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

JVM is an acronym for Java Virtual Machine, it is an abstract machine which provides the runtime environment in which java bytecode can be executed. It is a specification.

JVMs are available for many hardware and software platforms (so JVM is platform dependent).

**Java memory allocation**

**Java Heap Space**

Java Heap space is used by java runtime to allocate memory to Objects and JRE classes. Whenever we create any object, it’s always created in the Heap space. Garbage Collection runs on the heap memory to free the memory used by objects that doesn’t have any reference. Any object created in the heap space has global access and can be referenced from anywhere of the application Heap memory is used by all the parts of the application whereas stack memory is used only by one thread of execution.

**Java Stack Memory**

Java Stack memory is used for execution of a thread. They contain method specific values that are short-lived and references to other objects in the heap that are getting referred from the method. Stack memory is always referenced in LIFO (Last-In-First-Out) order. Whenever a method is invoked, a new block is created in the stack memory for the method to hold local primitive values and reference to other objects in the method. As soon as method ends, the block becomes unused and become available for next method.Stack memory size is very less compared to Heap memory.

1. **What is Polymorphism and encapsulation?**

**Polymorphism**

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

**Encapsulation**

Encapsulation indicates internal behavior of the object. Implementation of abstraction is called encapsulation. Binding of data and code is called encapsulation.

1. **What is method overloading and Method over riding?**

**OverLoading**

Writing the method with the same name and different arguments is called method overloading.

**OverRiding**

Declaring a method in **subclass** which is already present in **parent class** is known as method overriding. Earlier we shared[**method overloading in java**](http://beginnersbook.com/2013/05/method-overloading/)

**4. Why string is Immutable?**

String is an immutable class in Java. An immutable class is simply a class whose instances cannot be modified. All information in an instance is initialized when the instance is created and the information can not be modified.

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

|  |  |  |
| --- | --- | --- |
| **No.** | **StringBuffer** | **StringBuilder** |
| 1) | StringBuffer is *synchronized* i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously. | StringBuilder is *non-synchronized* i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously. |
| 2) | StringBuffer is *less efficient* than String Builder. | StringBuilder is *more efficient* than StringBuffer. |

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

|  |  |  |
| --- | --- | --- |
|  | **Array** | **ArrayList** |
|  |  |  |
| Resizable | No | Yes |
|  |  |  |
| Primitives | Yes | No |
|  |  |  |
| Iterating values | for, for each | Iterator , for each |
|  |  |  |
| Length | length variable | size method |
|  |  |  |
| Performance | Fast | Slow in comparision |
|  |  |  |
| Multidimensional | Yes | No |
| Add Elements | Assignment operator | add method |

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

|  |  |
| --- | --- |
| **HashMap** | **Hashtable** |
| 1) HashMap is **non synchronized**. It is not-thread safe and can't be shared between many threads without proper synchronization code. | Hashtable is **synchronized**. It is thread-safe and can be shared with many threads. |
| 2) HashMap **allows one null key and multiple null values**. | Hashtable **doesn't allow any null key or value**. |
| 3) HashMap is a **new class introduced in JDK 1.2**. | Hashtable is a **legacy class**. |
| 4) HashMap is **fast**. | Hashtable is **slow**. |
| 5) We can make the HashMap as synchronized by calling this code Map m = Collections.synchronizedMap(hashMap); | Hashtable is internally synchronized and can't be unsynchronized. |
| 6) HashMap is **traversed by Iterator**. | Hashtable is **traversed by Enumerator and Iterator**. |
| 7) Iterator in HashMap is **fail-fast**. | Enumerator in Hashtable is **not fail-fast**. |
| 8) HashMap inherits **AbstractMap** class. | Hashtable inherits **Dictionary** class. |

**8. What is a vector in Java?**

Vector implements a dynamic array. It is similar to ArrayList, Vector implements List Interface. Like ArrayList it also maintains insertion order but it is rarely used in non-thread environment as it is synchronized and due to which it gives poor performance in searching, adding, delete and update of its elements.

