**Java Refresher**

1. **Flow control (if, for, while, etc.), Input output.**

**Introduction:**

In java the flow control refers to the order in which the individual statements are executed, and in java the different control statements are if, if-else, while, do-while, for and switch-case.

**If:**

This statement is also known as if-then statement and this is the most basic of all flow control statements which executes a particular section of code only if a particular given condition is true.

The syntax of this statement:

If(condition 1)

{……

// Block or section of code to be executed when the condition 1 is true.

……..

}

**If-else:**

This statement is also known as if-then-else statement and it has two sections of code one section is under the if clause to be executed when the statement is true and the other section is under the else statement to be executed when the condition is false.

The syntax of this statement:

if(condition 1)

{……

// Block or section of code to be executed when the condition 1 is true.

……..

}

else

{……

// Block or section of code to be executed when the condition 1 is false.

……..

}

**While:**

This statement continuously executes a section or block of code as long as a condition is true.

Its syntax is as follows:

while(condition 1)  
{…….

// Block or section of code to be executed continuously as long as the condition 1 is true and the // control exits the loop only when the condition becomes false.

}

**Do-while:**

The difference between do-while and while is that do-while evaluates its condition at the bottom of the loop instead of the top. Therefore, the statements within the do block are always executed at least once irrespective of the condition being true or false.

Syntax of the do-while loop:

do{

…………

// block of code to be executed at least once irrespective of the condition 1.

}

While(condition 1);

**For**:

This statement is used when we want to iterate a section of code a particular number of times, the block of code to be iterated is kept in the for loop.

Syntax:

for (initialization; termination condition; increment or decrement)

{……..

// block of code.

………

}

Here the initialization statement we initialize the loop variable and then check the condition and if the condition is true then only the control will enter the loop and after the execution of the loop the increment or decrement statement is executed and the loop variable will be incremented or decremented and then the control checks the condition again and so on the process continues and the iterations in the loops take place as long as the condition is true and exists the loop once the condition is false.

**Switch-case:**

Switch-case statement is the selection statement which selects a path of execution from many available paths of execution.

Syntax:

switch(variable)

{

case 1 :

// block of code when variable = 1

...

break;

.

.

case n :

// block of code when variable = n

...

break;

default :

// block of code for all the other cases.

...

}

In the above syntax the block of code is executed depending on the value of the variable.

**Programming question description:**

Q) Write a Java program to print all the prime numbers less than or equal to a number n where the value of n is read from the user.

**Source code:**

**package** myproject;

**import** java.io.\*;

**class** Prime {

@SuppressWarnings("deprecation")

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

String s;

**int** no,i,j;

DataInputStream in=**new** DataInputStream(System.*in*);

System.*out*.print("Enter a value: \n");

s=in.~~readLine~~();

no=Integer.*parseInt*(s);

System.*out*.println("the prime numbers below "+ no+ " are as follows:");

**for**(i=no;i>=1;i--)

{

**for**(j=2;j<i;j++)

{

**if**(i%j==0) **break**;

}

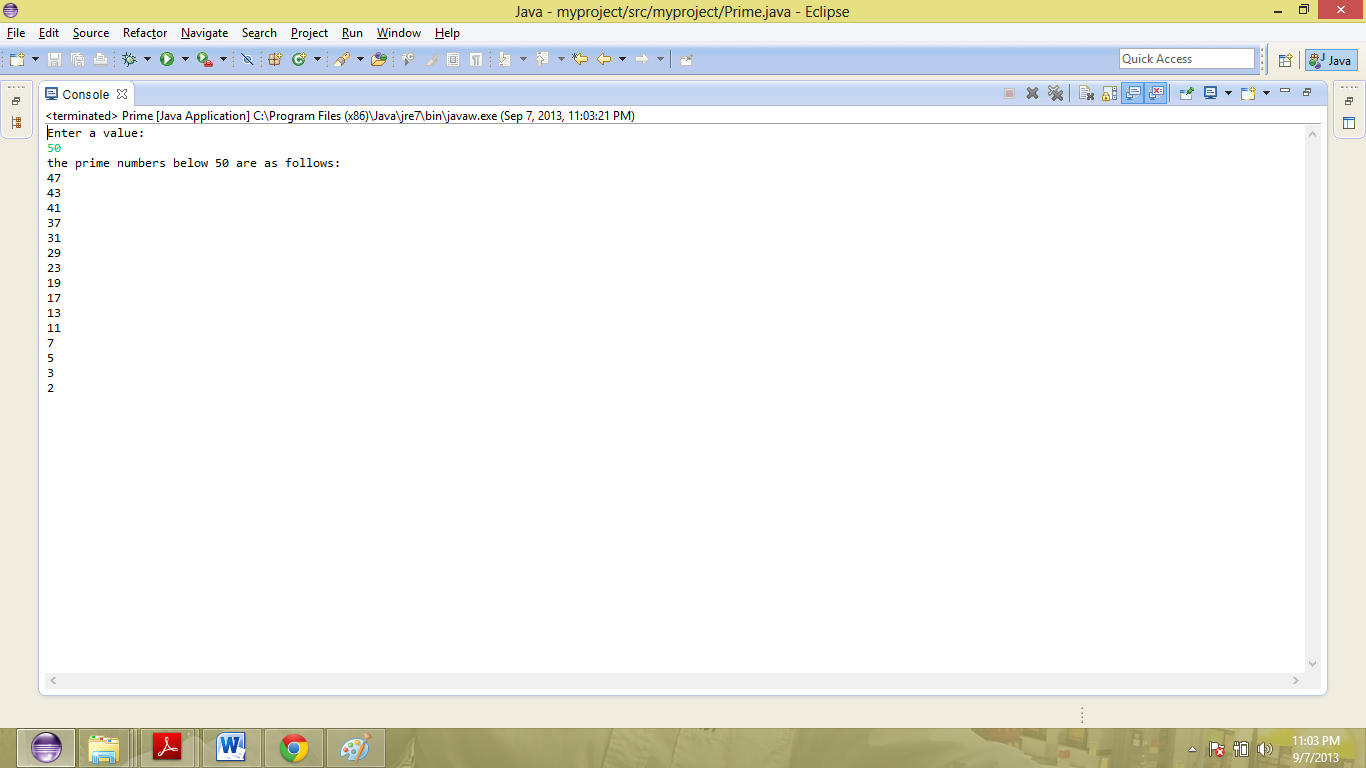
**if**(i==j) System.*out*.println(i);

}

}

}

**Output:**



1. **class String:**

**Introduction:**

In java String is a class and every string we define is an object of this String class so just like other objects, you can create an instance of a string using new keyword, as follows:

String s = new String();

This line of code creates a new object of class String, and assigns it to the reference variable s. The String class is immutable; so that once it is created a String object cannot be changed. If there is a necessity to make a lot of modifications to Strings of characters, then you should use String Buffer & String Builder Classes.

Methods used to obtain information about an object are known as assessor methods, the string class has many assessor methods:

String length:

s.length();

Here the above statement returns the length of the string s.

Concatenating Strings:

The String class includes a method for concatenating two strings:

string1.concat(string2);

String chatAt() method:

This method returns the character located at the String's specified index. The string indexes start from zero.

Syntax:

public char charAt(int index);

This method returns the char at the particular index value given.

**Programming question description:**

Write a java program to arrange the given strings in alphabetical order which takes both the number of strings and also the strings as an input from the user.

