Q1.Write a program how do you use implementation of particular collection in your bean tag? using iterators print collection elements?

==========>https://www.tutorialspoint.com/spring/spring_injecting_collection.htm

```
<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

       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">

  <!-- Definition for javaCollection -->

  <bean id = "javaCollection" class = "com.tutorialspoint.JavaCollection">

    <!-- results in a setAddressList(java.util.List) call -->

    <property name = "addressList">

      <list>

        <value>INDIA</value>

        <value>Pakistan</value>

        <value>USA</value>

        <value>USA</value>

      </list>

    </property>

  </bean>

</beans>
```

Q2.Find the all pairs of elements sum is equal to given number in an array?

```
int[]array={0,1,1,2,3,4,5};

int count=0;

for(int i=0;i<array.length;i++){

        if(array[count]+array[i])

}
```

Q3.Methods present in set interface

A Set is a Collection that cannot contain duplicate elements.It models the mathematical set abstraction. The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited. Set also adds a stronger contract on the behavior of the equals and hashCode operations, allowing Set instances to be compared meaningfully even if their implementation types differ.

1. **add( )** Adds an object to the collection.

2. **clear( )** Removes all objects from the collection.

3. **contains( )** Returns true if a specified object is an element within the collection.

4. **isEmpty(**) Returns true if the collection has no elements.

5. **iterator( )** Returns an Iterator object for the collection, which may be used to retrieve an object.

6. **remove( )** Removes a specified object from the collection.

7. **size( )** Returns the number of elements in the collection.

Q4.Object class and methods

The Object class defines the basic state and behavior that all objects must have,such as the ability to compare oneself to another object, to convert to a string, to wait on a condition variable, to notify other objects that a condition variable has changed, and to return the object's class.

1. toString()
2. hashCode()
3. equals()
4. getClass()
5. finalize()
6. clone()
7. wait()
8. notify()
9. notifyAll()

Q5.Thread life cycle.

Thread:

A multi-threaded program contains two or more parts that can run concurrently and each part can handle a different task at the same time making optimal use of the available resources

LifeCycle::

1.created & begins lifecycle and still not in execution

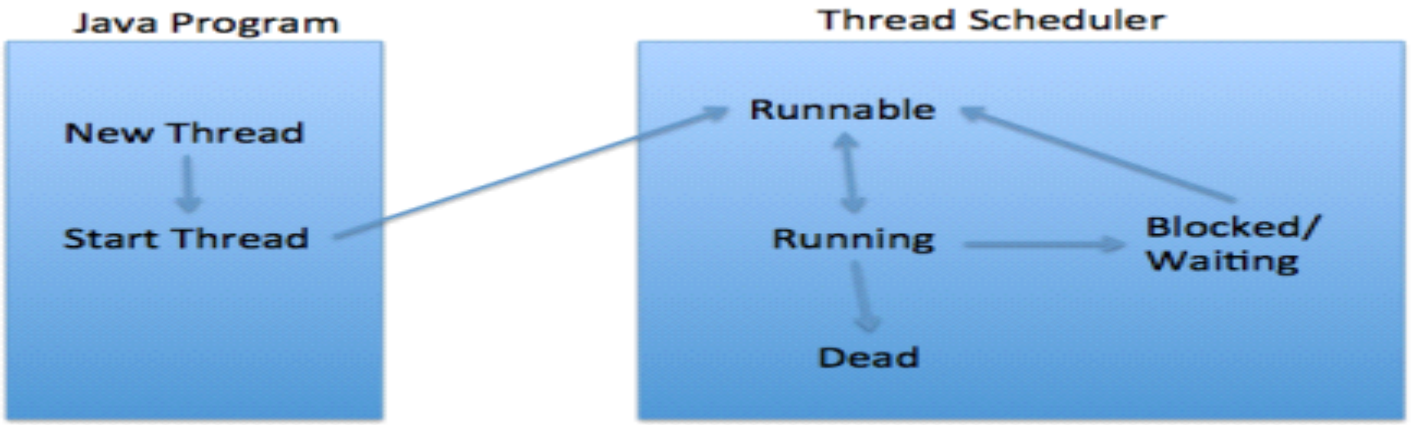
2.started and begins work under runnable state