**9. What is set in java?**

A collection class which implements set Interface and do not allow to insert duplicate values.

**10. What is an abstract class?**

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.

An *abstract method* is a method that is declared without an implementation (without braces, and

followed by a semicolon), like this:

abstract void moveTo(double deltaX, double deltaic);

OR

Abstract classes are classes that contain one or more abstract methods. An abstract method is a method that is declared, but contains no implementation. Abstract classes may not be instantiated, and require subclasses to provide implementations for the abstract methods.

**11. What is an interface?**

Set of rules is called specification.

Interface is a contract between service provider and service consumer

Java interface can only contain method signatures and fields.

By default, the methods of the interface are public and abstract.

To implement an interface implements keyword is used

Fields declared inside the class are public static and final

**12. Why Java is Platform independent?**

Java code can be run on multiple platforms e.g. Windows,Linux,Sun Solaris,Mac/OS etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere(WORA).

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

The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.

There are 4 types of java access modifiers:

1. private
2. default
3. protected
4. public

OR

Access modifiers specifies the access levels of a variable or method. Java access modifiers are public, private, protected, default modifier (Default access modifier).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Modifiers** | **Same Class** | **Same Package** | **Subclass** | **Other packages** |
| public | Yes | Yes | Yes | Yes |
| protected | Yes | Yes | Yes | No |
| default | Yes | Yes | No | No |
| private | Yes | No | No | No |

**14. What are java exceptions? Give me an example**

Run time errors which can be handled by the programmer is called exception. When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore these exceptions are to be handled.

**Checked exceptions:** A checked exception is an exception that occurs at the compile time, these are also called as compile time exceptions. These exceptions cannot simply be ignored at the time of compilation, the Programmer should take care of (handle) these exceptions

For example, if you use **FileReader** class in your program to read data from a file, if the file specified in its constructor doesn't exist, then an *FileNotFoundException* occurs, and compiler prompts the programmer to handle the exception.

**Unchecked exceptions:** An Unchecked exception is an exception that occurs at the time of execution, these are also called as Runtime Exceptions, these include programming bugs, such as logic errors or improper use of an API. runtime exceptions are ignored at the time of compilation.

For example, if you have declared an array of size 5 in your program, and trying to call the 6th element of the array then an *ArrayIndexOutOfBoundsExceptionexception* occurs.

**What is the difference between throws and throwable?**

## throws In Java :

**throws** is also a keyword in java which is used in the method signature to indicate that this method may throw mentioned exceptions. The caller to such methods must handle the mentioned exceptions either using try-catch blocks or using throws keyword. Below is the syntax for using throws keyword.

class ThrowsExample

{

void methodOne() throws SQLException

{

//This method may throw SQLException

}

void methodTwo() throws IOException

{

//This method may throw IOException

}

void methodThree() throws ClassNotFoundException

{

//This method may throw ClassNotFoundException

}

}

## Throwable In Java :

**Throwable** is a super class for all types of errors and exceptions in java. This class is a member of **java.lang** package. Only instances of this class or it’s sub classes are thrown by the java virtual machine or by the throw statement. The only argument of catch block must be of this type or it’s sub classes. If you want to create your own customized exceptions, then your class must extend this class.

|  |
| --- |
| class MyException extends Throwable  {  //Customized Exception class  }  class ThrowAndThrowsExample  {  void method() throws MyException  {  MyException e = new MyException();  throw e;  }  } |

**16. What is the difference between Error and exception?**

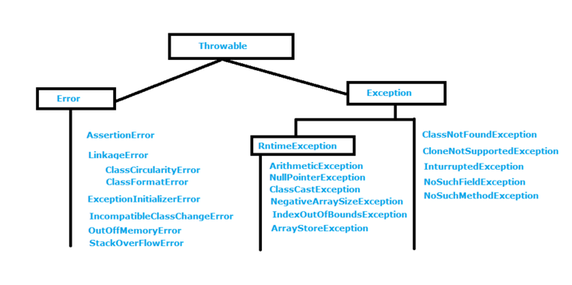
An error is an irrecoverable condition occurring at runtime like out of memory error. These kind of jvm errors cannot be handled at runtime.