**Source Code:**

**package** myproject;

**import** java.io.\*;

**class** Strings {

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

**int** i,j,n;

String swp;

DataInputStream in=**new** DataInputStream(System.*in*);

System.*out*.println("Enter the number of strings u want to enter:");

n=Integer.*parseInt*(in.~~readLine~~());

String s[]=**new** String[n];

System.*out*.println("Enter the " +n + " strings :");

**for**(i=0;i<=n-1;i++) s[i]=in.~~readLine~~();

**for**(i=0;i<=n-2;i++) {

**for**(j=i+1;j<=n-1;j++)

**if**(s[i].compareTo(s[j])>0)

{

swp=s[i];

s[i]=s[j];

s[j]=swp;

}

}

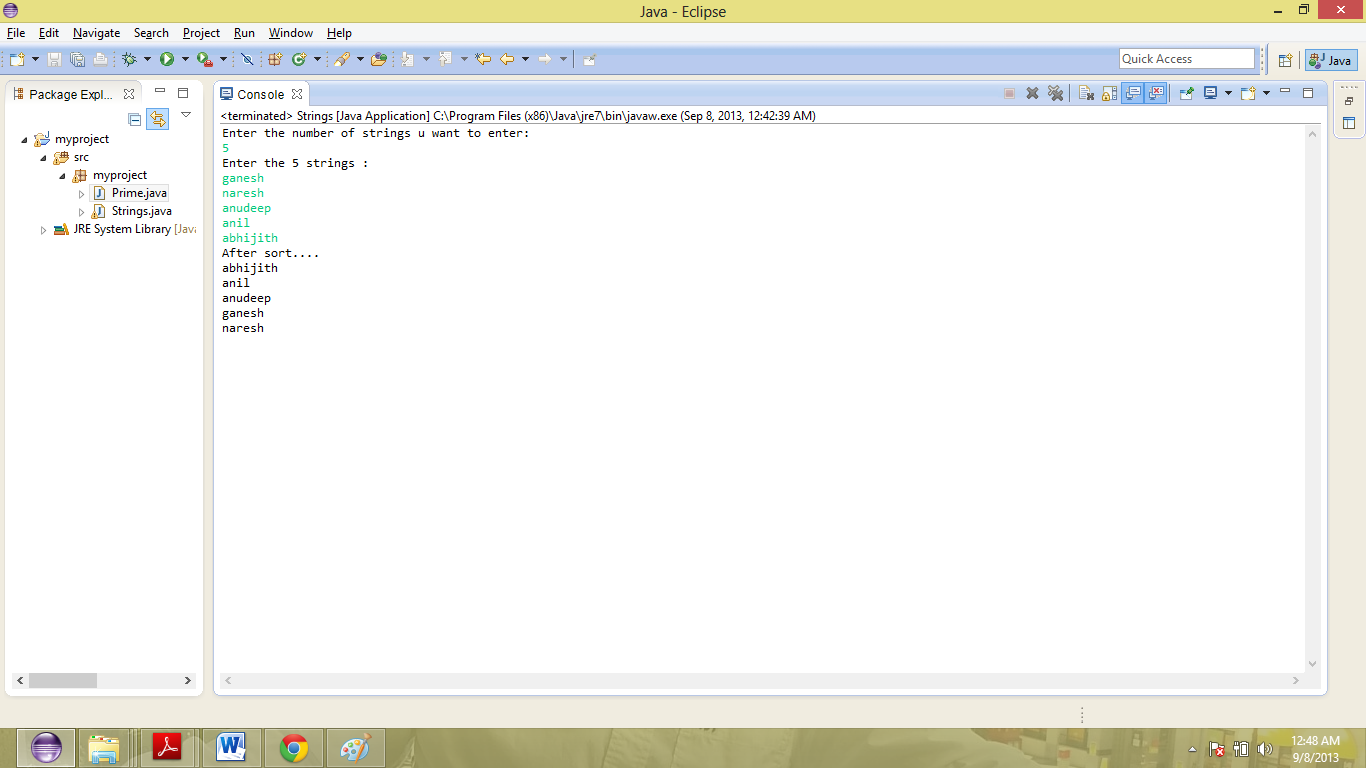
System.*out*.println("After sort....");

**for**(i=0;i<=n-1;i++)

System.*out*.println(s[i]);

}

}

**Output**:

1. **Queue:**

**Introduction:**

Queue is a particular kind of abstract data type or collection in which the entities in the collection are kept in a particular order and the operations on the collection are the addition of entities to the end of the queue, known as enqueue, and removal of entities from the front terminal position, known as dequeue. This makes the queue a First-In-First-Out (FIFO) data structure. In a FIFO data structure, the first element added to the queue will be the first one to be removed. This is equivalent to the requirement that once a new element is added, all elements that were added before have to be removed before the new element can be removed. A queue is an example of a linear data structure, or more abstractly a sequential collection.

**Programming question description :**

Q) The following program does various operations on queue like the insertion, displaying and deletion, the type of operation to be performed is selected by the user each time and display option displays all the elements in the queue.

**Source Code:**

**package** myproject;

**import** java.io.\*;

**import** java.lang.\*;

**class** clrqueue

{

DataInputStream get=**new** DataInputStream(System.*in*);

**int** a[];

**int** i,front=0,rear=0,n,item,count=0;

**void** getdata()

{

**try**

{

System.*out*.println("Enter the limit");

n=Integer.*parseInt*(get.~~readLine~~());

a=**new** **int**[n];

}

**catch**(Exception e)

{

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

}

}

**void** enqueue()

{

**try**

{

**if**(count<n)

{

System.*out*.println("Enter the element to be added:");

item=Integer.*parseInt*(get.~~readLine~~());

a[rear]=item;

rear++;

count++;

}

**else**

System.*out*.println("QUEUE IS FULL");

}

**catch**(Exception e)

{

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

}

}

**void** dequeue()

{

**if**(count!=0)

{

System.*out*.println("The item deleted is:"+a[front]);

front++;

count--;

}

**else**

System.*out*.println("QUEUE IS EMPTY");

**if**(rear==n)

rear=0;

}

**void** display()

{

**int** m=0;

**if**(count==0)

System.*out*.println("QUEUE IS EMPTY");

**else**

{

**for**(i=front;m<count;i++,m++)

System.*out*.println(" "+a[i%n]);

}

}

}

**class** myclrqueue

{

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

{

DataInputStream get=**new** DataInputStream(System.*in*);

**int** ch;

clrqueue obj=**new** clrqueue();

obj.getdata();

**try**

{

**do**

{

System.*out*.println(" 1.Enqueue 2.Dequeue 3.Display 4.Exit");

System.*out*.println("Enter the choice");

ch=Integer.*parseInt*(get.~~readLine~~());

**switch** (ch)

{

**case** 1:

obj.enqueue();

**break**;

**case** 2:

obj.dequeue();

**break**;

**case** 3:

obj.display();

