**Vision of the Department**

Providing quality education to enable the generation of socially conscious software engineers who can contribute to the advancement in the field of computer science and engineering.

**Mission of the Department**

* To equip the graduates with the knowledge and skills required to enable them to be industry ready.
* To train socially responsible, disciplined engineers who work with good leadership skills and can contribute for nation building.
* To make our graduates proficient in cutting edge technologies through student centric teaching-learning process and empower them to contribute significantly to the software industry.
* To shape the department into a Centre of academic and research excellence.

**Program Educational Objectives**

**PEO-1**

To provide the graduates with solid foundation in Computer Science and Engineering along with the fundamentals of Mathematics and Sciences with a view to impart in them high quality technical skills like modelling, analysing, designing, programming and implementation with global competence and helps the graduates for life-long learning.

**PEO-2**

To prepare and motivate graduates with recent technological developments related to core subjects like Programming, Databases, Design of Compilers and Network Security aspects and future technologies so as to contribute effectively for Research & Development by participating in professional activities like publishing and seeking copy rights.

**PEO-3**

To train graduates to choose a decent career option either in high degree of employability/Entrepreneur or, in higher education by empowering students with ethical administrative acumen, ability to handle critical situations and training to excel in competitive examinations.

**PEO-4**

To train the graduates to have basic interpersonal skills and sense of social responsibility that paves them a way to become good team members and leaders.

**Program Outcomes (POs)**

**1. Engineering knowledge:** apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.

**2. Problem analysis:** identify, formulate, review research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural science and engineering sciences.

**3. Design/development of solutions:** design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal and environmental considerations.

**4. Conduct investigations of complex problems:** use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

**5. Modern tool usage:** create, select and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

**6. The engineer and society:** apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice**.**

**7. Environment sustainability:** understand the impact of the professional engineering solutions in the societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

**8. Ethics:** apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

**9. Individual and team work:** function effectively as an individual and as a member or leader in diverse teams, and in multidisciplinary settings.

**10. Communication:** communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

**11. Project management and finance:** demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

**12. Lifelong learning:** recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broader context of technological change**.**

**Program Specific Outcomes (PSOs)**

**PSO-1: Professional Skills:** The ability to understand, analyse and develop computer programs in the areas related to algorithms, system software, multimedia, web design, big data analytics, and networking for efficient design of computer based systems of varying complexity.

**PSO-2: Successful Career and Entrepreneurship:** The ability to employ modern computer languages, environments, and platforms in creating innovative career paths to be an entrepreneur and a zest for higher studies/employability in the field of Computer Science & Engineering.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **II- Year I- Semester** | **Name of the Course** | **L** | **T** | **P** | **C** |
|  | **Java Programming Lab** | **0** | **0** | **3** | **1.5** |

**Course Objectives:**

1. To write programs using abstract classes.
2. To write programs for solving real world problems using java collection frame work.
3. To write multithreaded programs.
4. To design GUI application using swing controls.
5. To introduce java compiler and eclipse platform
6. To impart hands on experience with java programming.

**Course Outcomes:** at the end of the lab, the student will be able to

CO1: Implement object-oriented programming concepts, and apply them in solving problems. (Apply)

CO2: Experiment the implementation of packages and interfaces. (Apply)

CO3: Experiment the concept of multithreading over single threaded programming. (Analyze)

CO4: Use generic data structures of collection framework to manipulate data. (Apply)

CO5: Test the GUI based network applications among multiple users through network programming. (Analyze)

**CO-PO mapping Table**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CO/ PO-PSO | PO1 | PO2 | PO3 | PO4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO 11 | PO 12 | PSO1 | PSO2 |
| CO1 | 2 |  | 2 |  | 3 | 1 |  | 2 | 2 | 2 |  |  | 2 | 2 |
| CO2 | 2 |  | 2 |  | 3 | 1 |  | 2 | 2 | 2 |  |  | 2 | 2 |
| CO3 | 2 | 2 | 2 |  | 3 | 1 |  | 2 | 2 | 2 |  |  | 2 | 2 |
| CO4 | 2 |  | 2 |  | 3 | 1 |  | 2 | 3 | 2 |  | 2 | 2 | 2 |
| CO5 | 2 | 2 | 2 |  | 3 | 1 |  | 2 | 3 | 2 |  | 2 | 2 | 2 |

# Note:

Mandatory to follow test driven development with Eclipse IDE empowered JUnit testing framework and code coverage plugin.

The list suggests the minimum program set. Hence, the concerned staff is requested to add more problems to the list as needed.

**List of Experiments**

1. Create a class called Invoice that a hardware store might use to represent an invoice for an item sold at the store. An Invoice should include four pieces of information as instance variables-a part number (type String),a part description(type String),a quantity of the item being purchased (type int) and a price per item (double). Your class should have a constructor that initializes the four instance variables. Provide a set and a get method for each instance variable. In addition, provide a method named getInvoiceAmount() that calculates the invoice amount (i.e., multiplies the quantity by the price per item), then returns the amount as a double value. If the quantity is not positive, it should be set to 0. If the price per item is not positive, it should be set to 0.0. Write a test application named InvoiceTest that demonstrates class Invoice’s capabilities. [CO1]

2. Develop a Java application to generate Electricity bill. Create a class with the following members: Consumer no., consumer name, previous month reading, current month reading, and type of EB connection (i.e. domestic or commercial). Compute the bill amount using the following tariff. [CO1]

If the type of the EB connection is domestic, calculate the amount to be paid as follows:

* 1. First 100 units - Rs. 1 per unit
  2. 101-200units - Rs. 2.50 per unit
  3. 201 -500 units - Rs. 4 per unit
  4. >501 units - Rs. 6 per unit

If the type of the EB connection is commercial, calculate the amount to be paid as follows:

* 1. First 100 units - Rs. 2 per unit
  2. 101-200units - Rs. 4.50 per unit
  3. 201 -500 units - Rs. 6 per unit
  4. >501 units - Rs. 7 per unit

3. Create class Savings Account. Use a static variable annualInterestRate to store the annual interest rate for all account holders. Each object of the class contains a private instance variable savingsBalance indicating the amount the saver currently has on deposit. Provide method calculateMonthlyInterest to calculate the monthly interest by multiplying the savingsBalance by annualInterestRate divided by 12 this interest should be added to savings Balance. Provide a static method modifyInterestRate that sets the annualInterestRate to a new value. Write a program to test class SavingsAccount. Instantiate two savingsAccount objects, saver1 and saver2, with balances of $2000.00 and $3000.00, respectively. Set annualConcentration Rate to 4%, then calculate the monthly interest and print the new balances for both savers. Then set the annualInterestRate to 5%, calculate the next month’s interest and print the new balances for both savers. [CO1]

4. Create a class called Book to represent a book. A Book should include four pieces of information as instance variables; a book name, an ISBN number, an author name and a publisher. Your class should have a constructor that initializes the four instance variables. Provide a mutator method and accessor method (query method) for each instance variable. In addition, provide a method named getBookInfo that returns the description of the book as a String (the description should include all the information about the book). You should use this keyword in member methods and constructor. Write a test application named BookTest to create an array of object for 30 elements for class Book to demonstrate the class Book's capabilities. [CO1].

5. Write a JAVA program to search for an element in a given list of elements using binary search mechanism. [CO1]

6. Write a Java program that implements Merge sort algorithm for sorting and also shows the number of interchanges occurred for the given set of integers. [CO1]

7. Write a java program to make rolling a pair of dice 10,000 times and counts the number of times doubles of are rolled for each different pair of doubles. Hint: Math.random() [CO1].

8. Develop a java application with Employee class with Emp\_name, Emp\_id, Address, Mail\_id, Mobile\_no as members. Inherit the classes, Programmer, Assistant Professor, Associate Professor and Professor from employee class. Add Basic Pay (BP) as the member of all the inherited classes with 97% of BP as DA, 10 % of BP as HRA, 12% of BP as PF, 0.1% ofBP for staff club fund. Generate pay slips for the employees with their gross and net salary. [CO1]

9. Write a Java Program to create an abstract class named Shape that contains two integers and an empty method named print Area(). Provide three classes named Rectangle, Triangle and Circle such that each one of the classes extends the class Shape. Each one of the classes contains only the method print Area () that prints the area of the given shape.[CO2]

10. Develop a java application to implement currencyconverter(DollartoINR, EURO toINR,YentoINR and vice versa), distance converter (meter to KM, miles to KM and vice versa), timeconverter (hours to minutes, seconds and vice versa) using packages. [CO1]

11. Write a Java Program to Handle Arithmetic Exceptions and InputMisMatchExceptions. [CO1]

12. Write a multi-threaded Java program to print all numbers below 100,000 that are both prime and Fibonacci number (some examples are 2, 3, 5, 13, etc.). Design a thread that generates prime numbers below 100,000 and writes them into a pipe. Design another thread that generates Fibonacci numbers and writes them to another pipe. The main thread should read both the pipes to identify numbers common to both. [CO3].

13. Write a java program that implements a multi-threaded application that has three threads. First thread generates a random integer every 1 second and if the value is even, second thread computes the square of the number and prints. If the value is odd, the third thread will print the value of cube of the number. [CO3].

14. Write a Java program that correctly implements the producer – consumer problem using the concept of inter-thread communication. [CO3].

15. Write a Java program that reads a file name from the user, displays information about whether the file exists, whether the file is readable, or writable, the type of file and the length of the file inbytes. [CO1].

16. Write a Java program to build a Calculator in Swings. [CO4]

17. Write a Java program to implement JMenu to draw all basic shapes using Graphics. [CO4]

18. Write a Java program to implement JTable and JTree. [CO4]

19. Write a Java program to implement JTabbedPane. [CO4]

20. Write a Java Program that implements a simple client/server application. The client sends data to a server. The server receives the data, uses it to produce a result and then sends the result back to the client. The client displays the result on the console. For ex: The data sent from the client is the radius of a circle and the result produced by the server is the area of the circle. [CO5]

**List of Additional Experiment:**

**1 . Write a java program to implement the functions in String class.**

**2 . Write a java program to implement the functions in StringBuffer class.**

**3 . Write a java program to implement stacks.**

**4 . Write a java program to implement queues.**

**5 . Write a java program to demonstrate the usage of ByteStream classes.**

**6 . Write a java program to demonstrate the usage of CharacterStream classes.**

**7 . Write a java program to demonstrate Serialization and Deserialization.**

CERTIFICATE

Name of the Lab : JAVA PROGRAMMING

Name of the Student : K . BHARATH ESWAR

Student Regd. No. : 20BQ1A0593

CLASS : II B.TECH. I SEM CSE – B

GIT HUB LINK:

INDEX

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| S. No. | Experiment Number | DATE OF COMPLETION | DATE OF SUBMISSION | PAGE NO | Remarks | Signature of Faculty |
| 1 | Experiment -1 | 12-10-2021 | 18-10-2021 |  |  |  |
| 2 | Experiment -2 | 12-10-2021 | 18-10-2021 |  |  |  |
| 3 | Experiment -3 | 26-10-2021 | 28-10-2021 |  |  |  |
| 4 | Experiment -4 | 26-10-2021 | 28-10-2021 |  |  |  |
| 5 | Experiment -5 | 02-11-2021 | 05-11-2021 |  |  |  |
| 6 | Experiment -6 | 02-11-2021 | 05-11-2021 |  |  |  |
| 7 | Experiment -7 | 09-11-2021 | 11-11-2021 |  |  |  |
| 8 | Experiment -8 | 09-11-2021 | 11-11-2021 |  |  |  |
| 9 | Experiment -9 | 16-11-2021 | 19-11-2021 |  |  |  |
| 10 | Experiment -10 | 16-11-2021 | 19-11-2021 |  |  |  |
| 11 | Experiment -11 | 23-11-2021 | 27-11-2021 |  |  |  |
| 12 | Experiment -12 | 23-11-2021 | 27-11-2021 |  |  |  |
| 13 | Experiment -13 | 30-11-2021 | 13-12-2021 |  |  |  |
| 14 | Experiment -14 | 30-11-2021 | 13-12-2021 |  |  |  |
| 15 | Experiment -15 | 30-11-2021 | 13-12-2021 |  |  |  |
| 16 | Experiment -16 | 14-12-2021 | 28-12-2021 |  |  |  |
| 17 | Experiment -17 | 14-12-2021 | 28-12-2021 |  |  |  |
| 18 | Experiment -18 | 14-12-2021 | 28-12-2021 |  |  |  |
| 19 | Experiment -19 | 04-01-2022 | 06-01-2022 |  |  |  |
| 20 | Experiment -20 | 04-01-2022 | 06-01-2022 |  |  |  |
| **Additional Experiments** | | | | | | |
| 21 | Experiment -1 | 02-11-2021 | 05-11-2021 |  |  |  |
| 22 | Experiment -2 | 09-11-2021 | 11-11-2021 |  |  |  |
| 23 | Experiment -3 | 16-11-2021 | 19-11-2021 |  |  |  |
| 24 | Experiment -4 | 23-11-2021 | 27-11-2021 |  |  |  |
| 25 | Experiment -5 | 30-11-2021 | 13-12-2021 |  |  |  |
| 26 | Experiment -6 | 14-12-2021 | 28-12-2021 |  |  |  |
| 27 | Experiment -7 | 04-01-2022 | 06-01-2022 |  |  |  |

**EXPERIMENT NO: 1**

**AIM:** Create a class called Invoice that a hardware store might use to represent an invoice for an item sold at the store. An Invoice should include four pieces of information as instance variables-a part number (type String),a part description(type String),a quantity of the item being purchased (type int) and a price per item (double). Your class should have a constructor that initializes the four instance variables. Provide a set and a get method for each instance variable. In addition, provide a method named getInvoiceAmount() that calculates the invoice amount (i.e., multiplies the quantity by the price per item), then returns the amount as a double value. If the quantity is not positive, it should be set to 0. If the price per item is not positive, it should be set to 0.0. Write a test application named InvoiceTest that demonstrates class Invoice’s capabilities. [CO1]

**DESCRIPTION:**

GETTER AND SETTER :

Getter and Setter are methods used to protect your data and make your code more secure. Getter returns the value (accessors), it returns the value of data type int, String, double, float, etc. For the convenience of the program, getter starts with the word “get” followed by the variable name.

While Setter sets or updates the value (mutators). It sets the value for any variable which is used in the programs of a class. and starts with the word “set” followed by the variable name. Getter and Setter make the programmer convenient in setting and getting the value for a particular data type. In both getter and setter, the first letter of the variable should be capital.

**SYNTAX:**

**class GetSetEx {**

**private String name;**

**public String getName() {**

**return name;**

**}**

**public void setName(String Newname)**

**{ this.name = Newname;**

**}**

**}**

**PROGRAM:**

**import** java.util.Scanner;

**class** Invoice

{

**private** String partno;

**private** String desc;

**private** **int** quant;

**private** **double** price;

Invoice ()

{

System.***out***.println ("Hello customer");

}

**void** setpart (String partno)

{

**this**.partno = partno;

}

**void** setdesc (String desc)

{

**this**.desc = desc;

}

**void** setquant (**int** quant)

{

**if** (quant <= 0)

**this**.quant = 0;

**else**

**this**.quant = quant;

}

**void** setprice (**double** price)

{

**if** (price <= 0.0)

**this**.price = 0.0;

**else**

**this**.price = price;

}

String getpart ()

{

**return** partno;

}

String getdesc ()

{

**return** desc;

}

**int** getquant ()

{

**return** quant;

}

**double** getprice ()

{

**return** price;

}

**double** getInvoiceAmount ()

{

**return** (quant \* price);

}

}

**public** **class** InvoiceTest {

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

@SuppressWarnings("resource")

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

Invoice I = **new** Invoice ();

System.***out***.println ("Enter the part number");

String x = sc.next ();

I.setpart (x);

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

String y = sc.next ();

I.setdesc (y);

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

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

I.setquant (w);

System.***out***.println ("Enter the price of each item");

**double** r = sc.nextDouble ();

I.setprice (r);

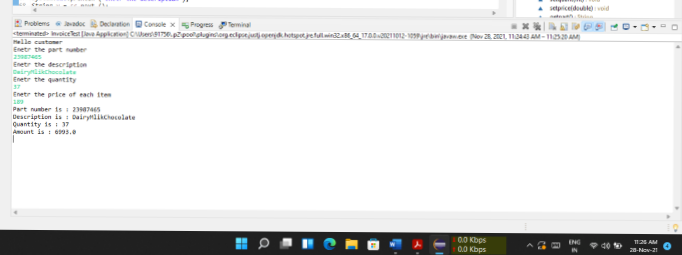
System.***out***.println ("Part number is:" + I.getpart ());

System.***out***.println ("Description is:" + I.getdesc ());

System.***out***.println ("Quantity is:" + I.getquant ());

System.***out***.println ("Amount is" + I.getInvoiceAmount ());

}

**OUTPUT:** 

**EXPERIMENT- 2**

**AIM :** Develop a Java application to generate Electricity bill. Create a class with the following members: Consumer no., consumer name, previous month reading, current month reading, and type of EB connection (i.e. domestic or commercial). Compute the bill amount using the following tariff. [CO1]

If the type of the EB connection is domestic, calculate the amount to be paid as follows:

* + 1. First 100 units - Rs. 1 per unit
    2. 101-200units - Rs. 2.50 per unit
    3. 201 -500 units - Rs. 4 per unit
    4. >501 units - Rs. 6 per unit

If the type of the EB connection is commercial, calculate the amount to be paid as follows:

* 1. First 100 units - Rs. 2 per unit
  2. 101-200units - Rs. 4.50 per unit
  3. 201 -500 units - Rs. 6 per unit
  4. >501 units - Rs. 7 per unit

**DESCRIPTION:**

The **Java String class equals()** method compares the two given strings based on the content of the string. If any character is not matched, it returns false. If all characters are matched, it returns true.

The String equals() method overrides the equals() method of the Object class.

**Example:**

**public** **class** EqualsEx{

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

String s1="javatpoint";

String s2="javatpoint";

String s3="JAVATPOINT";

System.out.println(s1.equals(s2)); //true because content and case is same

System.out.println(s1.equals(s3)); //false because case is not same

}}

**SYNTAX:**

publicboolean equals(Object anotherObject)

**Parameter: anotherObject** : another object, i.e., compared with this string.

**Return: true** if characters of both strings are equal otherwise **false**.

**PROGRAM:**

**import** java.util.Scanner;

**class** ElectBill

{

**int** ConsumerNo;

String ConsumerName;

**int** PrevReading;

**int** CurrReading;

String EBConn;

**double** Bill;

**void** consumerDetails()

{

@SuppressWarnings("resource")

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

System.***out***.println("Enter Consumer Number: ");

ConsumerNo = sc.nextInt();

System.***out***.println(" Enter Consumer Name: ");

ConsumerName = sc.next();

System.***out***.println("Enter Previous Units: ");

PrevReading = sc.nextInt();

System.***out***.println("Enter Current Units: ");

CurrReading = sc.nextInt();

System.***out***.println("Enter the types of EB Connection: ");

EBConn = sc.next();

}

**double** calculateBill()

{

**if**(EBConn.equals("domestic"))

{

**if**((CurrReading-PrevReading)<=100)

Bill=(CurrReading-PrevReading)\*1;

**else** **if**(((CurrReading-PrevReading)>=101)&&((CurrReading-PrevReading)<=200))

Bill=(CurrReading-PrevReading)\*2.50;

**else** **if**(((CurrReading-PrevReading)>=201)&&((CurrReading-PrevReading)<=300))

Bill=(CurrReading-PrevReading)\*4;

**else**

Bill=(CurrReading-PrevReading)\*6;

**return** Bill;

}

**else** **if**(EBConn.equals("commercial"))

{

**if**((CurrReading-PrevReading)<=100)

Bill=(CurrReading-PrevReading)\*2;

**else** **if**(((CurrReading-PrevReading)>=101)&&((CurrReading-PrevReading)<=200))

Bill=(CurrReading-PrevReading)\*4.50;

**else** **if**(((CurrReading-PrevReading)>=201)&&((CurrReading-PrevReading)<=300))

Bill=(CurrReading-PrevReading)\*6;

**else**

Bill=(CurrReading-PrevReading)\*7;

**return** Bill;

}

**return** Bill;

}

**void** display()

{

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

System.***out***.println("ELCTRICITY BILL");

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

System.***out***.println("Consumer Number: "+ConsumerNo);

System.***out***.println("Consumer Name: "+ConsumerName);

System.***out***.println("Consumer Previous Units: "+PrevReading);

System.***out***.println("Consumer Current Units: "+CurrReading);

System.***out***.println("Type of EBConnection: "+EBConn);

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

System.***out***.println("Total Amount(Rs.): "+Bill);

}

}

**class** ElecBillGen

{

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

{

ElectBill b=**new** ElectBill();

b.consumerDetails();

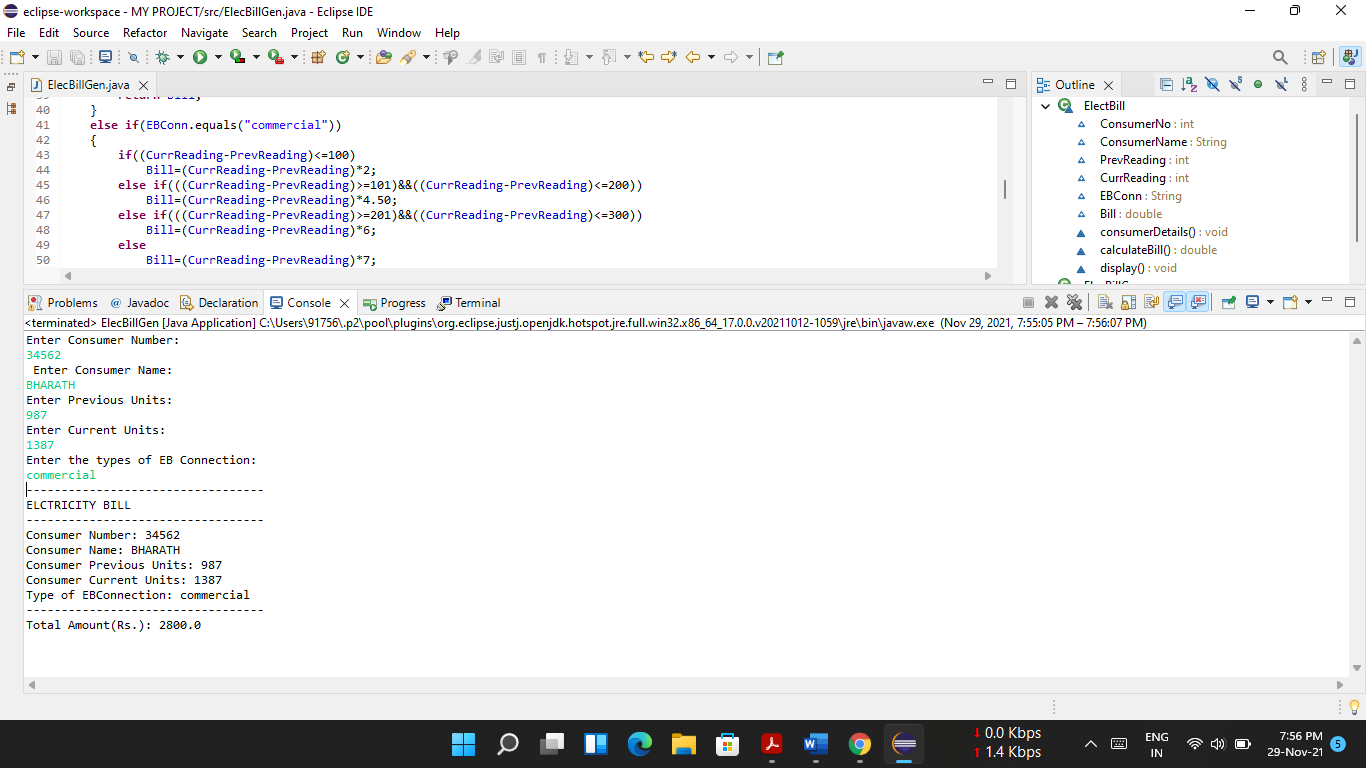
b.calculateBill();

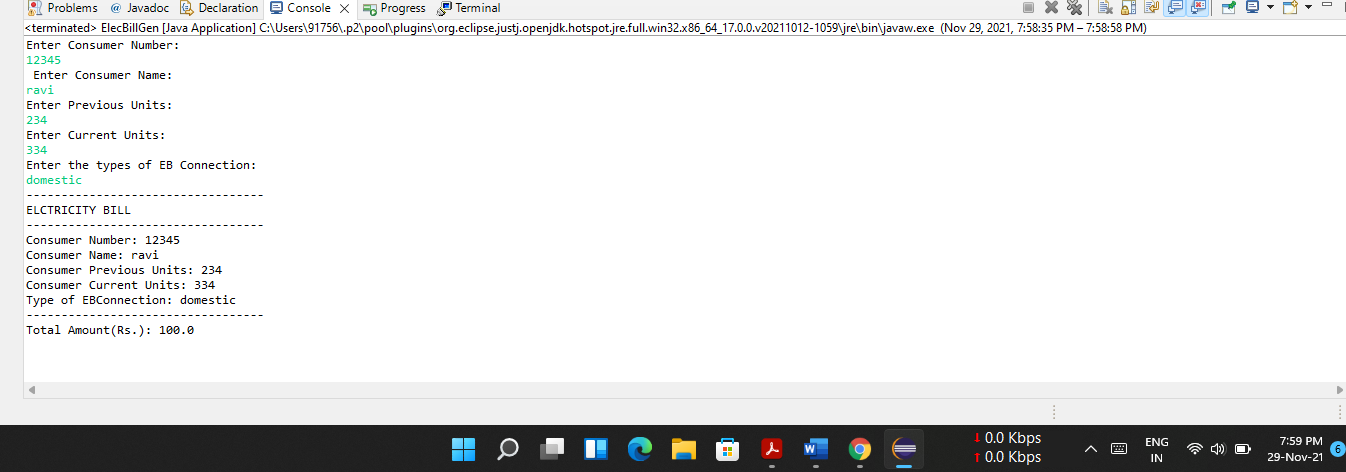
b.display();

