it construct them itself. It's a very useful technique for testing, since it allows dependencies to be mocked or stubbed out.

Dependencies can be injected into objects by many means (such as constructor injection or setter injection). One can even use specialized dependency injection frameworks (e.g. Spring) to do that, but they certainly aren't required. You don't need those frameworks to have dependency injection. Instantiating and passing objects (dependencies) explicitly is just as good an injection as injection by framework..

16) how many ways create auto wired.

Ans-  have @Autowired service which has to be used from within a static method. I know this is wrong but I cannot change the current design as it would require a lot of work, so I need some simple hack for that. I can't change randomMethod() to be non-static and I need to use this autowired bean. Any clues how to do that?

@Service

public class Foo {

public int doStuff() {

return 1;

}

}

public class Boo {

@Autowired

Foo foo;

public static void randomMethod() {

foo.doStuff();

}

}

17)which on better setter based or constructor based DI.

Ans- A class that takes a required dependency as a constructor argument can only be instantiated if that argument is provided (you should have a guard clause to make sure the argument is not null.) A constructor therefore enforces the dependency requirement whether or not you're using Spring, making it container-agnostic.

If you use setter injection, the setter may or may not be called, so the instance may never be provided with its dependency. The only way to force the setter to be called is using @Required or @Autowired , which is specific to Spring and is therefore not container-agnostic.

So to keep your code independent of Spring, use constructor arguments for injection.

18) how many ways create thread.

Ans- Threads can be created mainly in 3 different ways

1. **Extend** the java.lang.**Thread** class'

class SampleThread extends Thread {

//method where the thread execution will start

public void run(){

//logic to execute in a thread

}

//let’s see how to start the threads

public static void main(String[] args){

Thread t1 = new SampleThread();

Thread t2 = new SampleThread();

t1.start(); //start the first thread. This calls the run() method.

t2.start(); //this starts the 2nd thread. This calls the run() method.

}

}

1. **Implement** the java.lang.**Runnable** interface

class A implements Runnable{

@Override

public void run() {

// implement run method here

}

public static void main() {

final A obj = new A();

Thread t1 = new Thread(new A());

t1.start();

}

}

1. **Implement** the java.util.concurrent.**Callable** interface

class Counter implements Callable {

private static final int THREAD\_POOL\_SIZE = 2;

// method where the thread execution takes place

public String call() {

return Thread.currentThread().getName() + " executing ...";

}

public static void main(String[] args) throws InterruptedException,

ExecutionException {

// create a pool of 2 threads

ExecutorService executor = Executors

.newFixedThreadPool(THREAD\_POOL\_SIZE);

Future future1 = executor.submit(new Counter());

Future future2 = executor.submit(new Counter());

System.out.println(Thread.currentThread().getName() + " executing ...");

//asynchronously get from the worker threads

System.out.println(future1.get());

System.out.println(future2.get());

}

}

Favor Callable interface with the Executor framework for thread pooling.

The Runnable or Callable interface is preferred over extending the Thread class

19) which one beater implement the interface or extend thread.

Ans- Threading is a facility to allow multiple tasks to run concurrently within a single process. Threads are independent, concurrent execution through a program, and each thread has its own stack.

In Java threads can be implemented in two ways.

1)by Extending Thread Class

e.g.

class ThreadDemo extends Thread{

public void run(){

System.out.println("thread is running...");

}

public static void main(String args[]){

ThreadDemo t1=new ThreadDemo();

t1.start();

}

}

Output:thread is running...

2)by Implementing Runnable Interface

e.g.

class RunnableDemo implements Runnable{

public void run(){

System.out.println("thread is running...");

}

public static void main(String args[]){

RunnableDemo rd=new RunnableDemo();

Thread t1 =new Thread(rd);

t1.start();

}

}

Output:thread is running...

Creating a thread in Java

difference between extends thread and implements runnable

- The limitation with "extends Thread" approach is that if you extend Thread, you can not extend other class . Java does not support multiple inheritance.

While with implement runnable we can extend other class also.

- In "implements Runnable" , we are creating a different Runnable class for a specific task. It gives us the freedom to reuse the specific behavior task whenever required.

"extends Thread" contains both thread and job specific behavior code. Hence once thread completes execution , it can not be restart again.