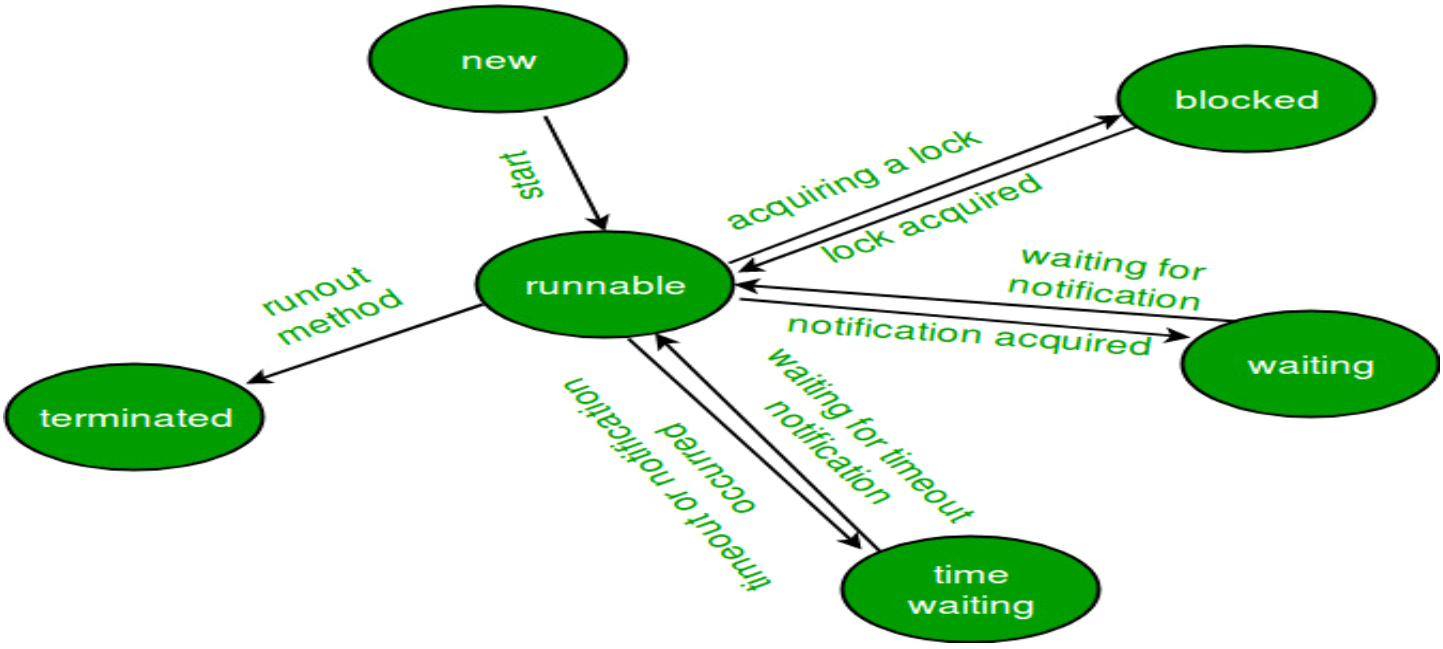
3.thread waits for another thread to perform a task.

comes back to the runnable state only when another thread signals it to continue executing.

4.in waiting state for fixed amount of time

5.thread is dead





Q6.Collection and collections differeces

Collection is a top level interface of java collection framework where as Collections is an utility class.

Collection Interface :

Collection is a root level interface of the Java Collection Framework. Most of the classes in Java Collection Framework inherit from this interface. List, Set and Queue are main sub interfaces of this interface. JDK doesn't provide any direct implementations of this interface. But, JDK provides direct implementations of it's sub interfaces. ArrayList, Vector, HashSet, LinkedHashSet, PriorityQueue are some indirect implementations of Collection interface. Map interface, which is also a part of java collection framework, doesn't inherit from Collection interface. Collection interface is a member of java.util package.

Collections Class:

Collections is an utility class in java.util package. It consists of only static methods which are used to operate on objects of type Collection. For example, it has the method to find the maximum element in a collection, it has the method to sort the collection, it has the method to search for a particular element in a collection. Below is the list of some important methods of Collections class.

	Collections	Collection
1	Class	Interface
2	Contains only static methods	Contains static and nonstatic methods
3	Used to operate on objects of type Collection	
4	Represents a single unit of objects	Its a Framework
5	Utiliy class that provide helping methods only	all operation like create and insert
6	It is a single class	Collection has several implamentation like list,queue and set
7	Methods Collections.max() Collections.min() Collections.sort() Collections.shuffle() Collections.synchronizedCollection() Collections.binarySearch() Collections.disjoint() Collections.copy() Collections.reverse()	Methods 1 public boolean add(Object element) 2 public boolean addAll(Collection c) 3 public boolean remove(Object element) 4 public boolean removeAll(Collection c) 5 public boolean retainAll(Collection c) 6 public int size() 7 public void clear() 8 public boolean contains(Object element) 9 public boolean containsAll(Collection c) 10 public Iterator iterator() 11 public Object[] toArray() 12 public boolean isEmpty() 13 public boolean equals(Object element) 14 public int hashCode()