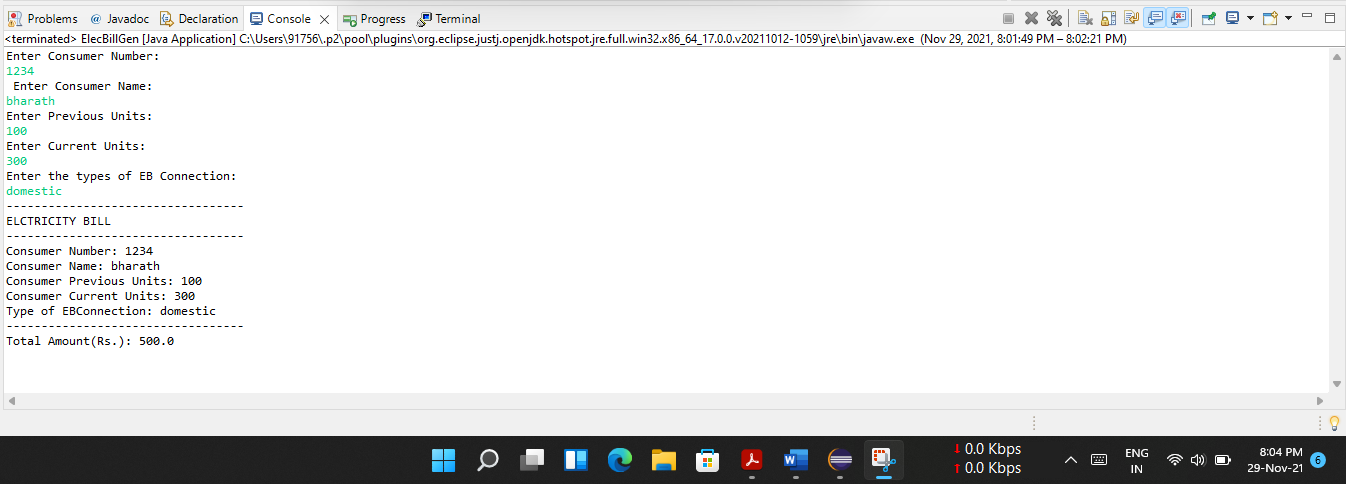
}

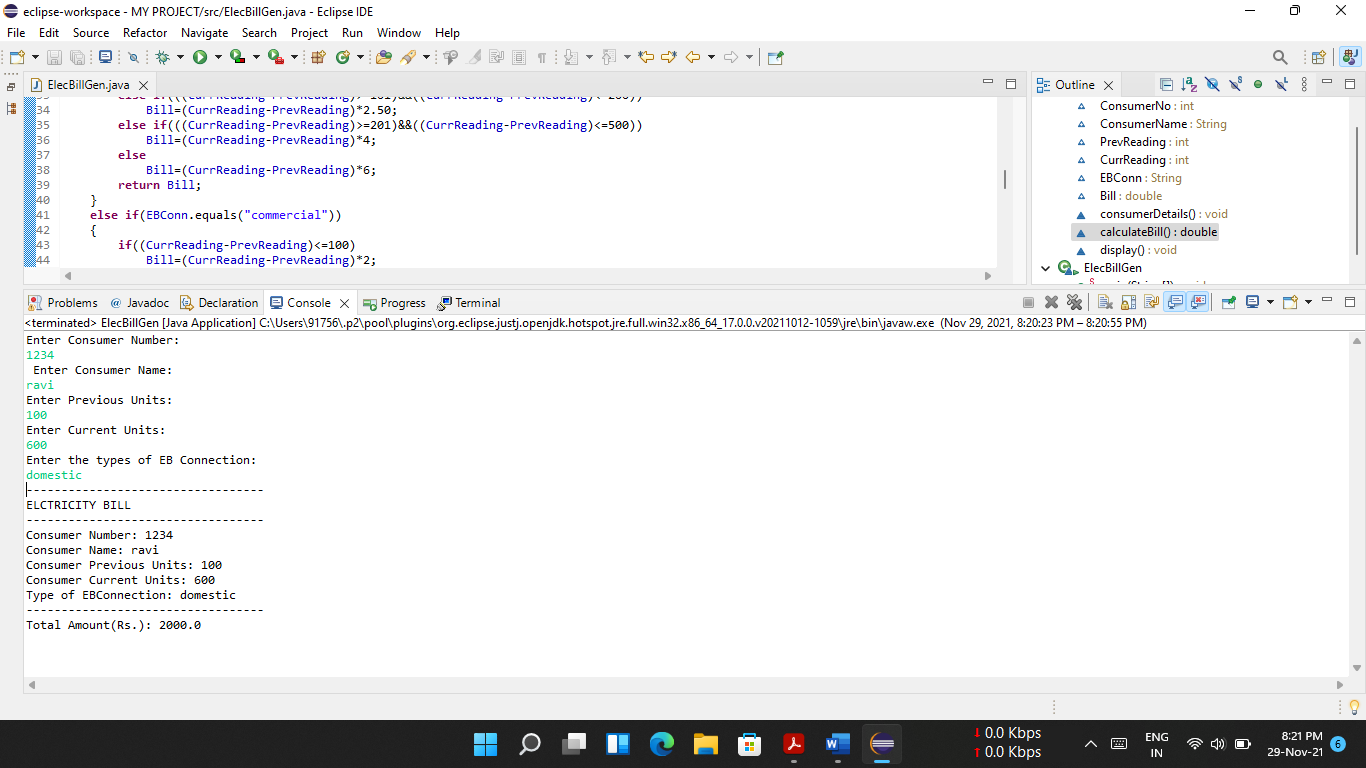
}

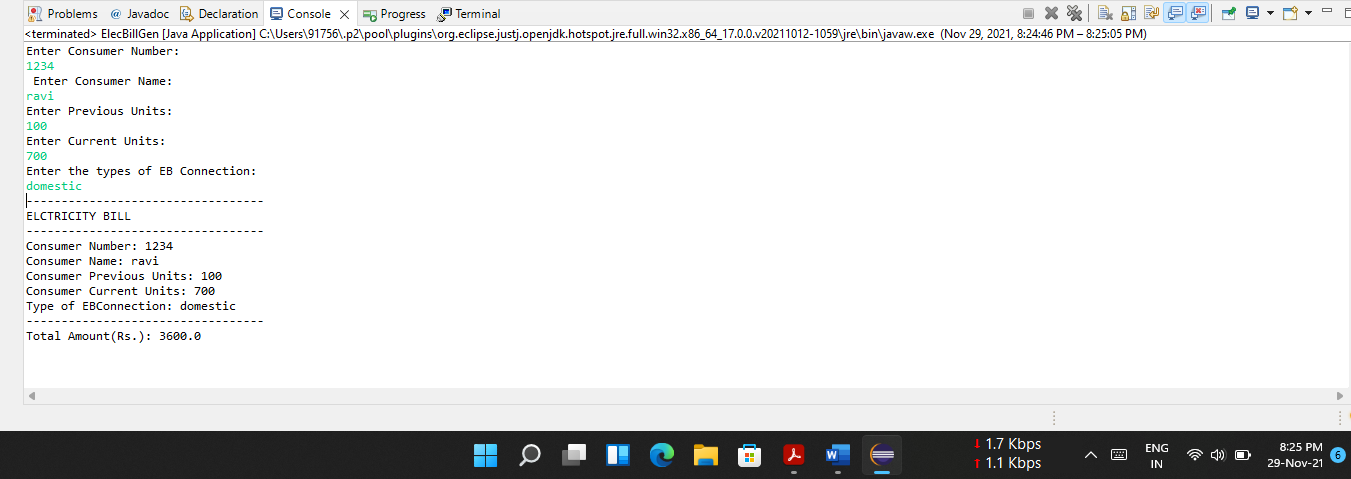
**OUTPUT:**

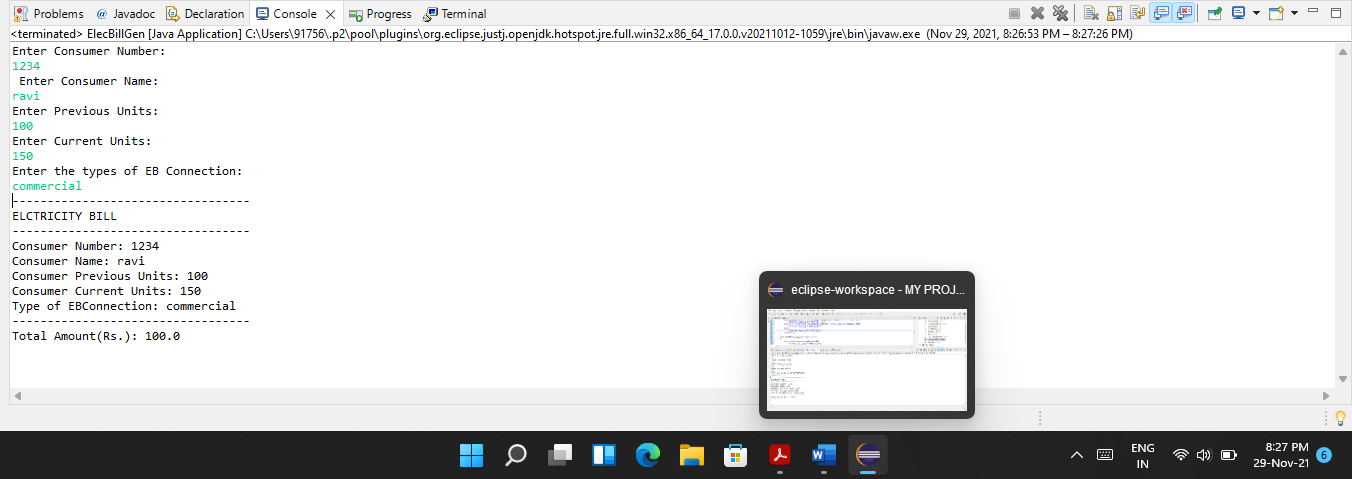


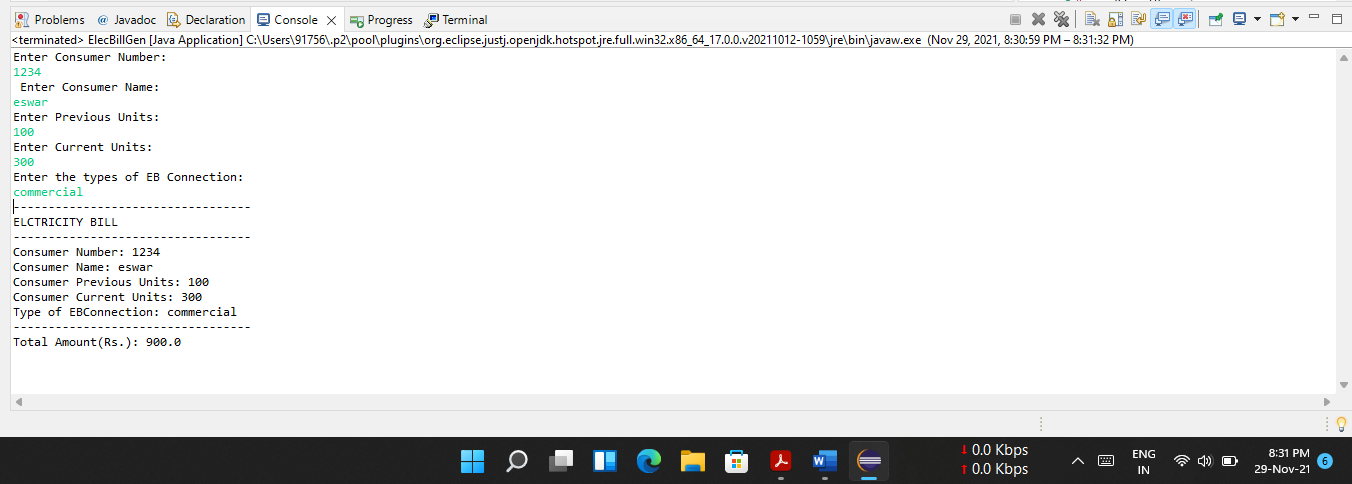


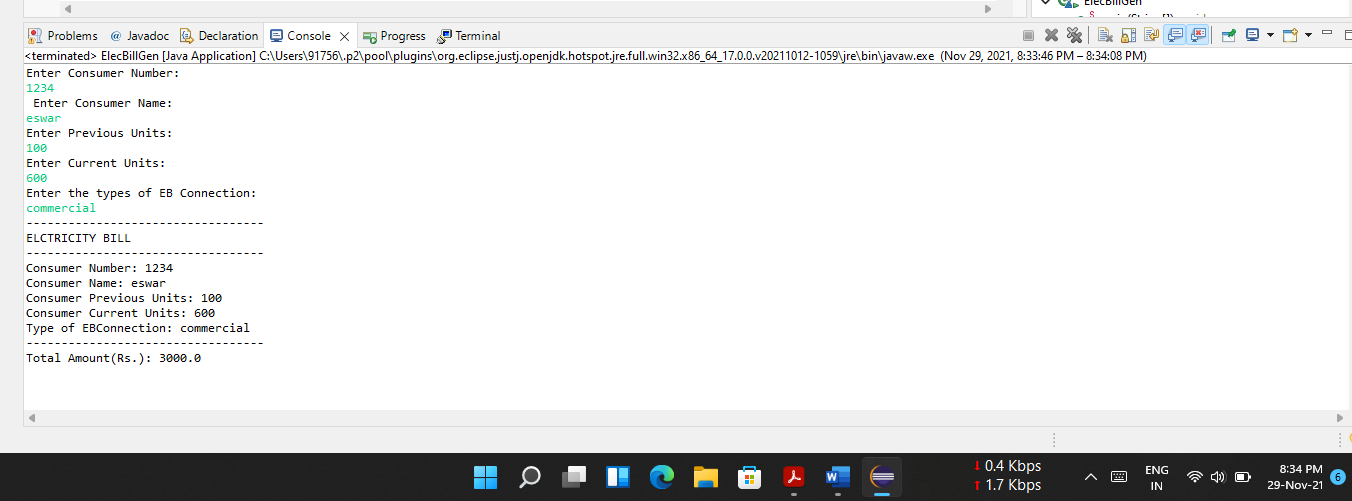












**EXPERIMENT – 3**

**AIM :** Create class Savings Account. Use a static variable annualInterestRate to store the annual interest rate for all account holders. Each object of the class contains a private instance variable savingsBalance indicating the amount the saver currently has on deposit. Provide method calculateMonthlyInterest to calculate the monthly interest by multiplying the savingsBalance by annualInterestRate divided by 12 this interest should be added to savings Balance. Provide a static method modifyInterestRate that sets the annualInterestRate to a new value. Write a program to test class SavingsAccount. Instantiate two savingsAccount objects, saver1 and saver2, with balances of $2000.00 and $3000.00, respectively. Set annualConcentration Rate to 4%, then calculate the monthly interest and print the new balances for both savers. Then set the annualInterestRate to 5%, calculate the next month’s interest and print the new balances for both savers. [CO1]

**DESCRIPTION:**

**JAVA STATIC KEYWORD:**

The **static keyword** in java is used for memory management mainly. We can apply static keyword with variable methods, blocks and nested classes. The static keyword belongs to the class than an instance of the class.

**JAVA STATIC VARIABLE:**

1. If you declare any variable as static, it is known as a static variable.

2. The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.

3. The static variable gets memory only once in the class area at the time of class loading.

**JAVA STATIC METHOD :**

If you apply static keyword with any method, it is known as static method.

1. A static method belongs to the class rather than the object of a class.

2. A static method can be invoked without the need for creating an instance of a class.

3. A static method can access static data member and can change the value of it.

**JAVA STATIC BLOCK:**

1. Is used to initialize the static data member.

2. It is executed before the main method at the time of classloading.

**SYNTAX:**

**Static block:**

**class** A2{

**static**{System.out.println("static block is invoked");}

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

   //Statements

}

}

Static method:

**class** Calculate{

**static** **int** cube(**int** x){

**//Statements;**

  }

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

**int** result=Calculate.cube(5);

  //Statements;

  }

}

Static Variable:

**class** Counter{

**static** **int** count=0;//will get memory only once and retain its value

Counter(){

count++;//incrementing the value of static variable

System.out.println(count);

}

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

//creating objects

Counter c1=**new** Counter();

}

}

**PROGRAM:**

**public** **class** SavingsAccount {

**static** **private** **double** *annualInterestRate*;

**private** **double** savingBalance;

**public** SavingsAccount(**double** savingBalance)

{

**this**.savingBalance=savingBalance;

}

**public** **double** getSavingBalance()

{

**return** savingBalance;

}

// Modify interest rate by setting annual interest rate to a new value

**public** **static** **void** modifyInterestRate(**double** newInterestRate)

{

*annualInterestRate*=newInterestRate;

}

**public** **void** calculateMonthlyInterest()

{

**double** monthlyI;

monthlyI= (**double**)(savingBalance\**annualInterestRate*/12);

savingBalance+=monthlyI;

}

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

// To test the class designed above

//Instantiate 2 saving account objects saver1 and saver2

SavingsAccount saver1, saver2;

saver1 = **new** SavingsAccount (2000.0);

saver2= **new** SavingsAccount (3000.0);

@SuppressWarnings("unused")

**int** total = 0;

//Set the annual interest rate to 4%=0.04

SavingsAccount.*modifyInterestRate* (0.04);

//Calculate monthly interest

saver1.calculateMonthlyInterest();

saver2.calculateMonthlyInterest();

//Print out the new balances for both savers

System.***out***.println("This month:\nSaver 1 balance= "+ saver1.getSavingBalance());

System.***out***.println("Saver 2 balance= "+ saver2.getSavingBalance());

//Change annual interest rate to 5%=0.05

SavingsAccount.*modifyInterestRate*(0.05);

//Calculate the next month interest rate and print out balances

saver1.calculateMonthlyInterest();

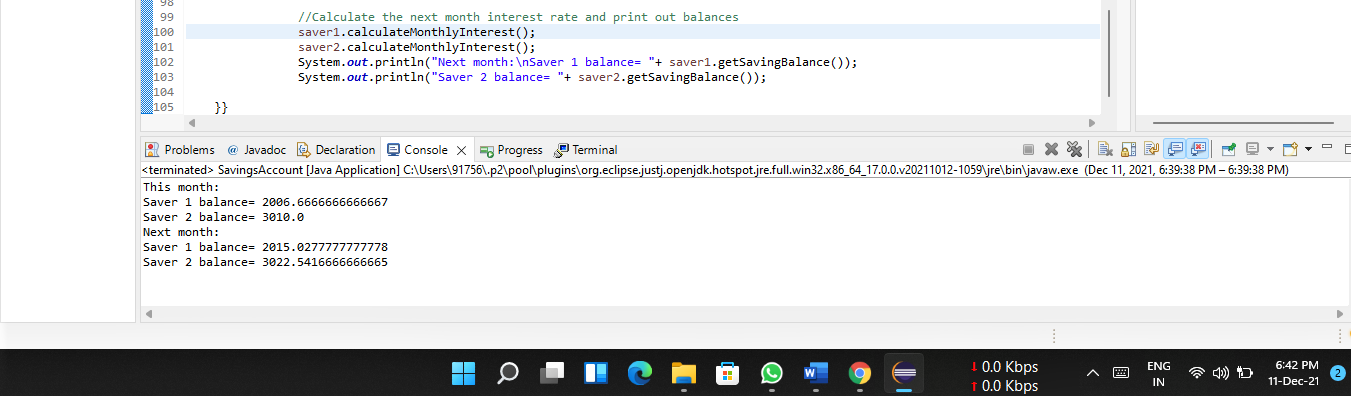
saver2.calculateMonthlyInterest();

System.***out***.println("Next month:\nSaver 1 balance= "+ saver1.getSavingBalance());

System.***out***.println("Saver 2 balance= "+ saver2.getSavingBalance());

}}

**OUTPUT:**

****

**EXPERIMENT – 4**

**AIM:** Create a class called Book to represent a book. A Book should include four pieces of information as instance variables; a book name, an ISBN number, an author name and a publisher. Your class should have a constructor that initializes the four instance variables. Provide a mutator method and accessor method (query method) for each instance variable. In addition, provide a method named getBookInfo that returns the description of the book as a String (the description should include all the information about the book). You should use this keyword in member methods and constructor. Write a test application named BookTest to create an array of object for 30 elements for class Book to demonstrate the class Book's capabilities. [CO1].

**DESCRIPTION:**

**Array of objects:**

The array of Objects the name itself suggests that it stores an array of objects. Unlike the traditional array stores values like String, integer, Boolean, etc an Array of Objects stores objects that mean objects are stored as elements of an array. Note that when we say Array of Objects it is not the object itself that is stored in the array but the reference of the object.

**Creating an array of objects:**

An Array of Objects is created using the object class, and we know Object class is the root class of all Classes.

We use the Class\_Name followed by a square bracket [] then object reference name to create an Array of Objects.

**SYNTAX:**

**Single dimensional Array:**

Array\_Name = new Data\_Type[size];

**PROGRAM:**

**class** Book

{

**private** String name;

**private** String ISBN;

**private** String author;

**private** String publisher;

Book (String authorname, String isbn, String au, String pub)

{

name = authorname;

ISBN = isbn;

author = au;

publisher = pub;

}

**void** setname (String authorname) {

name = authorname;

}

**void** setisbn (String isbn) {

ISBN = isbn;

}

**void** setauthor (String au) {

author = au;

}

**void** setpublisher (String pub) {

publisher = pub;

}

String getname () {

**return** name;

}

String getisbn () {

**return** ISBN;

}

String getauthor () {

**return** author;

}

String getpublisher () {

**return** publisher;

}

**void** getBookInfo () {

System.***out***.println ("Book name is:" + name);

System.***out***.println ("ISBN number is:" + ISBN);

System.***out***.println ("Author name is:" + author);

System.***out***.println ("Publisher is:" + publisher);

}

}

**public** **class** BookTest4 {

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

Book b [] = **new** Book [30];

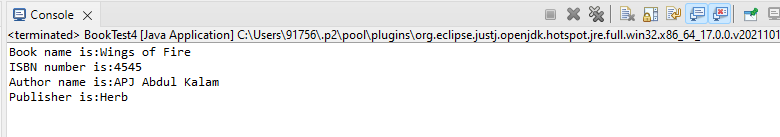
b [1] = **new** Book ("Wings of Fire", "4545", "APJ Abdul Kalam", "Herb");

b [1] .getBookInfo ();

}

}

**OUTPUT:**

****

**EXPERIMENT – 5**

**AIM:** Write a JAVA program to search for an element in a given list of elements using binary search mechanism. [CO1]

**DESCRIPTION:**

# Binary Search Algorithm

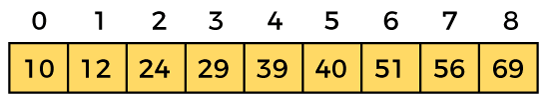
In this article, we will discuss the Binary Search Algorithm. Searching is the process of finding some particular element in the list. If the element is present in the list, then the process is called successful, and the process returns the location of that element. Otherwise, the search is called unsuccessful .Binary search is the search technique that works efficiently on sorted lists. Hence, to search an element into some list using the binary search technique, we must ensure that the list is sorted. Binary search follows the divide and conquer approach in which the list is divided into two halves, and the item is compared with the middle element of the list. If the match is found then, the location of the middle element is returned. Otherwise, we search into either of the halves depending upon the result produced through the match.