**break**;

}

}

**while**(ch!=4);

}

**catch**(Exception e)

{

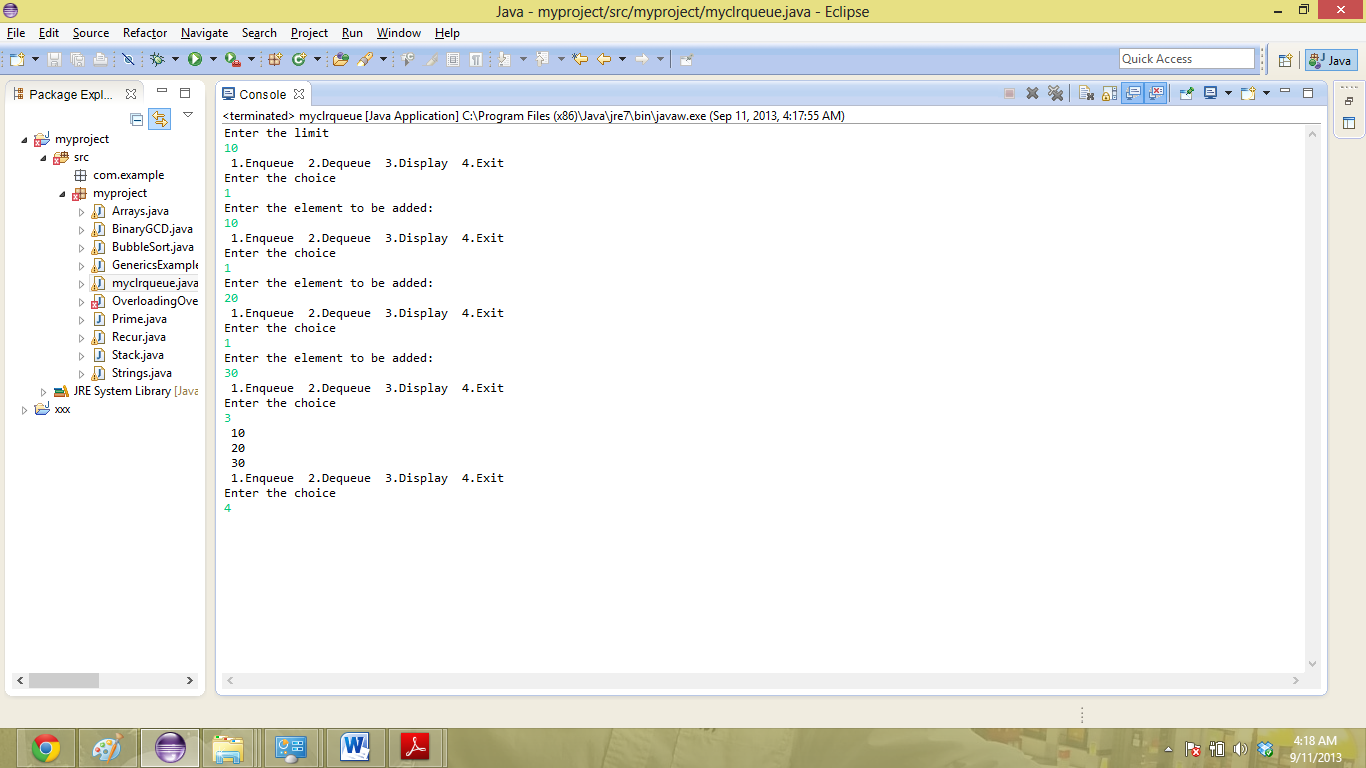
System.*out*.println(e.getMessage());

}

}

}

**Output**:



1. **Stack :**

**Introduction:**

A stack is a particular kind of abstract data type or collection in which the only operations on the collection are the addition of an entity to the collection, known as push and removal of an entity, known as pop. The relation between the push and pop operations is such that the stack is a Last-In-First-Out (LIFO) data structure. In a LIFO data structure, the last element added to the structure must be the first one to be removed. Stack is considered as a linear data structure, or more abstractly a sequential collection, the push and pop operations occur only at one end of the structure, referred to as the top of the stack. The addition and deletion of the elements for the stack is done only through the top of the stack.

**Programming question description :**

Q) The following converts the and given Infix expression to a post fix expression, the value of the infix expression can be given by the user as an input.

**Source Code:**

package myproject;

import java.io.\*;

class stack

{

char stack1[]=new char[20];

int top;

void push(char ch)

{

top++;

stack1[top]=ch;

}

char pop()

{

char ch;

ch=stack1[top];

top--;

return ch;

}

int pre(char ch)

{

switch(ch)

{

case '-':return 1;

case '+':return 1;

case '\*':return 2;

case '/':return 2;

}

return 0;

}

boolean operator(char ch)

{

if(ch=='/'||ch=='\*'||ch=='+'||ch=='-')

return true;

else

return false;

}

boolean isAlpha(char ch)

{

if(ch>='a'&&ch<='z'||ch>='0'&&ch=='9')

return true;

else

return false;

}

void postfix(String str)

{

char output[]=new char[str.length()];

char ch;

int p=0,i;

for(i=0;i<str.length();i++)

{

ch=str.charAt(i);

if(ch=='(')

{

push(ch);

}

else if(isAlpha(ch))

{

output[p++]=ch;

}

else if(operator(ch))

{

if(stack1[top]==0||(pre(ch)>pre(stack1[top]))||stack1[top]=='(')

{

push(ch);

}

}

else if(pre(ch)<=pre(stack1[top]))

{

output[p++]=pop();

push(ch);

}

else if(ch=='(')

{

while((ch=pop())!='(')

{

output[p++]=ch;

}

}

}

while(top!=0)

{

output[p++]=pop();

}

for(int j=0;j<str.length();j++)

{

System.out.print(output[j]);

}

}

}

class intopost

{

public static void main(String[] args)throws Exception

{

String s;

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

stack b=new stack();

System.out.println("Enter an Infix expression");

s=br.readLine();

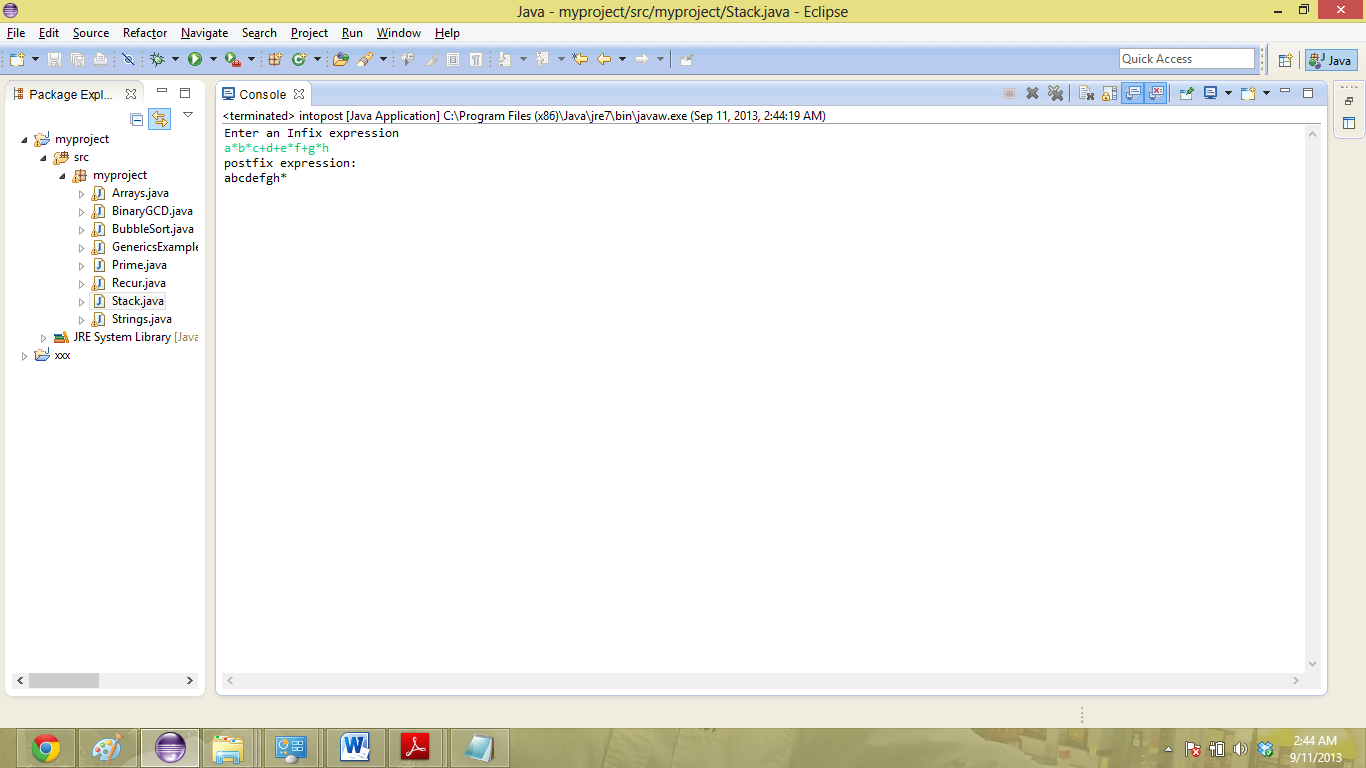
//System.out.println("Input String:"+s);