Exceptions are because of condition failures, which can be handled easily at runtime.

Exception are of type java.lang.Exception

Errors are of type java.lang.Error

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



* Exception and Error both are sub classes of java.lang.Throwable class.
* We can handle Exceptions at runtime but Errors we can not handle.
* If exception occurs we can handle it by using try and catch block. If Error occurs we can handle it program execution need to be terminated.
* In Exception we have two types 1. Checked Exception 2.Unchecked Exceptions
* Error are by default unchecked exceptions.
* Exception are of type java.lang.Exception
* Errors are of type java.lang.Error
* Error will run at run time.
* In Exceptions Checked Exceptions will known to compiler so we need to handle these exceptions at compile time itself otherwise compile time Error will come.
* Unchecked Exception will come at run time need to handle by using try and catch blocks.

**18. What are collection APIs, give me an example**

**Collections in java** is a framework that provides an architecture to store and manipulate the group of objects.

All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections. Java Collection simply means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc).

**19. What is the difference between final and finally?**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **final** |  | |  |  | | --- | --- | | **finally** |  | |
| 1) | Final is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden and final variable value can't be changed. | Finally is used to place important code, it will be executed whether exception is handled or not. |  |
| 2) | Final is a keyword. | Finally is a block. |  |

**20. Will java supports multiple inheritance?**

No java does not support multiple inheritance

**21. What are the different types of interface?**

List, Queue and Set

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

Everything in java is an object, except primitives. Primitives are int, short, long, boolean, etc. Since they are not objects, they cannot return as objects, and collection of objects. To support this, java provides wrapper classes to move primitives to objects. Some of the wrapper classes are Integer, Long, Boolean, etc. **Wrapper class in java** provides the mechanism to convert primitive into object and object into primitive**.**

int i = 26; // Primitive data type 'int'

// **Integer Wrapper class** instantiation

Integer i\_Obj = new Integer(i);

// Unwrapping primitive data 'int' from wrapper object

int i2 = i\_Obj.intValue();

**Important :** The primitive data type values will be stored in **Stack Memory** whereas wrapper class objects (like any other java objects) are stored in **Heap Memory**.

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

**Boxing:** It is a process of converting value of primitive type into object is called boxing.

int i = 10;

Integer num1 = new Integer(i);

**UnBoxing:** It is a process of converting object to primitive type is called unboxing

int i = 10;

Integer num1 = new Integer(i);

Int num2 = num1.intValue();

**24. Explain for each loop**

The for-each loop is used to traverse array or collection in java. It is easier to use than simple for loop because we don't need to increment value and use subscript notation. It works on elements basis not index. It returns element one by one in the defined variable. foreach should always be chosen over an iterator, as it is more convenient and concise.

for(int i : intList) {

System.out.println("An element in the list: " + i);

}

**25. What are iterators, explain with an example**

Iterators allows to traverse elements in collection framework.

Iterators are of two type

1]Iterators which allow us to traverse elements in forward direction only.

Example:

Iterator<Integer> it = range. Iterator();

while(it.hasNext()) {

int cur = it.next();

System.out.println(cur);

}

2]List Iterator which allows us to traverse elements in forward as well as backward direction ie, bidirectional.

Iterator<Integer> it = range. Iterator();

while(it.hasNext()) {

int cur = it.next();

System.out.println(cur);

}

Iterator<Integer> it = range. Iterator();

while(it.hasPrevious()) {

int cur = it.previous();

System.out.println(cur);

}

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

We can access private variables inside in different class by providing get() and set() methods in private variables class.