## **Working of Binary search**

Now, let's see the working of the Binary Search Algorithm. To understand the working of the Binary search algorithm, let's take a sorted array. It will be easy to understand the working of Binary search with an example. There are two methods to implement the binary search algorithm -

* Iterative method
* Recursive method

The recursive method of binary search follows the divide and conquer approach.Let the elements of array are -



Let the element to search is, **K = 56**

We have to use the below formula to calculate the **mid** of the array -

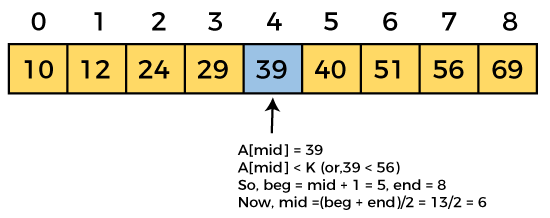
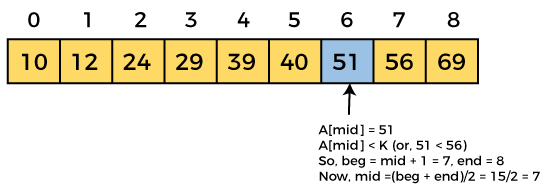
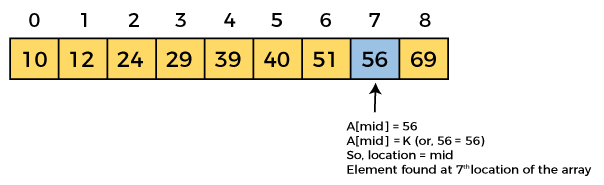
1. mid = (beg + end)/2

So, in the given array -

**beg** = 0

**end** = 8

**mid** = (0 + 8)/2 = 4. So, 4 is the mid of the array.

**SYNTAX:**

**PROGRAM:**

**import** java.util.Scanner;

**class** Binary

{

**int** bsearch (**int** a [], **int** n, **int** key)

{

**int** lb = 0;

**int** ub = n-1;

**while** (lb <= ub)

{

**int** mid = (lb + ub) / 2;

**if** (a [mid] == key)

**return** mid;

**else** **if** (a [mid] <key)

lb = mid + 1;

**else**

ub = mid + 1;

}

**return** -1;

}

}

**public** **class** BinarySearch5 {

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

// **TODO** Auto-generated method

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

System.***out***.println ("enter no.of elements");

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

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

System.***out***.println ("enter elements in sorted order");

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

a [i] = sc.nextInt ();

Binary b1 = **new** Binary ();

System.***out***.println ("enter the element you want to search");

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

**int** z = b1.bsearch (a, n, p);

**if** (z!= - 1)

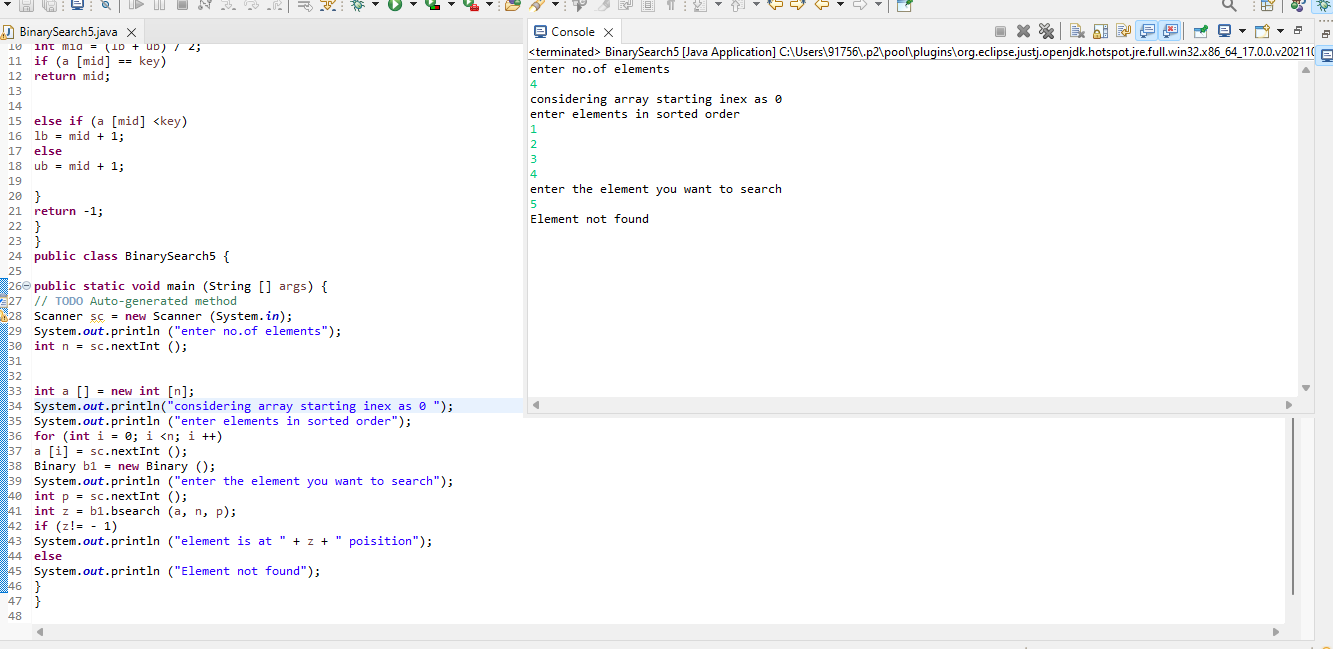
System.***out***.println ("element is at " + z + " poisition");

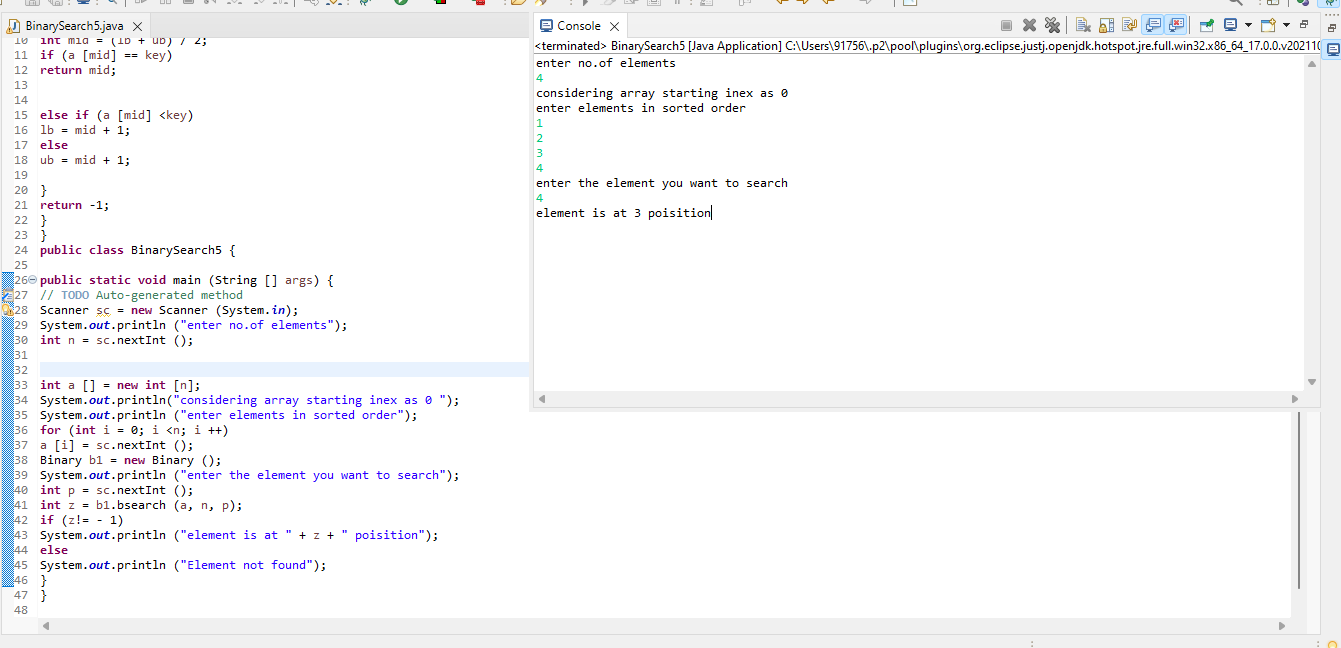
**else**

System.***out***.println ("Element not found");

}

}

**OUTPUT: **

****

**EXPERIMENT – 6**

**AIM:** Write a Java program that implements Merge sort algorithm for sorting and also shows the number of interchanges occurred for the given set of integers. [CO1]

**DESCRIPTION:**

### Merge Sort

Like QuickSort, Merge Sort is a Divide and Conquer algorithm. It divides the input array into two halves, calls itself for the two halves, and then merges the two sorted halves. The merge() function is used for merging two halves. The merge(arr, l, m, r) is a key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merges the two sorted sub-arrays into one. See the following C implementation for details.

MergeSort(arr[], l, r)

If r > l

1. Find the middle point to divide the array into two halves:

middle m = l+ (r-l)/2

2. Call mergeSort for first half:

Call mergeSort(arr, l, m)

3. Call mergeSort for second half:

Call mergeSort(arr, m+1, r)

4. Merge the two halves sorted in step 2 and 3:

Call merge(arr, l, m, r)

The following diagram shows the complete merge sort process for an example array {38, 27, 43, 3, 9, 82, 10}. If we take a closer look at the diagram, we can see that the array is recursively divided into two halves till the size becomes 1. Once the size becomes 1, the merge processes come into action and start merging arrays back till the complete array is merged.



**SYNTAX:**

**PROGRAM:**

**import** java.util.Scanner;

**class** Sort

{

**void** mersort (**int** a [], **int** lb, **int** ub)

{

**if** (lb!= ub)

{

**int** mid = (lb + ub) / 2;

mersort (a, lb, mid);

mersort (a, mid + 1, ub);

mpass (a, lb, mid, ub);

}

}

**void** mpass (**int** a [], **int** lb, **int** mid, **int** ub)

{

**int** temp [] = **new** **int** [100];

**int** i = lb;

**int** j = mid + 1;

**int** k = lb;

**while** ((i <= mid) && (j <= ub))

{

**if** (a [i]> a [j])

temp [k ++] = a [j ++];

**else**

temp [k ++] = a [i ++];

}

**while** (i <= mid)

temp [k ++] = a [i ++];

**while** (j <= ub)

temp [k ++] = a [j ++];

**for** (i = lb; i <= ub; i ++)

a [i] = temp [i];

}

}

**public** **class** MergeSort6 {

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

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

**int** i = 0;

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

System.***out***.println("Enter the no. of elements you want to sort");

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

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

System.***out***.println ("Enetr the elemnts");

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

p [i] = sc.nextInt ();

System.***out***.println ("Before sorting");

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

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

Sort s = **new** Sort ();

s.mersort (p, 0, n-1);

System.***out***.println ("Afetr sorting sorting");

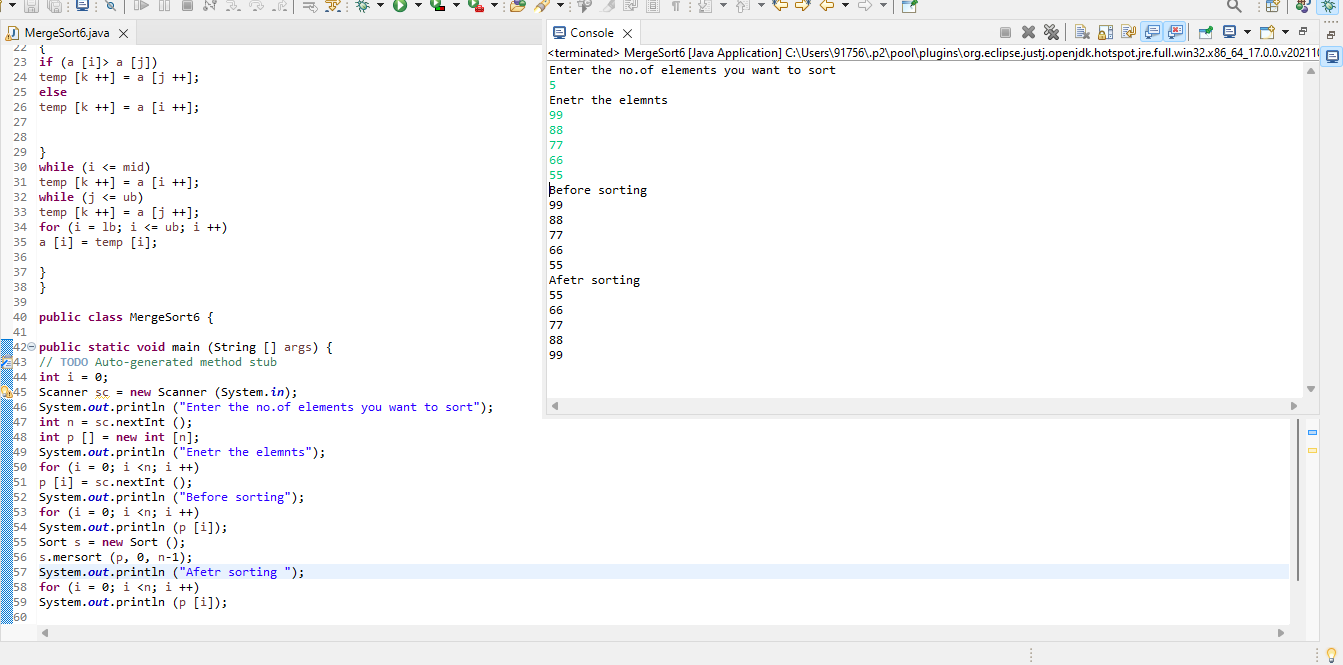
**for** (i = 0; i <n; i ++)

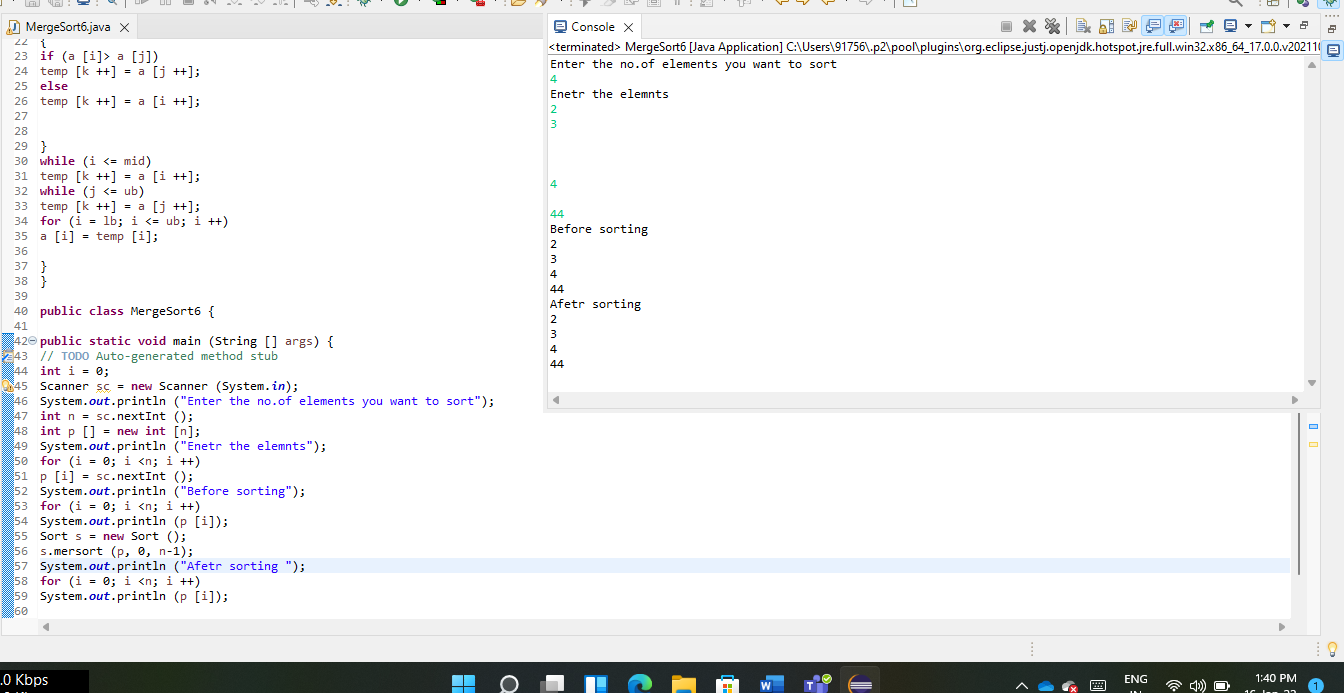
System.***out***.println (p [i]);

}

}

**OUTPUT:**

****

****

**EXPERIMENT – 7**

**AIM:** Write a java program to make rolling a pair of dice 10,000 times and counts the number of times doubles of are rolled for each different pair of doubles. Hint: Math.random() [CO1].

**DESCRIPTION:**

**math.random()**

The java.lang.Math.random() method returns a pseudorandom double type number greater than or equal to 0.0 and less than 1.0. . When this method is first called, it creates a single new pseudorandom-number generator,

exactly as if by the expression new java.util.Random.

**Example code:**

import java.lang.Math;

class Gfg1 {

// driver code

public static void main(String args[])

{

// Generate random number

double rand = Math.random();

// Output is different everytime this code is executed

System.out.println("Random Number:" + rand);

}

}

Output: 0.5568515217910215

**SYNTAX:**

public static double random()

Return:

This method returns a pseudorandom double greater than or equal to 0.0 and less than 1.0.

**PROGRAM:**

import java.util.Random;

public class Roll{

public static void main( String[ ] args){

Random randomNumbers = new Random( );

int frequency1 = 0;

int frequency2 = 0;

int frequency3 = 0;

int frequency4 = 0;

int frequency5 = 0;

int frequency6 = 0;

int dice1,dice2;

for ( int roll = 1; roll <= 10000; roll++ )

{

dice1 = 1 + randomNumbers.nextInt( 6 );

dice2 = 1 + randomNumbers.nextInt( 6 );

if(dice1==dice2)

{

switch(dice1)

{

case 1: ++frequency1;

break;

case 2: ++frequency2;

break;

case 3: ++frequency3;

break;

case 4: ++frequency4;

break;

case 5: ++frequency5;

break;

case 6: ++frequency6;

break;

}

}

}

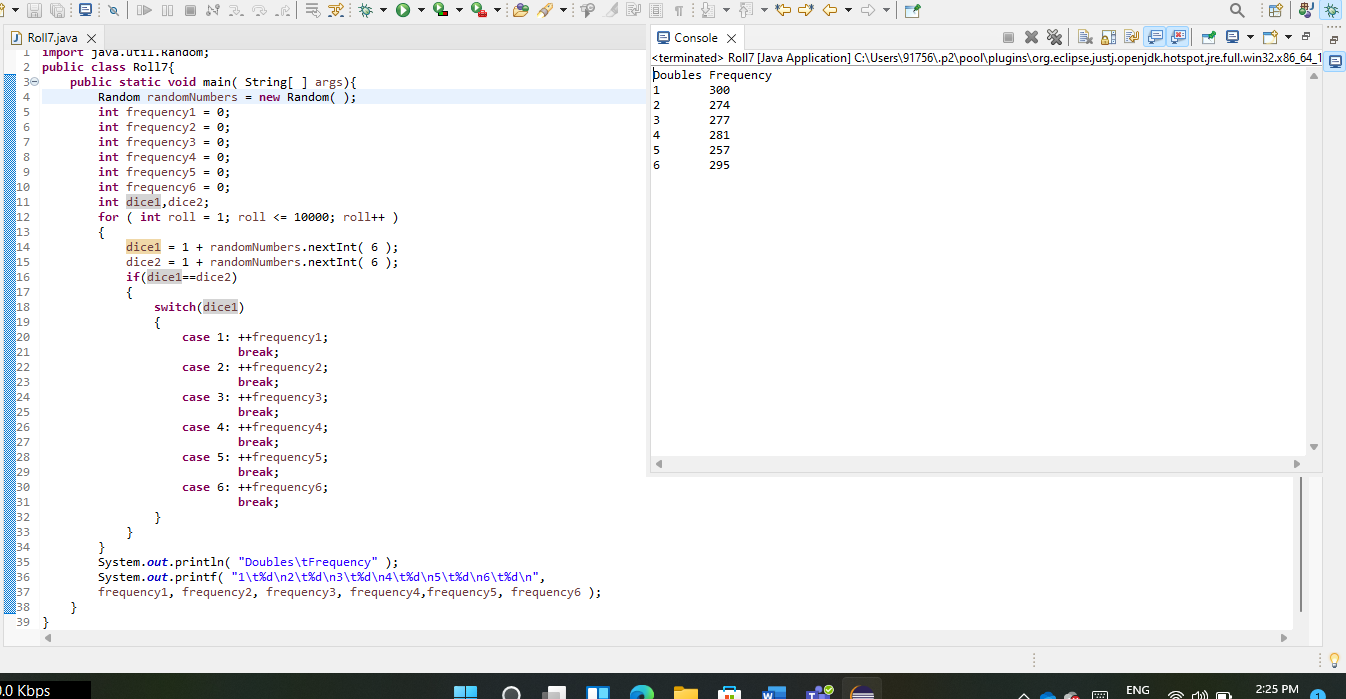
System.out.println( "Doubles\tFrequency" );

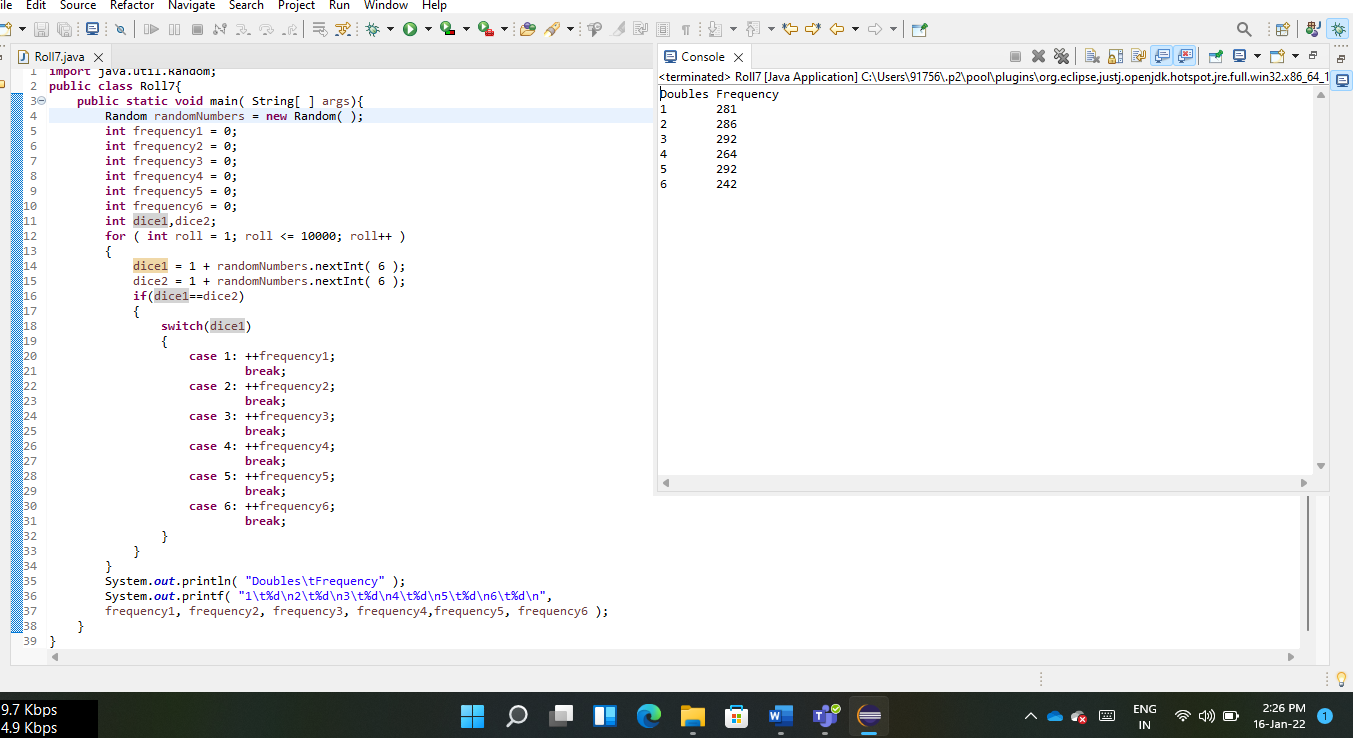
System.out.printf( "1\t%d\n2\t%d\n3\t%d\n4\t%d\n5\t%d\n6\t%d\n",

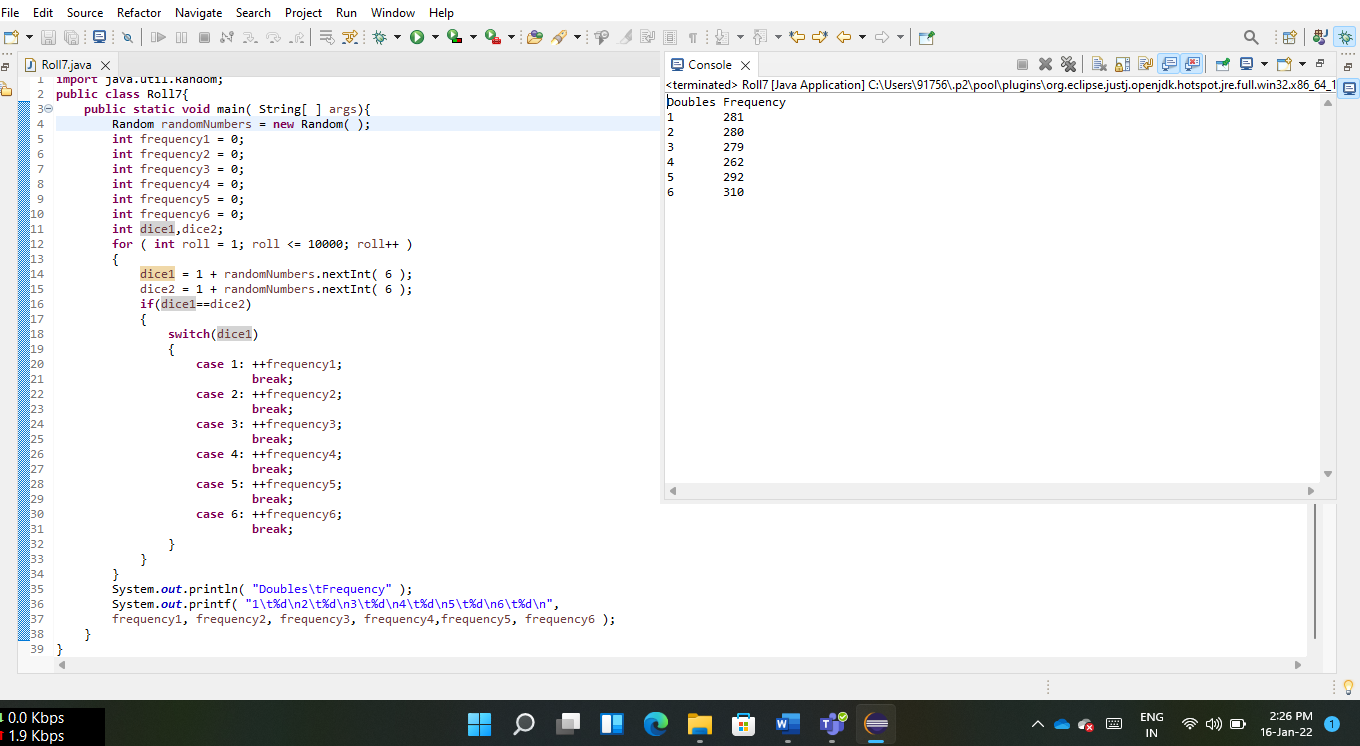
frequency1, frequency2, frequency3, frequency4,frequency5, frequency6 );

}

**OUTPUT:**

****

****

****

**EXPERIMENT – 8**

**AIM:** Develop a java application with Employee class with Emp\_name, Emp\_id, Address, Mail\_id, Mobile\_no as members. Inherit the classes, Programmer, Assistant Professor, Associate Professor and Professor from employee class. Add Basic Pay (BP) as the member of all the inherited classes with 97% of BP as DA, 10 % of BP as HRA, 12% of BP as PF, 0.1% ofBP for staff club fund. Generate pay slips for the employees with their gross and net salary. [CO1]

**DESCRIPTION:**

Inheritance in Java

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

## Why use inheritance in java?

For Method Overriding (so runtime polymorphism can be achieved).