- "implements Runnable" makes the code loosely-coupled and easier to read .Because the code is split into two classes . Thread class for the thread specific code and your Runnable implementation class for your task that should be run by a thread code.

"extends Thread" makes the code tightly coupled . Single class contains the thread code as well as the task that needs to be done by the thread.

there are many other difference also.

As per he above difference second approach has many advantages over the first approach so second approach is preferable.

[Pratik Dhanvani](https://www.quora.com/profile/Pratik-Dhanvani)

20) how to connect dispatcher servlet to ViewandModel.

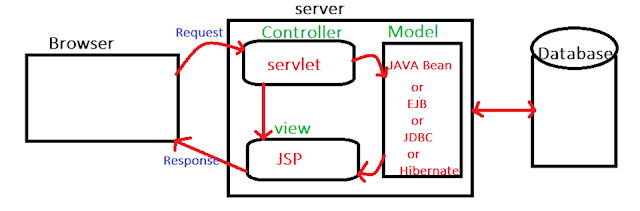
Ans- <https://dzone.com/articles/how-spring-mvc-really-works>

21) Explain Ur project Flow.

Ans- Project Architecture Means  what are the layers in our project with flow diagram. Here tier can be referred as a 'layer'.  suppose our project contains 3 layer like client layer, business layer and data layer then we draw the all layers flow.  
  
In IT industry N-Tier architecture is an industry-proven software architecture model. It is suitable to support enterprise level client-server applications by   
providing solutions on scalability, security, fault tolerance, re-usability and maintainability. It helps developers to create flexible and reusable applications.

**The Three-tier Architecture is divided into 3 layers:**  
  
Presentation layer(Client or Browser)  
Business layer(Struts or Spring) we used Server(either Tomcat (or) weblogic )  
Database layer( Oracle (or) MySQL)

presentation layer:  
  
It is also known as Client layer. Top most layer of an application. The main functionality of this layer is to communicate with Application layer. For example, login page of Facebook where an end user could see text boxes and buttons to enter user id, password and to click on sign-in. This is also called view of the application.  
  
Example: JSP  
  
**Business layer:**  
  
It is also called business logic layer or logical layer. As per the Facebook login page example, once user clicks on the login button, business layer interacts with Database layer and sends required information to the Presentation layer. It controls an application’s functionality by performing detailed processing. This layer acts as a mediator between the Presentation and the Database layer. Complete business logic will be written in this layer.  
  
**Example:** STRUTS,SPRING  
  
**Data layer:**  
  
The data is stored in this layer. Business layer communicates with Database layer to retrieve the data. It contains methods that connects the database and performs required action e.g.: insert, update, delete etc.  
  
**Example:** oracle, MySQL

[](https://2.bp.blogspot.com/-ApUODq-B3UI/WyoVLdMYCgI/AAAAAAAAEjo/xje8bZOC9IMsbqCd1toHpCrPIlb9X8EnACEwYBhgL/s1600/mvc.png)

**Step 1:** When ever we submitting the request from client(Browser) to Server component then controller receives the request from Browser. So you must remember every request will reach to controller component.

**Step 2:**The controller receives the request from Browser then Controller has to identify a particular model. Then it will execute business logic. Here model is might be java class or business logic's available. Here controller acts as mediator between view and the model. Finally, after performing some action on server side then controller identify respective view component. That means JSP page has view part.

**Step 3:** After executing model part if it is required to interact with database then model part interact with database.

**step 4:** Then view layer displays of the model to the end user. Then generate the require dynamic web page return to Browser. Finally view part send to the response to the client(Browser)

22) what is iterator

Ans- Iterators are used in [Collection framework](https://www.geeksforgeeks.org/collections-in-java-2/) in Java to retrieve elements one by one. There are three iterators.