Q7. JDBC Connections Steps

1. Register the driver class Class.forName("oracle.jdbc.driver.OracleDriver");

2. Creating connection

Connection con= DriverManager.getConnection("jdbc:oracle:thin:localhost:1521:xe","user","password");

3. Creating statement Statement stmt=con.createStatement();

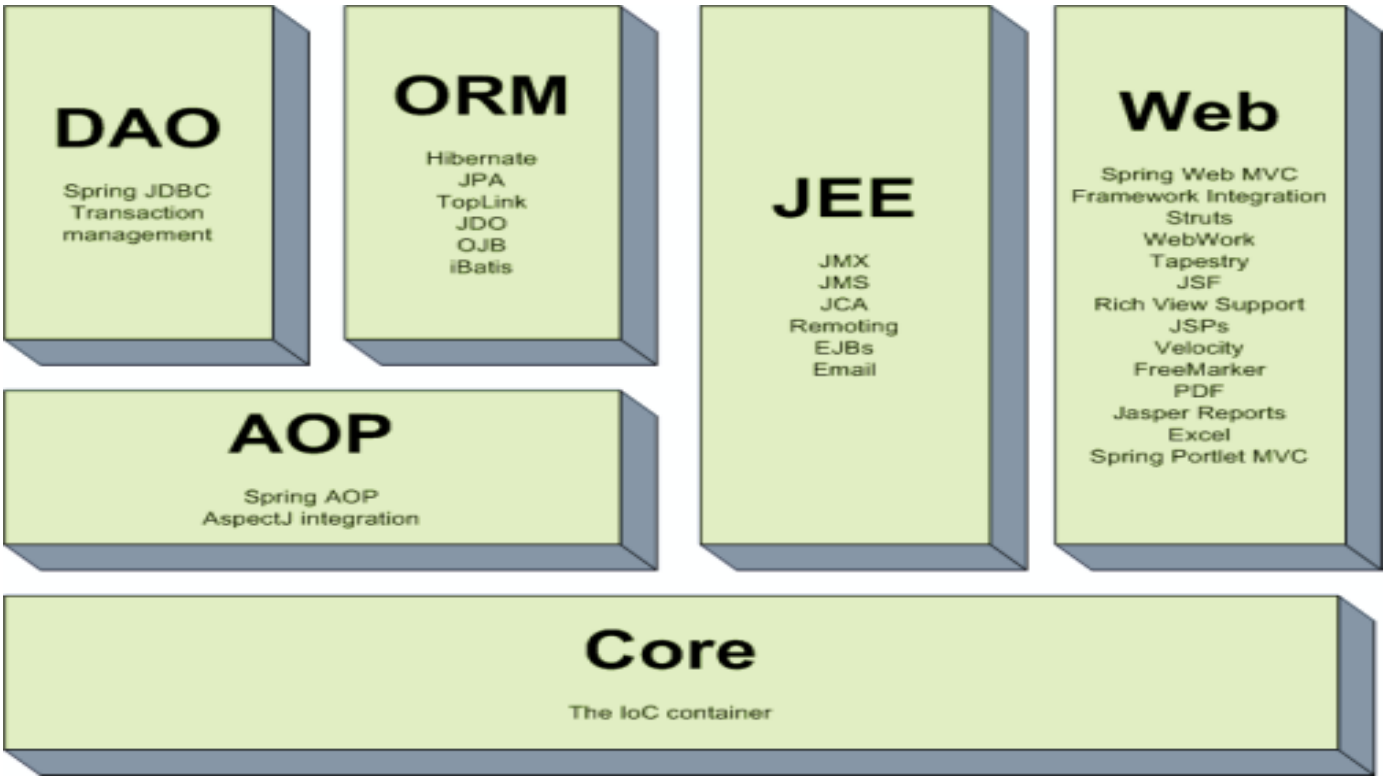
4. Executing queries ResultSet rs=stmt.executeQuery("select * from emp");

5. Closing connection con.close();

Q8. Given array of elements [1,2,34,45,4,6,7,8,9] store these elements in table (3x3)

```
int size=3;    int [] arr={1,2,34,45,4,6,7,8,9};   int [][] result=new int[size][size];

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

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

          result[i][j]=arr[i][j];

      }

 }
```

Q9. Spring modules and versions?



The Core Module: Provides the Dependency Injection (DI) feature which is the basic concept of the Spring framework. This module contains the ***BeanFactory,*** an implementation of Factory Pattern which creates the bean as per the configurations provided by the developer in an XML file.

AOP Module: The Aspect Oriented Programming module allows developers to define method-interceptors and point cuts to keep the concerns apart. It is configured at run time so the compilation step is skipped. It aims at declarative transaction management which is easier to maintain.