System.out.println("postfix expression:");

b.postfix(s);

}

}

**Output**:

1. **Overloading**:

**Introduction:**

In Java it is possible to define two or more methods within the same class that share the same name, as long as their parameter declarations are different. When this is the case, the methods are said to be overloaded, and the process is referred to as method overloading. Method overloading is one of the ways that Java implements polymorphism. Overloaded methods must differ in the type and/or number of their parameters. While overloaded methods may have different return types, the return type alone is insufficient to distinguish two versions of a method.

Overloading constructors:

It's common to overload constructors - define multiple constructors which differ in number and/or types of parameters. Just similar to method overloading also in constructor overloading the return type alone is insufficient to distinguish two versions of a constructor..

**Programming question description:**

Q) The following program explains the method overloading in java, here there is method named test which have 3 signatures.

**Source Code:**

package myproject;

public class Squid {

public void test(Object x) {

System.out.println("in the test method that takes an object");

}

public void test(String x) {

System.out.println("in the test method that takes a string");

}

public static void main(String[] args) {

Squid s = new Squid();

String x = "hi there";

Object y = x;

s.test(x);

s.test(y);

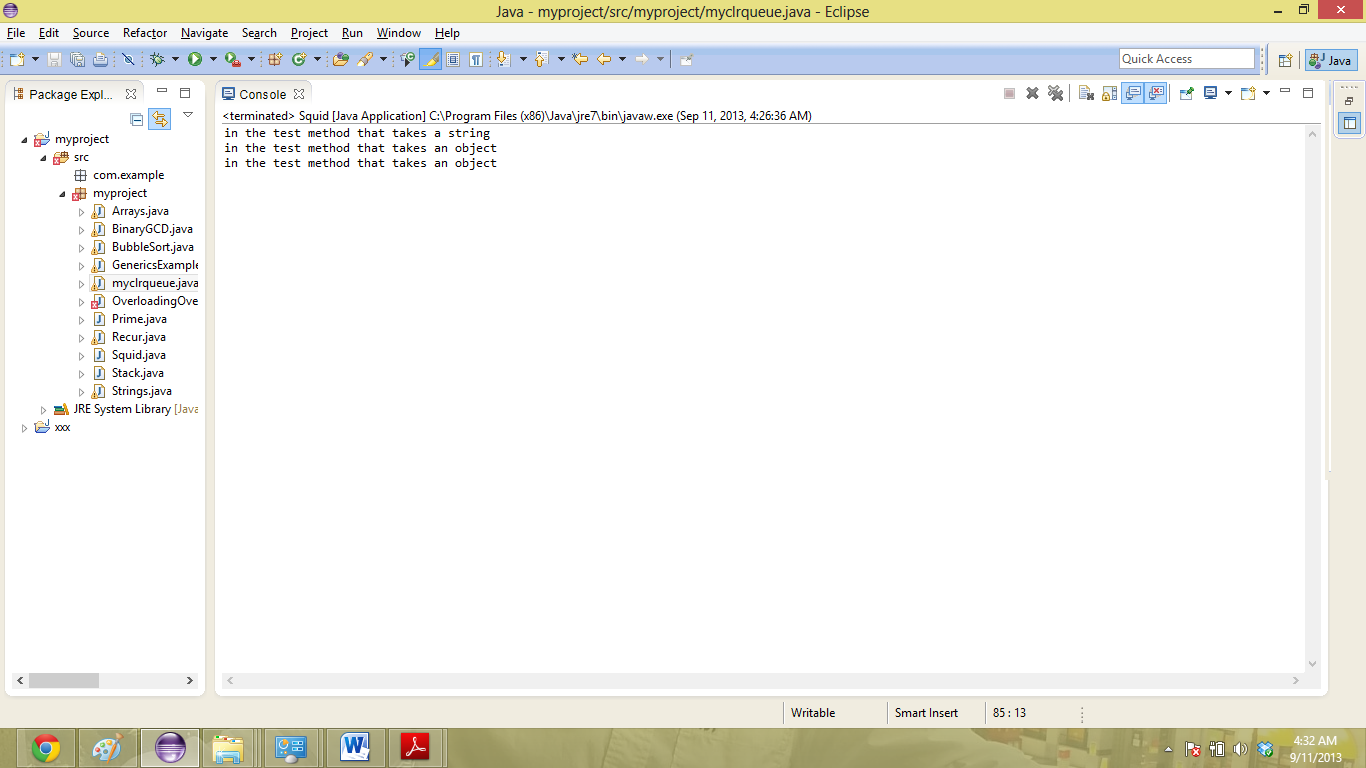
// note that here the upcast has an effect!!

s.test( (Object) x);

}

}

**Output:**



1. **Constructors, Static methods:**

**Introduction:**

**Constructors:**

In java the constructor is a special type of method called to create an object, by using the concept of constructor overloading there can be more than one constructor for a class, some constructors accept the parameters and some are called without any parameters. The name of the constructor is same as the name of its class. Parameterized constructors are the ones which accept the parameters.

Syntax:

The syntax of defining a constructor is as follows:

ClassName();

The syntax of creating an object is:

ClassName obj= new ClassName();

**Static methods:**

Java is object oriented but once in a while we have a special case that some methods can be called without an instance of a class that is without a class (like the math method). Any methos can be declared as static keeping the word static in front of it for example:

Public static int max(int a, int b) (

//this method returns the max of the two values a and b.

)

**Programming question description:**

Q) The following program takes two arrays of integers as arguments and returns true if they contain the same number of elements and all corresponding pairs of elements are equal an false if they are not and this uses a static method eq.

