Theory Question:

1. What is JVM and explain me the Java memory allocation

It is an acronym for java virtual machine. It provides the run time environment for the java byte code to be executed. It is platform dependent meaning to say that it is different for different operating systems, its implementation is known as java runtime environment or JRE. Whenever we run the Java class an instance of JVM is created, coming to the functionalities- it loads code, verifies code and executes code.

2. What is Polymorphism and encapsulation?

Polymorphism is an important Object oriented concept and widely used in Java and other programming language. Polymorphism is briefly described as "one interface, many implementations". Polymorphism is a characteristic of being able to assign a different meaning or usage to something in different contexts - specifically, to allow an entity such as a variable, a function, or an object to have more than one form. Ability to process an Object differently depending upon thier Data Type or Class. Ability of an object to take more than one form is known as polymorphism.

Types of POLYMORPHISM

1) Compile time Polymorphism - The compiler know that the way of execution of the program , means - which method have to be invoked at compilation time. It is achieved by

Method Overloading - Same method name, may be different FORMAL ARGUMENT LIST, may be different data type, may be different return type. Method Overloading perform only inside of the class.

2) Run-Time Polymorphism - The compiler doesn't know the way of execution of the program. It will take the decision for execute the program at Run-Time. It is achieved by

Method Overriding - Same Method name, same signatures, similar DataType, Return type also should be the same. Method Overriding perform at Subclass.

Encapsulation is the mechanism that binds together code and data it manipulates and keeps both safe from outside interface and misuse. Inheritence is the process by which one object acquires the properties of another object. Polymorphism is the feature that allows one interface to be used for general class actions.

The whole idea behind encapsulation is to hide the implementation details from users. If a data member is private it means it can only be accessed within the same class. No outside class can access private data member (variable) of other class. However if we setup public getter and setter methods to update (for e.g. void setSSN(int ssn))and read (for e.g. int getSSN()) the private data fields then the outside class can access those private data fields via public methods. This way data can only be accessed by public methods thus making the private fields and their implementation hidden for outside classes. That's why encapsulation is known as **data hiding.**

3. What is method overloading and Method over riding?

Method Overloading is a feature that allows a class to have two or more methods having same name, but different argument lists. may be different data type, may be different return type. Method Overloading perform only inside of the class.

Declaring a method in subclass or child class which is already present in parent class is known as method overriding. Hence, the method name and argument lists are same. The parent class method is called as over ridden method.

4. Why string is Immutable?

1) The string is Immutable in Java because String objects are cached in String pool. Since cached String literals are shared between multiple clients there is always a risk, where one client's action would affect all another client. For example, if one client changes the value of String "Test" to "TEST", all other clients will also see that value as explained in the first example. Since caching of String objects was important from performance reason this risk was avoided by making String class Immutable. At the same time, String was made final so that no one can compromise invariant of String class e.g. Immutability, Caching, hashcode calculation etc by extending and overriding behaviors. Another reason of why String class is immutable could die due to HashMap.

2)String has been widely used as parameter for many Java classes e.g. for opening network connection, you can pass hostname and port number as string , you can pass database URL as string for opening database connection, you can open any file in java by passing name of file as argument to File I/O classes.

In case, if String is not immutable, this would lead serious security threat , I mean some one can access to any file for which he has authorization, and then can change the file name either deliberately or accidentally and gain access of those file. Because of immutability, you don't need to worry about those kind of threats. This reason also gel with, Why String is final in Java, by makingjava.lang.String final, Java designer ensured that no one overrides any behavior of String class.

3)Since String is immutable it can safely be shared between many threads, which is very important for multithreaded programming and to avoid any synchronization issues in Java, Immutability also makes String instance thread-safe in Java, means you don't need to synchronize String operation externally. Another important point to note about String is memory leak caused by substring, which is not a thread related issues but something to be aware of.

4) Another reason of Why String is immutable in Java is to allow String to cache its hashcode , being immutable String in Java caches its hashcode, and do not calculate every time we call hashcode method of String, which makes it very fast as hashmap key to be used in hashmap in

Java. This one is also suggested by Jaroslav Sedlacek in comments below. In short because String is immutable, no one can change its contents once created which guarantees hashCode of String to be same on multiple invocation.

5) Another good reason of Why String is immutable in Java suggested by Dan Bergh Johnsson on comments is: The absolutely most important reason that String is immutable is that it is used by the class loading mechanism, and thus have profound and fundamental security aspects. Had String been mutable, a request to load "java.io.Writer" could have been changed to load "mil.vogoon.DiskErasingWriter".

5. What is the difference between String and String buffer?

String Buffer is A thread-safe, mutable sequence of characters, for concatenation opeeations this is the right choice

String is also thread safe, but it is mmutable sequence of characters.

String builder is a mutale sequence of characters but it is not thread safe, when using a single thread for a particular operation this is the right choice

6. What is the difference between array and array list?

First and Major difference between Array and ArrayList in Java is that Array is a fixed length data structure while ArrayList is a variable length collection class. You can not change length of Array once created in Java but ArrayList re-size itself when gets full depending upon capacity and load factor. Since ArrayList is internally backed by Array in Java, any resize operation in ArrayList will slow down performance as it involves creating new Array and copying content from old array to new array.

Another difference between Array and ArrayList in Java is that you can not use generics along with Array, as Array instance knows about what kind of type it can hold and throws ArrayStoreException, if you try to store type which is not convertible into type of Array. ArrayList allows you to use Generics to ensure type-safety.

You can also compare Array vs ArrayList on How to calculate length of Array or size of ArrayList. All kinds of Array provideslength variable which denotes length of Array while ArrayList provides size() method to calculate size of ArrayList in Java.

Generic in Java is added to provide compile time type-safety of code and removing risk of ClassCastException at runtime.

One more major difference between ArrayList and Array is that, you can not store primitives in ArrayList, it can only contain Objects. While Array can contain both primitives and Objects in

Java. Though Autoboxing of Java 5 may give you an impression of storing primitives in ArrayList, it actually automatically converts primitives to Object. e.g.