For Code Reusability.

"In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass."

**SYNTAX:**

The syntax of Java Inheritance

class Subclass-name extends Superclass-name

{

//methods and fields

}

The ‘extends’ keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

**PROGRAM:**

**import** java.util.Scanner;

**class** Employee

{

String Employee\_name;

**int** Emp\_id;

String Address;

String Mail\_id;

**long** Mobile\_no;

Employee(String e\_name,**int** eid ,String address,String mailid,**long** mobileno)

{

**this**.Employee\_name=e\_name;

**this**.Emp\_id=eid;

**this**.Address=address;

**this**.Mail\_id=mailid;

**this**.Mobile\_no=mobileno;

}

}

**class** Programmer **extends** Employee

{

Programmer(String e\_name, **int** eid, String address, String mailid, **long** mobileno)

{

**super**(e\_name, eid, address, mailid, mobileno);

}

**double** basicpay=1\_00\_000.00;

**double** da()

{

**return** (0.97)\*basicpay;

}

**double** hra()

{

**return** (0.10)\*basicpay;

}

**double** pf()

{

**return** (0.12)\*basicpay;

}

**double** scf()

{

**return** (0.001)\*basicpay;

}

**double** gross()

{

**return** basicpay+basicpay\*(0.97+0.10+0.12);

}

**double** netSalary()

{

**return** (basicpay+basicpay\*(0.97+0.10+0.12))-(0.12+0.001)\*basicpay;

}

}

**class** AssistantProffesor **extends** Employee

{

AssistantProffesor(String e\_name, **int** eid, String address, String mailid, **long** mobileno)

{

**super**(e\_name, eid, address, mailid, mobileno);

}

**double** basicpay=95\_000.00;

**double** da()

{

**return** (0.97)\*basicpay;

}

**double** hra()

{

**return** (0.10)\*basicpay;

}

**double** pf()

{

**return** (0.12)\*basicpay;

}

**double** scf()

{

**return** (0.001)\*basicpay;

}

**double** gross()

{

**return** basicpay+basicpay\*(0.97+0.10+0.12);

}

**double** netSalary()

{

**return** (basicpay+basicpay\*(0.97+0.10+0.12))-(0.12+0.001)\*basicpay;

}

}

**class** AssociateProfessor **extends** Employee

{

AssociateProfessor(String e\_name, **int** eid, String address, String mailid, **long** mobileno)

{

**super**(e\_name, eid, address, mailid, mobileno);

}

**double** basicpay=1\_25\_000.00;

**double** da()

{

**return** (0.97)\*basicpay;

}

**Double** hra()

{

**return** (0.10)\*basicpay;

}

**double** pf()

{

**return** (0.12)\*basicpay;

}

**double** scf()

{

**return** (0.001)\*basicpay;

}

**double** gross()

{

**return** basicpay+basicpay\*(0.97+0.10+0.12);

}

**double** netSalary()

{

**return** (basicpay+basicpay\*(0.97+0.10+0.12))-(0.12+0.001)\*basicpay;

}

}

**class** Professor **extends** Employee

{

Professor(String e\_name, **int** eid, String address, String mailid, **long** mobileno)

{

**super**(e\_name, eid, address, mailid, mobileno);

}

**double** basicpay=88\_000.00;

**double** da()

{

**return** (0.97)\*basicpay;

}

**double** hra()

{

**return** (0.10)\*basicpay;

}

**double** pf()

{

**return** (0.12)\*basicpay;

}

**double** scf()

{

**return** (0.001)\*basicpay;

}

**double** gross()

{

**return** basicpay+basicpay\*(0.97+0.10+0.12);

}

**double** netSalary()

{

**return** (basicpay+basicpay\*(0.97+0.10+0.12))-(0.12+0.001)\*basicpay;

}

}

**public** **class** Payslip8 {

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

Programmer p1=**new** Programmer("Bharath ",593,"gnt","bharath@gmail.com",7569869646L);

System.***out***.println("########## PROGRAMMER ###########");

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

System.***out***.println("employee name: "+p1.Employee\_name);

System.***out***.println("employee id: "+p1.Emp\_id);

System.***out***.println("employee ADDRESS: "+p1.Address);

System.***out***.println("employee mail id: "+p1.Mail\_id);

System.***out***.println("employee mobile no: "+p1.Mobile\_no);

System.***out***.println("enter basicpay: "+p1.basicpay);

System.***out***.println("DA : "+p1.da());

System.***out***.println("HRA : "+p1.hra());

System.***out***.println("PF : "+p1.pf());

System.***out***.println("SCF : "+p1.scf());

System.***out***.println("GROSS : "+p1.gross());

System.***out***.println("NET SALARY : "+p1.netSalary());

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

AssistantProffesor p2=**new** AssistantProffesor("eswar ",594,"CHPET","ESWAR@gmail.com",9876543219L);

System.***out***.println("########## AssistantProffesor ###########");

System.***out***.println("employee name: "+p2.Employee\_name);

System.***out***.println("employee id: "+p2.Emp\_id);

System.***out***.println("employee ADDRESS: "+p2.Address);

System.***out***.println("employee mail id: "+p2.Mail\_id);

System.***out***.println("employee mobile no: "+p2.Mobile\_no);

System.***out***.println("enter basicpay: "+p2.basicpay);

System.***out***.println("DA : "+p2.da());

System.***out***.println("HRA : "+p2.hra());

System.***out***.println("PF : "+p2.pf());

System.***out***.println("SCF : "+p2.scf());

System.***out***.println("GROSS : "+p2.gross());

System.***out***.println("NET SALARY : "+p2.netSalary());

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

AssociateProfessor p3=**new** AssociateProfessor("kalyan ",595,"CHPET","kalyan@gmail.com",8989898989L);

System.***out***.println("########## AssociateProfessor ###########");

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

System.***out***.println("employee name: "+p3.Employee\_name);

System.***out***.println("employee id: "+p3.Emp\_id);

System.***out***.println("employee ADDRESS: "+p3.Address);

System.***out***.println("employee mail id: "+p3.Mail\_id);

System.***out***.println("employee mobile no: "+p3.Mobile\_no);

System.***out***.println("enter basicpay: "+p3.basicpay);

System.***out***.println("DA : "+p3.da());

System.***out***.println("HRA : "+p3.hra());

System.***out***.println("PF : "+p3.pf());

System.***out***.println("SCF : "+p3.scf());

System.***out***.println("GROSS : "+p3.gross());

System.***out***.println("NET SALARY : "+p3.netSalary());

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

Professor p4=**new** Professor(" Hari ",595,"HYD","hari@gmail.com",2347728999L);

System.***out***.println("########## Professor ###########");

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

System.***out***.println("employee name: "+p4.Employee\_name);

System.***out***.println("employee id: "+p4.Emp\_id);

System.***out***.println("employee ADDRESS: "+p4.Address);

System.***out***.println("employee mail id: "+p4.Mail\_id);

System.***out***.println("employee mobile no: "+p4.Mobile\_no);

System.***out***.println("enter basicpay: "+p4.basicpay);

System.***out***.println("DA : "+p4.da());

System.***out***.println("HRA : "+p4.hra());

System.***out***.println("PF : "+p4.pf());

System.***out***.println("SCF : "+p4.scf());

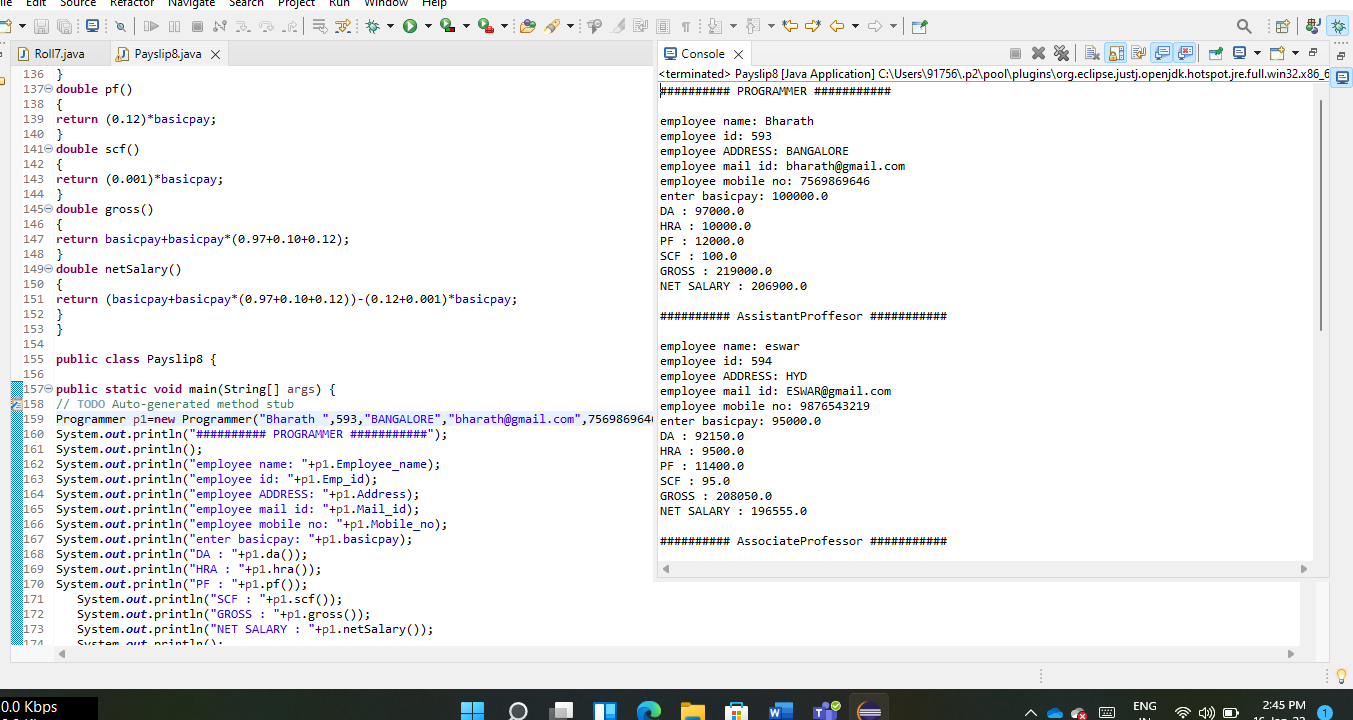
System.***out***.println("GROSS : "+p4.gross());

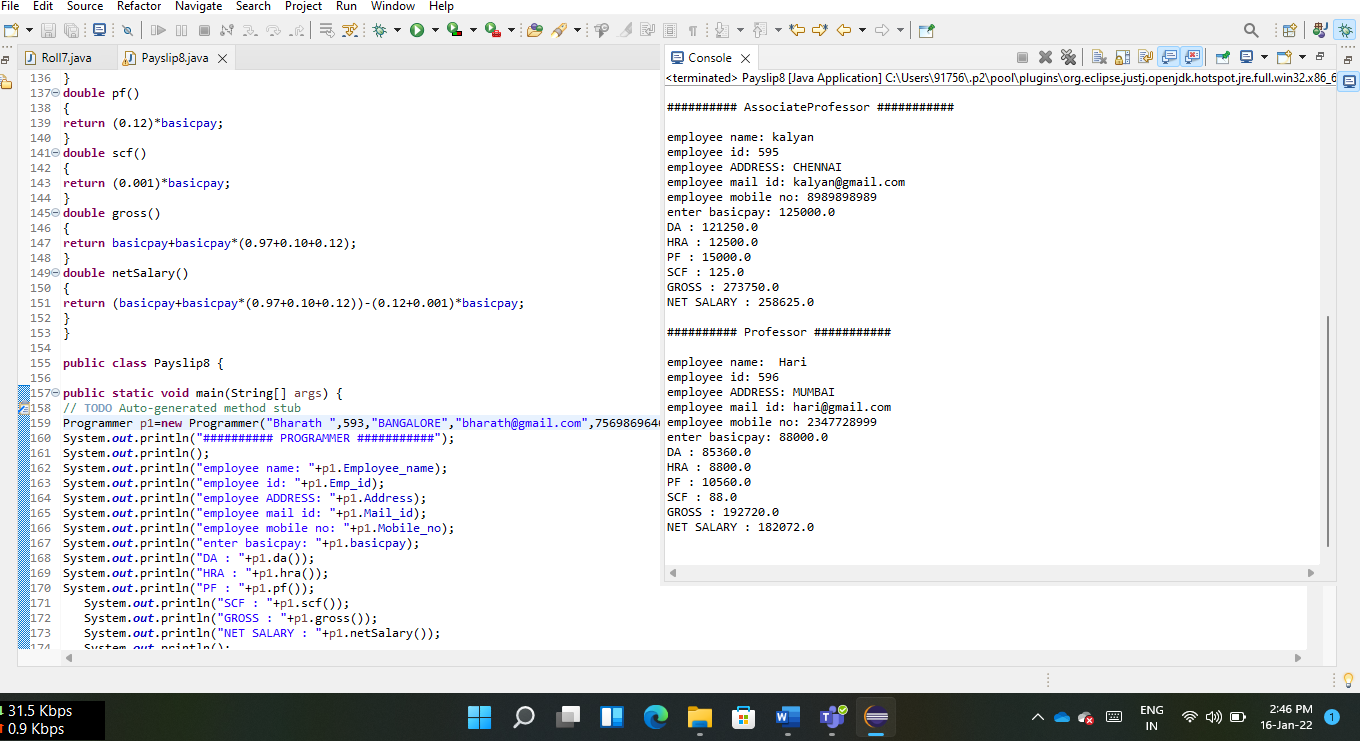
System.***out***.println("NET SALARY : "+p4.netSalary());

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

}//main

}//class Payslip8 close

**OUTPUT:**

****

**EXPERIMENT – 9**

**AIM:** Write a Java Program to create an abstract class named Shape that contains two integers and an empty method named print Area(). Provide three classes named Rectangle, Triangle and Circle such that each one of the classes extends the class Shape. Each one of the classes contains only the method print Area () that prints the area of the given shape.[CO2]

**DESCRIPTION:**

Abstract class in Java

A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body).

### Abstraction in Java

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

**SYNTAX:**

abstract class A

{

//STATEMENTS AND METHODS

}

**PROGRAM:**

**abstract** **class** Shape

{

**double** d1,d2,b;

Shape(**double** a,**double** b)

{

d1=a;

d2=b;

}

Shape(**double** a)

{

b=a;

}

**abstract** **void** printArea();

}

**class** Rectangle **extends** Shape

{

Rectangle(**double** x,**double** y)

{

**super**(x,y);

}

**void** printArea()

{

System.***out***.println("Area of rectangle is:"+d1\*d2);

}

}

**class** Triangle **extends** Shape

{

Triangle(**double** x,**double** y)

{

**super**(x,y);

}

**void** printArea()

{

System.***out***.println("Area of triangle is:"+(0.5\*d1\*d2));

}

}

**class** Circle **extends** Shape

{

Circle(**double** p)

{

**super**(p);

}

**void** printArea()

{

System.***out***.println("Area of circle is:"+(3.14\*b\*b));

}

}

**public** **class** Area9 {

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

{

Rectangle rect=**new** Rectangle(2,3);

rect.printArea();

Triangle tri=**new** Triangle(5.5,6);

tri.printArea();

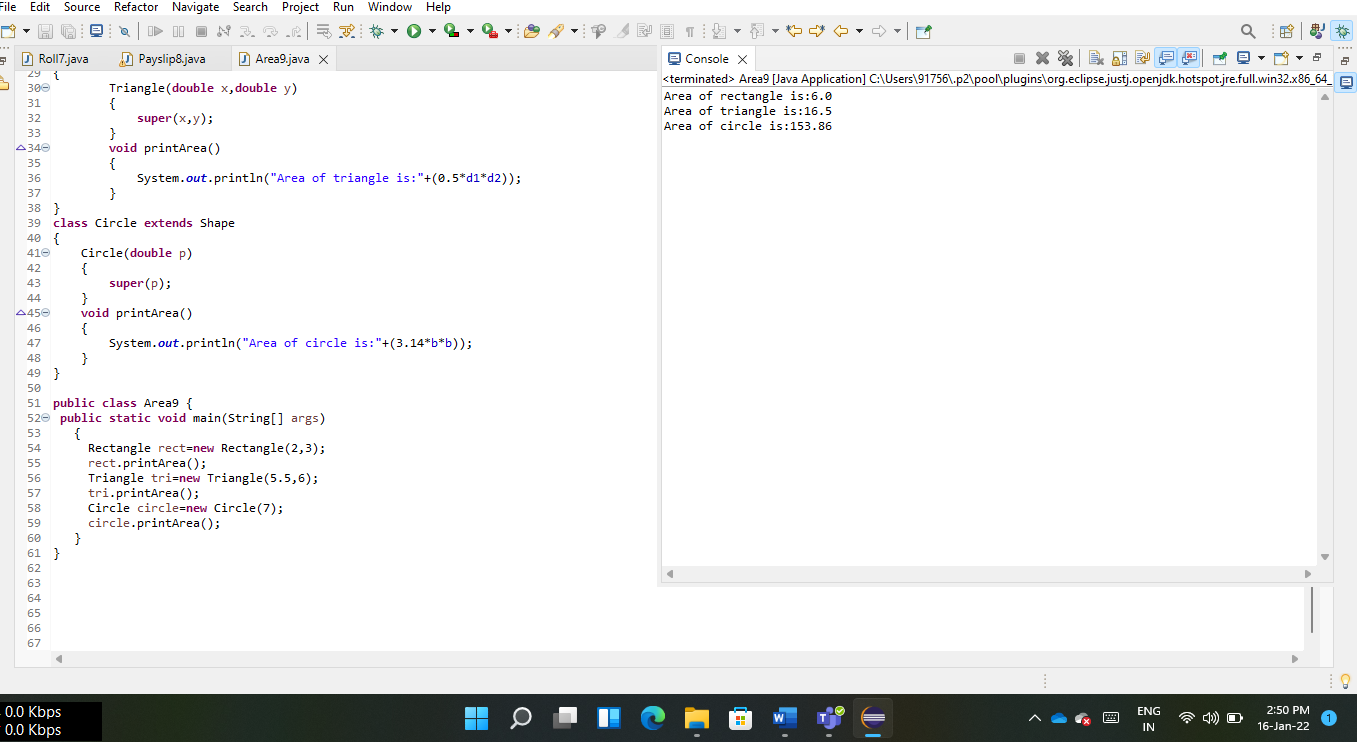
Circle circle=**new** Circle(7);

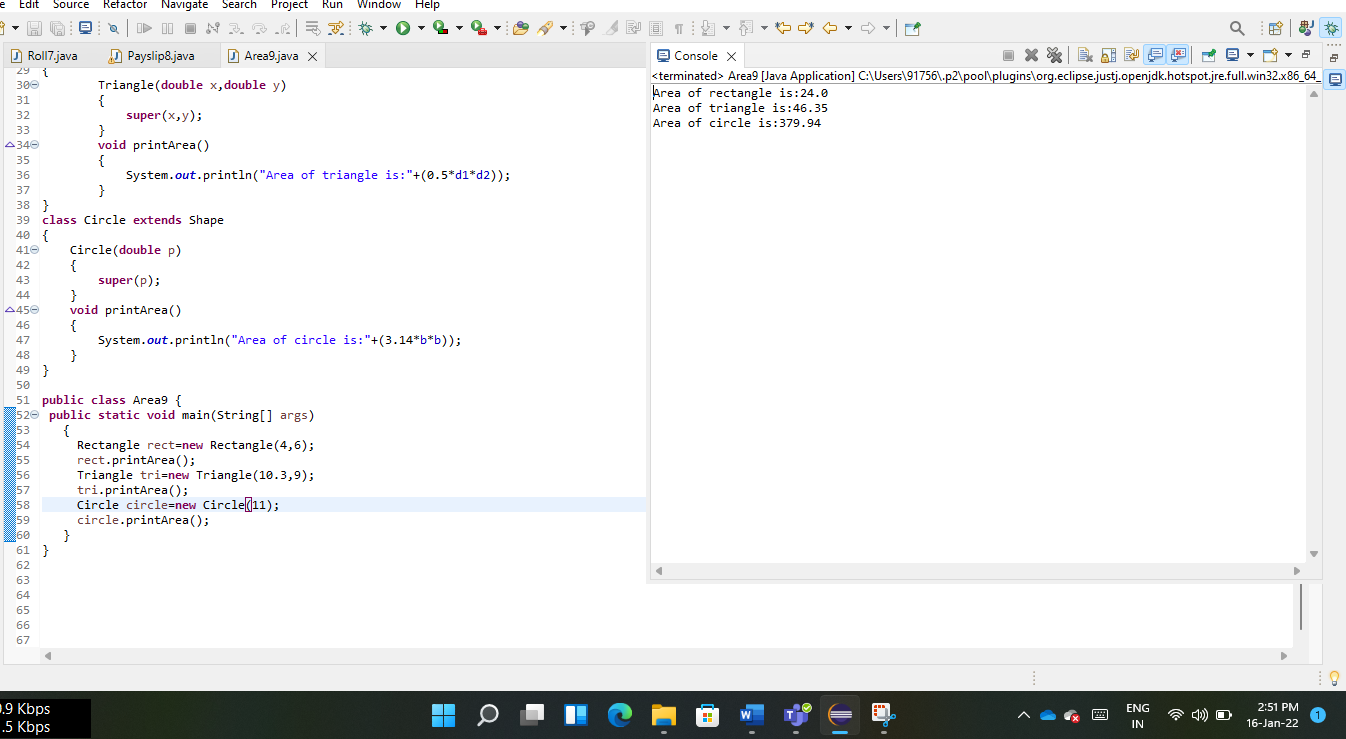
circle.printArea();

}

}

**OUTPUT:**

****

****

**EXPERIMENT – 10**

**AIM:** Develop a java application to implement currencyconverter(DollartoINR, EURO toINR,YentoINR and vice versa), distance converter (meter to KM, miles to KM and vice versa), timeconverter (hours to minutes, seconds and vice versa) using packages. [CO1]

**DESCRIPTION:**

**PACKAGE IN JAVA :**

Package in Java is a mechanism to encapsulate a group of classes, sub packages and interfaces. Packages are used for:

Preventing naming conflicts. For example there can be two classes with name Employee in two packages, college.staff.cse.Employee and college.staff.ee.Employee

Making searching/locating and usage of classes, interfaces, enumerations and annotations easier

Providing controlled access: protected and default have package level access control. A protected member is accessible by classes in the same package and its subclasses. A default member (without any access specifier) is accessible by classes in the same package only.

Packages can be considered as data encapsulation (or data-hiding).

All we need to do is put related classes into packages. After that, we can simply write an import class from existing packages and use it in our program. A package is a container of a group of related classes where some of the classes are accessible are exposed and others are kept for internal purpose.

We can reuse existing classes from the packages as many times as we need it in our program.