**Source Code:**

package myproject;

public class ArrayEquals {

// return true if two integer arrays have same length and all

// corresponding pairs of integers are equal

public static boolean eq(int[] a, int[] b) {

// same length?

if (a.length != b.length) return false;

// check each corresponding pair

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

if (a[i] != b[i]) return false;

}

// all elements must be equal

return true;

}

// test client

public static void main(String[] args) {

int[] a = { 3, 1, 4, 1, 5 };

int[] b = { 3, 1, 4, 1 };

int[] c = { 3, 1, 4, 1, 5 };

int[] d = { 2, 7, 1, 8, 2 };

System.out.println(eq(a, a));

System.out.println(eq(a, b));

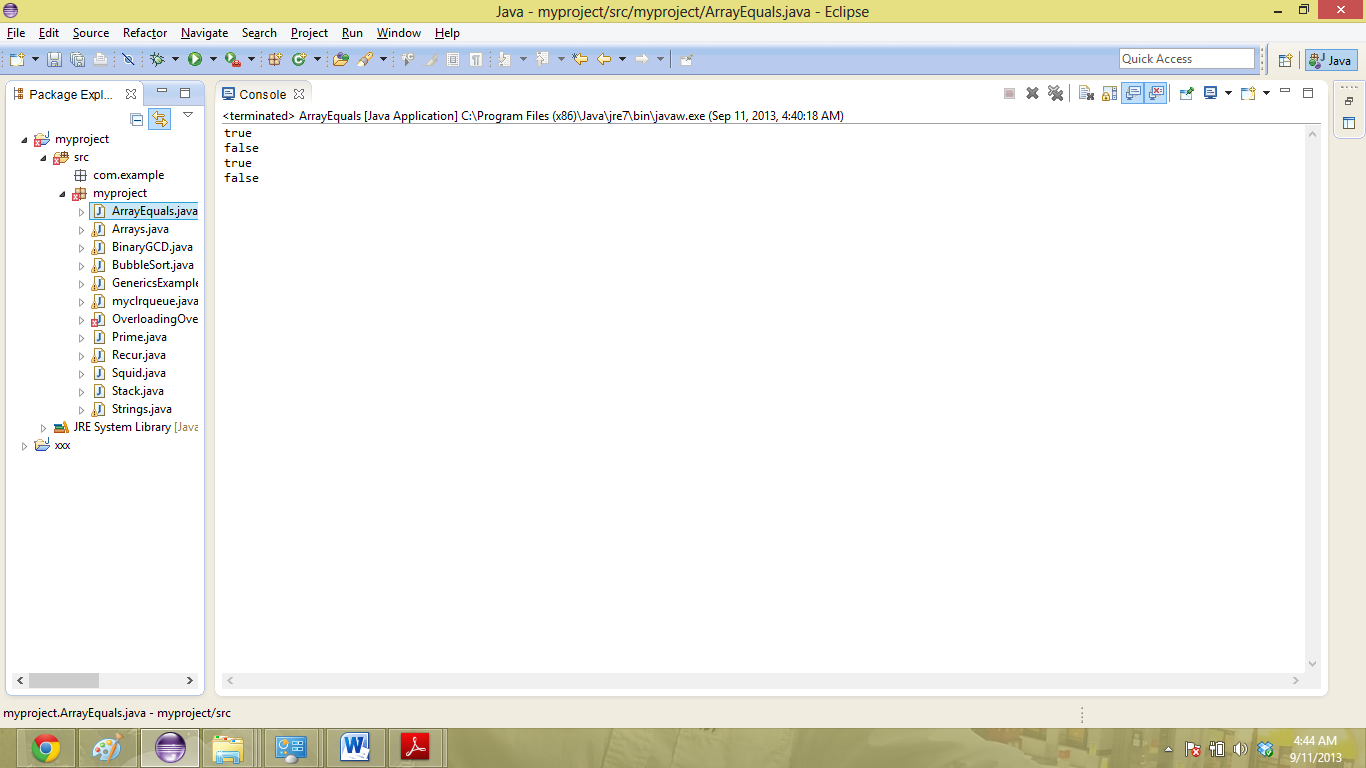
System.out.println(eq(a, c));

System.out.println(eq(a, d));

}

}

**Output**:



1. **Inheritance:**

**Introduction:**

Inheritance is the concept of deriving the properties form super class to the child class, a class that is derived from another class is called a subclass (also a derived class, extended class, or child class). The class from which the subclass is derived is called a superclass (also a base class or a parent class). The idea of inheritance is simple but powerful: When you want to create a new class and there is already a class that includes some of the code that you want, you can derive your new class from the existing class. In doing this, you can reuse the fields and methods of the existing class without having to write (and debug!) them yourself.

Syntax:

class SubClass extends SuperClass {

……

}

The extends keyword is used to inherit the properties of the super class by the sub class.

Example:

public class Animal{

}

public class Mammal extends Animal{

}

public class Reptile extends Animal{

}

public class Dog extends Mammal{

}

In the designing point of view inheritance is “**IS-A relationship”**.

**Programming question description:**

Q) The below program explains the concept of inheritance in the object oriented programming, in which there are 2 classes A and B where A is a super class and B is a subclass, and the subclass B inherits the super class A using the extend keyword.

**Source Code:**

package myproject;

class A{

public static void printStatic(){

System.out.println("this is In class A");

}}

class B extends A{

public static void printStatic(){

System.out.println("this is In B");

}

}

class Test{

@SuppressWarnings("static-access")

public static void main(String[] args){

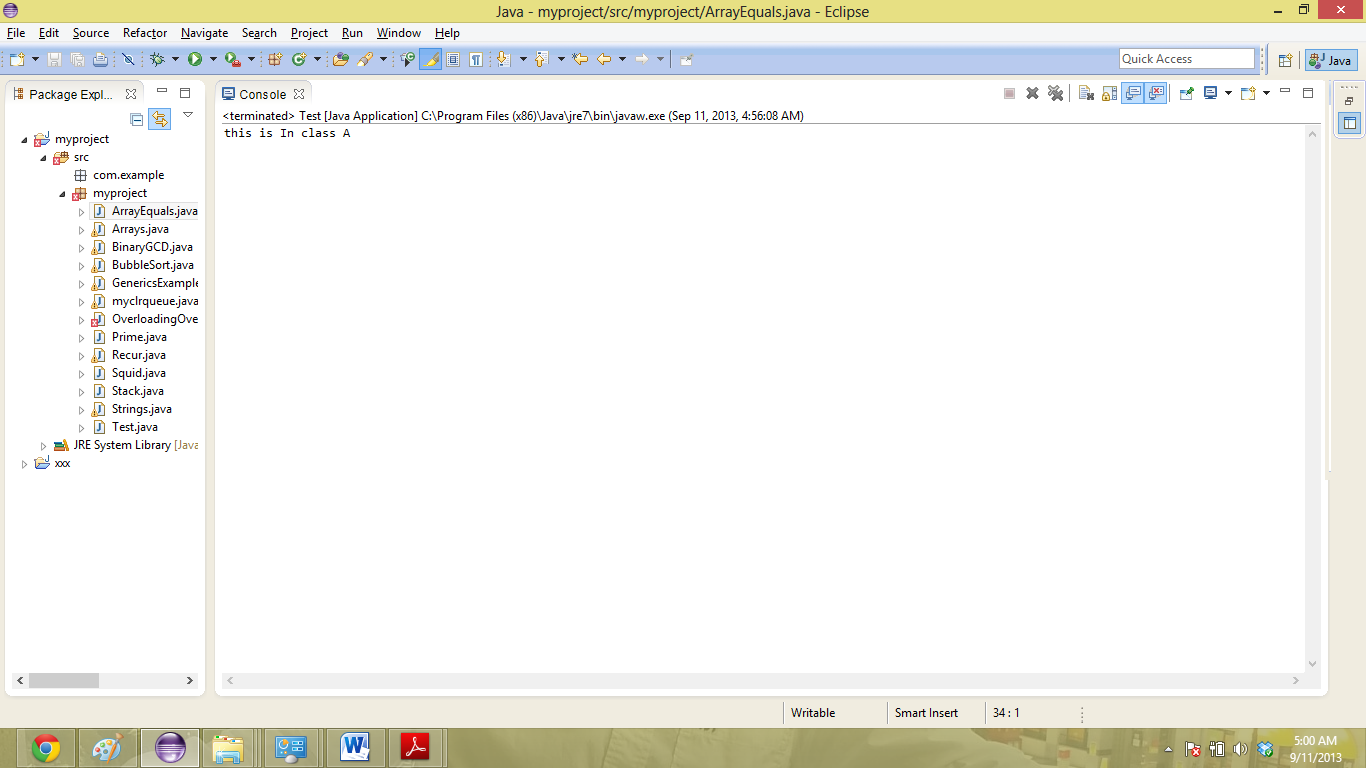
A a=new B();