DAO Module: This provides an abstraction layer to the low level task of creating a connection, releasing it etc. It also maintains a hierarchy of meaningful exceptions rather than throwing complicated error codes from specific database vendors. It uses AOP to manage transactions. Transactions can also be managed programmatically.

ORM Module: Spring doesn't provides its own ORM implementation but offers integrations with popular Object Relational mapping tools like Hibernate, iBATIS SQL Maps, Oracle TopLink and JPA etc.

JEE Module: It also provides support for JMX, JCA, EJB and JMS etc. In lots of cases, JCA (Java EE Connection API) is much like JDBC, except where JDBC is focused on database JCA focus on connecting to legacy systems.

Web Module: Spring comes with MVC framework which eases the task of developing web applications. It also integrates well with the most popular MVC frameworks like Struts, Tapestry, JSF, Wicket etc.

Version	**Date**
1.0	2004
2.0	2006

3.0	2009
4.0	2013
5.0	2017

Q10. Hibernate version in your project.

Q11. How many types hibernate relationships.

- One-To-One
- Many-To-One
- Many-To-Many
- One-To-Many

Q12. Is Spring beans are thread safe?

- **Spring** framework does not do anything under the hood concerning the multi-threaded behavior of a singleton **bean**. It is the developer's responsibility to deal with concurrency issue and **thread safety** of the singleton **bean**. While practically, most **spring beans** have no mutable state, and as such are trivially **thread safe**

Q13. Scopes of spring bean

Singleton: This scopes the bean definition to a single instance per Spring IoC container (default)

Prototype: This scopes a single bean definition to have any number of object instances.

Request: This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext.

Session: This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

Global-session: This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

Q14. **Spring Bean Life Cycle & Methods**

Life of traditional java objects starts on calling new operator which instantiates the object and finalize() method is getting called when the object is eligible for garbage collection. Life cycle of Spring beans are different as compared to traditional java objects.

Spring framework provides the following ways which can be used to control the lifecycle of bean:

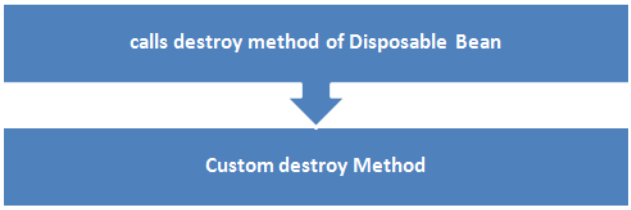
1. InitializingBean and DisposableBean callback interfaces
2. Bean Name, bean factory and Application Context Aware interfaces for specific behavior
3. custom init() and destroy() methods in bean configuration file

For annotation based configurations -

@PostConstruct and @PreDestroy annotations

Below diagram shows the complete lifecycle methods (from instantiate to Ready To use)

Following diagram shows the method calling at the time of destruction.



Sources: http://www.wideskills.com/spring/spring-bean-lifecycle

Q15. What is RequestDispatcher: **It** is an interface, implementation of which defines an object which can dispatch request to any resources(such as HTML, Image, JSP, Servlet) on the server.

Methods: 1> void `forward(ServletRequest request, ServletResponse response)`

2> void `include(ServletRequest request, ServletResponse response)`

Example:
```
RequestDispatcher rs = request.getRequestDispatcher("hello.html");
rs.forward(request,response);
```

Q16. What is Implicit object ?

Those are objects which are already been placed in the scope by the servlet container, so that it's accessible by EL (Expression Language), such as the `PageContext`, `HttpServletRequest#getParameter()`, `HttpServletRequest#getHeader()` and so on. Those are just for convenience so that you don't need to use old-fahioned *scriptlets* to grab them.

So instead of for example

```
<%= pageContext.getSession().getMaxInactiveInterval() %><br>
<%= request.getParameter("foo") %><br>
<%= request.getHeader("user-agent") %><br>
<%  for (Cookie cookie : request.getCookies()) { // Watch out with NPE!
        if (cookie.getName().equals("foo")) {
            out.write(cookie.getValue());
        }
    }
%><br>
```

you can just do

```
${pageContext.session.maxInactiveInterval}<br>
${param.foo}<br>
${header['user-agent']}<br>
${cookie.foo}<br>
```

JSP Implicit Objects are the Java objects that the JSP Container makes available to developers in each page and developer can call them directly without being explicitly declared. JSP Implicit Objects are also called pre-defined variables.