[**Enumeration**](https://www.geeksforgeeks.org/iterators-in-java/#Enumeration)**:**

It is a interface used to get elements of legacy collections(Vector, Hashtable). Enumeration is the first iterator present from JDK 1.0, rests are included in JDK 1.2 with more functionality. Enumerations are also used to specify the input streams to a SequenceInputStream. We can create Enumeration object by calling elements() method of vector class on any vector object

// Here "v" is an Vector class object. e is of

// type Enumeration interface and refers to "v"

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

There are **two** methods in Enumeration interface namely :

// Tests if this enumeration contains more elements

**public boolean hasMoreElements();**

// Returns the next element of this enumeration

// It throws **NoSuchElementException**

// if no more element present

**public Object nextElement();**

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Java program to demonstrate Enumeration  import java.util.Enumeration;  import java.util.Vector;    public class Test  {      public static void main(String[] args)      {          // Create a vector and print its contents          Vector v = new Vector();          for (int i = 0; i < 10; i++)              v.addElement(i);          System.out.println(v);            // At beginning e(cursor) will point to          // index just before the first element in v          Enumeration e = v.elements();            // Checking the next element availability          while (e.hasMoreElements())          {              // moving cursor to next element              int i = (Integer)e.nextElement();                System.out.print(i + " ");          }      }  } |

Output:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

0 1 2 3 4 5 6 7 8 9

**Limitations of Enumeration :**

* Enumeration is for **legacy** classes(Vector, Hashtable) only. Hence it is not a universal iterator.
* Remove operations can’t be performed using Enumeration.
* Only forward direction iterating is possible.

[**Iterator**](https://www.geeksforgeeks.org/iterators-in-java/#Iterator)**:**

It is a **universal** iterator as we can apply it to any Collection object. By using Iterator, we can perform both read and remove operations. It is improved version of Enumeration with additional functionality of remove-ability of a element.

Iterator must be used whenever we want to enumerate elements in all Collection framework implemented interfaces like Set, List, Queue, Deque and also in all implemented classes of Map interface. Iterator is the **only** cursor available for entire collection framework.

Iterator object can be created by calling iterator() method present in Collection interface.

// Here "c" is any Collection object. itr is of

// type Iterator interface and refers to "c"

Iterator itr = c.**iterator**();

Iterator interface defines **three** methods:

// Returns true if the iteration has more elements

**public boolean hasNext();**

// Returns the next element in the iteration

// It throws **NoSuchElementException** if no more

// element present

**public Object next();**

// Remove the next element in the iteration

// This method can be called only once per call

// to next()

**public void remove();**

remove() method can throw two exceptions

* UnsupportedOperationException : If the remove operation is not supported by this iterator
* IllegalStateException : If the next method has not yet been called, or the remove method has already been called after the last call to the next method

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Java program to demonstrate Iterator  import java.util.ArrayList;  import java.util.Iterator;    public class Test  {      public static void main(String[] args)      {          ArrayList al = new ArrayList();            for (int i = 0; i < 10; i++)              al.add(i);            System.out.println(al);            // at beginning itr(cursor) will point to          // index just before the first element in al          Iterator itr = al.iterator();            // checking the next element availabilty          while (itr.hasNext())          {              //  moving cursor to next element              int i = (Integer)itr.next();                // getting even elements one by one              System.out.print(i + " ");                // Removing odd elements              if (i % 2 != 0)                 itr.remove();          }          System.out.println();          System.out.println(al);      }  } |

Output:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

0 1 2 3 4 5 6 7 8 9

[0, 2, 4, 6, 8]

**Limitations of Iterator :**

* Only forward direction iterating is possible.
* Replacement and addition of new element is not supported by Iterator.

[**ListIterator**](https://www.geeksforgeeks.org/iterators-in-java/#ListIterator)**:**

It is only applicable for List collection implemented classes like arraylist, linkedlist etc. It provides bi-directional iteration.

ListIterator must be used when we want to enumerate elements of List. This cursor has more functionality(methods) than iterator.

ListIterator object can be created by calling listIterator() method present in List interface.

// Here "l" is any List object, ltr is of type

// ListIterator interface and refers to "l"

ListIterator ltr = l.**listIterator**();

ListIterator interface extends Iterator interface. So all three methods of Iterator interface are available for ListIterator. In addition there are **six** more methods.

// Forward direction

// Returns true if the iteration has more elements

**public boolean hasNext();**

// same as next() method of Iterator

**public Object next();**

// Returns the next element index

// or list size if the list iterator

// is at the end of the list

**public int nextIndex();**

// Backward direction

// Returns true if the iteration has more elements

// while traversing backward

**public boolean hasPrevious();**

// Returns the previous element in the iteration

// and can throws **NoSuchElementException**

// if no more element present

**public Object previous();**

// Returns the previous element index

// or -1 if the list iterator is at the

// beginning of the list

**public int previousIndex();**

// Other Methods

// same as remove() method of Iterator

**public void remove();**

// Replaces the last element returned by

// next() or previous() with the specified element

**public void set(Object obj);**

// Inserts the specified element into the list at

// position before the element that would be returned

// by next(),

**public void add(Object obj);**

Clearly the three methods that ListIterator inherits from Iterator (hasNext(), next(), and remove()) do exactly the same thing in both interfaces. The hasPrevious() and the previous operations are exact analogues of hasNext() and next(). The former operations refer to the element before the (implicit) cursor, whereas the latter refer to the element after the cursor. The previous operation moves the cursor backward, whereas next moves it forward.