1. **What is Constructor Over loading?**

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists.The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

Example

class Student5{

int id;

String name;

int age;

Student5(int i,String n){

id = i;

name = n;

}

Student5(int i,String n,int a){

id = i;

name = n;

age=a;

}

void display(){System.out.println(id+" "+name+" "+age);}

public static void main(String args[]){

Student5 s1 = new Student5(111,"Karan");

Student5 s2 = new Student5(222,"Aryan",25);

s1.display();

s2.display();

}

}

1. **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.Whenever you create the instance of subclass, an instance of parent class is created implicitly i.e. referred by super reference variable.

## Usage of java super Keyword

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

**Programing Questions:**

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

**package** pkg;

**import** java.util.Calendar;

**public** **class** DifferenceInDays {

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

// **TODO** Auto-generated method stub

Calendar c1 = Calendar.*getInstance*();

Calendar c2 = Calendar.*getInstance*();

c1.set(2015, 8, 22);

c2.set(2016, 8, 22);

**long** milli1 =c1.getTimeInMillis();

**long** milli2 = c2.getTimeInMillis();

**long** diff = milli2-milli1;

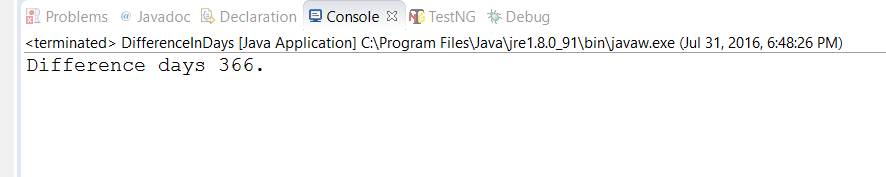
**long** diffInDays = diff/(24\*60\*60\*1000);

System.***out***.println("Difference days "+diffInDays+ ".");

}

}

**OUTPUT**



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

**package** pkg;

**import** java.util.Scanner;

**public** **class** DivisionWithoutOperator {

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

**int** num,div,res,qui = 0,rem = 0;

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("Enter a number: ");

num=sc.nextInt();

System.***out***.println("Enter a divisor: ");

div = sc.nextInt();

**for**(**int** i=0; i<num; i++){

res = div\*i;

**if**(res>num){

res = res-div;

qui=i-1;

rem = num-res;

**break**;

}

}

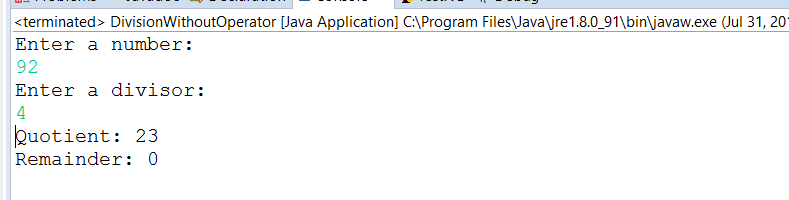
System.***out***.println("Quotient: "+qui);

System.***out***.println("Remainder: "+rem);

}

}

**OUTPUT**



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

**package** pkg;

**public** **class** MultiplyWithoutOperator {

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

// **TODO** Auto-generated method stub

**int** a =2, b=4, mul=0;

**for**(**int** i=1; i<=a; i++){

mul = mul+b;

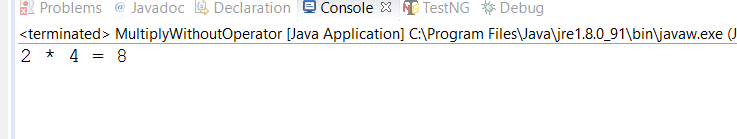
}

System.***out***.println(a +" \* "+b+" = "+mul);

}

}

**OUTPUT**



1. **How to make a list immutable?**

package pkg;

