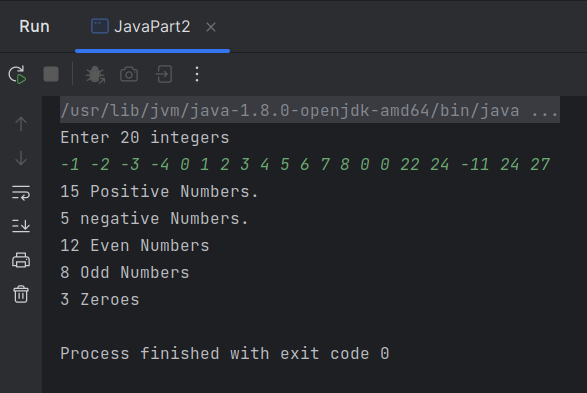
**Java Part-3 Assignment**

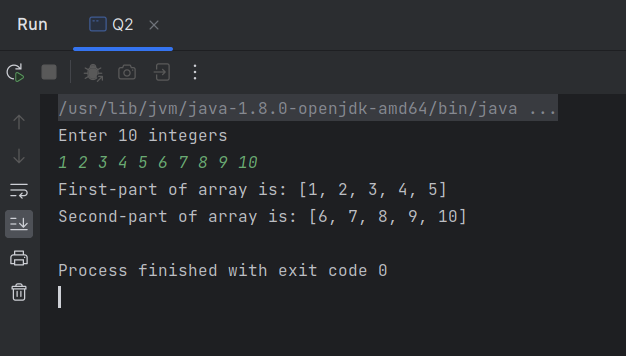
Q1.)

import java.util.\*;  
class PositiveOrNegative{  
 int positiveCount=0;  
 int negativeCount=0;  
 public void count(List<Integer> a) {  
 for (Integer i : a) {  
 if (i >= 0) positiveCount++;  
 else negativeCount++;  
 }  
 System.*out*.println(positiveCount+" Positive Numbers.");  
 System.*out*.println(negativeCount+" negative Numbers.");  
 }  
}  
  
class OddOrEven{  
 int oddCount=0;  
 int evenCount=0;  
 public void count(List<Integer> a){  
 for (Integer i : a) {  
 if (i%2==0) evenCount++;  
 else oddCount++;  
 }  
 System.*out*.println(evenCount+" Even Numbers");  
 System.*out*.println(oddCount+" Odd Numbers");  
 }  
}  
  
class ZeroOrNot{  
 int zeroCount=0;  
 public void count(List<Integer> a){  
 for (Integer i : a) {  
 if (i == 0) zeroCount++;  
 }  
 System.*out*.println(zeroCount+" Zeroes");  
 }  
}  
  
public class JavaPart2 {  
 public static void main(String[] args){  
 List<Integer> inputArray=new ArrayList(20);  
 Scanner inp=new Scanner(System.*in*);  
 System.*out*.println("Enter 20 integers");  
  
 for(int i=0;i<20;i++){  
 inputArray.add(inp.nextInt());  
 }  
 PositiveOrNegative pn=new PositiveOrNegative();  
 pn.count(inputArray);  
  
 OddOrEven oe=new OddOrEven();  
 oe.count(inputArray);  
  
 ZeroOrNot zn=new ZeroOrNot();  
 zn.count(inputArray);  
  
  
 }  
}



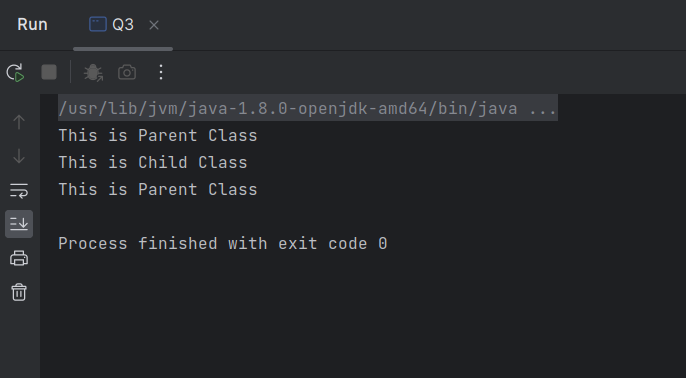
Q2.)

import java.util.ArrayList;  
import java.util.List;  
import java.util.Scanner;  
  
  
public class Q2 {  
 public static void main(String[] args){  
 List<Integer> inputArray = new ArrayList(20);  
 List<Integer> splitArray1 = new ArrayList(5);  
 List<Integer> splitArray2 = new ArrayList(5);  
 Scanner inp = new Scanner(System.*in*);  
 System.*out*.println("Enter 10 integers");  
 int inputValue;  
 for (int i = 0; i < 10; i++) {  
 inputValue=inp.nextInt();  
 inputArray.add(inputValue);  
 if(i<5) splitArray1.add(inputValue);  
 else splitArray2.add(inputValue);  
 }  
 System.*out*.println("First-part of array is: "+splitArray1);  
 System.*out*.println("Second-part of array is: "+splitArray2);  
 }  
}