JSP supports **nine** Implicit Objects which are listed below:

- **request** : This is the **HttpServletRequest** object associated with the request.
- **response** : This is the **HttpServletResponse** object associated with the response to the client.
- **out** : This is the **PrintWriter** object used to write any data to buffer.
- **session** : This is the **HttpSession** object associated with the request.
- **application** : This is the **ServletContext** object associated with application context.
- **config** : This is the **ServletConfig** object associated with the page.
- **pageContext** : This encapsulates use of server-specific features like higher performance JspWriters.
- **page** : This is simply a synonym for this, and is used to call the methods defined by the translated servlet class.
- **Exception** : The Exception object allows the exception data to be accessed by designated JSP.

Q17. Give list of Controller classes in spring

Q18. SQL and its advantages:

Advantages of SQL:

i) SQL Queries can be used to retrieve large amounts of records from a database quickly and efficiently.

ii) SQL is used to view the data without storing the data into the object.

iii) SQL joins two or more tables and show it as one object to user.

iv) SQL databases use long-established standard, which is being adopted by ANSI & ISO. Non-SQL databases do not adhere to any clear standard.

v) Using standard SQL it is easier to manage database systems without having to write substantial amount of code.

vi) SQL restricts the access of a table so that nobody can insert the rows into the table.

Disadvantages of SQL?

i) Interfacing an SQL database is more complex than adding a few lines of code.

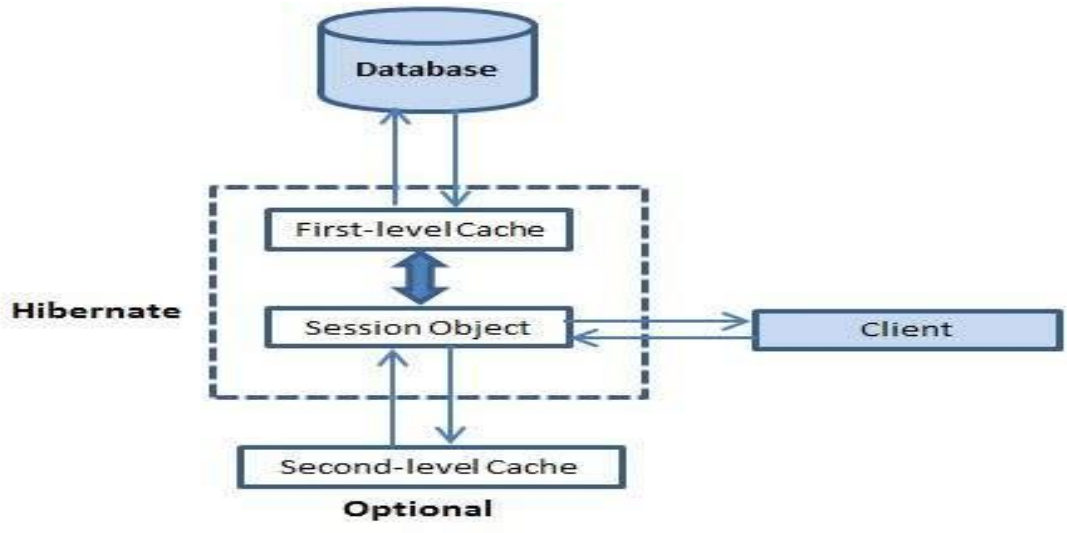
ii) When table is dropped view becomes inactive. It depends on the table objects.

iii) Although SQL databases conform to ANSI & ISO standards, some databases go for proprietary extensions to standard SQL to ensure vendor lock-in.

iv) It is an object so it occupies space.

Q19. What are available Cache in Hibernate ?

Caching is a mechanism to enhance the performance of a system. It is a buffer memory that lies between the application and the database. Cache memory stores recently used data items in order to reduce the number of database hits as much as possible.



First-level Cache

The first-level cache is the Session cache and is a mandatory cache through which all requests must pass. The Session object keeps an object under its own power before committing it to the database. If you issue multiple updates to an object, Hibernate tries to delay doing the update as long as possible to reduce the number of update SQL statements issued. If you close the session, all the objects being cached are lost and either persisted or updated in the database.

Second-level Cache

Second level cache is an optional cache and first-level cache will always be consulted before any attempt is made to locate an object in the second-level cache. The second level cache can be configured on a per-class and per-collection basis and mainly responsible for caching objects across sessions. Any third-party cache can be used with Hibernate. An **org.hibernate.cache.CacheProvider** interface is provided, which must be implemented to provide Hibernate with a handle to the cache implementation.

Query-level Cache

Hibernate also implements a cache for query resultsets that integrates closely with the second-level cache. This is an optional feature and requires two additional physical cache regions that hold the cached query results and the timestamps when a table was last updated. This is only useful for queries that are run frequently with the same parameters.