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

public class ImmutableList {

public static void main(String[] args) {

// TODO Auto-generated method stub

List<String> list = new ArrayList<String>();

list.add("India");

list.add("USA");

list.add("UK");

list.add("China");

for(String l:list){

System.out.println(l);

}

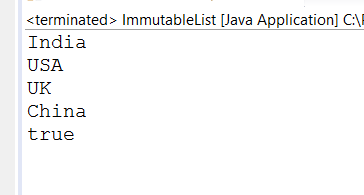
List<String> immutablelist = Collections.unmodifiableList(list);

//immutablelist.add("China");

System.out.println(immutablelist.contains("India"));

}

}



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

**package** pkg;

**public** **class** ReverseLinkedList {

**static** Node *head*;

**static** **class** Node {

**int** data;

Node next;

Node(**int** d) {

data = d;

next = **null**;

}

}

Node reverse(Node node){

Node prev =**null**;

Node current =node;

Node next = **null**;

**while**(current != **null**){

next = current.next;

current.next=prev;

prev=current;

current=next;

}

node=prev;

**return** node;

}

**void** printList(Node node) {

**while** (node != **null**) {

System.***out***.print(node.data + " ");

node = node.next;

}

}

}

1. **Write a program to implement ArrayList and Linked list**

**package** pkgLinkedList;

**class** linkedList

{

**protected** Node start;

**protected** Node end ;

**public** **int** size ;

/\* Constructor \*/

**public** linkedList()

{

start = **null**;

end = **null**;

size = 0;

}

/\* Function to check if list is empty \*/

**public** **boolean** isEmpty()

{

**return** start == **null**;

}

/\* Function to get size of list \*/

**public** **int** getSize()

{

**return** size;

}

/\* Function to insert an element at begining \*/

**public** **void** insertAtStart(**int** val)

{

Node nptr = **new** Node(val, **null**);

size++ ;

**if**(start == **null**)

{

start = nptr;

end = start;

}

**else**

{

nptr.setLink(start);

start = nptr;

}

}

/\* Function to insert an element at end \*/

**public** **void** insertAtEnd(**int** val)

{

Node nptr = **new** Node(val,**null**);

size++ ;

**if**(start == **null**)

{

start = nptr;

end = start;

}

**else**

{

end.setLink(nptr);

end = nptr;

}

}

/\* Function to insert an element at position \*/

**public** **void** insertAtPos(**int** val , **int** pos)

{

Node nptr = **new** Node(val, **null**);

Node ptr = start;

pos = pos - 1 ;

**for** (**int** i = 1; i < size; i++)

{

**if** (i == pos)

{

Node tmp = ptr.getLink() ;

ptr.setLink(nptr);

nptr.setLink(tmp);

**break**;

}

ptr = ptr.getLink();

}

size++ ;

}

/\* Function to delete an element at position \*/

**public** **void** deleteAtPos(**int** pos)

{

**if** (pos == 1)

{

start = start.getLink();

size--;

**return** ;

}

**if** (pos == size)

{

Node s = start;

Node t = start;

**while** (s != end)

{

t = s;

s = s.getLink();

}

end = t;

end.setLink(**null**);

size --;

**return**;

}

Node ptr = start;

pos = pos - 1 ;

**for** (**int** i = 1; i < size - 1; i++)

{

**if** (i == pos)

{

Node tmp = ptr.getLink();

tmp = tmp.getLink();

ptr.setLink(tmp);

**break**;

}

ptr = ptr.getLink();

}

size-- ;

}

/\* Function to display elements \*/

**public** **void** display()

{

System.***out***.print("\nSingly Linked List = ");

**if** (size == 0)

{

System.***out***.print("empty\n");

**return**;

}

**if** (start.getLink() == **null**)

{

System.***out***.println(start.getData() );

**return**;

}

Node ptr = start;