ArrayList<Integer>integerList = new ArrayList<Integer>();

integerList.add(1); //here we are not storing primitive in ArrayList, instead autoboxing will convert int primitive to Integer object.
Java provides add() method to insert element into ArrayList and you can simply use assignment operator to store element into Array e.g. In order to store Object to specified position use
One more difference on Array vs ArrayList is that you can create instance of ArrayList without specifying size, Java will create Array List with default size but its mandatory to provide size of Array while creating either directly or indirectly by initializing Array while creating it. By the way you can also initialize ArrayList while creating it.

7. What is the difference between hash map and Hash table?

- One of the major differences between HashMap and Hashtable is that HashMap is non-synchronized whereas Hashtable is synchronized, which means Hashtable is thread-safe and can be shared between multiple threads but HashMap cannot be shared between multiple threads without proper synchronization. Java 5 introduced ConcurrentHashMap which is an alternative of Hashtable and provides better scalability than Hashtable in Java.Synchronized means only one thread can modify a hash table at one point of time. Basically, it means that any thread before performing an update on a hashtable will have to acquire a lock on the object while others will wait for lock to be released.

- The HashMap class is roughly equivalent to Hashtable, except that it permits nulls. (HashMap allows null values as key and value whereas Hashtable doesn't allow nulls).

- The third significant difference between HashMap vs Hashtable is that Iterator in the HashMap is a fail-fast iterator while the enumerator for the Hashtable is not and throw ConcurrentModificationException if any other Thread modifies the map structurally by adding or removing any element except Iterator's own remove() method. But this is not a guaranteed behavior and will be done by JVM on best effort. This is also an important difference between Enumeration and Iterator in Java.

- One more notable difference between Hashtable and HashMap is that because of thread-safety and synchronization Hashtable is much slower than HashMap if used in Single threaded environment. So if you don’t need synchronization and HashMap is only used by one thread, it out perform Hashtable in Java.

- HashMap does not guarantee that the order of the map will remain constant over time.

Note that HashMap can be synchronized by

Map m = Collections.synchronizedMap(hashMap);

8. What is a vector in Java?
Vector implements a dynamic array. It is similar to ArrayList, but with two differences:

- Vector is synchronized.
- Vector contains many legacy methods that are not part of the collections framework.

Vector proves to be very useful if you don't know the size of the array in advance or you just need one that can change sizes over the lifetime of a program.

9. What is set in java?

IT IS AN INTERFACE OF THE COLLECTIONS FRAMEWORK that can contain no duplicate elements for example if e1 and e2 are two elements of the collections framework then e1.equals(e2) shud never be true and it can contain atmost one null value.

10. What is an abstract class?

1. An **abstract class** is a **class** that is declared **abstract** —it may or may not include **abstract** methods. **Abstract classes** cannot be instantiated, but they can be subclassed. When an **abstract class** is subclassed, the subclass usually provides implementations for all of the **abstract**methods in its parent **class**.
2. **Abstraction** is a process of hiding the implementation details and showing only functionality to the user.
3. Another way, it shows only important things to the user and hides the internal details for example sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery.
4. Abstraction lets you focus on what the object does instead of how it does it

11. What is an interface?

An *interface* in Java is similar to a class, but the body of an interface can include only abstract methods and final fields (constants). A class implementsan interface by providing code for each method declared by the interface.

12. Why Java is Platform independent?

"*Write once, run everywhere*."

Before going into the detail,first you have to understand what is the mean of platform? Platform consists of the computer hardware(mainly architecture of the microprocessor) and OS. Platform=hardware+Operating System

Anything that is platform indepedent can run on any operating system and hardware.

Java is platform indepedent so java can run on any operating system and hardware. Now question is how it is platform independent?

This is because of the magic of Byte Code which is OS indepedent. When java compiler compile any code then it generate the byte code not the machine native code(unlike C compiler).Now this byte code need a interpreter to execute on a machine.This interpreter is JVM.So JVM read that byte code(that is machine indepedent) amd execute it. Different JVM is designed for different OS and byte code is able to run on different OS.

In case of C or C++(language that are not platform indepedent) compiler generate the .exe file that is OS depedent so when we run this .exe file on another OS it will not run because this file is OS depedent so is not compatible with the another OS.

Finally an intermediate OS indepedent Byte code make the java platform independent.

13. What are access modifiers? Give me an example?

Access level modifiers determine whether other classes can use a particular field or invoke a particular method. There are two levels of access control:

- At the top level—public, or *package-private* (no explicit modifier).
- At the member level—public, private, protected, or *package-private* (no explicit modifier).
- A class may be declared with the modifier public, in which case that class is visible to all classes everywhere. If a class has no modifier (the default, also known as *package-private*), it is visible only within its own package (packages are named groups of related classes — you will learn about them in a later lesson.)
- At the member level, you can also use the public modifier or no modifier (*package-private*) just as with top-level classes, and with the same meaning. For members, there are two

additional access modifiers: private and protected. The private modifier specifies that the member can only be accessed in its own class. The protected modifier specifies that the member can only be accessed within its own package (as with *package-private*) and, in addition, by a subclass of its class in another package.

- The following table shows the access to members permitted by each modifier.

Access Levels

Modifier	**Class**	**Package**	**Subclass**	**World**
Public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

14. What are java exceptions? give me an example

An *exception* is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.when an exception occurs within a method the method creates an exception object and hands it over to the run time environment, the exception object contains information about the exception and its type and then the runtime system searches for an appropriate exception handler associated with this exception if it does not find one, the code terminates. Creating an exception object and handing it over to the run time system is known as throwing an exception.

The first step in constructing an exception handler is to enclose the code that might throw an exception within a try block.

Each catch block is an exception handler that handles the type of exception indicated by its argument.

The finally block always executes when the try block exits.

The main difference between checked and unchecked exception is that the checked exceptions are checked at compile-time while unchecked exceptions are checked at runtime.