ListIterator has no current element; its cursor position always lies between the element that would be returned by a call to previous() and the element that would be returned by a call to next()

set() method can throw four exceptions

* UnsupportedOperationException – if the set operation is not supported by this list iterator
* ClassCastException : If the class of the specified element prevents it from being added to this list
* IllegalArgumentException : If some aspect of the specified element prevents it from being added to this list
* IllegalStateException : If neither next nor previous have been called, or remove or add have been called after the last call to next or previous

add() method can throw three exceptions

* UnsupportedOperationException : If the add method is not supported by this list iterator
* ClassCastException : If the class of the specified element prevents it from being added to this list
* IllegalArgumentException : If some aspect of this element prevents it from being added to this list

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Java program to demonstrate ListIterator  import java.util.ArrayList;  import java.util.ListIterator;    public class Test  {      public static void main(String[] args)      {          ArrayList al = new ArrayList();          for (int i = 0; i < 10; i++)              al.add(i);            System.out.println(al);            // at beginning ltr(cursor) will point to          // index just before the first element in al          ListIterator ltr = al.listIterator();            // checking the next element availabilty          while (ltr.hasNext())          {              //  moving cursor to next element              int i = (Integer)ltr.next();                // getting even elements one by one              System.out.print(i + " ");                // Changing even numbers to odd and              // adding modified number again in              // iterator              if (i%2==0)              {                  i++;  // Change to odd                  ltr.set(i);  // set method to change value                  ltr.add(i);  // to add              }          }          System.out.println();          System.out.println(al);      }  } |

Output:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

0 1 2 3 4 5 6 7 8 9

[1, 1, 1, 3, 3, 3, 5, 5, 5, 7, 7, 7, 9, 9, 9]

**Limitations of ListIterator :** It is the most powerful iterator but it is only applicable for List implemented classes, so it is not a universal iterator.

**Important Common Points**

**1 :** Please note that initially any iterator reference will point to the index just before the index of first element in a collection.

**2 :** We don’t create objects of Enumeration, Iterator, ListIterator because they are interfaces. We use methods like elements(), iterator(), listIterator() to create objects. These methods have anonymous [Inner classes](https://www.geeksforgeeks.org/inner-class-java/) that extends respective interfaces and return this class object. This can be verified by below code. For more on inner class refer

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Java program to demonstrate iterators references  import java.util.Enumeration;  import java.util.Iterator;  import java.util.ListIterator;  import java.util.Vector;    public class Test  {      public static void main(String[] args)      {          Vector v = new Vector();            // Create three iterators          Enumeration e = v.elements();          Iterator  itr = v.iterator();          ListIterator ltr = v.listIterator();            // Print class names of iterators          System.out.println(e.getClass().getName());          System.out.println(itr.getClass().getName());          System.out.println(ltr.getClass().getName());      }  } |

Output:

java.util.Vector$1

java.util.Vector$Itr

java.util.Vector$ListItr

24) explain annotations in spring

## Ans- Spring Annotations

* [Spring framework](https://www.journaldev.com/16922/spring-framework) implements and promotes the principle of control inversion (IOC) or dependency injection (DI) and is in fact an IOC container.
* Traditionally, Spring allows a developer to manage bean dependencies by using XML-based configuration.
* There is an alternative way to define beans and their dependencies. This method is a Java-based configuration.
* Unlike the XML approach, Java-based configuration allows you to manage bean components programmatically. That’s why Spring annotations were introduced.

In this article we will explore most commonly used Spring Annotations and also look at some example program.