System.***out***.print(start.getData()+ "->");

ptr = start.getLink();

**while** (ptr.getLink() != **null**)

{

System.***out***.print(ptr.getData()+ "->");

ptr = ptr.getLink();

}

System.***out***.print(ptr.getData()+ "\n");

}

}

**package** pkgLinkedList;

**import** java.util.Scanner;

/\* Class Node \*/

**class** Node

{

**protected** **int** data;

**protected** Node link;

/\* Constructor \*/

**public** Node()

{

link = **null**;

data = 0;

}

/\* Constructor \*/

**public** Node(**int** d,Node n)

{

data = d;

link = n;

}

/\* Function to set link to next Node \*/

**public** **void** setLink(Node n)

{

link = n;

}

/\* Function to set data to current Node \*/

**public** **void** setData(**int** d)

{

data = d;

}

/\* Function to get link to next node \*/

**public** Node getLink()

{

**return** link;

}

/\* Function to get data from current Node \*/

**public** **int** getData()

{

**return** data;

}

}

**package** pkgLinkedList;

**import** java.util.Scanner;

**public** **class** SinglyLinkedList

{

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

{

Scanner scan = **new** Scanner(System.***in***);

/\* Creating object of class linkedList \*/

linkedList list = **new** linkedList();

System.***out***.println("Singly Linked List Test\n");

**char** ch;

/\* Perform list operations \*/

**do**

{

System.***out***.println("\nSingly Linked List Operations\n");

System.***out***.println("1. insert at begining");

System.***out***.println("2. insert at end");

System.***out***.println("3. insert at position");

System.***out***.println("4. delete at position");

System.***out***.println("5. check empty");

System.***out***.println("6. get size");

**int** choice = scan.nextInt();

**switch** (choice)

{

**case** 1 :

System.***out***.println("Enter integer element to insert");

list.insertAtStart( scan.nextInt() );

**break**;

**case** 2 :

System.***out***.println("Enter integer element to insert");

list.insertAtEnd( scan.nextInt() );

**break**;

**case** 3 :

System.***out***.println("Enter integer element to insert");

**int** num = scan.nextInt() ;

System.***out***.println("Enter position");

**int** pos = scan.nextInt() ;

**if** (pos <= 1 || pos > list.getSize() )

System.***out***.println("Invalid position\n");

**else**

list.insertAtPos(num, pos);

**break**;

**case** 4 :

System.***out***.println("Enter position");

**int** p = scan.nextInt() ;

**if** (p < 1 || p > list.getSize() )

System.***out***.println("Invalid position\n");

**else**

list.deleteAtPos(p);

**break**;

**case** 5 :

System.***out***.println("Empty status = "+ list.isEmpty());

**break**;

**case** 6 :

System.***out***.println("Size = "+ list.getSize() +" \n");

**break**;

**default** :

System.***out***.println("Wrong Entry \n ");

**break**;

}

/\* Display List \*/

list.display();

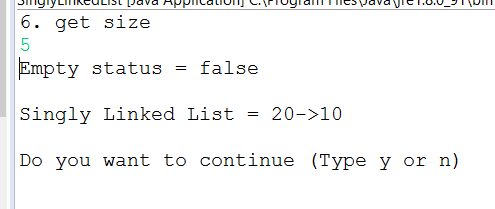
System.***out***.println("\nDo you want to continue (Type y or n) \n");

ch = scan.next().charAt(0);

} **while** (ch == 'Y'|| ch == 'y');

}

}



1. **Write a program for Insertion Sort in java.**

**package** pkg;

**public** **class** InsertionSort {

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

// **TODO** Auto-generated method stub

**int**[] data ={9,6,5,0,7,2,3,1};

*printNumbers*(data);

**int**[] result =*insertionSort*(data);

*printNumbers*(result);

}

**public** **static** **void** printNumbers(**int**[] input){

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

System.***out***.print(input[i]+" ");