- SQLException
- IOException
- DataAccessException
- ClassNotFoundException
- InvocationTargetException

15. What is the difference between throws and throwable?

The Throwable class is the superclass of all errors and exceptions in the Java language. Only objects that are instances of this class (or one of its subclasses) are thrown by the Java Virtual Machine or can be thrown by the Java throw statement. Similarly, only this class or one of its sub classes can be the argument type in a catch clause.
The key word **throws** is used in method declaration, this specify what kind of exception [Throwable class] we may expect from this method.

16. What is the difference between Error and exception?

Exceptions are caused by our program/code.

Exceptions are recoverable, give filenotfoundexception as example

Errors are not caused by our program/code, these are due to lack of system resources.eg:if sufficient heap memory is not available it, outofmemoryerror, jvm crashes etc, errors are not recoverable.

Errors	Exceptions
1. Errors in java are of type java.lang.Error.	Exceptions in java are of type java.lang.Exception.
2. All errors in java are unchecked type.	Exceptions include both checked as well as unchecked type.
3. Errors happen at run time. They will not be known to compiler.	Checked exceptions are known to compiler where as unchecked exceptions are not known to compiler because they occur at run time.
4. It is impossible to recover from errors.	You can recover from exceptions by handling them through try-catch blocks.
5.Errors are mostly caused by the environment in which application is running.	Exceptions are mainly caused by the application itself.
Examples : java.lang.StackOverflowError, java.lang.OutOfMemoryError	Examples : Checked Exceptions : SQLException, IOException Unchecked Exceptions :

	ArrayIndexOutOfBoundException, ClassCastException, NullPointerException

17. What is the difference between Error, throwable and exception?

Errors are not caused by our program/code, these are due to lack of system resources.eg:if sufficient heap memory is not available it, outofmemoryerror, jvm crashes etc, errors are not recoverable.
The Throwable class is the superclass of all errors and exceptions in the Java language. Only objects that are instances of this class (or one of its subclasses) are thrown by the Java Virtual Machine or can be thrown by the Java throw statement. Similarly, only this class or one of its sub classes can be the argument type in a catch clause.
Exceptions are caused by our program/code. Exceptions are recoverable, give filenotfoundexception as example

18. What are collection APIs, give me an example

The Collection API is a set of classes and interfaces that support operation on collections of objects. These classes and interfaces are more flexible, more powerful, and more regular than the vectors, arrays, and hashtables if effectively replaces.
Example of classes: HashSet, HashMap, ArrayList, LinkedList, TreeSet and TreeMap. Example of interfaces: Collection, Set, List and Map.
Primitive types are types that the compiler gives to us, non primitive types are type that’s we create, Collections cannot store primitive types, they are part of the java.util.Collection package,
Collections is an interface that contains a group of objects as a single entity, suppose we want an interface that allowa duplicates and the nsertion order has to be preserved we make use of the list interface, it can implemented with the classes like arraylist, linkedlist, stack and vector set-interface of collections framework that is not ordered,means insertion order is not preserved but ifu want the insertion order to be preserved we must go for sortedset interface and duplicates are not allowed, implementation classes are hashset and linkedhashset map-it is a set of keys to look up values(names and numbers). With linked list we can add elements in the middle, wid array we cannot do that, these are both data structures and not part of the collections framework

19. What is the difference between final and finally?

No.	final	finally
1)	Final is used to apply restrictions on class, method and variable. Final class can't be	Finally is used to place important code, it will be executed whether exception is handled or not.

	inherited, final method can't be overridden and final variable value can't be changed.	
2)	Final is a keyword.	Finally is a block.

20. Will java supports multiple inheritance?

Java does not support multiple inheritance. It is just to remove ambiguity, because multiple inheritance can cause ambiguity in few scenarios. Hence, this enforces simplicity which is one of the feature of Java.

21. What are the different types of interface? (Ans List, set, Queue)

22. What are wrapper class? Give me an example.

Wrapper class in java provides the mechanism to convert primitive into object and object into primitive.
There are mainly two uses with wrapper classes.

- To convert simple data types into objects, that is, to give object form to a data type; here constructors are used.
- To convert strings into data types (known as parsing operations), here methods of type parseXXX() are used.

Example: Int k = 100;

```
Integer it1 = new Integer(k);
```

The int data type k is converted into an object, it1 using Integer class. The it1 object can be used in Java programming wherever k is required an object.

The following code can be used to unwrap (getting back int from Integer object) the object it1.

```
int m = it1.intValue();

System.out.println(m*m); // prints 10000
```

intValue() is a method of Integer class that returns an int data type.

23. What is boxing and unboxing in Java? Explain with an example

Boxing is the automatic conversion that the java compiler makes between the primitive types and their corresponding object wrapper classes.

Example: Integer obj=new int(5) wrapping/boxing

Converting an object of a wrapper type (Integer) to its corresponding primitive (int) value is called unboxing.

Int j=obj.intvalue()unwrapping/unboxing

24. Explain for each loop

It is an evolution of for loop designed specifically for working with arrays, lists. It does not have an explicit counter, its always 1 unlike in a for loop.

Syntax for (type var:name of array or list)

25. What are iterators, explain with an example

Iterators are used to traverse the list interface of the collection framework , specifically arraylists and vectors. It can be used to delete elements from a list or add elements to the list with the help of list iterator.

26. How do you access Private variables in different class?

If we setup public getter and setter methods to update the private data fields then the outside class can access those private data fields via public methods. This way data can only be accessed by public methods thus making the private fields and their implementation hidden for outside classes.

27. Prepare for one java program to write on the board

28. What is Constructor Over loading?

Constructor overloading is way of having more than one constructor which does different-2 tasks. For e.g. Vector class has 4 types of constructors. If you do not want to specify the initial capacity and capacity increment then you can simply use default constructor of Vector class like this Vector v = new Vector(); however if you need to specify the capacity and increment then you call the parameterized constructor with two int args like this: Vector v= new Vector(10, 5)

29. Without using sync key word how do you perform synchronization?

30. What is Super keyword ? when and where do you use it ?

The super keyword in java is a reference variable that is used to refer immediate parent class object.