**SYNTAX:**

**Naming a Package:**

Package package\_name;

**Accessing classes inside a package:**

Consider following two statements:

// import the Swing class from util package.

import java.util.Swing;

// import all the classes from util package

import java.util.\*;

**PROGRAM:**

**package** CurrencyConverter;

**public** **class** Currency {

**public** **double** DollarToInr (**double** x)

{

**return** (x \* 74.42);

}

**public** **double** InrToDollar (**double** x)

{

**return** (x \* 0.013);

}

**public** **double** EuroToInr (**double** x)

{

**return** (x \* 84.37);

}

**public** **double** InrToEuro (**double** x)

{

**return** (x \* 0.012);

}

**public** **double** YenToInr (**double** x)

{

**return** (x \* 0.65);

}

**public** **double** InrToYen (**double** x)

{

**return** (x \* 1.54);

}

}

**package** DistanceConverter;

**public** **class** Distance {

**public** **double** MeterToKM(**double** x)

{

**return** (x\*0.001);

}

**public** **double** KMtoMeter(**double** x)

{

**return** (x\*1000);

}

**public** **double** milesToKM(**double** x)

{

**return** (x\*1.609);

}

**public** **double** KMtoMiles(**double** x)

{

**return** (x\*0.621);

}

}

**package** TimeConverter;

**public** **class** Time {

**public** **double** hoursToMinutes(**double** x)

{

**return** (x\*60);

}

**public** **double** minutesToHours(**double** x)

{

**return** (x\*0.0167);

}

**public** **double** hoursToSeconds(**double** x)

{

**return** (x\*3600);

}

**public** **double** secondsToHours(**double** x)

{

**return** (x\*0.00027778);

}

}

**import** java.util.Scanner;

**import** CurrencyConverter.\*;

**import** DistanceConverter.\*;

**import** TimeConverter.\*;

**public** **class** Conversions10 {

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

{

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

Currency c=**new** Currency();

Distance d=**new** Distance();

Time t=**new** Time();

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

System.***out***.println("1.Dollar to Rupee");

System.***out***.println("2.Rupee to Dollar");

System.***out***.println("3.Euro to Rupee");

System.***out***.println("4.Rupee to Euro");

System.***out***.println("5.Yen to Rupee");

System.***out***.println("6.Rupee to Yen");

System.***out***.println("7.Meter to km");

System.***out***.println("8.KM to meter");

System.***out***.println("9.Miles to KM");

System.***out***.println("10.KM to miles");

System.***out***.println("11.Hours to minutes");

System.***out***.println("12.Minutes to Hours");

System.***out***.println("13.Hours to seconds");

System.***out***.println("14.Seconds to hours");

System.***out***.println("15.Exit");

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

**int** x=0;

**try**

{

x=sc.nextInt();

**if**(x<0)

**throw** **new** Exception("Choice cannot be negative");

}

**catch**(Exception e )

{

System.***out***.println(e+" pls enter the choice");

x=sc.nextInt();

}

**switch**(x)

{

**case** 1:

System.***out***.println("Enetr the no.of dollars:");

**double** y=sc.nextDouble();

System.***out***.println("Rupees are:"+c.DollarToInr(y));

**break**;

**case** 2:

System.***out***.println("Enetr the no.of rupees:");

**double** z=sc.nextDouble();

System.***out***.println("Dollars are:"+c.InrToDollar(z));

**break**;

**case** 3:

System.***out***.println("Enetr the no.of Euros:");

**double** w=sc.nextDouble();

System.***out***.println("Rupees are:"+c.EuroToInr(w));

**break**;

**case** 4:

System.***out***.println("Enetr the no.of Rupees:");

**double** q=sc.nextDouble();

System.***out***.println("Euros are:"+c.InrToEuro(q));

**break**;

**case** 5:

System.***out***.println("Enetr the no.of Yens:");

**double** e=sc.nextDouble();

System.***out***.println("Rupees are:"+c.YenToInr(e));

**break**;

**case** 6:

System.***out***.println("Enetr the no.of rupees:");

**double** r=sc.nextDouble();

System.***out***.println("Yens are:"+c.InrToYen(r));

**break**;

**case** 7:

System.***out***.println("Enetr the no.of meters:");

**double** C=sc.nextDouble();

System.***out***.println("kilo meters are:"+d.MeterToKM(C));

**break**;

**case** 8:

System.***out***.println("Enetr the no.of kilometers:");

**double** Y=sc.nextDouble();

System.***out***.println("meters are:"+d.KMtoMeter(Y));

**break**;

**case** 9:

System.***out***.println("Enetr the no.of miles:");

**double** T=sc.nextDouble();

System.***out***.println("kilo meters are:"+d.milesToKM(T));

**break**;

**case** 10:

System.***out***.println("Enetr the no.of kilometers:");

**double** P=sc.nextDouble();

System.***out***.println("miles are:"+d.KMtoMiles(P));

**break**;

**case** 11:

System.***out***.println("Enetr the no.of hours:");

**double** L=sc.nextDouble();

System.***out***.println("minutes are:"+t.hoursToMinutes(L));

**break**;

**case** 12:

System.***out***.println("Enetr the no.of minutes:");

**double** M=sc.nextDouble();

System.***out***.println("Hours are:"+t.minutesToHours(M));

**break**;

**case** 13:

System.***out***.println("Enetr the no.of hours:");

**double** N=sc.nextDouble();

System.***out***.println("seconds are:"+t.hoursToSeconds(N));

**break**;

**case** 14:

System.***out***.println ("Enetr the no.of seconds:");

**double** B = sc.nextDouble ();

System.***out***.println ("kilo meters are:" + t.secondsToHours (B));

**break**;

**case** 15:

System.***out***.println ("YOU CHOOSED EXIT. TATA GOODBYE!!!");

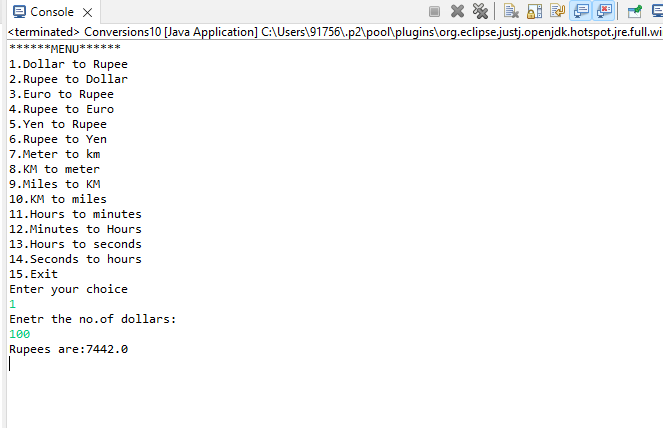
**break**;

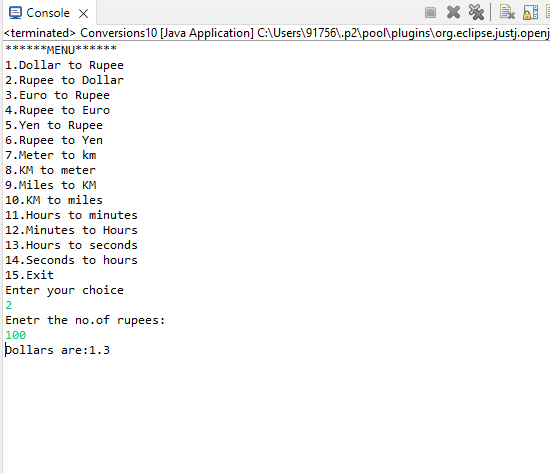
}//switch

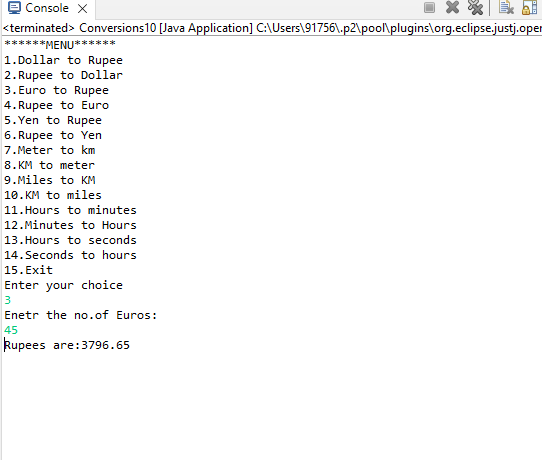
}//main

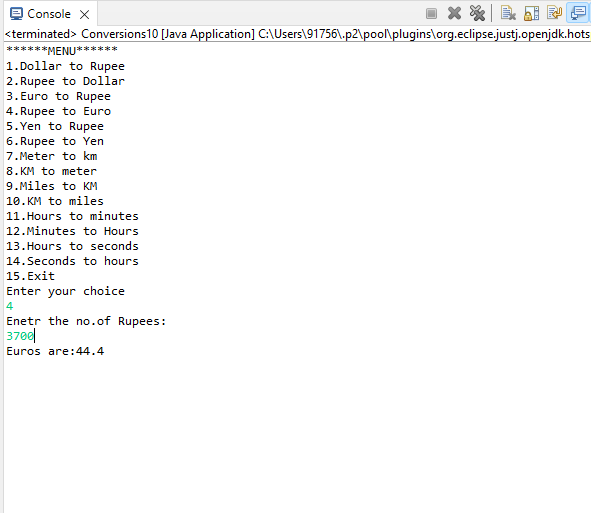
}//class

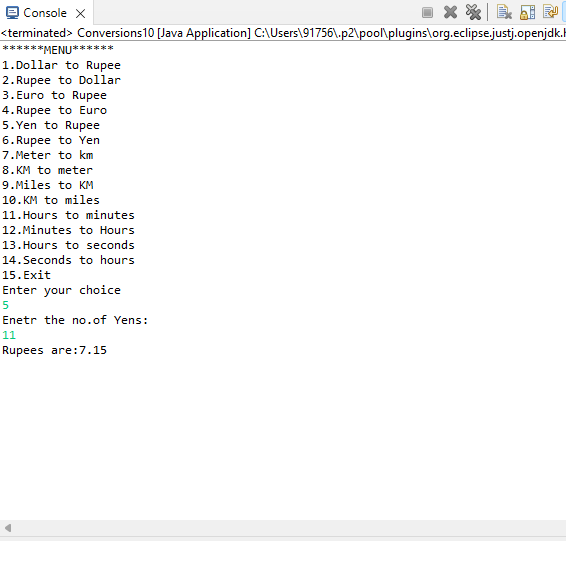
**OUTPUT:**

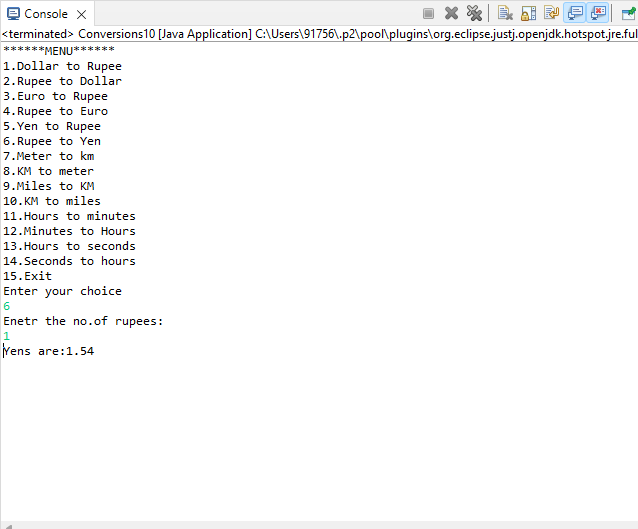
****

****

****

****

****

****

**EXPERIMENT – 11**

**AIM:** Write a Java Program to Handle Arithmetic Exceptions and InputMisMatchExceptions. [CO1]

**DESCRIPTION:**

An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e at run time, that disrupts the normal flow of the program’s instructions.

Exceptions can be handled by using try-catch blocks. The try block contains the code which may raise an exception and the catch is used to handle the exception. A code or set of statements can be written in the catch block. The catch block is executed when there is an exception in the try block. There can be multiple catch blocks to handle multiple exceptions or a single catch block to handle multiple exceptions. The catch block is then followed by finally block. The finally block executes even if the program terminates.

**SYNTAX:**

try {

    //code that may raise an exception

}catch(<exceptionname> en1) {

    //code that must be executed incase of an exception

}catch(<exeptionname> en2) {

     //code that must be executed incase of an exception

}

finally {

    //this block executes even if the program terminates

}

To catch multiple exceptions using single catch block try {    // catch block

}catch(ExceptionType1 | Exceptiontype2 ex) {

// catch block

  }

**PROGRAM:**

**import** java.util.Scanner;

**import** java.util.InputMismatchException;

**class** Exception11

{

**int** a, b, c;

**void** inputvalues()

{

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

**try**

{

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

a = sc.nextInt ();

}

**catch** (InputMismatchException e)

{

System.***out***.println ("excepted input is integer " + e);

}

**try**

{

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

b = sc.nextInt ();

}

**catch** (InputMismatchException e)

{

System.***out***.println ("excepted input is integer "+ e);

}

operation(a, b);//operation fn calls

}

**void** operation(**int** a, **int** b)

{

**try**

{

**int** c = a / b;

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

}

**catch** (ArithmeticException e)

{

System.***out***.println ("division is not possible "+ e);

}

}

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

{

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

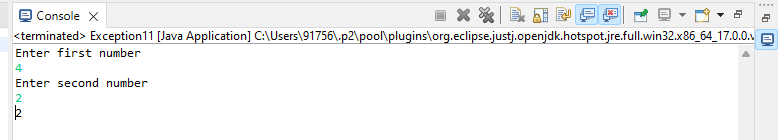
Exception11 t = **new** Exception11 ();

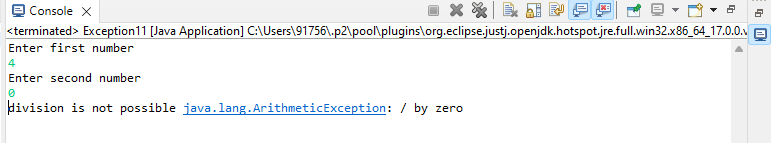
t.inputvalues();

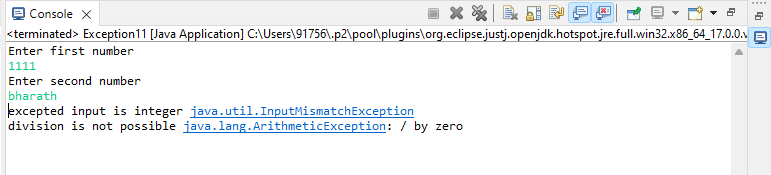
}

}

**OUTPUT:**

****

****

****

**EXPERIMENT – 12**

**AIM:** Write a multi-threaded Java program to print all numbers below 100,000 that are both prime and Fibonacci number (some examples are 2, 3, 5, 13, etc.). Design a thread that generates prime numbers below 100,000 and writes them into a pipe. Design another thread that generates Fibonacci numbers and writes them to another pipe. The main thread should read both the pipes to identify numbers common to both. [CO3].

**DESCRIPTION:**

**Java - PipedReader**

The PipedReader class is used to read the contents of a pipe as a stream of characters. This class is used generally to read text.

PipedReader class must be connected to the same PipedWriter and are used by different threads.

**Java - PipedWriter**

The PipedWriter class is used to write java pipe as a stream of characters. This class is used generally for writing text. Generally PipedWriter is connected to a PipedReader and used by different threads.

**Java Threads**

Threads allows a program to operate more efficiently by doing multiple things at the same time.

Threads can be used to perform complicated tasks in the background without interrupting the main program.

**SYNTAX:**

PipedReader(int pipeSize)

PipedWriter()//constructor public class Main extends Thread { public void run() {

System.out.println("This code is running in a thread");

}

}

public class Main implements Runnable { public void run() {

System.out.println("This code is running in a thread");

}

}

**PROGRAM:**

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

**import** java.io.PipedWriter;

**import** java.io.PipedReader;

**class** fibonacci **extends** Thread

{//85

PipedWriter fw=**new** PipedWriter();

**public** PipedWriter getwrite()

{

**return** fw;

}

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

{

**super**.run();

fibo();

}

**int** f(**int** n)

{

**if**(n<2)

**return** n;

**else**

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

}

**void** fibo()

{

**for**(**int** i=2,fibv=0;(fibv=f(i))<100000;i++)

{

**try**{

fw.write(fibv);

}

**catch**(IOException e)

{}

}

}

}

**class** receiver **extends** Thread

{

PipedReader fibr,primer;

**public** receiver(fibonacci fib,prime pr)**throws** IOException

{

fibr=**new** PipedReader(fib.getwrite());

primer=**new** PipedReader(pr.getwrite());

}

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

{

**int** p=0,f=0;

**try**{

p=primer.read();

f=fibr.read();

}

**catch**(IOException e)

{}

**while**(**true**)

{

**try**{

**if**(p==f)

{

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

p=primer.read();

f=fibr.read();

}

**else** **if**(f<p)

f=fibr.read();

**else**

p=primer.read();

}

**catch**(IOException e)

{

System.*exit*(-1);

}

}

}

}

**class** prime **extends** Thread

{

PipedWriter pw=**new** PipedWriter();

**public** PipedWriter getwrite()

{

**return** pw;

}

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

{

**super**.run();

prim();

}

**public** **void** prim()

{**for**(**int** i=2;i<100000;i++)

{**if**(isprime(i))

{

**try**{pw.write(i);}

**catch**(IOException e){}

}

}

}

**boolean** isprime(**int** n)

{

**boolean** p=**true**;

**int** s=(**int**)Math.*sqrt*(n);

**for**(**int** i=2;i<=s;i++)

{

**if**(n%i==0)

p=**false**;

}

**return** p;

}

}

**public** **class** Fibprime

{

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

fibonacci fi=**new** fibonacci();

prime pri=**new** prime();

receiver r=**new** receiver(fi,pri);

fi.start();

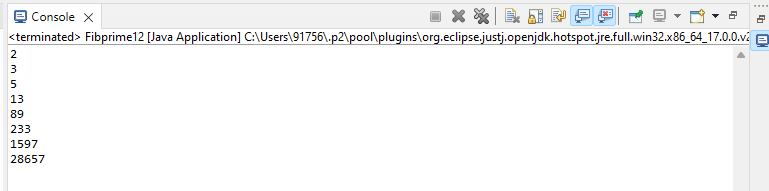
pri.start();

r.start();

}

}

**OUTPUT:**

****

**EXPERIMENT – 13**

**AIM:** Write a java program that implements a multi-threaded application that has three threads. First thread generates a random integer every 1 second and if the value is even, second thread computes the square of the number and prints. If the value is odd, the third thread will print the value of cube of the number. [CO3].

**DESCRIPTION:**

**MultiThreading:**

Multithreading in java is a process of executing multiple threads simultaneously. A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking. we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and contextswitching between the threads takes less time than process. Java Multithreading is mostly used in games, animation, etc.

**Thread :**

A thread is a lightweight subprocess, the smallest unit of processing. It is a separate path of execution.Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved in two ways:

Process-based Multitasking (Multiprocessing)

Thread-based Multitasking (Multithreading) **Methods in thread class:**

|  |  |  |
| --- | --- | --- |
| 1.start() 2.run() 3.sleep() | 4.currentThread() | 5.join() |
| 6.setPriority() 7.getPriority() | 8.getState() etc.. |  |

**SYNTAX:**

public class Main extends Thread { public void run() {

System.out.println("This code is running in a thread");

}

}

public class Main implements Runnable {

public void run() {

System.out.println("This code is running in a thread");

}

}

**PROGRAM:**

**import** java.util.Random;

**class** Square **extends** Thread

{

**int** x;

Square(**int** n)

{

x = n;

}

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

{

**int** sqr = x \* x;

System.***out***.println("Square of " + x + " = " + sqr );

}

}

**class** Cube **extends** Thread

{

**int** x;

Cube(**int** n)

{

x = n;

}

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

{

**int** cub = x \* x \* x;

System.***out***.println("Cube of " + x + " = " + cub );

}

}

**class** Number **extends** Thread

{

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

{

Random random = **new** Random();

**for**(**int** i =0; i<5; i++)

{

**int** randomInteger = random.nextInt(100);

System.***out***.println("Random Integer generated : " + randomInteger);

**if**(randomInteger%2==0)

{

Square s = **new** Square(randomInteger);

s.start();

}

**else**

{

Cube c = **new** Cube(randomInteger);

c.start();

}

**try** {

Thread.*sleep*(1000);

}

**catch** (InterruptedException ex)

{

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

}

}

}

}

**public** **class** Thr12 {

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

{

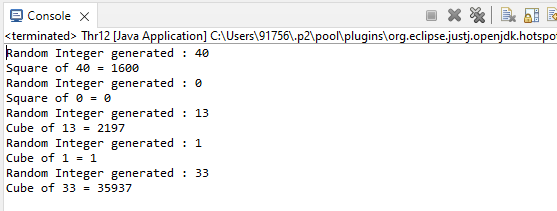
Number n = **new** Number();

n.start();

}

}

**OUTPUT:**

****

**EXPERIMENT – 14**

**AIM:** . Write a Java program that correctly implements the producer – consumer problem using the concept of inter-thread communication. [CO3].

**DESCRIPTION:**

**Inter-thread communication** or Co-operation is all about allowing synchronized threads to communicate with each other.

Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.It is implemented by following methods of Object class:

1. wait()
2. notify()
3. notifyAll()

The wait() method causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.

The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.

The notify() method wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation.

Wakes up all threads that are waiting on this object's monitor.

Java programming language provides a very handy way of creating threads and synchronizing their task by using **synchronized** blocks

**SYNTAX:**

public final void notify() public final void notifyAll() try{ wait(); }

catch(Exception e)

{

}

synchronized(objectidentifier) {

// Access shared variables and other shared resources

}

**PROGRAM:**

**class** Thread1

{

**int** n;

**boolean** valueset=**false**; **synchronized** **int** get()

{

**if** (!valueset)

**try**

{

wait();

}

**catch** (Exception e)

{

System.***out***.println("Excepton occur at : "+e);

}

System.***out***.println("get" +n);

**try**

{ Thread.*sleep*(1000);

}

**catch** (Exception e)

{

System.***out***.println("Excepton occur at :"+e);

}

valueset=**false**;

notify(); **return** n;

}

**synchronized** **int** put(**int** n)

{

**if** (valueset)

**try**

{

wait();

}

**catch** (Exception e)

{

System.***out***.println("Excepton occur at : "+e);

}

**this**.n=n; valueset=**true**;

System.***out***.println("put"+n);

**try**

{

Thread.*sleep*(1000);

}

**catch** (Exception e)

{

System.***out***.println("Excepton occur at : "+e);

}

notify();

**return** n;

}

}

**class** Producer **implements** Runnable

{

Thread1 t;

Producer(Thread1 t)

{

**this**.t=t;

**new** Thread(**this**,"Producer").start();

}

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

{

**int** i=0; **while** (**true**)

{ t.put(i++); }

}

}

**class** Consumer **implements** Runnable

{

Thread1 t;

Consumer(Thread1 t)

{

**this**.t=t;

**new** Thread(**this**,"Consumer").start();

}

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

{

**int** i=0; **while** (**true**)

{

t.get();

}

}

}

**public** **class** ProducerConsumer

{

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

{ Thread1 t=**new** Thread1(); **new** Producer(t);

**new** Consumer(t);

System.***out***.println("Press Control+c to exit");}

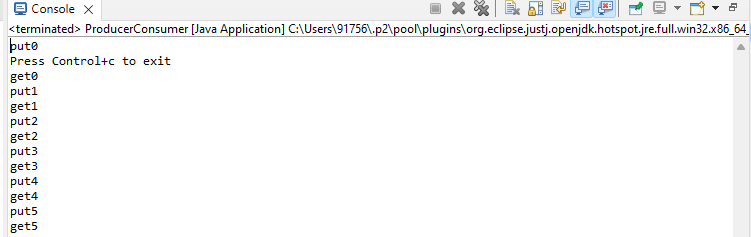
Thread1 t=**new** Thread1();{

**new** Producer(t);

**new** Consumer(t);

System.***out***.println("Press Control+c to exit");

} }

**OUTPUT:**

**EXPERIMENT – 15**

**AIM:** Write a Java program that reads a file name from the user, displays information about whether the file exists, whether the file is readable, or writable, the type of file and the length of the file inbytes. [CO1].

**DESCRIPTION:**

**Java File Handling**

The File class is an abstract representation of file and directory pathname. A pathname can be either absolute or relative.

The File class have several methods for working with directories and files such as creating new directories or files, deleting and renaming directories or files, listing the contents of a directory etc.

**Methods**: **getParent():**

It returns the pathname string of this abstract pathname's parent, or null if this pathname does not name a parent directory. **canWrite():**

It tests whether the application can modify the file denoted by this abstract pathname.String[] **canExecute():**

It tests whether the application can execute the file denoted by this abstract pathname. **isAbsolute():**

It tests whether this abstract pathname is absolute.

**SYNTAX:**

File(File parent, String child)// It creates a new File instance from a parent abstract pathname and a child pathname string.