a.printStatic();

}

}

**Output**:



**8)** **Arrays:**

**Introduction:**

Array is a container object which holds a fixed number of values of a single type and the length of an array is established when it is created and is fixed once it is created. Each element in an array is called an element and each element is accessed by its numerical index.

Syntax:

The following syntax is used to declare and initialize an array in Java.

int[] array\_name = new int[m];

the above command declares an array of size m of one dimension.

For declaring a two dimensional array the syntax is as follows :

int[][] array\_name = new int[m][n];

**Programming question description:**

Q) The problem here is to find the product of two matrices of any dimensions given by the user as an input and also with the user inputs of the matrix elements.

A) Here the user gives the dimensions of the matrixes as inputs m1,n1,m2,n2 and we have to check whether the matrix multiplication is possible or not, as the matrix multiplication is possible only if n1==m2. If the matrix multiplication is possible then the user can give the elements of the matrix as inputs and we use the following code to compute their product.

**Source Code:**

package myproject;

import java.io.\*;

import java.util.Scanner;

class Arrays {

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

int m1,n1,m2,n2;

System.out.println("enter the dimensions of matrix1, m1: ");

Scanner s= new Scanner(System.in);

m1=s.nextInt();

System.out.println("enter the dimensions of matrix1, n1: ");

n1=s.nextInt();

System.out.println("enter the dimensions of matrix2, m2: ");

m2=s.nextInt();

System.out.println("enter the dimensions of matrix2, n2: ");

n2=s.nextInt();

if(n1==m2&&m1>0&&n1>0&&m2>0&&n2>0)

{

int[][] mat1=new int[m1][n1];

int[][] mat2=new int[m2][n2];

int[][] mat3=new int[m1][n2];

int i,j,k;

System.out.println("enter the values of matrix 1: ");

for(i=0;i<m1;i++){

for(j=0;j<n1;j++){

mat1[i][j]= s.nextInt();

}

}

System.out.println("enter the values of matrix 2: ");

for(i=0;i<m1;i++){

for(j=0;j<n1;j++){

mat2[i][j]= s.nextInt();

}

}

for(i=0;i<=m1-1;i++){

for(j=0;j<=n1-1;j++) {

for(k=0;k<=n1-1;k++) {

mat3[i][j]+=mat1[i][k]\*mat2[k][j];

}

}

}

System.out.println("After Matrix Multiplicatin......");

for(i=0;i<=m1-1;i++){

for(j=0;j<=n1-1;j++) {

System.out.print(mat3[i][j]+"\t");

}

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

}

}

else {

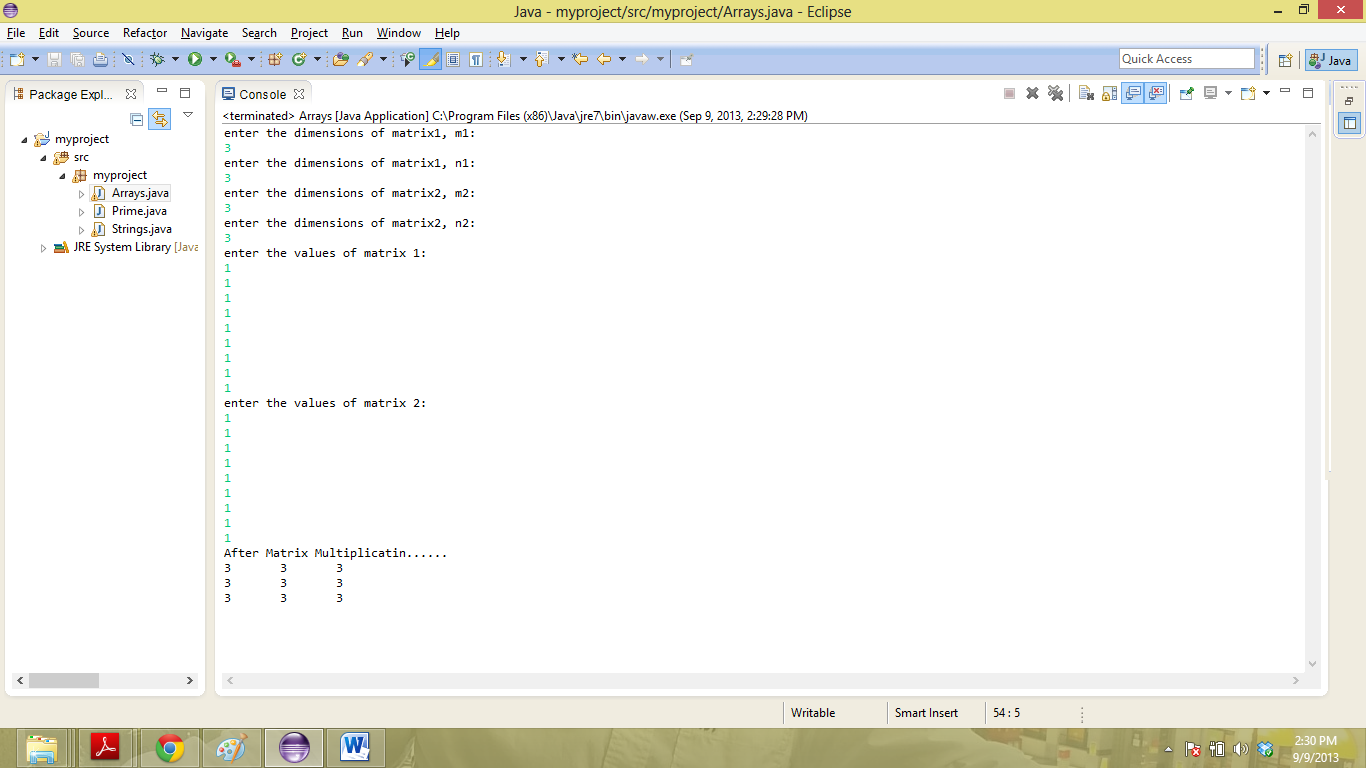
System.out.println("matrix multiplication is not possible");

}

}

}

**Output:**



1. **Polymorphism:**

**Introduction:**

In object oriented programming polymorphism is the ability to create a variable, a function, or an object that has more than one form. Polymorphism in Java has two types: Compile time polymorphism (static binding) and Runtime polymorphism (dynamic binding). Method overloading is an example of static polymorphism, while method overriding is an example of dynamic polymorphism.