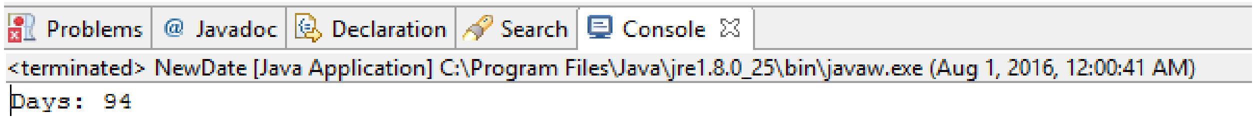
Usage of java super Keyword

- super is used to refer immediate parent class instance variable.
- super() is used to invoke immediate parent class constructor.
- super is used to invoke immediate parent class method.

Programing Questions:

1. Find out the number of days in between two given dates ?

```
package com.test;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.concurrent.TimeUnit;

public class NewDate{

public static void main(String[] args){
SimpleDateFormat myFormat = new SimpleDateFormat("dd MM yyyy");
String inputString1 = "23 01 2016";
String inputString2 = "27 04 2016";

try {
    Date date1 = myFormat.parse(inputString1);
    Date date2 = myFormat.parse(inputString2);
    long diff = date2.getTime() - date1.getTime();
    System.out.println ("Days: " + TimeUnit.DAYS.convert(diff,
TimeUnit.MILLISECONDS));
} catch (ParseException e) {
    e.printStackTrace();
}
}
}
```

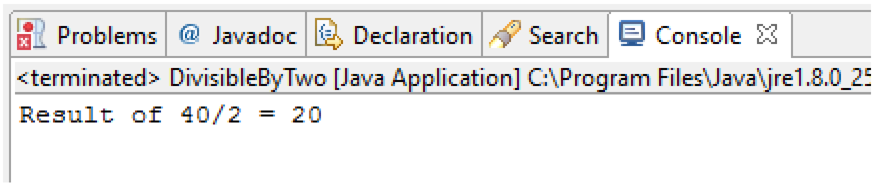


```
<terminated> NewDate [Java Application] C:\Program Files\Java\jre1.8.0_25\bin\javaw.exe (Aug 1, 2016, 12:00:41 AM)
Days: 94
```

2. How to divide a number by 2 without using / operator?

```
package com.test;

public class DivisibleByTwo {

public static void main(String[] args) {
```

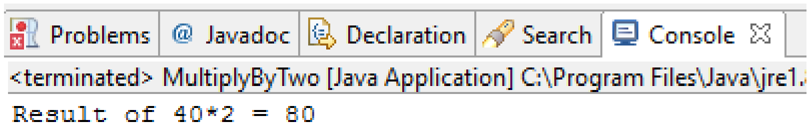
```
int num = 40;

int dv = devideByTwo (num);
System.out.println ("Result of " + num + "/2 = " + dv);
        }
public static int devideByTwo (int num) {
return (num >> 1);
        }
}
```



```
<terminated> DivisibleByTwo [Java Application] C:\Program Files\Java\jre1.8.0_25
Result of 40/2 = 20
```

3. How to multiply a number by 2 without using * operator?

```
package com.test;

public class MultiplyByTwo  {
public static void main (String[] args) {

    int num = 40;
    int mul = multiplyByTwo (num);
    System.out.println ("Result of " + num + "*2 = " + mul);
}
      public static int multiplyByTwo (int num) {
    return (num << 1);
}


}
```

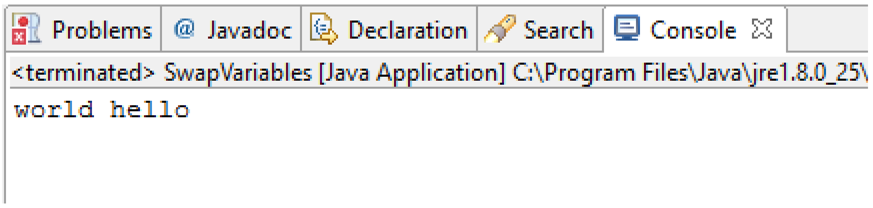


```
<terminated> MultiplyByTwo [Java Application] C:\Program Files\Java\jre1.
Result of 40*2 = 80
```

4. How to swap two variables, by using pass by reference method ?

```
package com.test;
public class SwapVariables {
public static void main(String[] args)  {

   String a[] = {"hello", "world"};
    swap(a);
    System.out.println(a[0] + " " + a[1]);
  }
    static void swap(String[] a) {
    String t = a[0];
   a[0] = a[1];
```

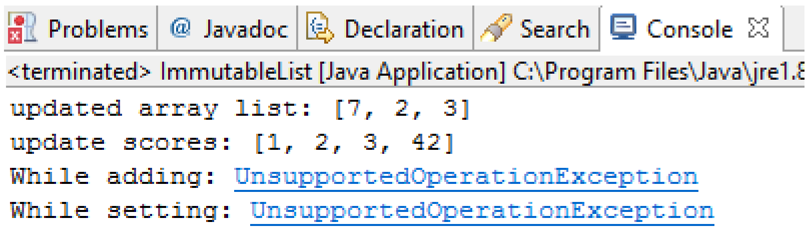
```
    a[1] = t;
   }
   }
```



```
<terminated> SwapVariables [Java Application] C:\Program Files\Java\jre1.8.0_25\
world hello
```