### Spring Annotations List

Some of the spring core framework [annotations](https://www.journaldev.com/721/java-annotations) are:

1. @Configuration: Used to indicate that a class declares one or more @Bean methods. These classes are processed by the Spring container to generate bean definitions and service requests for those beans at runtime.
2. @Bean: Indicates that a method produces a bean to be managed by the Spring container. This is one of the most used and important spring annotation. @Bean annotation also can be used with parameters like name, initMethod and destroyMethod.
   * name – allows you give name for bean
   * initMethod – allows you to choose method which will be invoked on context register
   * destroyMethod – allows you to choose method which will be invoked on context shutdown

For example:

@Configuration

public class AppConfig {

@Bean(name = "comp", initMethod = "turnOn", destroyMethod = "turnOff")

Computer computer(){

return new Computer();

}

}

public class Computer {

public void turnOn(){

System.out.println("Load operating system");

}

public void turnOff(){

System.out.println("Close all programs");

}

}

1. @PreDestroy and @PostConstruct are alternative way for bean initMethod and destroyMethod. It can be used when the bean class is defined by us. For example;
2. public class Computer {
3. @PostConstruct
4. public void turnOn(){
5. System.out.println("Load operating system");
6. }
7. @PreDestroy
8. public void turnOff(){
9. System.out.println("Close all programs");
10. }
11. }
12. @ComponentScan: Configures component scanning directives for use with @Configuration classes. Here we can specify the base packages to scan for spring components.
13. @Component: Indicates that an annotated class is a “component”. Such classes are considered as candidates for auto-detection when using annotation-based configuration and classpath scanning.
14. @PropertySource: provides a simple declarative mechanism for adding a property source to Spring’s Environment. There is a similar annotation for adding an array of property source files i.e @PropertySources.
15. @Service: Indicates that an annotated class is a “Service”. This annotation serves as a specialization of @Component, allowing for implementation classes to be autodetected through classpath scanning.
16. @Repository: Indicates that an annotated class is a “Repository”. This annotation serves as a specialization of @Component and advisable to use with [DAO](https://www.journaldev.com/16813/dao-design-pattern) classes.
17. @Autowired: [Spring @Autowired annotation](https://www.journaldev.com/2623/spring-autowired-annotation) is used for automatic injection of beans. Spring @Qualifier annotation is used in conjunction with Autowired to avoid confusion when we have two of more bean configured for same type.

### Spring MVC Annotations

Some of the important Spring MVC annotations are:

1. @Controller
2. @RequestMapping
3. @PathVariable
4. @RequestParam
5. @ModelAttribute
6. @RequestBody and @ResponseBody
7. @RequestHeader and @ResponseHeader

You can learn more about them at [Spring MVC Tutorial](https://www.journaldev.com/2433/spring-mvc-tutorial).

### Spring Transaction Management Annotations

@Transactional is the spring declarative transaction management annotation, read more at [Spring MVC Hibernate](https://www.journaldev.com/3531/spring-mvc-hibernate-mysql-integration-crud-example-tutorial).

### Spring Security Annotations

@EnableWebSecurity is used with @Configuration class to have the Spring Security configuration defined, read more at [Spring Security Example](https://www.journaldev.com/2715/spring-security-example-tutorial).