File(String pathname)// It creates a new File instance by converting the given pathname string into an abstract pathname. File f1=new File(fname); f1.getName() f1.canExecute() f1.canRead() f1.canWrite()

f1.getParentFile() f1.exists() f1.length()

**PROGRAM:**

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

**import** java.util.\*;

**public** **class** FileFunctions15 {

;

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

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

String fname;

Scanner sc=**new** Scanner(System.***in***); System.***out***.println("Enter the filenmae:");

fname=sc.nextLine();

File f1=**new** File(fname);

System.***out***.println("File name:"+f1.getName());

System.***out***.println("Executable File:"+f1.canExecute());

System.***out***.println("Readable:"+f1.canRead()); System.***out***.println("Writable:"+f1.canWrite());

System.***out***.println("Existance of the file:"+f1.exists());

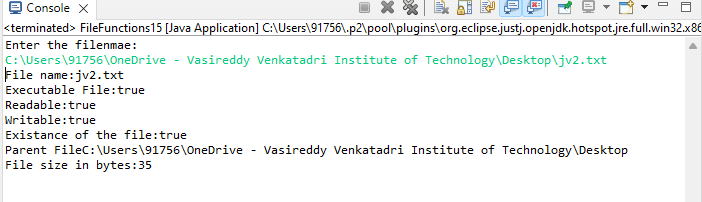
System.***out***.println("Parent File"+f1.getParentFile());

System.***out***.println("File size in bytes:"+f1.length());

}

}

**OUTPUT:**

****

**EXPERIMENT – 16**

**AIM:** . Write a Java program to build a Calculator in Swings. [CO4]

**DESCRIPTION:**

**SYNTAX:**

**FOR LABEL**:

JLabel jb = new JLabel(“content to display”)//constructor //

**FOR FRAME :**

JFrame jfrm = new Jframe(“frame title”)//constructor

**for button:**

JButton jbt = new JButton(“button content”)//constructor

**for the text field:**

JtextField jtf = new JTextFeild(22)//constructor specify number of letters to fit in text field

addWindowListener//interface

public interface WindowListener extends EventListener

addWindowAdapter//class

WindowAdapter() //to be implemented using windowlistener by anonymous function

**PROGRAM:**

**import** java.awt.\*;

**import** java.awt.event.\*;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**public** **class** MyCalculator **extends** Frame

{

**public** **boolean** setClear=**true**;

**double** number, memValue;

**char** op;

String digitButtonText[] = {"7", "8", "9", "4", "5", "6", "1", "2", "3", "0", "+/-", "." };

String operatorButtonText[] = {"/", "sqrt", "\*", "%", "-", "1/X", "+", "=" };

String memoryButtonText[] = {"MC", "MR", "MS", "M+" };

String specialButtonText[] = {"Backspc", "C", "CE" };

MyDigitButton digitButton[]=**new** MyDigitButton[digitButtonText.length];

MyOperatorButton operatorButton[]=**new** MyOperatorButton[operatorButtonText.length];

MyMemoryButton memoryButton[]=**new** MyMemoryButton[memoryButtonText.length];

MySpecialButton specialButton[]=**new** MySpecialButton[specialButtonText.length];

Label displayLabel=**new** Label("0",Label.***RIGHT***);

Label memLabel=**new** Label(" ",Label.***RIGHT***);

**final** **int** FRAME\_WIDTH=325,FRAME\_HEIGHT=325;

**final** **int** HEIGHT=30, WIDTH=30, H\_SPACE=10,V\_SPACE=10;

**final** **int** TOPX=30, TOPY=50;

///////////////////////////

MyCalculator(String frameText)//constructor

{

**super**(frameText);

**int** tempX=TOPX, y=TOPY;

displayLabel.setBounds(tempX,y,240,HEIGHT);

displayLabel.setBackground(Color.***BLUE***);

displayLabel.setForeground(Color.***WHITE***);

add(displayLabel);

memLabel.setBounds(TOPX, TOPY+HEIGHT+ V\_SPACE,WIDTH, HEIGHT);

add(memLabel);

// set Co-ordinates for Memory Buttons

tempX=TOPX;

y=TOPY+2\*(HEIGHT+V\_SPACE);

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

{

memoryButton[i]=**new** MyMemoryButton(tempX,y,WIDTH,HEIGHT,memoryButtonText[i], **this**);

memoryButton[i].setForeground(Color.***RED***);

y+=HEIGHT+V\_SPACE;

}

//set Co-ordinates for Special Buttons

tempX=TOPX+1\*(WIDTH+H\_SPACE); y=TOPY+1\*(HEIGHT+V\_SPACE);

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

{

specialButton[i]=**new** MySpecialButton(tempX,y,WIDTH\*2,HEIGHT,specialButtonText[i], **this**);

specialButton[i].setForeground(Color.***RED***);

tempX=tempX+2\*WIDTH+H\_SPACE;

}

//set Co-ordinates for Digit Buttons

**int** digitX=TOPX+WIDTH+H\_SPACE;

**int** digitY=TOPY+2\*(HEIGHT+V\_SPACE);

tempX=digitX; y=digitY;

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

{

digitButton[i]=**new** MyDigitButton(tempX,y,WIDTH,HEIGHT,digitButtonText[i], **this**);

digitButton[i].setForeground(Color.***BLUE***);

tempX+=WIDTH+H\_SPACE;

**if**((i+1)%3==0){tempX=digitX; y+=HEIGHT+V\_SPACE;}

}

//set Co-ordinates for Operator Buttons

**int** opsX=digitX+2\*(WIDTH+H\_SPACE)+H\_SPACE;

**int** opsY=digitY;

tempX=opsX; y=opsY;

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

{

tempX+=WIDTH+H\_SPACE;

operatorButton[i]=**new** MyOperatorButton(tempX,y,WIDTH,HEIGHT,operatorButtonText[i], **this**);

operatorButton[i].setForeground(Color.***RED***);

**if**((i+1)%2==0){tempX=opsX; y+=HEIGHT+V\_SPACE;}

}

addWindowListener(**new** WindowAdapter()

{

**public** **void** windowClosing(WindowEvent ev)

{System.*exit*(0);}

});

setLayout(**null**);

setSize(FRAME\_WIDTH,FRAME\_HEIGHT);

setVisible(**true**);

}

//////////////////////////////////

**static** String getFormattedText(**double** temp)

{

String resText=""+temp;

**if**(resText.lastIndexOf(".0")>0)

resText=resText.substring(0,resText.length()-2);

**return** resText;

}

////////////////////////////////////////

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

{

**new** MyCalculator("Calculator - JavaTpoint");

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**class** MyDigitButton **extends** Button **implements** ActionListener

{

MyCalculator cl;

//////////////////////////////////////////

MyDigitButton(**int** x,**int** y, **int** width,**int** height,String cap, MyCalculator clc)

{

**super**(cap);

setBounds(x,y,width,height);

**this**.cl=clc;

**this**.cl.add(**this**);

addActionListener(**this**);

}

////////////////////////////////////////////////

**static** **boolean** isInString(String s, **char** ch)

{

**for**(**int** i=0; i<s.length();i++) **if**(s.charAt(i)==ch) **return** **true**;

**return** **false**;

}

/////////////////////////////////////////////////

**public** **void** actionPerformed(ActionEvent ev)

{

String tempText=((MyDigitButton)ev.getSource()).getLabel();

**if**(tempText.equals("."))

{

**if**(cl.setClear)

{cl.displayLabel.setText("0.");cl.setClear=**false**;}

**else** **if**(!*isInString*(cl.displayLabel.getText(),'.'))

cl.displayLabel.setText(cl.displayLabel.getText()+".");

**return**;

}

**int** index=0;

**try**{

index=Integer.*parseInt*(tempText);

}**catch**(NumberFormatException e){**return**;}

**if** (index==0 && cl.displayLabel.getText().equals("0")) **return**;

**if**(cl.setClear)

{cl.displayLabel.setText(""+index);cl.setClear=**false**;}

**else**

cl.displayLabel.setText(cl.displayLabel.getText()+index);

}//actionPerformed

}//class defination

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**class** MyOperatorButton **extends** Button **implements** ActionListener

{

MyCalculator cl;

MyOperatorButton(**int** x,**int** y, **int** width,**int** height,String cap, MyCalculator clc)

{

**super**(cap);

setBounds(x,y,width,height);

**this**.cl=clc;

**this**.cl.add(**this**);

addActionListener(**this**);

}

///////////////////////

**public** **void** actionPerformed(ActionEvent ev)

{

String opText=((MyOperatorButton)ev.getSource()).getLabel();

cl.setClear=**true**;

**double** temp=Double.*parseDouble*(cl.displayLabel.getText());

**if**(opText.equals("1/x"))

{

**try**

{**double** tempd=1/(**double**)temp;

cl.displayLabel.setText(MyCalculator.*getFormattedText*(tempd));}

**catch**(ArithmeticException excp)

{cl.displayLabel.setText("Divide by 0.");}

**return**;

}

**if**(opText.equals("sqrt"))

{

**try**

{**double** tempd=Math.*sqrt*(temp);

cl.displayLabel.setText(MyCalculator.*getFormattedText*(tempd));}

**catch**(ArithmeticException excp)

{cl.displayLabel.setText("Divide by 0.");}

**return**;

}

**if**(!opText.equals("="))

{

cl.number=temp;

cl.op=opText.charAt(0);

**return**;

}

// process = button pressed

**switch**(cl.op)

{

**case** '+':

temp+=cl.number;**break**;

**case** '-':

temp=cl.number-temp;**break**;

**case** '\*':

temp\*=cl.number;**break**;

**case** '%':

**try**{temp=cl.number%temp;}

**catch**(ArithmeticException excp)

{cl.displayLabel.setText("Divide by 0."); **return**;}

**break**;

**case** '/':

**try**{temp=cl.number/temp;}

**catch**(ArithmeticException excp)

{cl.displayLabel.setText("Divide by 0."); **return**;}

**break**;

}//switch

cl.displayLabel.setText(MyCalculator.*getFormattedText*(temp));

//cl.number=temp;

}//actionPerformed

}//class

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**class** MyMemoryButton **extends** Button **implements** ActionListener

{

MyCalculator cl;

/////////////////////////////////

MyMemoryButton(**int** x,**int** y, **int** width,**int** height,String cap, MyCalculator clc)

{

**super**(cap);

setBounds(x,y,width,height);

**this**.cl=clc;

**this**.cl.add(**this**);

addActionListener(**this**);

}

////////////////////////////////////////////////

**public** **void** actionPerformed(ActionEvent ev)

{

**char** memop=((MyMemoryButton)ev.getSource()).getLabel().charAt(1);

cl.setClear=**true**;

**double** temp=Double.*parseDouble*(cl.displayLabel.getText());

**switch**(memop)

{

**case** 'C':

cl.memLabel.setText(" ");cl.memValue=0.0;**break**;

**case** 'R':

cl.displayLabel.setText(MyCalculator.*getFormattedText*(cl.memValue));**break**;

**case** 'S':

cl.memValue=0.0;

**case** '+':

cl.memValue+=Double.*parseDouble*(cl.displayLabel.getText());

**if**(cl.displayLabel.getText().equals("0") || cl.displayLabel.getText().equals("0.0") )

cl.memLabel.setText(" ");

**else**

cl.memLabel.setText("M");

**break**;

}//switch

}//actionPerformed

}//class

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**class** MySpecialButton **extends** Button **implements** ActionListener

{

MyCalculator cl;

MySpecialButton(**int** x,**int** y, **int** width,**int** height,String cap, MyCalculator clc)

{

**super**(cap);

setBounds(x,y,width,height);

**this**.cl=clc;

**this**.cl.add(**this**);

addActionListener(**this**);

}

//////////////////////

**static** String backSpace(String s)

{

String Res="";

**for**(**int** i=0; i<s.length()-1; i++) Res+=s.charAt(i);

**return** Res;

}

//////////////////////////////////////////////////////////

**public** **void** actionPerformed(ActionEvent ev)

{

String opText=((MySpecialButton)ev.getSource()).getLabel();

//check for backspace button

**if**(opText.equals("Backspc"))

{

String tempText=*backSpace*(cl.displayLabel.getText());

**if**(tempText.equals(""))

cl.displayLabel.setText("0");

**else**

cl.displayLabel.setText(tempText);

**return**;

}

//check for "C" button i.e. Reset

**if**(opText.equals("C"))

{

cl.number=0.0; cl.op=' '; cl.memValue=0.0;

cl.memLabel.setText(" ");

}

//it must be CE button pressed

cl.displayLabel.setText("0");cl.setClear=**true**;

}//actionPerformed

}//class

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Features not implemented and few bugs

i) No coding done for "+/-" button.

ii) Menubar is not included.

iii)Not for Scientific calculation

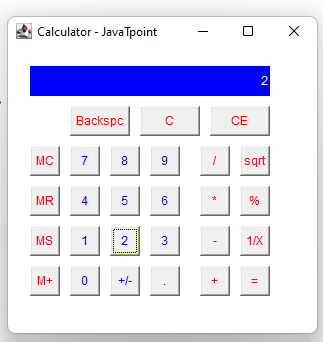
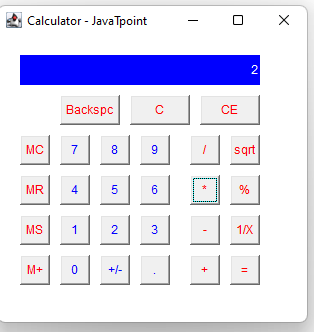
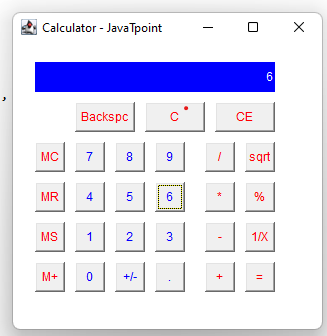
iv)Some of the computation may lead to unexpected result

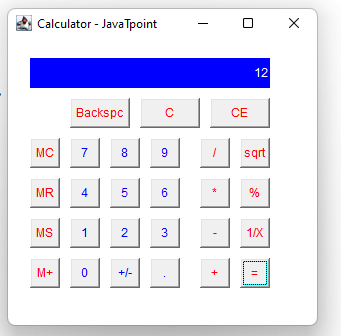
due to the representation of Floating point numbers in computer

is an approximation to the given value that can be stored

physically in memory.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**OUTPUT:**



**EXPERIMENT – 17**

**AIM:** Write a Java program to implement JMenu to draw all basic shapes using Graphics. [CO4]

**DESCRIPTION:**

Class Graphics [java.lang.Object](https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html) java.awt.Graphics public abstract class Graphics extends [Object](https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html)

The Graphics class is the abstract base class for all graphics contexts that allow an application to draw onto components that are realized on various devices, as well as onto off-screen images.

public [Graphics](https://docs.oracle.com/javase/7/docs/api/java/awt/Graphics.html) create(int x,

int y, int width, int height)

Creates a new Graphics object based on this Graphics object, but with a new translation and clip area. The new Graphics object has its origin translated to the specified point (x, y). Its clip area is determined by the intersection of the original clip area with the specified rectangle. The arguments are all interpreted in the coordinate system of the original Graphics object. The new graphics context is identical to the original, except in two respects:

The new graphics context is translated by (x, y). That is to say, the point (0, 0) in the new graphics context is the same as (x, y) in the original graphics context.

The new graphics context has an additional clipping rectangle, in addition to whatever (translated) clipping rectangle it inherited from the original graphics context. The origin of the new clipping rectangle is at (0, 0), and its size is specified by the width and height arguments.

Parameters:

x - the x coordinate. y - the y coordinate.

width - the width of the clipping rectangle. height - the height of the clipping rectangle.

**SYNTAX:**

Graphics create(int x, int y, int width, int height)

abstract void drawPolygon(int[] xPoints, int[] yPoints, int nPoints)

//Draws a closed polygon defined by arrays of x and y coordinates. void drawPolygon(Polygon p)

//Draws the outline of a polygon defined by the specified Polygon object.

abstract voiddrawPolyline(int[] xPoints, int[] yPoints, int nPoints)

//Draws a sequence of connected lines defined by arrays of x and y coordinates.

void drawRect(int x, int y, int width, int height)

//Draws the outline of the specified rectangle.

**PROGRAM:**

**import** java.awt.\*;

**import** java.awt.event.\*;

**import** javax.swing.\*;

**public** **class** Menu1 **extends** JFrame {

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

// A main routine that allows this class to be run

// as a stand-alone application. It just opens a frame.

**new** Menu1();

}

JRadioButtonMenuItem black, red, green, blue, cyan, magenta,

yellow, white, custom;

// Items for the "Color" menu, which controls the drawing color.

// They form a group in which only one item can be selected.

// When the user starts drawing, the color is determined by

// checking to see which of the items is selected.

JRadioButtonMenuItem curve, straightLine, rectangle, oval,

roundRect, filledRectangle, filledOval, filledRoundRect;

// Items for the "Shape" menu, which determine the shape to be drawn.

JRadioButtonMenuItem noSymmetry, twoWay, fourWay, eightWay;

// Items for the "Symmetry" menu, which determine which

// reflections of the basic figure should be drawn.

**public** **boolean** standAlone = **true**;

// If a frame is created by an applet, the applet should

// set this variable to false. Otherwise, an error will

// be generated when the user selects the "Quit" command,

// since that command will call System.exit() if standalone

// is true. The applet should also call the frame's

// setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE).

**public** Menu1() { // replaces init() method.

// Constructor creates a drawing area and uses it as its

// content pane. It also sets up the menu bar.

**super**("Graphics Menu"); // Set a title for the window.

Display canvas = **new** Display(); // The drawing area.

setContentPane(canvas);

// Create menu bar and menus.

JMenuBar menubar = **new** JMenuBar();

JMenu controlMenu = **new** JMenu("Control",**true**);

menubar.add(controlMenu);

JMenu colorMenu = **new** JMenu("Color",**true**);

menubar.add(colorMenu);

JMenu shapeMenu = **new** JMenu("Shape",**true**);

menubar.add(shapeMenu);

JMenu symmetryMenu = **new** JMenu("Symmetry",**true**);

menubar.add(symmetryMenu);

setJMenuBar(menubar);

// Set up the "Control" menu, and set the canvas to respond

// to commands from this menu. Add accelerators for some

// of the commands.

controlMenu.add("Fill with Black").addActionListener(canvas);

controlMenu.add("Fill with Red").addActionListener(canvas);

controlMenu.add("Fill with Green").addActionListener(canvas);

controlMenu.add("Fill with Blue").addActionListener(canvas);

controlMenu.add("Fill with Cyan").addActionListener(canvas);

controlMenu.add("Fill with Magenta").addActionListener(canvas);

controlMenu.add("Fill with Yellow").addActionListener(canvas);

controlMenu.add("Fill with White").addActionListener(canvas);

controlMenu.add("Fill with Custom").addActionListener(canvas);

controlMenu.addSeparator();

JMenuItem customItem = **new** JMenuItem("Set Custom Color...");

customItem.addActionListener(canvas);

customItem.setAccelerator( KeyStroke.*getKeyStroke*("ctrl T") );

controlMenu.add(customItem);

JMenuItem clearItem = **new** JMenuItem("Clear");

clearItem.addActionListener(canvas);

clearItem.setAccelerator( KeyStroke.*getKeyStroke*("ctrl K") );

controlMenu.add(clearItem);

JMenuItem undoItem = **new** JMenuItem("Undo");

undoItem.addActionListener(canvas);

undoItem.setAccelerator( KeyStroke.*getKeyStroke*("ctrl Z") );

controlMenu.add(undoItem);

JMenuItem quitItem = **new** JMenuItem("Quit");

quitItem.setAccelerator( KeyStroke.*getKeyStroke*("ctrl Q") );

quitItem.addActionListener(canvas);

controlMenu.add(quitItem);

// Set up the "Color" menu, with all the items in a button group.

ButtonGroup colorGroup = **new** ButtonGroup();

black = **new** JRadioButtonMenuItem("Black");

colorGroup.add(black);

colorMenu.add(black);

red = **new** JRadioButtonMenuItem("Red");

colorGroup.add(red);

colorMenu.add(red);

green = **new** JRadioButtonMenuItem("Green");

colorGroup.add(green);

colorMenu.add(green);

blue = **new** JRadioButtonMenuItem("Blue");

colorGroup.add(blue);

colorMenu.add(blue);

cyan = **new** JRadioButtonMenuItem("Cyan");

colorGroup.add(cyan);

colorMenu.add(cyan);

magenta = **new** JRadioButtonMenuItem("Magenta");

colorGroup.add(magenta);

colorMenu.add(magenta);

yellow = **new** JRadioButtonMenuItem("Yellow");

colorGroup.add(yellow);

colorMenu.add(yellow);

white = **new** JRadioButtonMenuItem("White");

colorGroup.add(white);

colorMenu.add(white);

custom = **new** JRadioButtonMenuItem("Custom Color");

colorGroup.add(custom);

colorMenu.add(custom);

black.setSelected(**true**);

// Set up the "Shape" menu.

ButtonGroup shapeGroup = **new** ButtonGroup();

curve = **new** JRadioButtonMenuItem("Curve");

shapeGroup.add(curve);

shapeMenu.add(curve);

straightLine = **new** JRadioButtonMenuItem("Straight Line");

shapeGroup.add(straightLine);

shapeMenu.add(straightLine);

rectangle = **new** JRadioButtonMenuItem("Rectangle");

shapeGroup.add(rectangle);

shapeMenu.add(rectangle);

oval = **new** JRadioButtonMenuItem("Oval");

shapeGroup.add(oval);

shapeMenu.add(oval);

roundRect = **new** JRadioButtonMenuItem("RoundRect");

shapeGroup.add(roundRect);

shapeMenu.add(roundRect);

filledRectangle = **new** JRadioButtonMenuItem("Filled Rectangle");

shapeGroup.add(filledRectangle);

shapeMenu.add(filledRectangle);

filledOval = **new** JRadioButtonMenuItem("Filled Oval");

shapeGroup.add(filledOval);

shapeMenu.add(filledOval);

filledRoundRect = **new** JRadioButtonMenuItem("Filled RoundRect");

shapeGroup.add(filledRoundRect);

shapeMenu.add(filledRoundRect);

curve.setSelected(**true**);

// Set up the "Symmetry" menu.

ButtonGroup symmetryGroup = **new** ButtonGroup();

noSymmetry = **new** JRadioButtonMenuItem("None");

noSymmetry.setAccelerator( KeyStroke.*getKeyStroke*("ctrl 0") );

symmetryGroup.add(noSymmetry);

symmetryMenu.add(noSymmetry);

twoWay = **new** JRadioButtonMenuItem("Two-way");

twoWay.setAccelerator( KeyStroke.*getKeyStroke*("ctrl 2") );

symmetryGroup.add(twoWay);

symmetryMenu.add(twoWay);

fourWay = **new** JRadioButtonMenuItem("Four-way");

fourWay.setAccelerator( KeyStroke.*getKeyStroke*("ctrl 4") );

symmetryGroup.add(fourWay);

symmetryMenu.add(fourWay);

eightWay = **new** JRadioButtonMenuItem("Eight-way");

eightWay.setAccelerator( KeyStroke.*getKeyStroke*("ctrl 8") );

symmetryGroup.add(eightWay);

symmetryMenu.add(eightWay);

noSymmetry.setSelected(**true**);

// Set size, etc., of frame and make it visible.

pack();

setLocation(75,50);

setResizable(**false**);

setDefaultCloseOperation(***EXIT\_ON\_CLOSE***);

~~show~~();

} // end constructor

**private** **class** Display **extends** JPanel

**implements** MouseListener, MouseMotionListener, ActionListener {

// Nested class Display represents the drawing surface of the

// applet. It lets the user use the mouse to draw colored curves

// and shapes. The current color is specified by the pop-up menu

// colorChoice. The current shape is specified by another pop-up menu,

// figureChoice. (These are instance variables in the main class.)

// The panel also listens for action events from buttons

// named "Clear" and "Set Background". The "Clear" button fills

// the panel with the current background color. The "Set Background"

// button sets the background color to the current drawing color and

// then clears. These buttons are set up in the main class.

**private** **final** **static** **int**

***CURVE*** = 0,

***LINE*** = 1,

***RECT*** = 2, // Some constants that code

***OVAL*** = 3, // for the different types of

***ROUNDRECT*** = 4, // figure the program can draw.

***FILLED\_RECT*** = 5,

***FILLED\_OVAL*** = 6,

***FILLED\_ROUNDRECT*** = 7;

**private** **final** **static** **int**

***NO\_SYMMETRY*** = 0, // Some constants that code for

***SYMMETRY\_2*** = 1, // the different symmetry styles.

***SYMMETRY\_4*** = 2,

***SYMMETRY\_8*** = 3;

Color customColor = Color.***gray***; // The custom color that is used

// when the user selects "Custom Color"

// as the drawing color or "Fill with Custom"

// from the "Control" menu. This color

// is changed when the user selects the

// "Set Custom Color..." command.

/\* Some variables used for backing up the contents of the panel. \*/

Image OSI; // The off-screen image (created in checkOSI()).

**int** widthOfOSI, heightOfOSI; // Current width and height of OSI. These

// are checked against the size of the applet,

// to detect any change in the panel's size.

// If the size has changed, a new OSI is created.

// The picture in the off-screen image is lost

// when that happens.

Image undoBuffer; // An off-screen image that is used to implement

// the undo operation. When the user begins

// a drawing operation, the OSI is copied to

// undoBuffer. If the user selects the "Undo"

// command, the OSI and the undoBuffer are swapped

// and the panel is repainted to show the previous image.

/\* The following variables are used when the user is sketching a

curve while dragging a mouse. \*/

**private** **int** mouseX, mouseY; // The location of the mouse.

**private** **int** prevX, prevY; // The previous location of the mouse.

**private** **int** startX, startY; // The starting position of the mouse.

// (Not used for drawing curves.)

**private** **boolean** dragging; // This is set to true when the user is drawing.

**private** **int** figure; // What type of figure is being drawn. This is

// specified by the figureChoice menu.

**private** **int** symmetry; // What type of symmetry style is being used. This is

// specified by the symmetryChoice menu.

**private** Graphics dragGraphics; // A graphics context for the off-screen image,

// to be used while a drag is in progress.

**private** Color dragColor; // The color that is used for the figure that is

// being drawn.

Display() {

// Constructor. When this component is first created, it is set to

// listen for mouse events and mouse motion events from

// itself. The initial background color is white.

addMouseListener(**this**);

addMouseMotionListener(**this**);

setBackground(Color.***white***);

setPreferredSize( **new** Dimension(450,450) );

}

**private** Color getSelectedColor() {

// Check the "Color" menu and return the color

// that is currently selected.

**if** (black.isSelected())

**return** Color.***black***;

**else** **if** (red.isSelected())

**return** Color.***red***;

**else** **if** (green.isSelected())

**return** Color.***green***;

**else** **if** (blue.isSelected())

**return** Color.***blue***;

**else** **if** (cyan.isSelected())

**return** Color.***cyan***;

**else** **if** (magenta.isSelected())

**return** Color.***magenta***;

**else** **if** (yellow.isSelected())

**return** Color.***yellow***;

**else** **if** (white.isSelected())

**return** Color.***white***;

**else**

**return** customColor;

}

**private** **int** getSelectedShape() {

// Check the "Shape" menu and return the code

// for the shape that is currently selected.

**if** (curve.isSelected())

**return** ***CURVE***;

**else** **if** (straightLine.isSelected())

**return** ***LINE***;

**else** **if** (rectangle.isSelected())

**return** ***RECT***;

**else** **if** (oval.isSelected())

**return** ***OVAL***;

**else** **if** (roundRect.isSelected())

**return** ***ROUNDRECT***;

**else** **if** (filledRectangle.isSelected())

**return** ***FILLED\_RECT***;

**else** **if** (filledOval.isSelected())

**return** ***FILLED\_OVAL***;

**else**

**return** ***FILLED\_ROUNDRECT***;

}

**private** **int** getSelectedSymmetry() {

// Check the "Symmetry" menu and return the code

// for the type of symmetry that is currently selected.

**if** (noSymmetry.isSelected())

**return** ***NO\_SYMMETRY***;

**else** **if** (twoWay.isSelected())

**return** ***SYMMETRY\_2***;

**else** **if** (fourWay.isSelected())

**return** ***SYMMETRY\_4***;

**else**

**return** ***SYMMETRY\_8***;

}

**private** **void** drawFigure(Graphics g, **int** shape, **int** x1, **int** y1, **int** x2, **int** y2) {

// This method is called to do ALL drawing in this applet!

// Draws a shape in the graphics context g.