Q20. Difference Between :

ClassNotFoundException	NoClassDefFoundError
It is an exception. It is of type java.lang.Exception.	It is an error. It is of type java.lang.Error.
It occurs when an application tries to load a class at run time which is not updated in the classpath.	It occurs when java runtime system doesn't find a class definition, which is present at compile time, but missing at run time.
It is thrown by the application itself. It is thrown by the methods like Class.forName(), loadClass() and findSystemClass().	It is thrown by the Java Runtime System.
It occurs when classpath is not updated with required JAR files.	It occurs when required class definition is missing at runtime.

Q21. How many ways are there to create object ?

1. **Using new keywords** : **Ex >** `Employee emp1 = new Employee();`
2. **Using newInstance() method of Class class :**

```
Ex> Employeeemp2=(Employee)
            Class.forName("org.programming.mitra.exercises.Employee").newInstance();
OR  Employee emp2 = Employee.class.newInstance();
```

3. Using newInstance() method of Constructor class :

```
Constructor<Employee> constructor = Employee.class.getConstructor();
Employee emp3 = constructor.newInstance();
```

4. **Using clone() method:** **EX >** `Employee emp4 = (Employee) emp3.clone();`
5. **Using deserialization: Ex >**

```
ObjectInputStream in = new ObjectInputStream(new FileInputStream("data.obj"));
Employee emp5 = (Employee) in.readObject();
public class ObjectCreation {
```

Example of creating object with all ways:

```
public static void main(String[]args) throws Exception {
        // By using new keyword
        Employee emp1 = new Employee();
        emp1.setName("Naresh");
        System.out.println(emp1 + ", hashcode : " + emp1.hashCode());
        // By using Class class's newInstance() method
        Employee emp2 = (Employee)
Class.forName("org.programming.mitra.exercises.Employee").newInstance();
        // Or we can simply do this
        // Employee emp2 = Employee.class.newInstance();
        emp2.setName("Rishi");
        System.out.println(emp2 + ", hashcode : " + emp2.hashCode());
        // By using Constructor class's newInstance() method
        Constructor<Employee> constructor = Employee.class.getConstructor();
        Employee emp3 = constructor.newInstance();
        emp3.setName("Yogesh");
        System.out.println(emp3 + ", hashcode : " + emp3.hashCode());
        // By using clone() method
        Employee emp4 = (Employee) emp3.clone();
        emp4.setName("Atul");
        System.out.println(emp4 + ", hashcode : " + emp4.hashCode());
        // By using Deserialization
        // Serialization
        ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream("data.obj"));
        out.writeObject(emp4);
        out.close();
        //Deserialization
        ObjectInputStream in = new ObjectInputStream(new FileInputStream("data.obj"));
        Employee emp5 = (Employee) in.readObject();
        in.close();
        emp5.setName("Akash");
        System.out.println(emp5 + ", hashcode : " + emp5.hashCode());
    }
}
```

This program will give the following output:

```
Employee Constructor Called...
Employee [name=Naresh], hashcode : -1968815046
Employee Constructor Called...
Employee [name=Rishi], hashcode : 78970652
Employee Constructor Called...
Employee [name=Yogesh], hashcode : -1641292792
Employee [name=Atul], hashcode : 2051657
Employee [name=Akash], hashcode : 63313419
```

Q22. How HashMap works internally .

Lets understand it with step by step. Let us add below keys and its values into HashMap using put function.

scores.put (“Rohit”, 140); scores.put (“Dinesh”, 70); scores.put (“Dhoni”, 90); scores.put (“Kholi”, 100);

scores.put (“Sachin”, 150); scores.put (“Dravid”, 130);