5. How to make a list immutable?

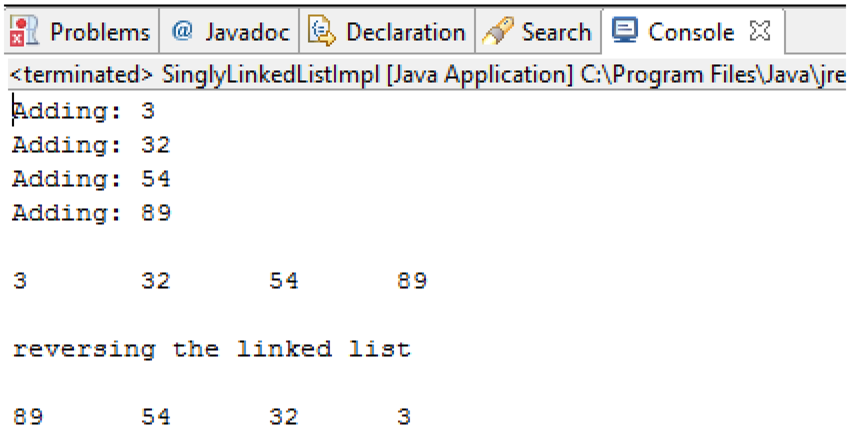
```
package com.test;

import java.util.Arrays;
import java.util.Collections;
import java.util.LinkedList;
import java.util.List;

public class ImmutableList {

    public static void main(String[] args) {
        List<Integer> arrayList = Arrays.asList(1, 2, 3);
        List<Integer> scores = new LinkedList<>(arrayList);

        System.out.println("scores: " + scores);

        arrayList.set(0, 7);
        System.out.println("updated array list: " + arrayList);

        scores.add(42);
        System.out.println("update scores: " + scores);

        scores = Collections.unmodifiableList(scores);

        try {
            scores.add(7);
        } catch (UnsupportedOperationException e) {
            System.out.println("While adding: " +
e.getClass().getSimpleName());
        }

        try {
            scores.set(0, 7);
        } catch (UnsupportedOperationException e) {
            System.out.println("While setting: " +
e.getClass().getSimpleName());
        }
    }


}
```


Problems @ Javadoc Declaration Search Console

```
<terminated> ImmutableList [Java Application] C:\Program Files\Java\jre1.8
updated array list: [7, 2, 3]
update scores: [1, 2, 3, 42]
While adding: UnsupportedOperationException
While setting: UnsupportedOperationException
```

6. **Write a sample code to reverse Singly Linked List by iterating through it only once.**

```
package com.test;

      public class SinglyLinkedListImpl<T> {

          private Node<T> head;

          public void add(T element){

              Node<T> nd = new Node<T>();
              nd.setValue(element);
              System.out.println("Adding: "+element);
              Node<T> tmp = head;
              while(true){
                  if(tmp == null){
                      //since there is only one element, both head and
                      //tail points to the same object.
                      head = nd;
                      break;
                  } else if(tmp.getNextRef() == null){
                      tmp.setNextRef(nd);
                      break;
                  } else {
                      tmp = tmp.getNextRef();
                  }
              }
          }

          public void traverse(){

              Node<T> tmp = head;
              while(true){
                  if(tmp == null){
                      break;
                  }
                  System.out.print(tmp.getValue()+"\t");
                  tmp = tmp.getNextRef();
              }
          }

          public void reverse(){
              System.out.println("\nreversing the linked list\n");
              Node<T> prev = null;
              Node<T> current = head;
              Node<T> next = null;
              while(current != null){
                  next = current.getNextRef();
                  current.setNextRef(prev);
                  prev = current;
                  current = next;
              }
              head = prev;
          }

          public static void main(String a[]){
              SinglyLinkedListImpl<Integer> sl = new
SinglyLinkedListImpl<Integer>();
              sl.add(3);
              sl.add(32);
              sl.add(54);
              sl.add(89);
              System.out.println();
              sl.traverse();
              System.out.println();
              sl.reverse();
              sl.traverse();
          }
      }

      class Node<T> implements Comparable<T> {

          private T value;
          private Node<T> nextRef;

          public T getValue() {
              return value;
          }
          public void setValue(T value) {
              this.value = value;
          }
          public Node<T> getNextRef() {
              return nextRef;
          }
          public void setNextRef(Node<T> ref) {
              this.nextRef = ref;
          }
          @Override
          public int compareTo(T arg) {
              if(arg == this.value){
                  return 0;
              } else {
                  return 1;
              }
          }
      }
```

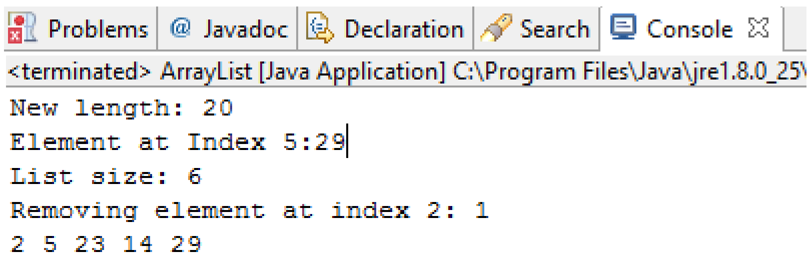


```
<terminated> SinglyLinkedListImpl [Java Application] C:\Program Files\Java\jre
Adding: 3
Adding: 32
Adding: 54
Adding: 89

3       32      54      89

reversing the linked list

89      54      32      3
```

7. Write a program to implement ArrayList and Linked list

```
package com.test;

import java.util.Arrays;

public class ArrayList {

    private Object[] myStore;
    private int actSize = 0;

    public ArrayList(){
        myStore = new Object[10];
    }

    public Object get(int index){
        if(index < actSize){
            return myStore[index];
        } else {
            throw new ArrayIndexOutOfBoundsException();
        }
    }

    public void add(Object obj){
        if(myStore.length-actSize <= 5){
            increaseListSize();
        }
        myStore[actSize++] = obj;
    }

    public Object remove(int index){
        if(index < actSize){
            Object obj = myStore[index];
            myStore[index] = null;
            int tmp = index;
            while(tmp < actSize){
                myStore[tmp] = myStore[tmp+1];
                myStore[tmp+1] = null;
                tmp++;
            }
            actSize--;
```