// The shape parameter tells what kind of shape to draw. This

// can be LINE, RECT, OVAL, ROUNTRECT, FILLED\_RECT,

// FILLED\_OVAL, or FILLED\_ROUNDRECT. (Note that a CURVE is

// drawn by drawing multiple LINES, so the shape parameter is

// never equal to CURVE.) For a LINE, a line is drawn from

// the point (x1,y1) to (x2,y2). For other shapes, the

// points (x1,y1) and (x2,y2) give two corners of the shape

// (or of a rectangle that contains the shape).

**if** (shape == ***LINE***) {

// For a line, just draw the line between the two points.

g.drawLine(x1,y1,x2,y2);

**return**;

}

**int** x, y; // Top left corner of rectangle that contains the figure.

**int** w, h; // Width and height of rectangle that contains the figure.

**if** (x1 >= x2) { // x2 is left edge

x = x2;

w = x1 - x2;

}

**else** { // x1 is left edge

x = x1;

w = x2 - x1;

}

**if** (y1 >= y2) { // y2 is top edge

y = y2;

h = y1 - y2;

}

**else** { // y1 is top edge.

y = y1;

h = y2 - y1;

}

**switch** (shape) { // Draw the appropriate figure.

**case** ***RECT***:

g.drawRect(x, y, w, h);

**break**;

**case** ***OVAL***:

g.drawOval(x, y, w, h);

**break**;

**case** ***ROUNDRECT***:

g.drawRoundRect(x, y, w, h, 20, 20);

**break**;

**case** ***FILLED\_RECT***:

g.fillRect(x, y, w, h);

**break**;

**case** ***FILLED\_OVAL***:

g.fillOval(x, y, w, h);

**break**;

**case** ***FILLED\_ROUNDRECT***:

g.fillRoundRect(x, y, w, h, 20, 20);

**break**;

}

}

**private** **void** putMultiFigure(Graphics g, **int** shape, **int** x1, **int** y1, **int** x2, **int** y2) {

// Draws the shape and possibly some of its reflections.

// The reflections that are drawn depend on the selected

// item in symmetryChoice. The shapes are drawn by calling

// the drawFigure method.

**int** width = getWidth();

**int** height = getHeight();

drawFigure(g,shape,x1,y1,x2,y2); // Draw the basic figure

**if** (symmetry >= ***SYMMETRY\_2***) { // Draw the horizontal reflection.

drawFigure(g, shape, width - x1, y1, width - x2, y2);

}

**if** (symmetry >= ***SYMMETRY\_4***) { // Draw the two vertical reflections.

drawFigure(g, shape, x1, height - y1, x2, height - y2);

drawFigure(g, shape, width - x1, height - y1, width - x2, height - y2);

}

**if** (symmetry == ***SYMMETRY\_8***) { // Draw the four diagonal reflections.

**int** a1 = (**int**)( ((**double**)y1 / height) \* width );

**int** b1 = (**int**)( ((**double**)x1 / width) \* height );

**int** a2 = (**int**)( ((**double**)y2 / height) \* width );

**int** b2 = (**int**)( ((**double**)x2 / width) \* height );

drawFigure(g, shape, a1, b1, a2, b2);

drawFigure(g, shape, width - a1, b1, width - a2, b2);

drawFigure(g, shape, a1, height - b1, a2, height - b2);

drawFigure(g, shape, width - a1, height - b1, width - a2, height - b2);

}

}

**private** **void** repaintRect(**int** x1, **int** y1, **int** x2, **int** y2) {

// Call repaint on a rectangle that contains the points (x1,y1)

// and (x2,y2). (Add a 1-pixel border along right and bottom

// edges to allow for the pen overhang when drawing a line.)

**int** x, y; // top left corner of rectangle that contains the figure

**int** w, h; // width and height of rectangle that contains the figure

**if** (x2 >= x1) { // x1 is left edge

x = x1;

w = x2 - x1;

}

**else** { // x2 is left edge

x = x2;

w = x1 - x2;

}

**if** (y2 >= y1) { // y1 is top edge

y = y1;

h = y2 - y1;

}

**else** { // y2 is top edge.

y = y2;

h = y1 - y2;

}

repaint(x,y,w+1,h+1);

}

**private** **void** repaintMultiRect(**int** x1, **int** y1, **int** x2, **int** y2) {

// Call repaint on a rectangle that contains the points (x1,y1)

// and (x2,y2). Also call repaint on reflections of this

// rectangle, depending on the type of symmetry. The

// rects are repainted by calling repaintRect().

**int** width = getWidth();

**int** height = getHeight();

repaintRect(x1,y1,x2,y2); // repaint the original rect

**if** (symmetry >= ***SYMMETRY\_2***) { // repaint the horizontal reflection.

repaintRect(width - x1, y1, width - x2, y2);

}

**if** (symmetry >= ***SYMMETRY\_4***) { // repaint the two vertical reflections.

repaintRect(x1, height - y1, x2, height - y2);

repaintRect(width - x1, height - y1, width - x2, height - y2);

}

**if** (symmetry == ***SYMMETRY\_8***) { // repaint the four diagonal reflections.

**int** a1 = (**int**)( ((**double**)y1 / height) \* width );

**int** b1 = (**int**)( ((**double**)x1 / width) \* height );

**int** a2 = (**int**)( ((**double**)y2 / height) \* width );

**int** b2 = (**int**)( ((**double**)x2 / width) \* height );

repaintRect(a1, b1, a2, b2);

repaintRect(width - a1, b1, width - a2, b2);

repaintRect(a1, height - b1, a2, height - b2);

repaintRect(width - a1, height - b1, width - a2, height - b2);

}

}

**private** **void** checkOSI() {

// This method is responsible for creating the off-screen image.

// It should be called before using the OSI. It will make a new OSI if

// the size of the panel changes.

**if** (OSI == **null** || widthOfOSI != getSize().width || heightOfOSI != getSize().height) {

// Create the OSI, or make a new one if panel size has changed.

OSI = **null**; // (If OSI already exists, this frees up the memory.)

undoBuffer = **null**; // (Free memory.)

widthOfOSI = getWidth();

heightOfOSI = getHeight();

OSI = createImage(widthOfOSI,heightOfOSI);

Graphics OSG = OSI.getGraphics(); // Graphics context for drawing to OSI.

OSG.setColor(getBackground());

OSG.fillRect(0, 0, widthOfOSI, heightOfOSI);

OSG.dispose();

undoBuffer = createImage(widthOfOSI,heightOfOSI);

OSG = undoBuffer.getGraphics(); // Graphics context for drawing to undoBuffer

OSG.setColor(getBackground());

OSG.fillRect(0, 0, widthOfOSI, heightOfOSI);

OSG.dispose();

}

}

**public** **void** paintComponent(Graphics g) {

// Copy the off-screen image to the screen,

// after checking to make sure it exists. Then,

// if a shape other than CURVE is being drawn,

// draw it on top of the image from the OSI.

checkOSI();

g.drawImage(OSI, 0, 0, **this**);

**if** (dragging && figure != ***CURVE***) {

g.setColor(dragColor);

putMultiFigure(g,figure,startX,startY,mouseX,mouseY);

}

}

**public** **void** actionPerformed(ActionEvent evt) {

// Respond when the user selects an item from the "Control" menu.

String command = evt.getActionCommand();

checkOSI();

**if** (command.equals("Fill with Black"))

clear(Color.***black***);

**else** **if** (command.equals("Fill with Red"))

clear(Color.***red***);

**else** **if** (command.equals("Fill with Green"))

clear(Color.***green***);

**else** **if** (command.equals("Fill with Blue"))

clear(Color.***blue***);

**else** **if** (command.equals("Fill with Cyan"))

clear(Color.***cyan***);

**else** **if** (command.equals("Fill with Magenta"))

clear(Color.***magenta***);

**else** **if** (command.equals("Fill with Yellow"))

clear(Color.***yellow***);

**else** **if** (command.equals("Fill with White"))

clear(Color.***white***);

**else** **if** (command.equals("Fill with Custom"))

clear(customColor);

**else** **if** (command.equals("Set Custom Color...")) {

Color c = JColorChooser.*showDialog*(**this**,"Select Custom Color",customColor);

**if** (c != **null**) {

// Change the custom color and select it for use as

// the drawing color.

customColor = c;

custom.setSelected(**true**);

}

}

**else** **if** (command.equals("Clear")) {

// Clear to current background color.

Graphics g = OSI.getGraphics();

g.setColor(getBackground());

g.fillRect(0,0,getSize().width,getSize().height);

g.dispose();

repaint();

}

**else** **if** (command.equals("Undo")) {

// Undo the most recent drawing operation

// by swapping OSI with undoBuffer.

Image temp = OSI;

OSI = undoBuffer;

undoBuffer = temp;

repaint();

}

**else** **if** (command.equals("Quit")) {

// Close the window and exit. Note: The

// exit command will cause an error when

// the frame is opened from an applet.

// An applet should set the frame's standAlone

// variable to false after creating the frame.

dispose();

**if** (standAlone)

System.*exit*(0);

}

}

**private** **void** clear(Color background) {

// Fill with the specified color. If the

// color is equal to the current drawing color, then

// the current drawing color is changed, so that

// drawing operations will not be invisible.

setBackground(background);

**if** (background.equals(getSelectedColor())) {

**if** (background.equals(Color.***black***))

white.setSelected(**true**); // On a black background, draw in white.

**else**

black.setSelected(**true**); // On other backgrounds, use black.

}

Graphics g = OSI.getGraphics();

g.setColor(getBackground());

g.fillRect(0,0,getSize().width,getSize().height);

g.dispose();

repaint();

}

**public** **void** mousePressed(MouseEvent evt) {

// This is called when the user presses the mouse on the

// panel. This begins a draw operation in which the user

// sketches a curve or draws a shape. (Note that curves

// are handled differently from other shapes. For CURVE,

// a new segment of the curve is drawn each time the user

// moves the mouse. For the other shapes, a "rubber band

// cursor" is used. That is, the figure is drawn between

// the starting point and the current mouse location.)

**if** (dragging == **true**) // Ignore mouse presses that occur

**return**; // when user is already drawing a curve.

// (This can happen if the user presses

// two mouse buttons at the same time.)

prevX = startX = evt.getX(); // Save mouse coordinates.

prevY = startY = evt.getY();

figure = getSelectedShape(); // Get data from menus for drawing.

symmetry = getSelectedSymmetry();

dragColor = getSelectedColor();

checkOSI();

Graphics undoGraphics = undoBuffer.getGraphics();

undoGraphics.drawImage(OSI,0,0,**null**); // Remember the current image,

// for "Undo" operations,

// before changing the image.

undoGraphics.dispose();

dragGraphics = OSI.getGraphics();

dragGraphics.setColor(dragColor);

dragging = **true**; // Start drawing.

} // end mousePressed()

**public** **void** mouseReleased(MouseEvent evt) {

// Called whenever the user releases the mouse button.

// If the user was drawing a shape, we make the shape

// permanent by drawing it to the off-screen image.

**if** (dragging == **false**)

**return**; // Nothing to do because the user isn't drawing.

dragging = **false**;

mouseX = evt.getX();

mouseY = evt.getY();

**if** (figure == ***CURVE***) {

// A CURVE is drawn as a series of LINEs

putMultiFigure(dragGraphics,***LINE***,prevX,prevY,mouseX,mouseY);

repaintMultiRect(prevX,prevY,mouseX,mouseY);

}

**else** **if** (figure == ***LINE***) {

repaintMultiRect(startX,startY,prevX,prevY);

**if** (mouseX != startX || mouseY != startY) {

// Draw the line only if it has non-zero length.

putMultiFigure(dragGraphics,figure,startX,startY,mouseX,mouseY);

repaintMultiRect(startX,startY,mouseX,mouseY);

}

}

**else** {

repaintMultiRect(startX,startY,prevX,prevY);

**if** (mouseX != startX && mouseY != startY) {

// Draw the shape only if both its height

// and width are both non-zero.

putMultiFigure(dragGraphics,figure,startX,startY,mouseX,mouseY);

repaintMultiRect(startX,startY,mouseX,mouseY);

}

}

dragGraphics.dispose();

dragGraphics = **null**;

}

**public** **void** mouseDragged(MouseEvent evt) {

// Called whenever the user moves the mouse while a mouse button

// is down. If the user is drawing a curve, draw a segment of

// the curve on the off-screen image, and repaint the part

// of the panel that contains the new line segment. Otherwise,

// just call repaint and let paintComponent() draw the shape on

// top of the picture in the off-screen image.

**if** (dragging == **false**)

**return**; // Nothing to do because the user isn't drawing.

mouseX = evt.getX(); // x-coordinate of mouse.

mouseY = evt.getY(); // y=coordinate of mouse.

**if** (figure == ***CURVE***) {

// A CURVE is drawn as a series of LINEs.

putMultiFigure(dragGraphics,***LINE***,prevX,prevY,mouseX,mouseY);

repaintMultiRect(prevX,prevY,mouseX,mouseY);

}

**else** {

// Repaint two rectangles: The one that contains the previous

// version of the figure, and the one that will contain the

// new version. The first repaint is necessary to restore

// the picture from the off-screen image in that rectangle.

repaintMultiRect(startX,startY,prevX,prevY);

repaintMultiRect(startX,startY,mouseX,mouseY);

}

prevX = mouseX; // Save coords for the next call to mouseDragged or mouseReleased.

prevY = mouseY;

} // end mouseDragged.

**public** **void** mouseEntered(MouseEvent evt) { } // Some empty routines.

**public** **void** mouseExited(MouseEvent evt) { } // (Required by the MouseListener

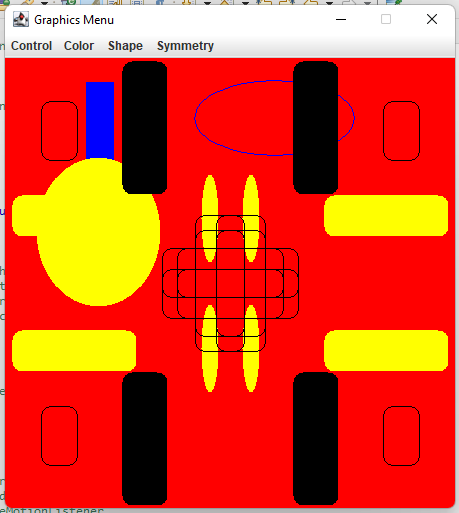
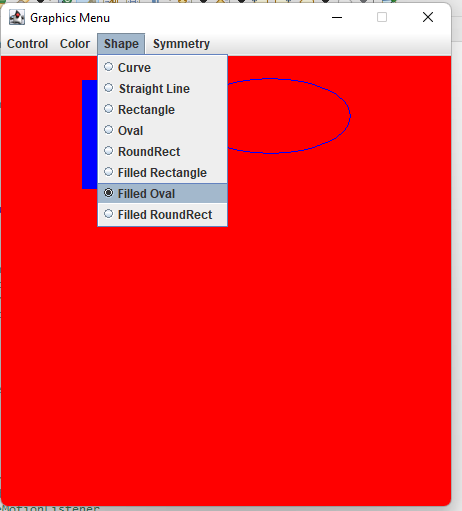
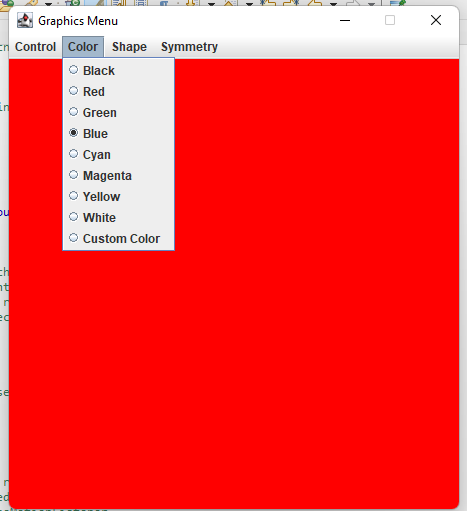
**public** **void** mouseClicked(MouseEvent evt) { } // and MouseMotionListener

**public** **void** mouseMoved(MouseEvent evt) { } // interfaces).

} // end nested class Display

} // end class Menu1

**OUTPUT:**

****

**EXPERIMENT – 18,19**

**AIM:** . Write a Java program to implement JTabbedPane , JTable and JTree. [CO4]

**DESCRIPTION:**

Class TreeSelectionEvent java.lang.Object java.util.EventObject javax.swing.event.TreeSelectionEvent

An event that characterizes a change in the current selection. The change is based on any number of paths. TreeSelectionListeners will generally query the source of the event for the new selected status of each potentially changed row.

To detect when the user selects a node in a tree, you need to register a tree selection listener. Here is an example, taken from the TreeDemo example discussed in Responding to Node Selection, of detecting node selection in a tree that can have at most one node selected at a time

The JTabbedPane class is used to switch between a group of components by clicking on a tab with a given title or icon. It inherits JComponent class.

**SYNTAX:**

public class TreeSelectionEvent extends EventObject

public class JTabbedPane extends JComponent implements Serializable, Accessible, SwingConstants

JTabbedPane() --Creates an empty TabbedPane with a default tab placement of JTabbedPane.Top.

JTabbedPane(int tabPlacement) --Creates an empty TabbedPane with a specified tab placement.

JTabbedPane(int tabPlacement, int tabLayoutPolicy) --Creates an empty TabbedPane with a specified tab placement and tab layout policy.

**PROGRAM:**

//Demonstrate JTabbedPane.

**import** javax.swing.\*;

**import** javax.swing.event.TreeSelectionEvent;

**import** javax.swing.event.TreeSelectionListener;

**import** javax.swing.tree.DefaultMutableTreeNode;

**import** java.awt.\*;

**public** **class** JTabbedPaneDemo {

**public** JTabbedPaneDemo() {

// Set up the JFrame.

JFrame jfrm = **new** JFrame("JTabbedPaneDemo");

jfrm.setLayout(**new** FlowLayout());

jfrm.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

jfrm.setSize(1000, 1000);

// Create the tabbed pane.

JTabbedPane jtp = **new** JTabbedPane();

jtp.addTab("Tree", **new** TreePanel());

jtp.addTab("Table", **new** TablePanel());

jfrm.add(jtp);

// Display the frame.

jfrm.setVisible(**true**);

}

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

{

// Create the frame on the event dispatching thread.

SwingUtilities.*invokeLater*(**new** Runnable()

{

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

{

**new** JTabbedPaneDemo();

}

}

);

}

}

//Make the panels that will be added to the tabbed pane.

**class** TreePanel **extends** JPanel {

**public** TreePanel() {

// Create top node of tree.

DefaultMutableTreeNode top = **new** DefaultMutableTreeNode("Options");

// Create subtree of "A".

DefaultMutableTreeNode a = **new** DefaultMutableTreeNode("A");

top.add(a);

DefaultMutableTreeNode a1 = **new** DefaultMutableTreeNode("A1");

a.add(a1);

DefaultMutableTreeNode a2 = **new** DefaultMutableTreeNode("A2");

a.add(a2);

// Create subtree of "B".

DefaultMutableTreeNode b = **new** DefaultMutableTreeNode("B");

top.add(b);

DefaultMutableTreeNode b1 = **new** DefaultMutableTreeNode("B1");

b.add(b1);

DefaultMutableTreeNode b2 = **new** DefaultMutableTreeNode("B2");

b.add(b2);

DefaultMutableTreeNode b3 = **new** DefaultMutableTreeNode("B3");

b.add(b3);

// Create the tree.

JTree tree = **new** JTree(top);

// Add the tree to a scroll pane.

JScrollPane jsp = **new** JScrollPane(tree);

// Add the scroll pane to the content pane.

add(jsp);

// Add the label to the content pane.

JLabel jlab = **new** JLabel();

add(jlab, BorderLayout.***SOUTH***);

// Handle tree selection events.

tree.addTreeSelectionListener(**new** TreeSelectionListener() {

**public** **void** valueChanged(TreeSelectionEvent tse) {

jlab.setText("Selection is " + tse.getPath());

}

});

}

}

**class** TablePanel **extends** JPanel {

**public** TablePanel() {

// Initialize column headings.

String[] colHeads = { "Name", "Extension", "ID#" };

// Initialize data.

Object[][] data = {

{ "Gail", "4567", "865" },

{ "Ken", "7566", "555" },

{ "Viviane", "5634", "587" },

{ "Melanie", "7345", "922" },

{ "Anne", "1237", "333" },

{ "John", "5656", "314" },

{ "Matt", "5672", "217" },

{ "Claire", "6741", "444" },

{ "Erwin", "9023", "519" },

{ "Ellen", "1134", "532" },

{ "Jennifer", "5689", "112" },

{ "Ed", "9030", "133" },

{ "Helen", "6751", "145" }

};

// Create the table.

JTable table = **new** JTable(data, colHeads);

// Add the table to a scroll pane.

JScrollPane jsp = **new** JScrollPane(table);

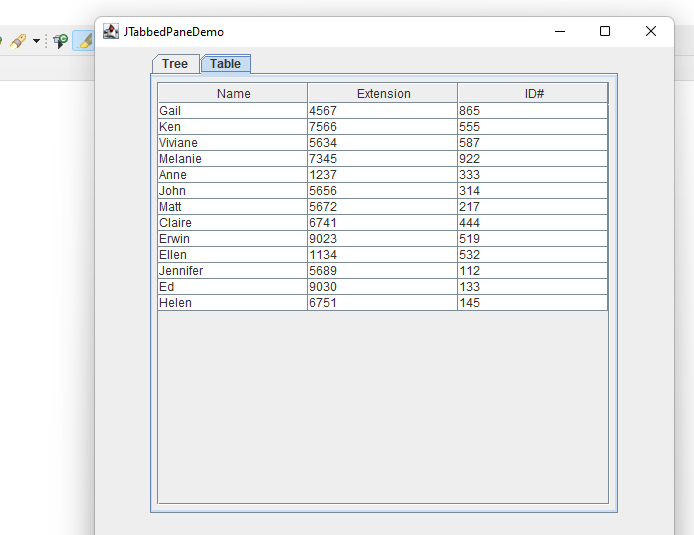
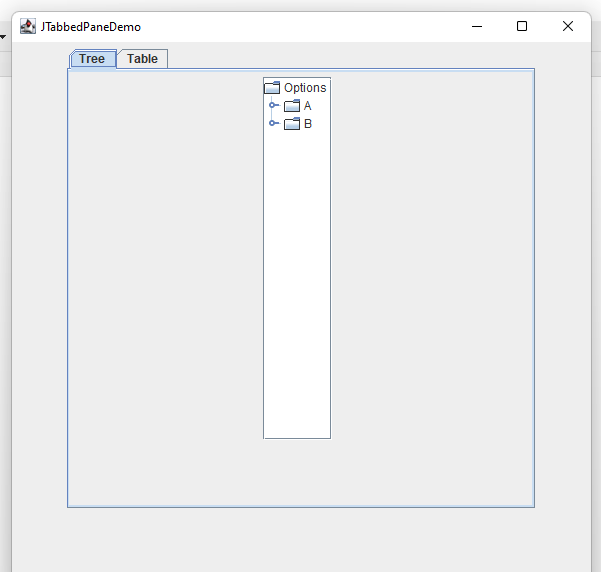
// Add the scroll pane to the content pane.

add(jsp);

}

}

**OUTPUT:**

****

**EXPERIMENT – 20**

**AIM:** . Write a Java Program that implements a simple client/server application. The client sends data to a server. The server receives the data, uses it to produce a result and then sends the result back to the client. The client displays the result on the console. For ex: The data sent from the client is the radius of a circle and the result produced by the server is the area of the circle. [CO5]

**DESCRIPTION:**

Socket Programming Socket programming is used to make a connection between two nodes namely server and client on a network. By using this we can create a two-way connection between multiple nodes.

Logic

1. Firstly we will use sockets to request a connection between the nodes by passing the port number and keeping the host as localhost.
2. Once the server accepts the connection, we will implement a Runnable interface and override its methods to display the messages between the nodes.
3. We have used ExecutorService to create a thread pool and to connect multiple clients with the server at a time.
4. We will be using Threads to handle multiple messages from clients at a time.
5. Once the message is sent by any node our program will stop.

**SYNTAX:**

Public Socket accept() // returns socket and establish connection between server and client Public synchronized void close() //closes the sever socket

**PROGRAM:**

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

**import** java.net.\*;

**public** **class** Client

{

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

{

Socket s=**new** Socket(InetAddress.*getLocalHost*(),1064);

BufferedReader br;

PrintStream ps;

String str;

System.***out***.println("Enter Radius :");

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

ps=**new** PrintStream(s.getOutputStream());

ps.println(br.readLine());

br=**new** BufferedReader(**new** InputStreamReader(s.getInputStream()));

str=br.readLine();

System.***out***.println("Area of the circle is : "+str);

br.close();

ps.close();

}

}

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

**import** java.net.\*;

**public** **class** Server

{

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

{

**try**

{

ServerSocket ss=**new** ServerSocket(1064);

System.***out***.println("Waiting for Client Request");

Socket s=ss.accept();

BufferedReader br;

PrintStream ps;

String str;

br=**new** BufferedReader(**new** InputStreamReader(s.getInputStream()));

str=br.readLine();

System.***out***.println("Received radius");

**double** r=Double.*parseDouble*(str);

**double** area=3.14\*r\*r;

ps=**new** PrintStream(s.getOutputStream());

ps.println(String.*valueOf*(area));

br.close();

ps.close();

s.close();

ss.close();

}

**catch**(Exception e)

{

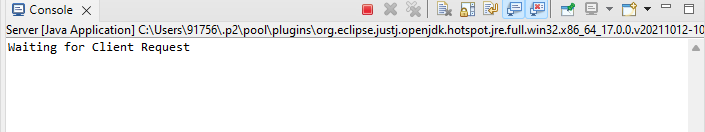
System.***out***.println(e);

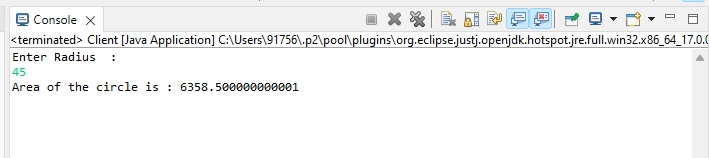
}

}

}

**OUTPUT:**

****

****

**Additional Experiment – 1**

**AIM:** Demonstrate the usage of methods in String class.

**DESCRIPTION:**

Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

The java.lang.String class implements Serializable, Comparable and CharSequence interfaces.