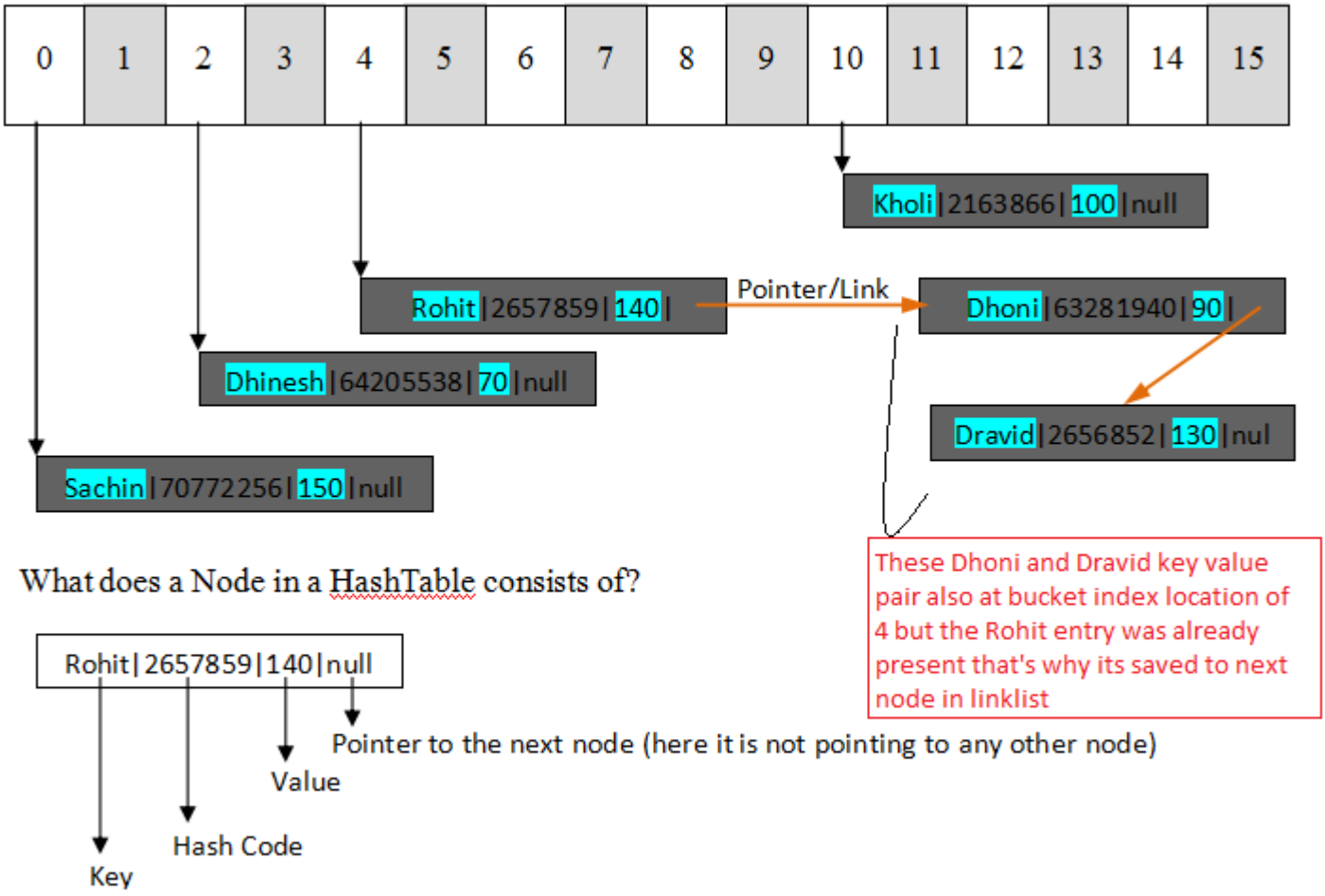
Let us understand at which location below key value pair will be saved into HashMap.

scores.put ("Rohit", 140);

- When you call the put function then it computes the hash code of the Key.

using **hash(k) function**, Lets suppose the Hash code of ("Rohit") is 2657860.

- ***[Most Important]*** Once the hash code is generated then HashMap find the bucket index value using a modular operation to find out where the key value is going to fit in the HashMap.
  - Index = Reminder of (hash code % size of HashMap which is 16 by default)
  - Index = 2657860%16 => 4
  - Index = 4, So 4 is the computed bucket index value where the entry will go sit as a node in the HashMap.
- Now bucket index value computed, so Key and Value pair will save at the bucket index 4 of the hashMap as a node as shown in the below picture, Please note: if the entry is already present then the value will save at next node in the linklist at 4 bucket location.
- Similarly all other entries will also enter into HashMao with same process of generating a hashcode and computing its corresponding index number.
- Java HashMap allows Null Keys (Key= Null) for which the hash code will be zero and it will always goes to index zero of the table.
- Shown below is how a HashMap will look like after entering different sets of Key value pairs.



Q23. What is CSS programming language?

Cascading Style Sheets (**CSS**) is a style sheet language used for describing the presentation of a document written in a markup language like HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript

Q24. How to use external javascript file?

- <script type="text/javascript" src="message.js"></script>

We can create external JavaScript file and embed it in many html page. It provides **code re usability** because single JavaScript file can be used in several html pages. An external JavaScript file must be saved by .js extension. It is recommended to embed all JavaScript files into a single file. It increases the speed of the webpage.