Static Polymorphism:

In Java, static polymorphism is achieved through method overloading. Method overloading means there are several methods present in a class having the same name but different types/order/number of parameters.

At compile time, Java knows which method to invoke by checking the method signatures. So, this is called compile time polymorphism or static binding.

Dynamic Polymorphism:

Suppose a sub class overrides a particular method of the super class. Let’s say, in the program we create an object of the subclass and assign it to the super class reference. Now, if we call the overridden method on the super class reference then the sub class version of the method will be called.

**Programming problem description:**

The following program will explain the concept of polymorphism

**Source Code:**

package myproject;

class Mode

{

public void display(int num)

{

System.out.println(num);

}

public void display(long num)

{

System.out.println(num);

}

public static void main(String args[])

{

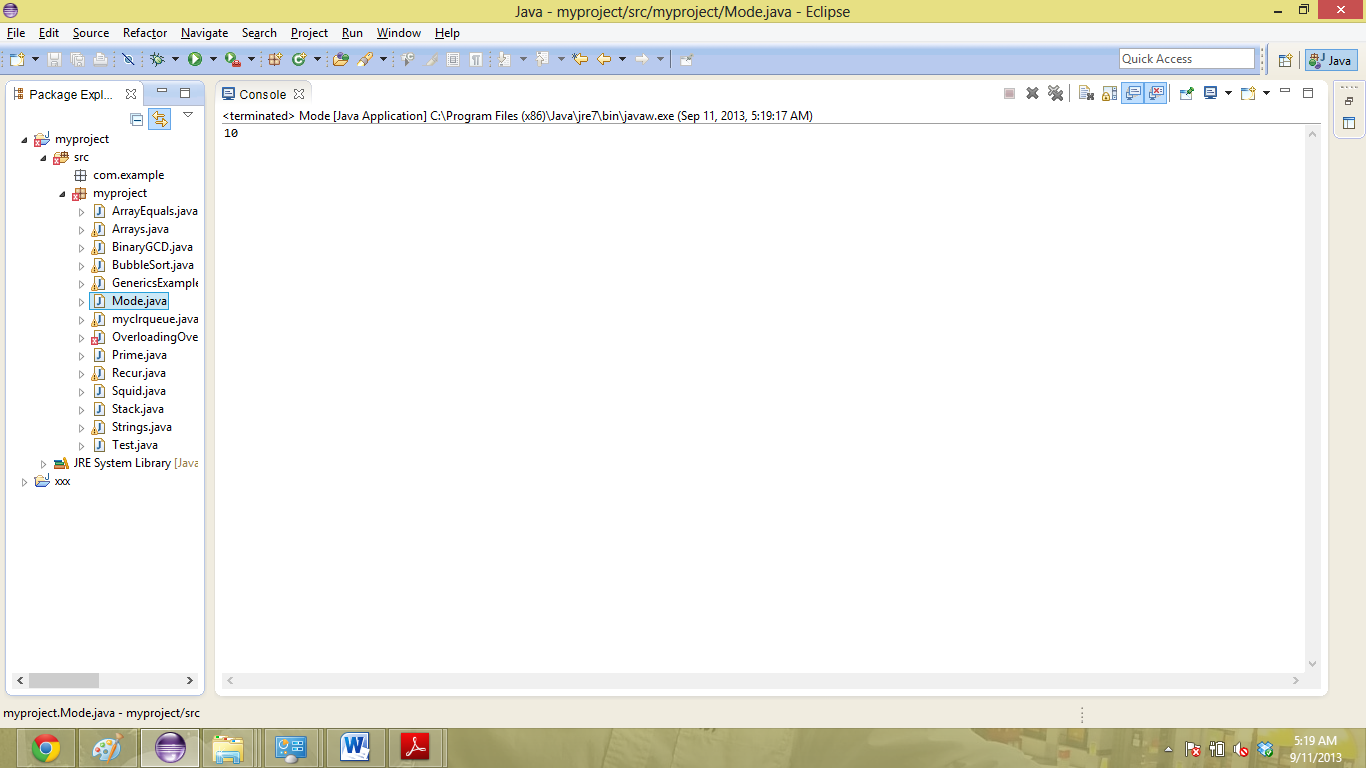
Mode a=new Mode();

a.display(10);

}

}

**Output:**



1. **Recursion:**

**Introduction:**

Recursion is when a method calls itself. It is sometimes used because some problems are easier to solve using a recursive solution than an iterative solution. There is no problem that absolutely requires the use of recursion. It’s just that for some problems it is easier to slove using recursion than to solve using iteration.

Recursion is less efficient and uses more memory than iterative solutions. This is because each time the method calls itself, a new stack frame is created with that method, and all of its state. In severe cases, this can result in overflowing the stack, or running out of memory. Because of this, recursion does have to be used with care.

**Programming problem description:**

Q) the following program takes input of two numbers from the user and prints the GCD(greatest common divisor) of the two numbers as output.

For doing this we use the binary GCD algorithm which is as follows:

gcd(p, q) =

* p if q = 0
* q if p = 0
* 2 \* gcd(p/2, q/2) if p and q are even
* gcd(p/2, q) if p is even and q is odd
* gcd(p, q/2) if p is odd and q is even
* gcd((p-q)/2, q) if p and q are odd and p >= q
* gcd(p, (q-p)/2) if p and q are odd and p < q.

**Source Code:**

package myproject;

import java.util.Scanner;

public class BinaryGCD {

public static int gcd(int p, int q) {

if (q == 0) return p;

if (p == 0) return q;

// p and q even

if ((p & 1) == 0 && (q & 1) == 0) return gcd(p >> 1, q >> 1) << 1;

// p is even, q is odd

else if ((p & 1) == 0) return gcd(p >> 1, q);

// p is odd, q is even

else if ((q & 1) == 0) return gcd(p, q >> 1);

// p and q odd, p >= q

else if (p >= q) return gcd((p-q) >> 1, q);

// p and q odd, p < q

else return gcd(p, (q-p) >> 1);

}

public static void main(String[] args) {

Scanner s= new Scanner(System.in);

System.out.println("Enter the first number");

int p = s.nextInt();