}

System.***out***.println("\n");

}

**public** **static** **int**[] insertionSort(**int**[] data){

**for**(**int** j=1; j<data.length; j++){

**int** key= data[j];

**int** i = j-1;

**while**((i>=0) &&(data[i])>key){

data[i+1] =data[i];

i--;

}

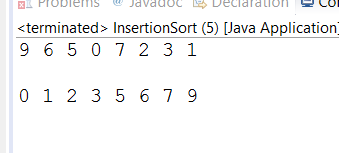
data[i+1] =key;

}

**return** data;

}

}



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

package pkg;

import java.io.BufferedReader;

import java.io.DataInputStream;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.ArrayList;

import java.util.List;

import java.util.StringTokenizer;

public class MyDistinctFileWords {

public static void main(String[] args) throws IOException {

List<String> array = new ArrayList<String>();

File file = new File("C:\\Users\\Fredrick\\Downloads\\Assignment 2.txt");

FileInputStream fin = new FileInputStream(file);

BufferedReader reader = new BufferedReader(new InputStreamReader(fin));

String line=null;

while((line = reader.readLine()) != null){

StringTokenizer st = new StringTokenizer(line, " ,.,:,/");

while(st.hasMoreTokens()){

String tmp = st.nextToken().toLowerCase();

if(!array.contains(tmp))

array.add(tmp);

}

}

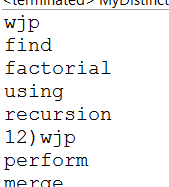
for(String str:array){

System.out.println(str);

}

}

}



1. **Find longest substring without repeating characters.**

package pkg;

import java.util.HashSet;

import java.util.Set;

public class MyLongestSubstr {

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

private int finalSubStrSize = 0;

public Set<String> getLongestSubstr(String input){

subStrList.clear();

finalSubStrSize = 0;

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();

}

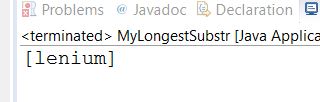
public static void main(String a[]){

MyLongestSubstr mls = new MyLongestSubstr();

System.out.println(mls.getLongestSubstr("Selenium"));

}

}



1. **Write a program to remove duplicates from sorted array**

**package** pkg;

**public** **class** RemoveDuplicateElements {

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

// **TODO** Auto-generated method stub

**int**[] A={1,2,2,3,17,17};

*printArray*(A);

**int** [] out=*removeDuplicates*(A);

*printArray*(out);

}

**public** **static** **void** printArray(**int** a[]){

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

System.***out***.print(a[i]+" ");

}

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

}

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

**int** i=1;

**int** j=0;

**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++){

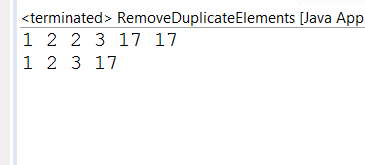
output[k]= input[k];

}

**return** output;

}

}



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

**package** pkg;

**import** java.util.Scanner;

**public** **class** Fibonacci {

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

// **TODO** Auto-generated method stub

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("Enter a number: ");

**int** n=sc.nextInt();

**for**(**int** i=0; i<n; i++){

System.***out***.println(*fibonacci*(i));

}

}

**public** **static** **long** fibonacci(**int** n){

**if**(n==0)

**return** 0;

**else** **if**(n==1)

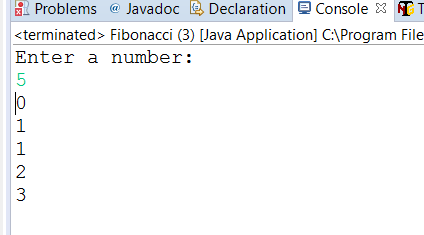
**return** 1;

**else**

**return** *fibonacci*(n-1)+*fibonacci*(n-2);

}

}



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

**package** pkg;

**public** **class** DuplicateCharsInString {

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

// **TODO** Auto-generated method stub

**int** counter;

String str = "aaabbccccccd";