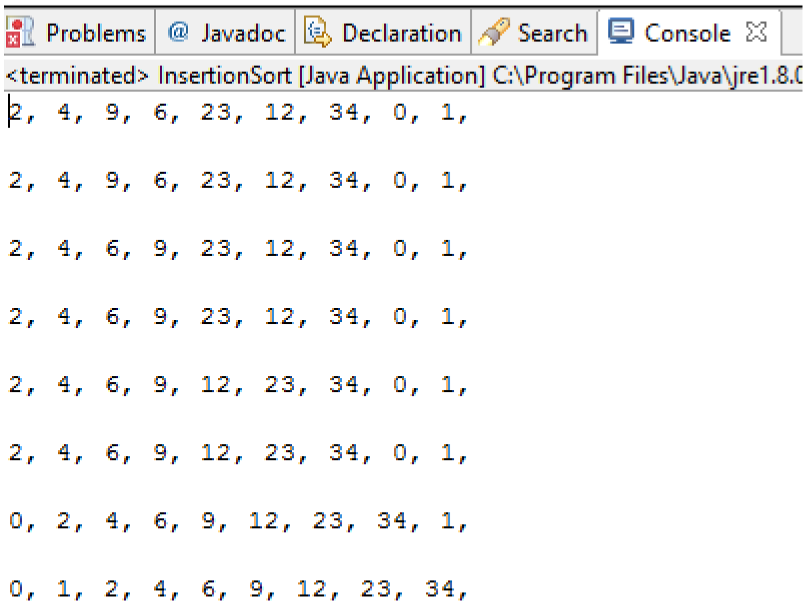
```java
            return obj;
        } else {
            throw new ArrayIndexOutOfBoundsException();
        }

    }

    public int size(){
        return actSize;
    }

    private void increaseListSize(){
        myStore = Arrays.copyOf(myStore, myStore.length*2);
        System.out.println("\nNew length: "+myStore.length);
    }

    public static void main(String a[]){
        ArrayList mal = new ArrayList();
        mal.add(new Integer(2));
        mal.add(new Integer(5));
        mal.add(new Integer(1));
        mal.add(new Integer(23));
        mal.add(new Integer(14));
        for(int i=0;i<mal.size();i++){
            System.out.print(mal.get(i)+" ");
        }
        mal.add(new Integer(29));
        System.out.println("Element at Index 5:"+mal.get(5));
        System.out.println("List size: "+mal.size());
        System.out.println("Removing element at index 2: "+mal.remove(2));
        for(int i=0;i<mal.size();i++){
            System.out.print(mal.get(i)+" ");
        }
    }
}
```



```
<terminated> ArrayList [Java Application] C:\Program Files\Java\jre1.8.0_25\
New length: 20
Element at Index 5:29
List size: 6
Removing element at index 2: 1
2 5 23 14 29
```

8. Write a program for Insertion Sort in java.

```java
package com.test;

public class InsertionSort {

    public static void main(String[] args) {

        int[] input = { 4, 2, 9, 6, 23, 12, 34, 0, 1 };
        insertionSort(input);
    }

    private static void printNumbers(int[] input) {

        for (int i = 0; i < input.length; i++) {
            System.out.print(input[i] + ", ");
        }
        System.out.println("\n");
    }

    public static void insertionSort(int array[]) {
        int n = array.length;
        for (int j = 1; j < n; j++) {
            int key = array[j];
            int i = j-1;
            while ( (i > -1) && ( array [i] > key ) ) {
                array [i+1] = array [i];
                i--;
            }
            array[i+1] = key;
            printNumbers(array);
        }
    }
}
```



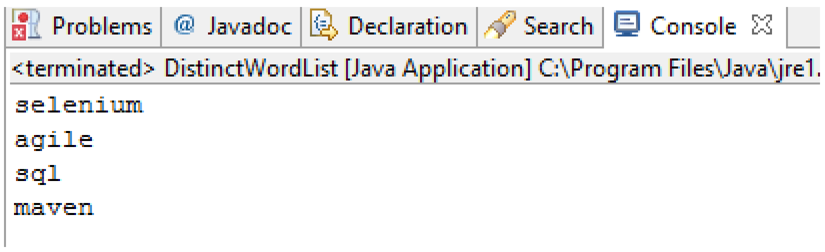
Problems | Javadoc | Declaration | Search | Console

```
<terminated> InsertionSort [Java Application] C:\Program Files\Java\jre1.8.(
2, 4, 9, 6, 23, 12, 34, 0, 1,

2, 4, 9, 6, 23, 12, 34, 0, 1,

2, 4, 6, 9, 23, 12, 34, 0, 1,

2, 4, 6, 9, 23, 12, 34, 0, 1,

2, 4, 6, 9, 12, 23, 34, 0, 1,

2, 4, 6, 9, 12, 23, 34, 0, 1,

0, 2, 4, 6, 9, 12, 23, 34, 1,

0, 1, 2, 4, 6, 9, 12, 23, 34,
```

9. Write a program to get distinct word list from the given file.

```
package com.test;

import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
```

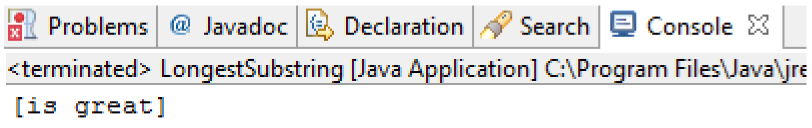
```java
import java.util.ArrayList;
import java.util.List;
import java.util.StringTokenizer;

public class DistinctWordList {

    public List<String> getDistinctWordList(String fileName){

        FileInputStream fis = null;
        DataInputStream dis = null;
        BufferedReader br = null;
        List<String> wordList = new ArrayList<String>();
        try {
            fis = new FileInputStream(fileName);
            dis = new DataInputStream(fis);
            br = new BufferedReader(new InputStreamReader(dis));
            String line = null;
            while((line = br.readLine()) != null){
                StringTokenizer st = new StringTokenizer(line, " ,.;:\"");
                while(st.hasMoreTokens()){
                    String tmp = st.nextToken().toLowerCase();
                    if(!wordList.contains(tmp)){
                        wordList.add(tmp);
                    }
                }
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally{
            try{if(br != null) br.close();}catch(Exception ex){}
        }
        return wordList;
    }

    public static void main(String a[]){

        DistinctWordList distFw = new DistinctWordList();
        List<String> wordList =
distFw.getDistinctWordList("C:/Users/pramu/Desktop/sample.txt");
        for(String str:wordList){
            System.out.println(str);
        }
    }
}
```



```
<terminated> DistinctWordList [Java Application] C:\Program Files\Java\jre1.
selenium
agile
sql
maven
```