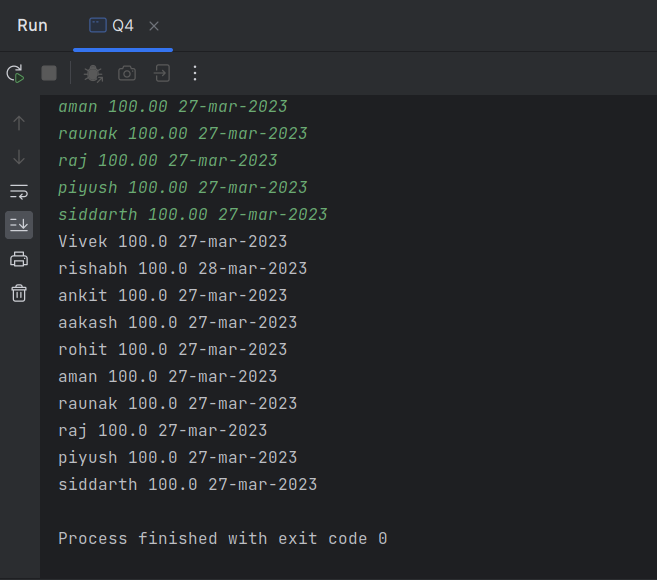
Q3.)

class ParentClass{  
 public void print(){  
 System.*out*.println("This is Parent Class");  
 }  
}  
  
class ChildClass extends ParentClass{  
 ChildClass(){};  
 ChildClass(int a){  
 super.print();  
 }  
 public void print(){  
 System.*out*.println("This is Child Class");  
 }  
}  
public class Q3 {  
 public static void main(String[] args){  
 ParentClass pc=new ParentClass();  
 pc.print();  
 ChildClass cc=new ChildClass();  
 cc.print();  
 ChildClass cc1=new ChildClass(1);  
  
  
 }  
}

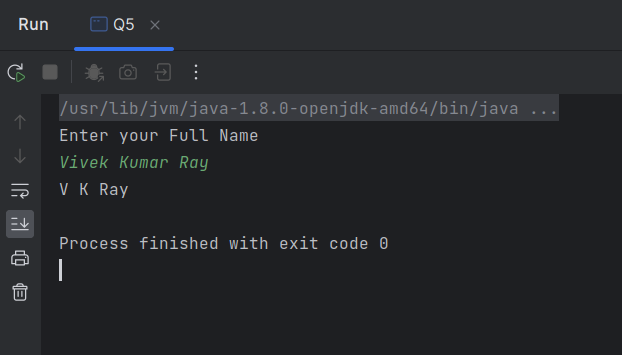


Q4.)

import java.util.ArrayList;  
import java.util.Date;  
import java.util.List;  
import java.util.Scanner;  
class Employee{  
 String name;  
 double salary;  
 String date;  
}  
public class Q4 {  
 public static void main(String[] args) {  
 List<Employee> inputArray = new ArrayList(10);  
 Scanner inp = new Scanner(System.*in*);  
 System.*out*.println("Enter Name, Salary and Date of joining of 10 Employee ");  
  
 for (int i = 0; i < 10; i++) {  
 Employee e=new Employee();  
 e.name=inp.next();  
 e.salary=inp.nextDouble();  
 e.date=inp.next();  
 inputArray.add(e);  
 }  
 for(Employee i:inputArray) {  
 System.*out*.println(i.name+" "+i.salary+" "+i.date);  
 }  
 }  
}



Q5.)

import java.util.Scanner;  
class Name{  
 String firstName;  
 String middleName;  
 String lastName;  
}  
public class Q5 {  
 public static void main(String[] args) {  
 System.*out*.println("Enter your Full Name");  
 Scanner inp = new Scanner(System.*in*);  
 for (int i = 0; i < 1; i++) {  
 Name e = new Name();  
 e.firstName = inp.next();  
 e.middleName = inp.next();  
 e.lastName = inp.next();  
 System.*out*.println(e.firstName.charAt(0)+ " " + e.middleName.charAt(0)+ " " + e.lastName);  
 }  
  
 }  
}

Q6) Both equals() method and the == operator are used to compare two objects in Java.

== is an operator and equals() is method. But == operator compares reference or memory

location of objects in the heap, whether they point to the same location or not.