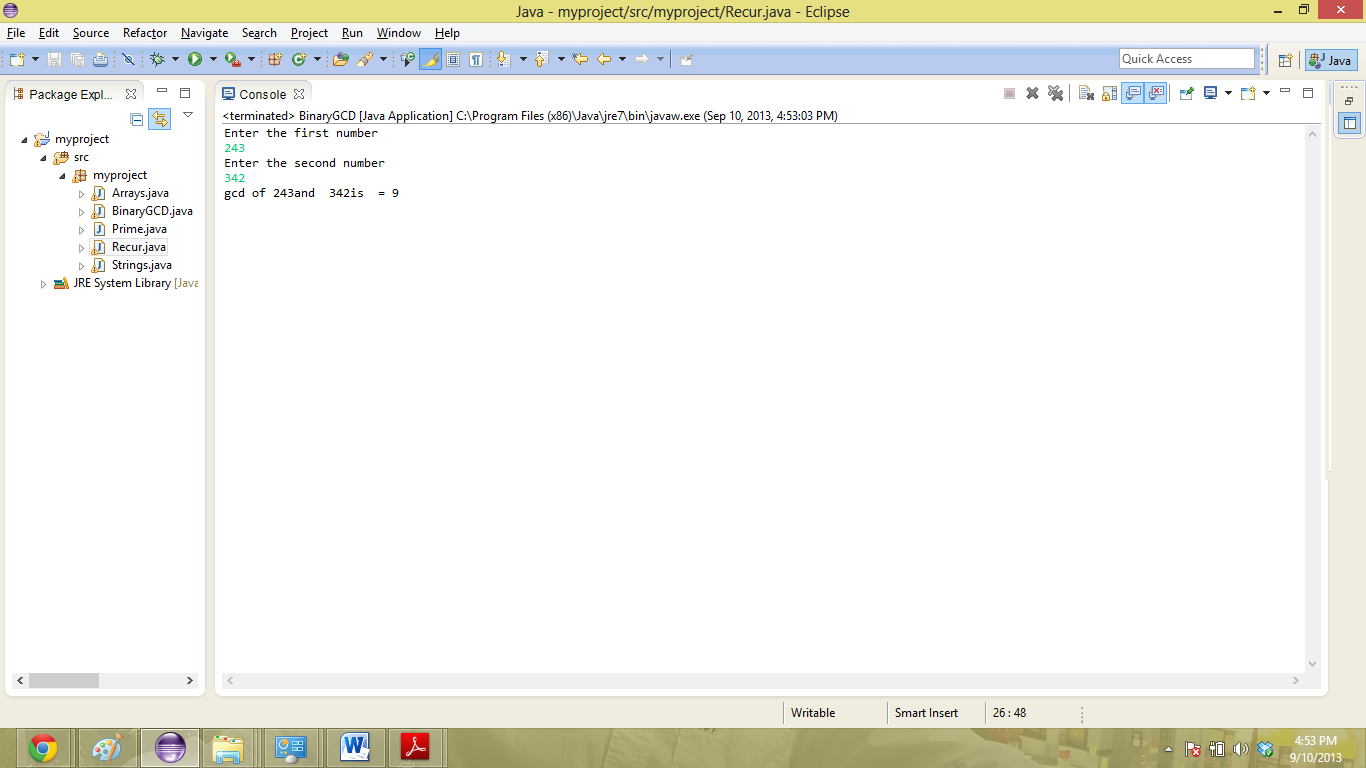
System.out.println("Enter the second number");

int q = s.nextInt();

System.out.println("gcd of " + p + "and " + q + "is = " + gcd(p, q));

}

}

**Output**:****

1. **Template:**

**Introduction:**

A class template or generic class is a blueprint that describes how members of a set of related classes are constructed, They allow "a type or method to operate on objects of various types while providing compile-time type safety. For example:

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

v.add("test");

Integer i = v.get(0);

The type parameter String within the angle brackets declares the ArrayList to be constituted of String. With generics it is no longer necessary to cast the third line to any particular type, because the result of v.get(0) is defined as String by the code generated by the compiler.

A class or an interface is generic if it has one or more type variable. Type variable are delimited by angle brackets and follow the class (or the interface) name:

public interface List<T> extends Collection<T> {

.........

}

**Programming problem description:**

Q) The following program performs a basic sorting operation on the generic list.

**Source Code:**

package myproject;

import java.util.Scanner;

public class BubbleSort<E> {

public static <E> void bubbleSort(E[] unsorted) {

for(int iter =1; iter< unsorted.length; iter++){

for(int inner = 0; inner < (unsorted.length - iter); inner ++){

if((((Comparable) (unsorted[inner])).compareTo(unsorted[inner+1])) > 0){

E tmp = unsorted[inner];

unsorted[inner] = unsorted[inner + 1];

unsorted[inner + 1] = tmp;

}

}

}

}

public static <E> void display(E[] unsorted) {

for(E i : unsorted){

System.out.print( " " + i);

}

System.out.println();

}

public static void main(String[] args) {

Integer[] unsorted = {9,8,7,6,5,4,3,2,1};

System.out.print( "unsorted list: ");

display(unsorted);

bubbleSort(unsorted);

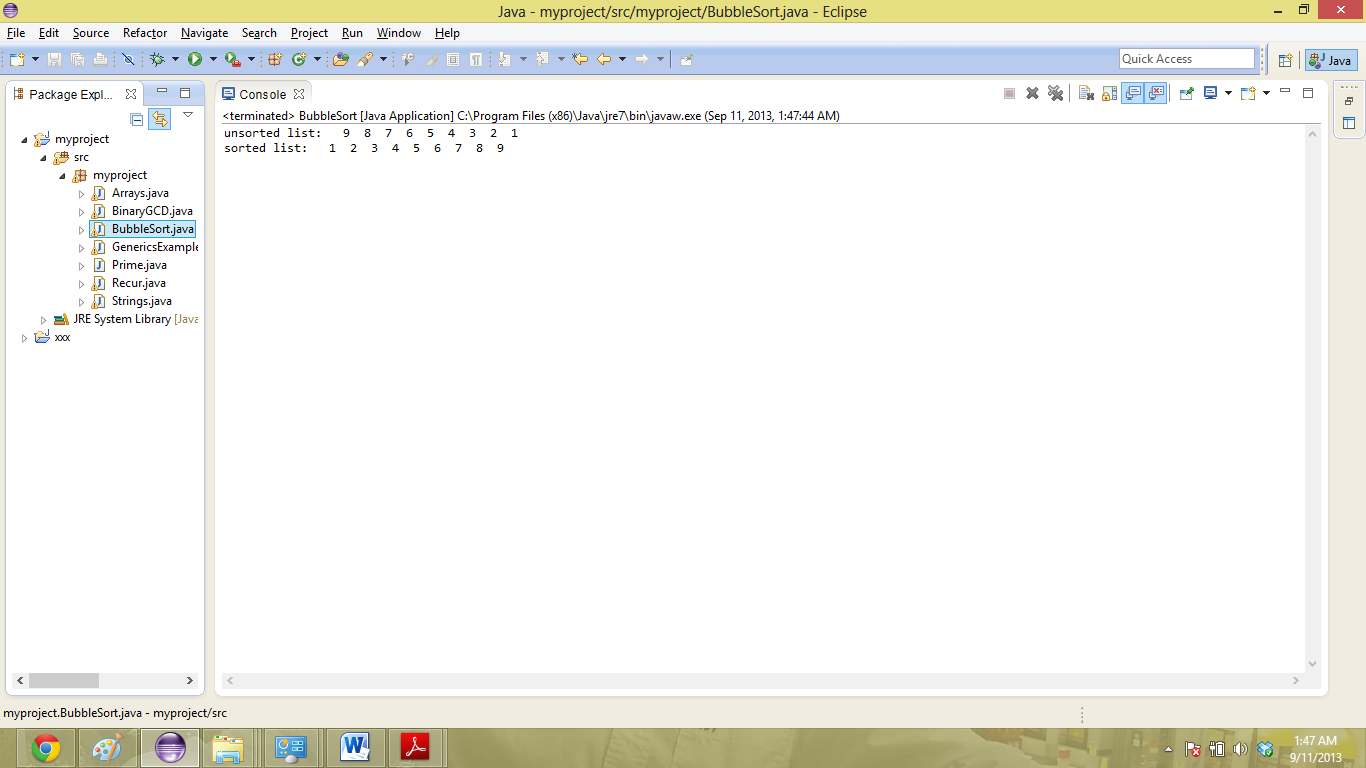
System.out.print( "sorted list: ");

display(unsorted);

}

}

**Output:**

****