**char**[] ch = str.toCharArray();

**int** sz = ch.length;

**for**(**int** i=0; i<sz; i++){

counter =0;

**for**(**int** j=0; j<sz; j++){

**if**(j<i && ch[i]==ch[j])

**break**;

**if**(ch[i]==ch[j])

counter++;

**if**(j==sz-1)

System.***out***.println(ch[i]+ " --> "+counter);

}

}

}

}



1. **Write a program to create deadlock between two threads**

**package** pkg;

**public** **class** DeadLock {

**public** **static** Object *Lock1* = **new** Object();

**public** **static** Object *Lock2* = **new** Object();

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

// **TODO** Auto-generated method stub

ThreadDemo1 T1 = **new** ThreadDemo1();

ThreadDemo2 T2 = **new** ThreadDemo2();

T1.start();

T2.start();

}

**private** **static** **class** ThreadDemo1 **extends** Thread {

**public** **void** run() {

**synchronized** (*Lock1*) {

System.***out***.println("Thread 1: Holding lock 1...");

**try** { Thread.*sleep*(10); }

**catch** (InterruptedException e) {}

System.***out***.println("Thread 1: Waiting for lock 2...");

**synchronized** (*Lock2*) {

System.***out***.println("Thread 1: Holding lock 1 & 2...");

}

}

}

}

**private** **static** **class** ThreadDemo2 **extends** Thread {

**public** **void** run() {

**synchronized** (*Lock2*) {

System.***out***.println("Thread 2: Holding lock 2...");

**try** { Thread.*sleep*(10); }

**catch** (InterruptedException e) {}

System.***out***.println("Thread 2: Waiting for lock 1...");

**synchronized** (*Lock1*) {

System.***out***.println("Thread 2: Holding lock 1 & 2...");

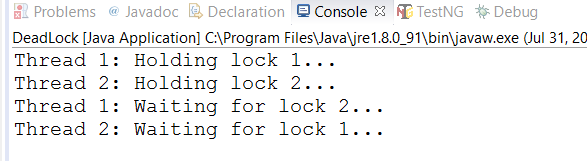
}

}

}

}

}



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

**package** pkg;

**public** **class** FindMiddleIndex {

**public** **static** **int** findMiddleIndex(**int**[] numbers){

**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**{

System.***out***.println("Enter proper array");

}

}

}

**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");

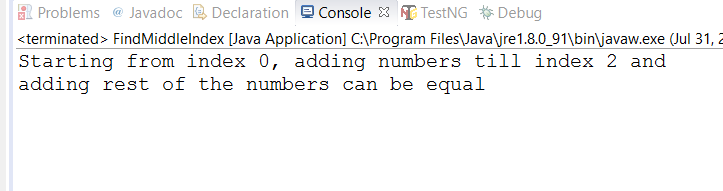
} **catch** (Exception ex) {

System.***out***.println(ex.getMessage());

}

}

}



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

**package** pkg;

**import** java.util.Scanner;

**public** **class** ArmstrongNumber {

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

// **TODO** Auto-generated method stub

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("Enter a number");

**int** n = sc.nextInt();

**int** temp =n;

**int** result = *reverse*(n);

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

**boolean** flag = *armstrongNumber*(temp, result);

**if**(flag == **true**){

System.***out***.println(n+" is a plaindrome number");

}**else**{

System.***out***.println(n+" is not a plaindrome number");

}

}

**public** **static** **int** reverse(**int** n){

**int** sum=0, rem;

**do**{

rem = n%10;

sum += Math.*pow*(rem, 3);

n = n/10;

}**while**(n!=0);

**return** sum;

}

**public** **static** **boolean** armstrongNumber(**int** temp, **int** sum){

**if**(sum == temp)

**return** **true**;

**else**

**return** **false**;

}

}