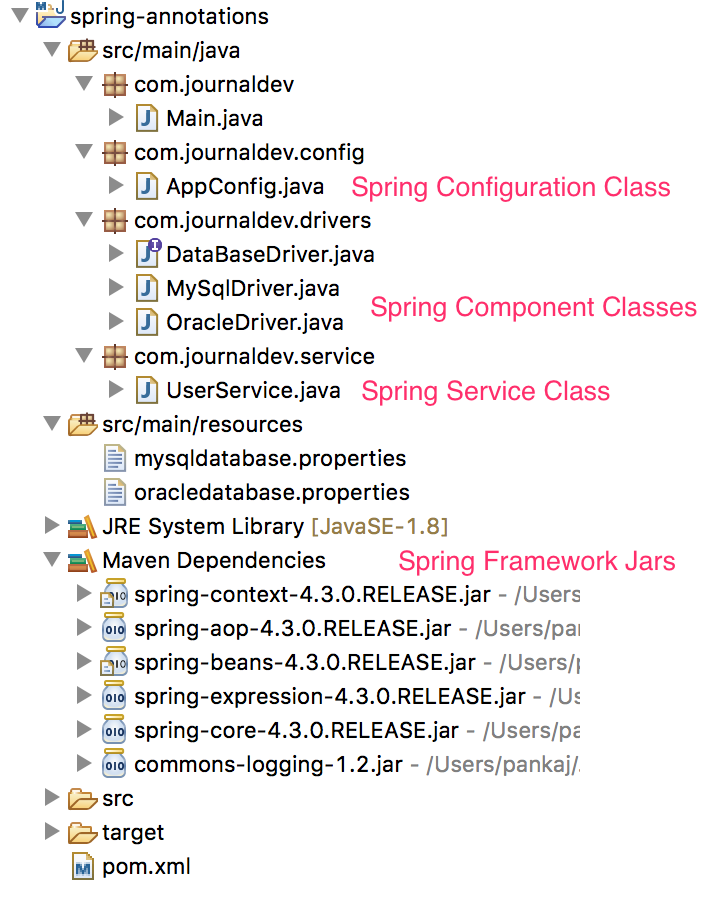
### Spring Boot Annotations

1. @SpringBootApplication
2. @EnableAutoConfiguration

Read more at [Spring Boot Example](https://www.journaldev.com/8225/spring-boot-initilizr-web-interface).

### Spring Annotations Example

Let’s look at a simple example where we will use Spring annotations in our application. Below image illustrates my Spring Annotations Example project.



#### Spring Framework Dependencies

I have created the maven project and added Spring Core Framework dependencies.

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.journaldev.spring</groupId>

<artifactId>spring-annotations</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>Spring Annotations</name>

<properties>

<spring.framework>4.3.0.RELEASE</spring.framework>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>${spring.framework}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>${spring.framework}</version>

</dependency>

</dependencies>

</project>

This will pull up all the spring core jars for our project.

#### Component Classes

Next step is to create component classes. Here I am imitating multiple database components, one for MySQL and another for Oracle.

package com.journaldev.drivers;

public interface DataBaseDriver {

public String getInfo();

}

DataBaseDriver is the base [interface](https://www.journaldev.com/1601/interface-in-java) that we will implement.

package com.journaldev.drivers;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.context.annotation.PropertySource;

import org.springframework.stereotype.Component;

@Component

@PropertySource("classpath:mysqldatabase.properties")

public class MySqlDriver implements DataBaseDriver {

@Value("${databaseName}")

private String databaseName;

@Value("${disableStatementPooling}")

private String disableStatementPooling;

public String getInfo() {

return "[ Driver: mySql" +

", databaseName: " + databaseName +

", disableStatementPooling: " + disableStatementPooling +

" ]";

}

}

Notice the use of @Component annotation to indicate spring framework to treat this class as a Component. We are also using @PropertySource and @Value annotations, Spring will use these at runtime to inject and set these variable values from specified property file. Below is the properties declared in mysqldatabase.properties file.

databaseName=school

disableStatementPooling=true

package com.journaldev.drivers;

public class OracleDriver implements DataBaseDriver {

protected String url;

protected String user;

protected String password;

protected String driver;

protected Integer port;

public String getUrl() {

return url;

}

public String getUser() {

return user;

}

public void setUser(String user) {

this.user = user;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public String getDriver() {

return driver;

}

public void setDriver(String driver) {

this.driver = driver;

}

public Integer getPort() {

return port;

}

public void setPort(Integer port) {

this.port = port;

}

public void setUrl(String url) {

this.url = url;

}

public String getInfo() {

return "[ Driver: Oracle" +

", url: " + url +

", port; " + port +

", user: " + user +

", password: " + password +

", driver: " + driver +

" ] ";

}

}

OracleDriver is a simple bean, we will use service class to inject properties to this bean.

#### Spring Service Class

package com.journaldev.service;

import com.journaldev.drivers.DataBaseDriver;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Qualifier;

import org.springframework.stereotype.Service;

@Service

public class UserService {

@Autowired

@Qualifier("oracleDriver")

private DataBaseDriver dataBaseDriver;

public String getDriverInfo(){

return dataBaseDriver.getInfo();

}

}

Here we are using @Service annotation to indicate Spring framework to treat this as a Service class. Then we are using @Autowired and @Qualifier("oracleDriver") annotations to tell spring framework to inject bean named oracleDriver to class property dataBaseDriver. Note that we haven’t yet created this spring bean.