10. Find longest substring without repeating characters.

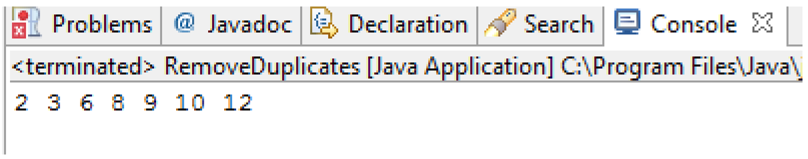
```
package com.test;

import java.util.HashSet;
import java.util.Set;

public class LongestSubstring {

    private Set<String> subStrList = new HashSet<String>();
    private int finalSubStrSize = 0;

    public Set<String> getLongestSubstr(String input){
        //reset instance variables
        subStrList.clear();
        finalSubStrSize = 0;
        // have a boolean flag on each character ascii value
        boolean[] flag = new boolean[256];
        int j = 0;
        char[] inputCharArr = input.toCharArray();
        for (int i = 0; i < inputCharArr.length; i++) {
            char c = inputCharArr[i];
            if (flag[c]) {
                extractSubString(inputCharArr,j,i);
                for (int k = j; k < i; k++) {
                    if (inputCharArr[k] == c) {
                        j = k + 1;
                        break;
                    }
                    flag[inputCharArr[k]] = false;
                }
            } else {
                flag[c] = true;
            }
        }
        extractSubString(inputCharArr,j,inputCharArr.length);
        return subStrList;
    }

    private String extractSubString(char[] inputArr, int start, int end){

        StringBuilder sb = new StringBuilder();
        for(int i=start;i<end;i++){
            sb.append(inputArr[i]);
        }
        String subStr = sb.toString();
        if(subStr.length() > finalSubStrSize){
            finalSubStrSize = subStr.length();
            subStrList.clear();
            subStrList.add(subStr);
        } else if(subStr.length() == finalSubStrSize){
            subStrList.add(subStr);
        }

        return sb.toString();
    }
    public static void main(String a[]){
        LongestSubstring mls = new LongestSubstring();
        System.out.println(mls.getLongestSubstr("God is great"));
    }
}
```



```
<terminated> LongestSubstring [Java Application] C:\Program Files\Java\jre
[is great]
```

11. Write a program to remove duplicates from sorted array

```
package com.test;

public class RemoveDuplicates {

    public static int[] removeDuplicates(int[] input){

        int j = 0;
        int i = 1;
        //return if the array length is less than 2
        if(input.length < 2){
            return input;
        }
        while(i < input.length){
            if(input[i] == input[j]){
                i++;
            }else{
                input[++j] = input[i++];
            }
        }
        int[] output = new int[j+1];
        for(int k=0; k<output.length; k++){
            output[k] = input[k];
        }

        return output;
    }

    public static void main(String a[]){
        int[] input1 = {2,3,6,6,8,9,10,10,10,12,12};
        int[] output = removeDuplicates(input1);
        for(int i:output){
            System.out.print(i+" ");
        }
    }
}
```

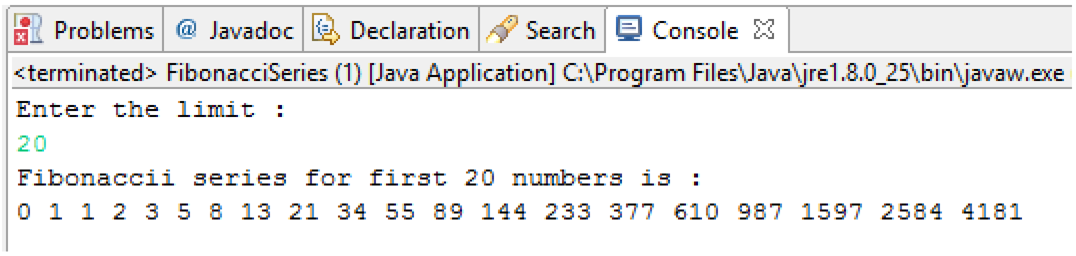


Problems @ Javadoc Declaration Search Console

```
<terminated> RemoveDuplicates [Java Application] C:\Program Files\Java\
2 3 6 8 9 10 12
```

12. **Write a program to print fibonacci series.**

```
package com.test;

import java.util.Scanner;

public class FibonacciSeries {

	public static void main(String[] args)
	{
		/*number of elements to generate in a series*/
		Scanner in = new Scanner(System.in);
		System.out.println("Enter the limit :");

		int limit = in.nextInt();
		int[] arrNum = new int[limit];

		/*create first 2 series elements*/
		arrNum[0] = 0; arrNum[1] = 1;

		 /*create the Fibonacci series and store it in an array*/
		for(int i = 2 ; i<limit ; i++)
		{
			arrNum[i] = arrNum[i-1]+arrNum[i-2];
		}

		/*Print the Fibonacci series numbers*/
		System.out.println("Fibonaccii series for first "+ limit +"
numbers is : ");
		for(int i=0 ; i<limit ; i++)
		{
			System.out.print(arrNum[i]+" ");
		}
	}
}
```

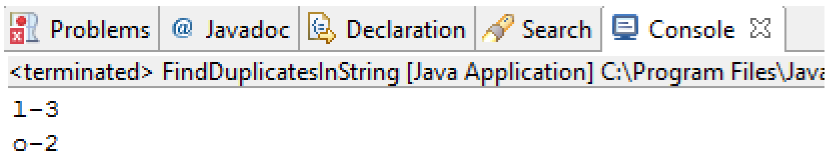


Problems @ Javadoc Declaration Search Console

```
<terminated> FibonacciSeries (1) [Java Application] C:\Program Files\Java\jre1.8.0_25\bin\javaw.exe
Enter the limit :
20
Fibonaccii series for first 20 numbers is :
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181
```