Q7)

* 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.

* StringBuffer is less efficient than StringBuilder.

StringBuilder is more efficient than StringBuffer.

Q8) The final keyword is a non-access modifier used for classes, attributes and methods, which makes them non-changeable (impossible to inherit or override).

Q9) The finally block may not execute if the JVM exits while the try or catch code is being executed. The try block of the writeList method opens a PrintWriter . The program should close that stream before exiting the writeList method.

Q10) In shallow copy, only fields of the primitive data type are copied while the objects' references are not copied. Deep copy involves the copy of primitive data types as well as to object references.

Q11) ArithematicException will be shown.

Q12) Clean up typically means closing db connections, closing IO streams etc. finally{} block is always invoked even if exception has occurred.

Q13) ArrayIndexOutOfBoundsException will be there

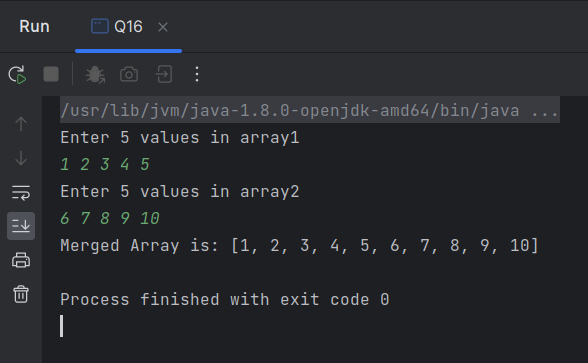
Q14) A-3 object

B-1 object

C-4 object

Q15) isEmpty() Method : The isEmpty() method checks whether a string is empty or not. This method returns true if the string is empty (length() is 0), and false if not.

Q16)

import java.util.ArrayList;  
import java.util.List;  
import java.util.Scanner;  
  
public class Q16 {  
 public static void main(String[] args) {  
 List<Integer> inputArray1 = new ArrayList(5);  
 List<Integer> inputArray2 = new ArrayList(5);  
 List<Integer> mergedArray = new ArrayList(10);  
 Scanner inp = new Scanner(System.*in*);  
  
 System.*out*.println("Enter 5 values in array1");  
 for(int i=0;i<5;i++) {  
 inputArray1.add(inp.nextInt());  
 }  
  
 System.*out*.println("Enter 5 values in array2");  
 for(int i=0;i<5;i++) {  
 inputArray2.add(inp.nextInt());  
 }  
  
 mergedArray.addAll(inputArray1);  
 mergedArray.addAll(inputArray2);  
  
 System.*out*.println("Merged Array is: " + mergedArray);  
  
  
 }  
}

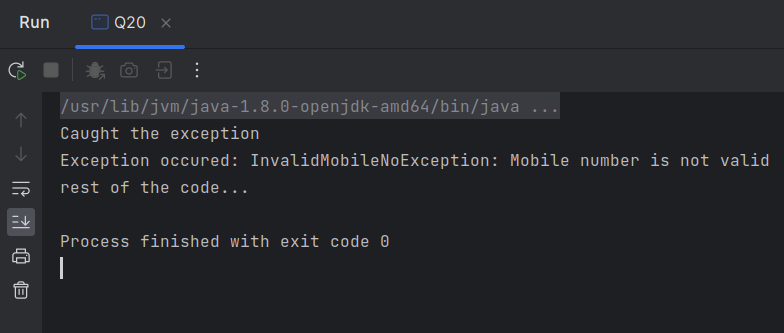
Q17) fill()

Q18) Generics

Q19) A comparable interface is implemented by all the String class and wrapper classes. Moreover, custom objects use a comparable interface for sorting. On the other hand, the comparator interface is chiefly used to sort the custom objects. You can use it to compare the objects of different classes too.

Q20)

class InvalidMobileNoException extends Exception{  
 public InvalidMobileNoException(String str){  
 super(str);  
 }  
}  
  
public class Q20 {  
 static void validate (String mobNo) throws InvalidMobileNoException{  
 if(mobNo.length()!=10){  
 throw new InvalidMobileNoException("Mobile number is not valid ");  
  
 }  
 else {  
  
 System.*out*.println("Mobile Number Registered");  
 }  
 }  
  
 public static void main(String[] args) {  
 try  
 {  
 // calling the method  
 *validate*("98875563456");  
 }  
 catch (InvalidMobileNoException ex)  
 {  
 System.*out*.println("Caught the exception");  
  
 // printing the message from InvalidMobileNoException object  
 System.*out*.println("Exception occured: " + ex);  
 }  
  
 System.*out*.println("rest of the code...");  
 }  
 }



Q21) false true