#### Spring Beans

Final step is to create our spring beans and configuration classes to glue everything together.

package com.journaldev.config;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.annotation.PropertySource;

import org.springframework.core.env.Environment;

import com.journaldev.drivers.DataBaseDriver;

import com.journaldev.drivers.MySqlDriver;

import com.journaldev.drivers.OracleDriver;

@Configuration

@ComponentScan("com.journaldev")

@PropertySource("classpath:oracledatabase.properties")

public class AppConfig {

@Autowired

Environment environment;

@Bean

DataBaseDriver oracleDriver() {

OracleDriver oracleDriver = new OracleDriver();

oracleDriver.setDriver(environment.getProperty("db.driver"));

oracleDriver.setUrl(environment.getProperty("db.url"));

oracleDriver.setPort(Integer.parseInt(environment.getProperty("db.port")));

oracleDriver.setUser(environment.getProperty("db.user"));

oracleDriver.setPassword(environment.getProperty("db.password"));

return oracleDriver;

}

@Bean

DataBaseDriver mysqlDriver() {

return new MySqlDriver();

}

}

Notice the bean definition for oracleDriver. In this method, we are reading properties from oracledatabase.properties file that is being set to environment variable by Spring framework.

Here is the properties defined in oracledatabase.properties file.

db.url=localhost

db.port=4444

db.user=vasiliy

db.password=yilisav

db.driver=driver\_name

Our spring annotations example project is ready to test. As a summary, we performed following steps:

1. Created maven project and added required spring dependencies.
2. Created component classes and inject properties from a resource file into it’s variable.
3. If we have a third party component, we can use Service class to inject dependencies into it. Just like we did for OracleDriver through UserService class.
4. Finally, we created Configuration class to define [spring beans](https://www.journaldev.com/2637/spring-bean-life-cycle) and set the base package to scan for spring component classes and configure them.

### Spring Annotations Example Testing

Here is our main class to test our Spring annotations example project.

package com.journaldev;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import org.springframework.context.support.AbstractApplicationContext;

import com.journaldev.config.AppConfig;

import com.journaldev.drivers.DataBaseDriver;

import com.journaldev.service.UserService;

public class Main {

public static void main(String[] args) {

AbstractApplicationContext appContext = new AnnotationConfigApplicationContext(AppConfig.class);

DataBaseDriver oracle = appContext.getBean("oracleDriver", DataBaseDriver.class);

DataBaseDriver mysql = appContext.getBean("mysqlDriver", DataBaseDriver.class);

System.out.println("Oracle driver info:");

System.out.println(oracle.getInfo());

System.out.println("MySQL driver info:");

System.out.println(mysql.getInfo());

System.out.println("UserService Information");

UserService userService = appContext.getBean(UserService.class);