Q26. Write a program to implement hashcode and equals.

```
import java.util.HashMap;
public class MyHashcodeImpl {
     public static void main(String a[]){
         HashMap<Price, String> hm = new HashMap<Price, String>();
         hm.put(new Price("Banana", 20), "Banana");
         hm.put(new Price("Apple", 40), "Apple");
         hm.put(new Price("Orange", 30), "Orange");
         //creating new object to use as key to get value
         Price key = new Price("Banana", 20);
         System.out.println("Hashcode of the key: "+key.hashCode());
         System.out.println("Value from map: "+hm.get(key));
     }
}
 class Price{
     private String item;
     private int price;
     public Price(String itm, int pr){
         this.item = itm;
         this.price = pr;
     }
     public int hashCode(){
         System.out.println("In hashcode");
         int hashcode = 0;
         hashcode = price*20;
         hashcode += item.hashCode();
         return hashcode;
     }
     public boolean equals(Object obj){
         System.out.println("In equals");
         if (obj instanceof Price) {
             Price pp = (Price) obj;
             return (pp.item.equals(this.item) && pp.price == this.price);
         } else {
             return false;
         }
     }
      public String getItem() {
         return item;
     }
     public void setItem(String item) {
         this.item = item;
     }
     public int getPrice() {
         return price;
     }
```

```
    public void setPrice(int price) {
        this.price = price;
    }

    public String toString(){
        return "item: "+item+"  price: "+price;
    }
}
```

Output:

```
In hashcode
In hashcode
In hashcode
In hashcode
Hashcode of the key: 1982479637
In hashcode
In equals
Value from map: Banana
```

Q27. Write a program to check the given number is binary number or not?

```
package com.java2novice.algos;

public class MyBinaryCheck {

    public boolean isBinaryNumber(int binary){

        boolean status = true;
        while(true){
            if(binary == 0){
                break;
            } else {
                int tmp = binary%10;
                if(tmp > 1){
                    status = false;
                    break;
                }
                binary = binary/10;
            }
        }
        return status;
    }

    public static void main(String a[]){
        MyBinaryCheck mbc = new MyBinaryCheck();
        System.out.println("Is 1000111 binary? :"+mbc.isBinaryNumber(1000111));
        System.out.println("Is 10300111 binary? :"+mbc.isBinaryNumber(10300111));
    }
}
```

Output:

```
Is 1000111 binary? :true
Is 10300111 binary? :false
```

Q28. Sort a Stack using a temporary Stack

```
// Java program to sort a stack using
// a auxiliary stack.
import java.util.*;

class SortStack
{
    // This function return the sorted stack
    public static Stack<Integer> sortstack(Stack<Integer>
                                           input)
    {
        Stack<Integer> tmpStack = new Stack<Integer>();
        while(!input.isEmpty())
        {
            // pop out the first element
            int tmp = input.pop();

            // while temporary stack is not empty and
            // top of stack is greater than temp
            while(!tmpStack.isEmpty() && tmpStack.peek()
                                                > tmp)
            {
                // pop from temporary stack and
                // push it to the input stack
            input.push(tmpStack.pop());
            }

            // push temp in tempory of stack
            tmpStack.push(tmp);
        }
        return tmpStack;
    }


    // Driver Code
    public static void main(String args[])
    {
        Stack<Integer> input = new Stack<Integer>();
        input.add(34);
        input.add(3);
        input.add(31);
        input.add(98);
        input.add(92);
        input.add(23);

        // This is the temporary stack
        Stack<Integer> tmpStack=sortstack(input);
        System.out.println("Sorted numbers are:");

        while (!tmpStack.empty())
        {
            System.out.print(tmpStack.pop()+" ");
        }
    }
}
```

Output:

```
Sorted numbers are:
98 92 34 31 23 3
```

Basic Questions:

Can you make a constructor final?
Can we have static constructor?
What is final method?
What is a static function?
What is difference between static and dynamic?
Can you override a private method?
Can a static method be inherited?
Can a class be static?
Can you override a final method?
Can we overload static main method in Java?
Can we inherit a static variable in Java?
What is meant by a static variable?
Why static member variables must be defined outside the class?
Can you call a static method from a non static method?
Can you call a static method from an instance?

Can we declare a static variable inside a non static method?
Can I use static variable in non static method?

Can you use this in a static method?
Can you access static method from non static method?
Can I call a static method from an object?
Can you call a static method from an instance method?
Why is the main method static?
Can we have static constructor?

- Can you use this in a static method?
- Why super is not used in static context?
- Can we use this and super together?
- Why do we use static keyword in Java?
- Can you access non static variable in static method?
- Why is the main method static?
- Can we use this () and super () both in a constructor?
- Can we change the return type in method overloading?
- Can we execute a program without main () method?
- What is a static variable?
- Can we use non static variable in static method?
- Can you call a non static method from a static method?
- What is the use of a static variable?
- Why we have to write public static void main?
- Can we overload static method in Java?
- Why we can not override static method?
- What do you mean by static variables?

- What is a static method?
- What is the difference between static and instance methods?
- Can you make a constructor final?