13. **Write a program to find out duplicate characters in a string**

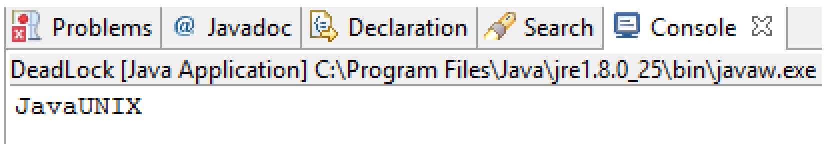
```java
package com.test;

import java.util.HashMap;
import java.util.Map;
import java.util.Set;

public class FindDuplicatesInString {

    public void findDuplicateChars(String str){

        Map<Character, Integer> dupMap = new HashMap<Character, Integer>();
        char[] chrs = str.toCharArray();
        for(Character ch:chrs){
            if(dupMap.containsKey(ch)){
                dupMap.put(ch, dupMap.get(ch)+1);
            } else {
                dupMap.put(ch, 1);
            }
        }
        Set<Character> keys = dupMap.keySet();
        for(Character ch:keys){
            if(dupMap.get(ch) > 1){
                System.out.println(ch+"-"+dupMap.get(ch));
            }
        }
    }

    public static void main(String a[]){
        FindDuplicatesInString dcs = new FindDuplicatesInString();
        dcs.findDuplicateChars("Hello World");
    }
}
```



```
<terminated> FindDuplicatesInString [Java Application] C:\Program Files\Java
l-3
o-2
```

14. Write a program to create deadlock between two threads

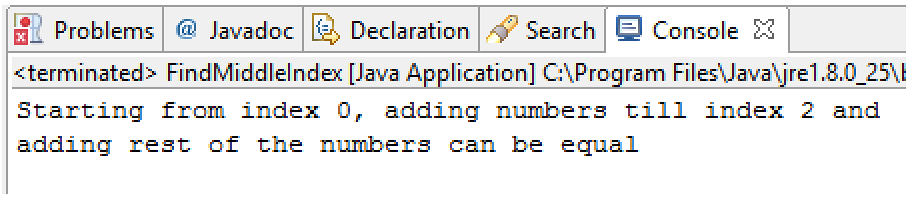
```java
package com.test;

public class DeadLock {

    String str1 = "Java";
    String str2 = "UNIX";

    Thread trd1 = new Thread("My Thread 1"){
        public void run(){
            while(true){
                synchronized(str1){
                    synchronized(str2){
                        System.out.println(str1 + str2);
                    }
                }
            }
        }
    };

    Thread trd2 = new Thread("My Thread 2"){
        public void run(){
            while(true){
                synchronized(str2){
                    synchronized(str1){
                        System.out.println(str2 + str1);
                    }
                }
            }
        }
    };

    public static void main(String a[]){
        DeadLock mdl = new DeadLock();
        mdl.trd1.start();
        mdl.trd2.start();
    }
}
```



```
DeadLock [Java Application] C:\Program Files\Java\jre1.8.0_25\bin\javaw.exe
JavaUNIX
```

15. Find out middle index where sum of both ends are equal

```
package com.test;

public class FindMiddleIndex {

    public static int findMiddleIndex(int[] numbers) throws Exception {

        int endIndex = numbers.length - 1;
        int startIndex = 0;
        int sumLeft = 0;
        int sumRight = 0;
        while (true) {
            if (sumLeft > sumRight) {
                sumRight += numbers[endIndex--];
            } else {
                sumLeft += numbers[startIndex++];
            }
            if (startIndex > endIndex) {
                if (sumLeft == sumRight) {
                    break;
                } else {
                    throw new Exception(
                            "Please pass proper array to match the
requirement");
                }
            }
        }
        return endIndex;
    }

    public static void main(String a[]) {
        int[] num = { 2, 4, 4, 5, 4, 1 };
        try {
            System.out.println("Starting from index 0, adding numbers till
index "
                            + findMiddleIndex(num) + " and");
            System.out.println("adding rest of the numbers can be equal");
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }
}
```

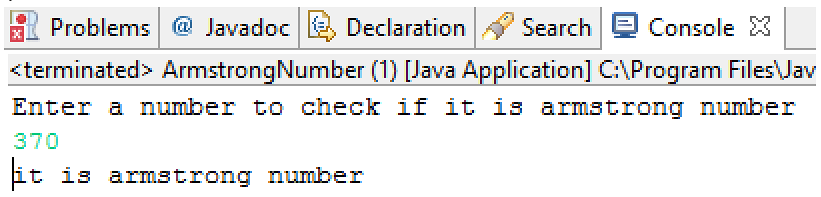

Problems @ Javadoc Declaration Search Console

```
<terminated> FindMiddleIndex [Java Application] C:\Program Files\Java\jre1.8.0_25\
Starting from index 0, adding numbers till index 2 and
adding rest of the numbers can be equal
```

16. Write a program to find the given number is Armstrong number or not?

```
package com.test;

import java.util.Scanner;

public class ArmstrongNumber {
      public static boolean isArmstrong(int input) {
            String inputAsString = input + "";
            int numOfdigits = inputAsString.length();
            int cinput = input;
            int sum = 0;
            while (cinput != 0) {
                  int lastDigit = cinput % 10;
                  sum = sum + (int) Math.pow(lastDigit, numOfdigits);
                  cinput = cinput / 10;
            }

            if (sum == input) {
                  return true;
            } else {
                  return false;
            }
      }
```

```
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		System.out.println("Enter a number to check if it is armstrong 
number");
		Scanner sc = new Scanner(System.in);
		int num = sc.nextInt();
		boolean result = isArmstrong(num);
		if (result) {
			System.out.println("it is armstrong number");
		} else {
			System.out.println("it is not armstrong number");
		}
	}

}
```



```
<terminated> ArmstrongNumber (1) [Java Application] C:\Program Files\Jav
Enter a number to check if it is armstrong number
370
it is armstrong number
```