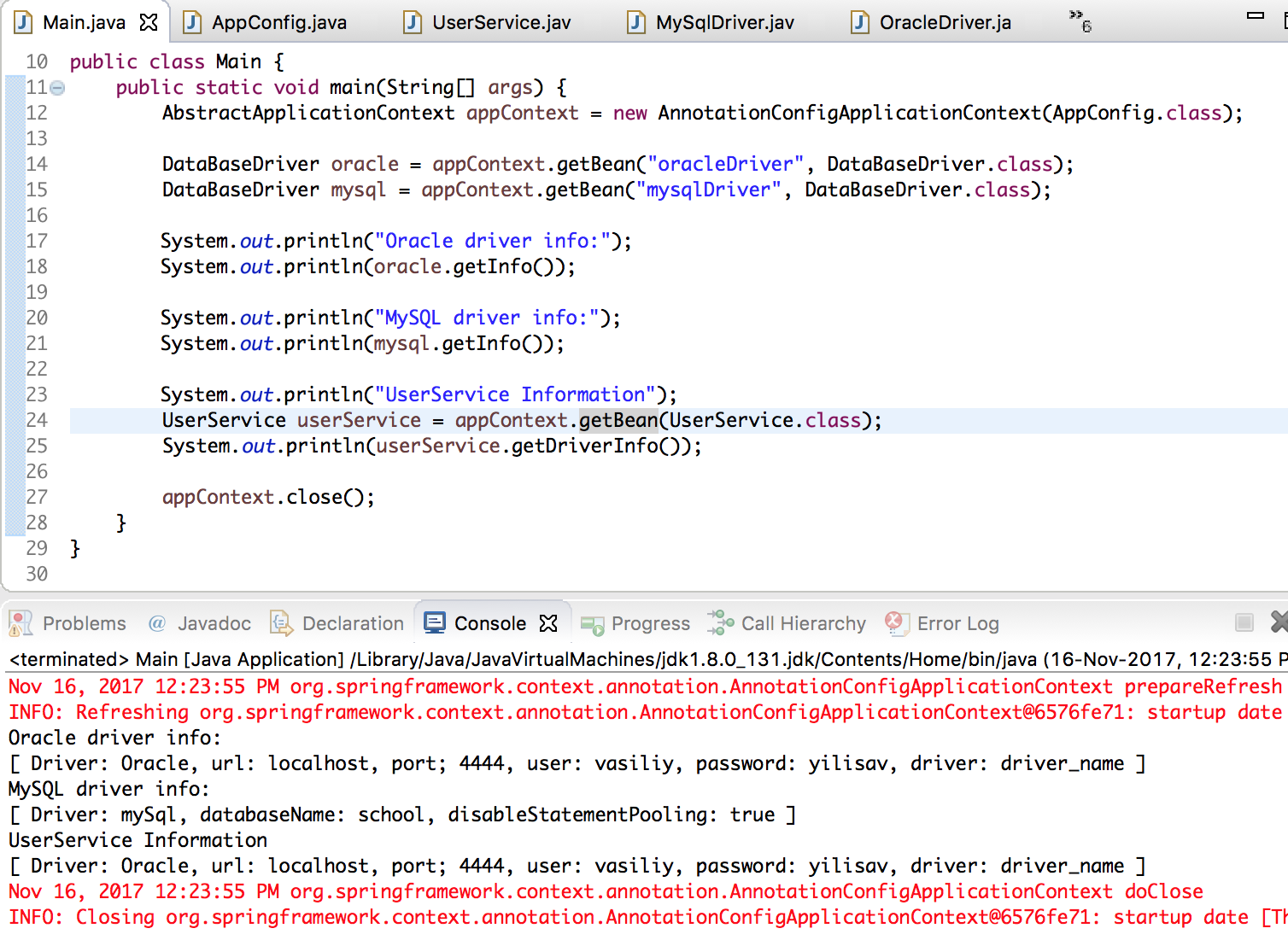
System.out.println(userService.getDriverInfo());

appContext.close();

}

}

Below image shows the output produced. Notice that we haven’t configured any logging framework, so all the spring framework logging is getting printed into console in red color.



25)Difference between SessionFactory and session.

Ans- Do we have any other differences other than the below? Also please validate whether the below are correct

1. SessionFactory objects are one per application and Session objects are one per client.
2. SessionFactory is to create and manage Sessions. Session is to provide a CRUD interface for mapped classes, and also access to the more versatile Criteria API.
3. SessionFactory is thread safe where as Session is not thread safe

26) Difference between save and persist

Ans- persist() is well defined. It makes a transient instance persistent. However, it doesn't guarantee that the identifier value will be assigned to the persistent instance immediately, the assignment might happen at flush time. The spec doesn't say that, which is the problem I have with persist() .

persist() also guarantees that it will not execute an INSERT statement if it is called outside of transaction boundaries. This is useful in long-running conversations with an extended Session/persistence context.

A method like persist() is required.

save() does not guarantee the same, it returns an identifier, and if an INSERT has to be executed to get the identifier (e.g. "identity" generator, not "sequence"), this INSERT happens immediately, no matter if you are inside or outside of a transaction. This is not good in a long-running conversation with an extended Session/persistence context.

27)write a program how many words are repeated in your name.

Ans-

|  |
| --- |
| import java.io.\*;  import java.util.\*;    class Test {        static int findRepeatFirstN2(String s)      {            // this is O(N^2) method          int p = -1, i, j;          for (i = 0; i < s.length(); i++)          {              for (j = i + 1; j < s.length(); j++)              {                  if (s.charAt(i) == s.charAt(j))                  {                      p = i;                      break;                  }              }              if (p != -1)                  break;          }            return p;      }        // Driver code      static public void main (String[] args)      {          String str = "geeksforgeeks";          int pos = findRepeatFirstN2(str);            if (pos == -1)              System.out.println("Not found");          else          System.out.println( str.charAt(pos));      }  }    // This code is contributed by anuj\_67. |

**Output:**