No. Method Description

1. char charAt(int index) It returns char value for the particular

index

1. int length() It returns string length
2. static String format(String format, Object... args) It returns a formatted string.
3. String substring(int beginIndex) It returns substring for given begin index.
4. String substring(int beginIndex, int endIndex) It returns substring for given begin index

and end index.

1. String concat(String str) It concatenates the specified string.
2. String replace(char old, char new) It replaces all occurrences of the

specified char value.

1. String replace(CharSequence old, CharSequence new) It replaces all occurrences of the

specified CharSequence.

1. static String equalsIgnoreCase(String another) It compares another string. It doesn't

check case.

|  |  |  |
| --- | --- | --- |
| 10 | String[] split(String regex) | It returns a split string matching regex. |
| 11 | int indexOf(int ch) | It returns the specified char value index. |
| 12 | int indexOf(String substring) | It returns the specified substring index. |
| 13 | String toLowerCase() | It returns a string in lowercase. |
| 14 | String toUpperCase() | It returns a string in uppercase. | |
| 15 | String trim() | It removes beginning and ending spaces | |

of this string.

**SYNTAX:**

String s=new String("Welcome"); //creates two objects and one reference variable

**PROGRAM:**

**class** StringAdditional {

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

//String Length

**char** chars[]= {'c','s','e'};

String s=**new** String(chars);

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

//String Literals

**char** chars1[]= {'j','a','v','a'};

String s1=**new** String(chars1);

System.***out***.println(s1);

//String concatenation

String line="Welcome "+"to "+"string "

+"functions";

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

String a="Strength "+ 6+ 4;

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

String b="Strength "+ (6+ 4);

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

//Character Extraction

**char** x;

x="Bharath Eswar".charAt(8);

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

//getChars()

String q="string functions in java";

**int** start=7;

**int** end=9;

**char** buf[]=**new** **char**[end-start];

q.getChars(start,end,buf,0);

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

//get bytes

**byte** k[]=q.getBytes();

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

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

//tochar array

**char** c[]=q.toCharArray();

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

//equals() and equalsIgnoreclass()

String o="java lab";

System.***out***.println(o.equals("JAVA LAB"));

System.***out***.println(o.equalsIgnoreCase("JAVA LAB"));

//region matches

System.***out***.println("matches?"+q.regionMatches(0, o, 0, 3));

//starts with and ends with

System.***out***.println("starts with java?"+o.startsWith("java"));

System.***out***.println("endswith java?"+q.endsWith("java"));

//index position

System.***out***.println("index of j in java lab"+o.indexOf('j'));

System.***out***.println("last occurence of a in java lab"+o.lastIndexOf('a'));

//replace

System.***out***.println("Replacing l with k in hello:"+"hello".replace('l','k'));

//upper and lower case

System.***out***.println("converting 'eswar' to uppercase:"+"eswar".toUpperCase());

System.***out***.println("converting 'komali' to lower case:"+"komali".toUpperCase());

//additional string methods

System.***out***.println("the unicode value of s in 'ramesh'is:"+"ramesh".codePointAt(1));

System.***out***.println("'java lab' contains la?:"+"java lab".contains("la"));

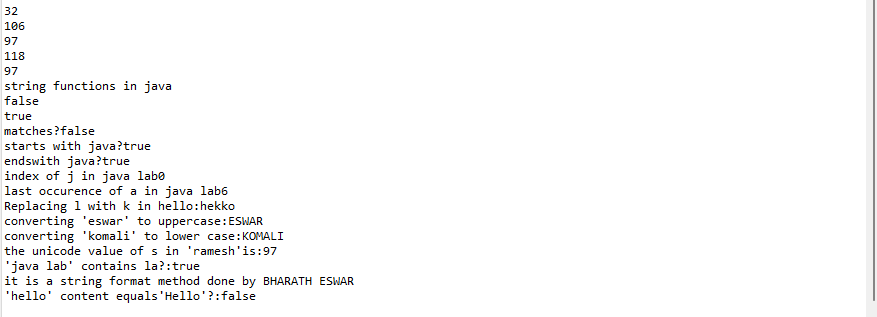
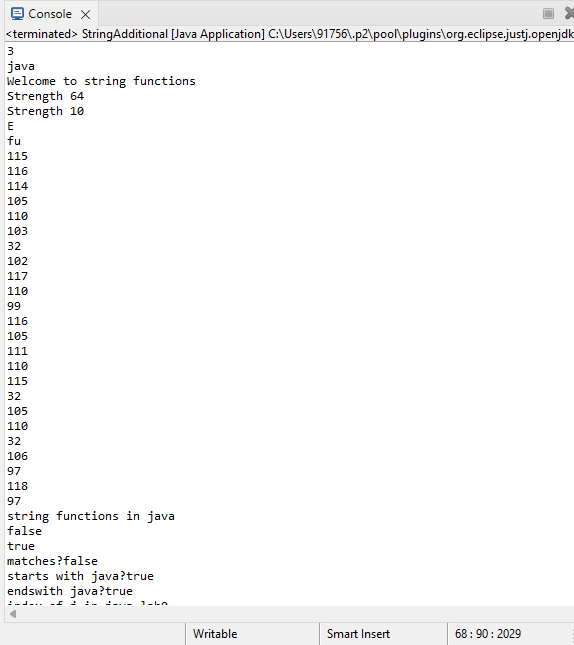
System.***out***.println(String.*format*("it is a string format method done by %s","BHARATH ESWAR"));

System.***out***.println("'hello' content equals'Hello'?:"+"hello".contentEquals("Hello"));

}

}

**OUTPUT:**

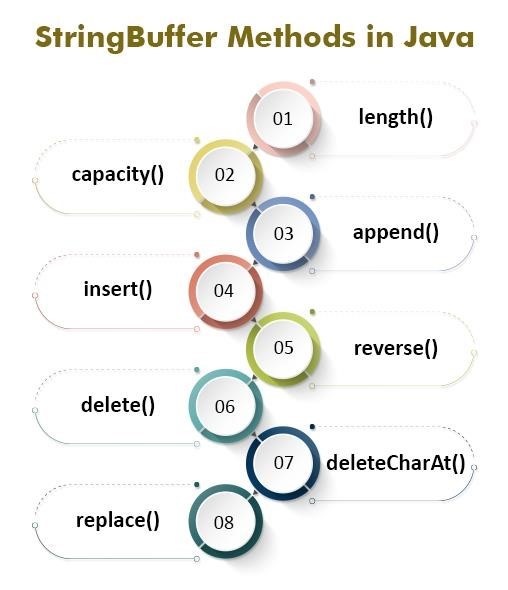
****

**Additional Experiment – 2**

**AIM**: Write a java program to implement the functions in StringBuffer class.

**DESCRIPTION:**

Java StringBuffer class is used to create mutable (modifiable) String objects. The StringBuffer class in Java is the same as String class except it is mutable i.e. it can be changed.



**SYNTAX:**

StringBuffer()

StringBuffer(String str) StringBuffer(int capacity)

**PROGRAM:**

**public** **class** StringBufferAdditional1 {

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

{

StringBuffer s = **new** StringBuffer ("Hello");

StringBuffer p = **new** StringBuffer ();

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

// length and capacity

System.***out***.println ("length of the string 'hello':" + s.length ());

System.***out***.println ("capacity of the string 'hello':" + s.capacity ());

System.***out***.println ("capacity of a an empty string:" + p.capacity ());

// ensure capacity and set length

p.ensureCapacity (100);

System.***out***.println ("capicity:" + p.capacity ());

s.setLength (3);

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

s.setCharAt (1, 'g');

System.***out***.println ("replacing a character 'e' with 'g' in 'Hello'" + s);

// append

StringBuffer k = **new** StringBuffer ("Java");

System.***out***.println ("appending 'record' to 'Java':" + k.append ("record"));

// insert

System.***out***.println ("inserting 'lab' in between 'java record':" + k.insert (4, "lab"));

// reverse

StringBuffer g = **new** StringBuffer ("Hello");

System.***out***.println ("reverse order of 'Hello':" + g.reverse ());

// delete

System.***out***.println ("deleting a character:" + g.deleteCharAt (1));

// replace

StringBuffer z = **new** StringBuffer ("bharath");

System.***out***.println ("index position:" + z.indexOf ("i"));

System.***out***.println ("last index position:" + z.lastIndexOf ("i"));

StringBuffer q = **new** StringBuffer ("hello");

System.***out***.println ("before trimming");

System.***out***.println ("capacity is:" + q.capacity ());

q.trimToSize ();

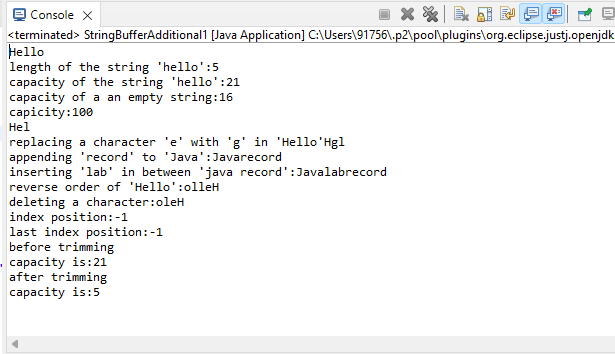
System.***out***.println ("after trimming");

System.***out***.println ("capacity is:" + q.capacity ());

}

}

**OUTPUT:**

****

**Additional Experiment – 3**

**AIM**: Write a java program to implement stacks.

**DESCRIPTION:**

The stack is a linear data structure that is used to store the collection of objects. It is based on LastIn-First-Out (LIFO). Java collection framework provides many interfaces and classes to store the collection of objects. One of them is the Stack class that provides different operations such as push, pop, search, etc.

**SYNTAX:**

empty() The method checks the stack is empty or not.

pop() The method removes an element from the top of the stack and returns the same element as the value of that function. peek() The method looks at the top element of the stack without removing it.

**PROGRAM:**

import java.util.Scanner;  
class StackTest  
{  
  
int top = -1, n = 1000;  
Scanner sc = new Scanner (System.in);

int stack [] = new int [n];  
  
  
void push ()  
{  
System.out.println ("enter how many elements you want to push");  
int k = sc.nextInt ();  
System.out.println ("enter the elemnts");  
  
for (int i = 0; i <k; i ++)  
{  
if (isFull ())  
{  
System.out.println ("Stack is full");  
  
  
}  
else {  
int e [] = new int [k];  
e [i] = sc.nextInt ();  
stack [++ top] = e [i];  
}  
  
}  
  
}  
int pop ()  
{  
if (isEmpty ())  
{  
System.out.println ("Stack is empty");  
return 0;  
  
}  
else {  
return stack [top--];  
}  
}

boolean isFull()  
{  
return ((top==n-1)?true:false);  
}  
boolean isEmpty() {

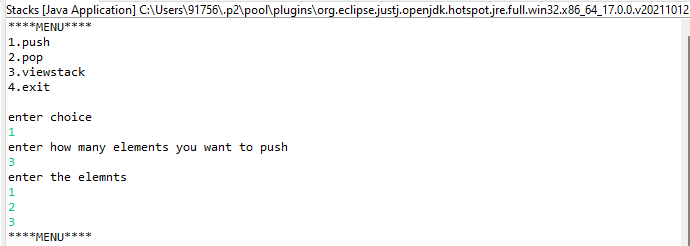
return ((top==-1)?true:false);  
}  
void display()  
{  
if(isEmpty())  
{  
System.out.println("there is no content");  
return ;  
}  
System.out.println("contents are");  
for(int j=top;j>=0;j--)  
{  
System.out.println(stack[j]);  
}  
}  
  
}  
public class Stacks {

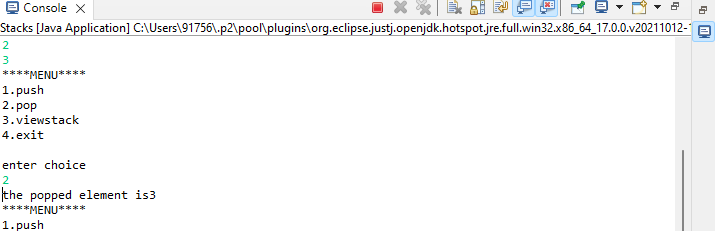
public static void main(String[] args) {  
// TODO Auto-generated method stub  
StackTest k=new StackTest();  
  
Scanner p=new Scanner(System.in);  
  
first:while(true)  
{  
System.out.println("\*\*\*\*MENU\*\*\*\*");  
System.out.println("1.push\n"+"2.pop\n"+"3.viewstack\n"+"4.exit\n");  
System.out.println("enter choice");  
int x=p.nextInt();  
switch(x)  
{  
case 1:  
k.push();  
break;  
case 2:  
System.out.println("the popped element is"+k.pop());  
break;  
case 3:  
k.display();  
break;  
case 4:  
System.out.println("Have a nice day");  
  
break first;  
  
  
}  
}

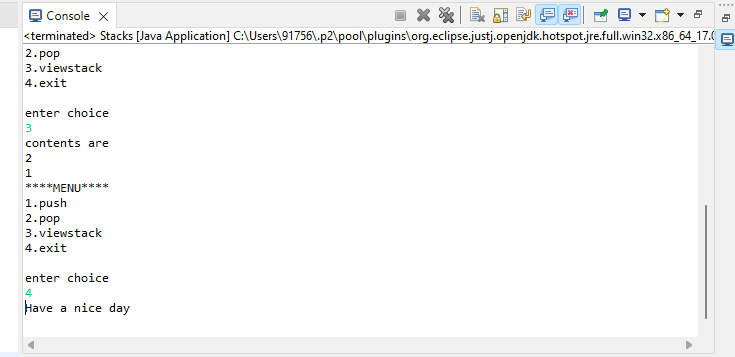
}

}

**OUTPUT:**

****

****

****

**Additional Experiment – 4**

**AIM:** Write a java program to implement queues.

**DESCRIPTION:**

The Queue interface present in the java.util package and extends the Collection interface is used to hold the elements about to be processed in FIFO(First In First Out) order. It is an ordered list of objects with its use limited to insert elements at the end of the list and deleting elements from the start of the list, (i.e.), it follows the FIFO or the First-In-First-Out principle.

**SYNTAX:**

|  |  |
| --- | --- |
| Method | Description |
| add(int index, element) | This method is used to add an element at a particular index in the list. When a single parameter is passed, it simply adds the element at the end of the list. |
| addAll(int index, Collection collection) | This method is used to add all the elements in the given collection to the list. When a single parameter is passed, it adds all the elements of the given collection at the end of the list. |
| size() | This method is used to return the size of the list. |
| clear() | This method is used to remove all the elements in the list. However, the reference of the list created is still stored. |
| remove(int index) | This method removes an element from the specified index. It shifts subsequent elements(if any) to left and decreases their indexes by 1. |
| remove(element) | This method is used to remove the first occurrence of the given element in the list. |
| get(int index) | This method returns elements at the specified index. |

**PROGRAM:**

import java.util.Scanner;  
class QueueTest  
{  
  
int front=0,rear=0, n=1000;  
Scanner sc=new Scanner(System.in);

int queue[]=new int[n];  
  
  
void enqueue()  
{  
System.out.println("enter how many elements you want to enqueue");  
int k= sc.nextInt();  
System.out.println("enter the elemnts");  
  
for (int i=0;i<k;i++)  
{  
if(isFull())  
{  
System.out.println("Stack is full");  
  
  
}  
else {  
int e[]=new int[k];  
e[i]=sc.nextInt();  
queue[rear++]=e[i];  
}  
  
}  
  
}  
int dequeue()  
{  
if(isEmpty())  
{  
System.out.println("Stack is empty");  
return 0;  
  
}  
else {  
return queue[front++];  
}  
}

boolean isFull()  
{  
return ((rear==n)?true:false);  
}  
boolean isEmpty() {

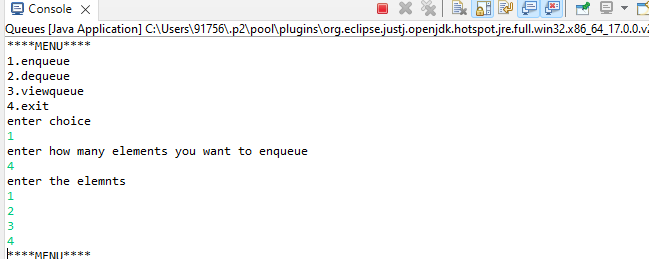
return ((rear==front)?true:false);  
}  
void display()  
{  
if(isEmpty())  
{  
System.out.println("there is no content");  
return ;  
}  
System.out.println("contents are");  
for(int j=front;j<rear;j++)  
{  
System.out.println(queue[j]);  
}  
}  
  
}  
public class Queues {

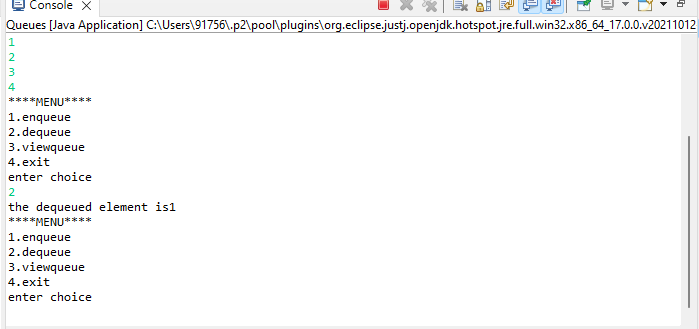
public static void main(String[] args) {  
// TODO Auto-generated method stub  
QueueTest k=new QueueTest();  
  
Scanner p=new Scanner(System.in);  
  
first:while(true)  
{  
System.out.println("\*\*\*\*MENU\*\*\*\*");  
System.out.println("1.enqueue\n"+"2.dequeue\n"+"3.viewqueue\n"+"4.exit");  
System.out.println("enter choice");  
int x=p.nextInt();  
switch(x)  
{  
case 1:  
k.enqueue();  
break;  
case 2:  
System.out.println("the dequeued element is"+k.dequeue());  
break;  
case 3:  
k.display();  
break;  
case 4:  
System.out.println("Have a nice day");  
  
break first;  
  
  
}  
}

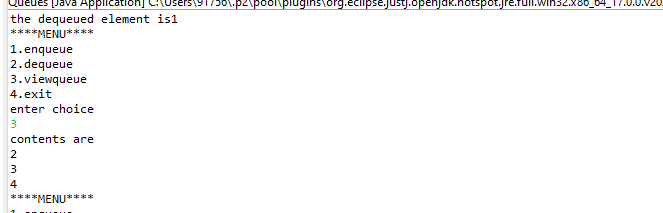
}

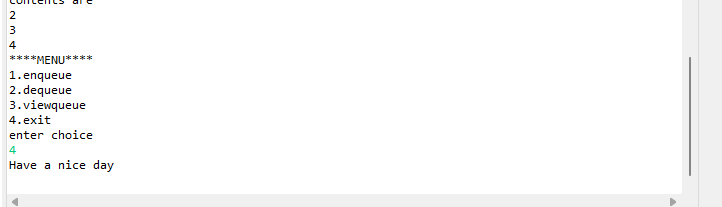
}

**OUTPUT:**

****

****

****

****

**Additional Experiment – 5**

**AIM:** Write a java program to demonstrate the usage of ByteStream classes.

**DESCRIPTION:**

ByteStream classes are used to read bytes from the input stream and write bytes to the output stream. In other words, we can say that ByteStream classes read/write the data of 8-bits. We can store video, audio, characters, etc., by using ByteStream classes. These classes are part of the java.io package.

The ByteStream classes are divided into two types of classes, i.e., InputStream and OutputStream. These classes are abstract and the super classes of all the Input/Output stream classes.

InputStream Class

The InputStream class provides methods to read bytes from a file, console or memory. It is an abstract class and can't be instantiated; however, various classes inherit the InputStream class and override its methods

**SYNTAX:**

SN Method Description

1. int read() ---This method returns an integer, an integral representation of the next available byte of the input. The integer -1 is returned once the end of the input is encountered.
2. int read (byte buffer []) ---This method is used to read the specified buffer length bytes from the input and returns the total number of bytes successfully read. It returns -1 once the end of the input is encountered.
3. int read (byte buffer [], int loc, int nBytes) ---This method is used to read the 'nBytes' bytes from the buffer starting at a specified location, 'loc'. It returns the total number of bytes successfully read from the input. It returns -1 once the end of the input is encountered.
4. int available () ---This method returns the number of bytes that are available to read.
5. Void mark(int nBytes) ---This method is used to mark the current position in the input stream until the specified nBytes are read.
6. void reset () ---This method is used to reset the input pointer to the previously set mark.
7. long skip (long nBytes) ---This method is used to skip the nBytes of the input stream and returns the total number of bytes that are skipped.
8. void close () ---This method is used to close the input source. If an attempt is made to read even after the closing, IOException is thrown by the method.

ByteArrayInputStream inputStream = new ByteArrayInputStream(content);

**PROGRAM:**

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

**public** **class** ByteStreamAdditional

{

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

{

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

String s="JAVA IS AN OBJECT ORIENTED LANGUAGE AND EASY TO LEARN";

**byte** b1[]=s.getBytes();

**for**(**int** i=0;i<b1.length;i++)

{

System.***out***.print((**char**)b1[i]);

}

//ByteArrayInputStream obj creation

ByteArrayInputStream bis=**new** ByteArrayInputStream(b1);

**int** c;

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

**while**((c=bis.read())!=-1)

{

**char** ch;

ch=(**char**)c;

System.***out***.print(Character.*toLowerCase*(ch));

}

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

//ByteArrayOutputStream obj creation

ByteArrayOutputStream bos=**new** ByteArrayOutputStream();

bos.write(b1);

String s1=bos.toString();

System.***out***.printf(s1);

DataOutputStream dos=**new** DataOutputStream(**new** FileOutputStream(

"C:\\Users\\91756\\OneDrive - Vasireddy Venkatadri Institute of Technology\\Desktop\\jv2.txt"));

dos.writeInt(32);

dos.writeChar('p');

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

DataInputStream dow=**new** DataInputStream(**new** FileInputStream(

"C:\\\\Users\\\\91756\\\\OneDrive - Vasireddy Venkatadri Institute of Technology\\\\Desktop\\\\jv2.txt"));

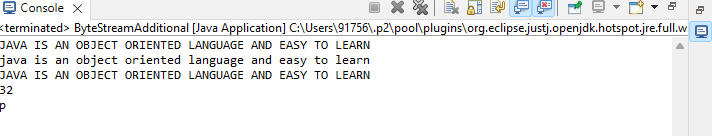
System.***out***.println(dow.readInt());

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

}

}

**OUTPUT:**

****

**Additional Experiment – 6**

**AIM:** Write a java program to demonstrate the usage of CharacterArrayStream classes.

**DESCRIPTION:** The java.io package provides CharacterStream classes to overcome the limitations of ByteStream classes, which can only handle the 8-bit bytes and is not compatible to work directly with the Unicode characters. CharacterStream classes are used to work with 16-bit Unicode characters. They can perform operations on characters, char arrays and Strings.

However, the CharacterStream classes are mainly used to read characters from the source and write them to the destination. For this purpose, the CharacterStream classes are divided into two types of classes, I.e., Reader class and Writer class.

**Reader Class**

Reader class is used to read the 16-bit characters from the input stream. However, it is an abstract class and can't be instantiated, but there are various subclasses that inherit the Reader class and override the methods of the Reader class. All methods of the Reader class throw an IOException.

**SYNTAX:**

int read() ---returns the integral representation of the next available character of input. It returns

-1 when end of file is encountered

int read (char buffer []) ----attempts to read buffer. length characters into the buffer and returns the total number of characters successfully read. It returns -I when end of file is encountered

**PROGRAM:**

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

**public** **class** CharacterStreamAdditional

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

//Creating FileReader object

File file = **new** File("C:\\\\Users\\\\91756\\\\OneDrive - Vasireddy Venkatadri Institute of Technology\\\\Desktop\\\\jv2.txt");

FileReader reader = **new** FileReader(file);

**char** chars[] = **new** **char**[(**int**) file.length()];

reader.read(chars);

File out = **new** File("C:\\\\Users\\\\91756\\\\OneDrive - Vasireddy Venkatadri Institute of Technology\\\\Desktop\\\\jv2.txt");

FileWriter writer = **new** FileWriter(out);

writer.write(chars);

System.***out***.println("Data successfully written in the specified file");

writer.close();

reader.close();

String s="Hiii. I Am Learning Java.";

**char** c[]=**new** **char**[s.length()];

s.getChars(0, s.length(), c, 0);

BufferedReader br=**new** BufferedReader(**new** CharArrayReader(c));

**int** x;

**while**((x=br.read())>=0)

System.***out***.print((**char**)x);

BufferedWriter bw=**new** BufferedWriter(**new** FileWriter("C:\\\\Users\\\\91756\\\\OneDrive - Vasireddy Venkatadri Institute of Technology\\\\Desktop\\\\jv2.txt"));

bw.write(s);

bw.close();

br.close();

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

String s1="VIVA VVIT 2021";

**char** c1[]=**new** **char**[s1.length()];

s.getChars(0,s1.length(),c1,0);

CharArrayWriter cw=**new** CharArrayWriter();

cw.write(c1);

String s2=cw.toString();

System.***out***.println("Data in stream is:");

CharArrayReader cr=**new** CharArrayReader(c1);

**int** x1;

**while**((x1=cr.read())>=0)

System.***out***.print((**char**)x1);

//System.out.println(s2);

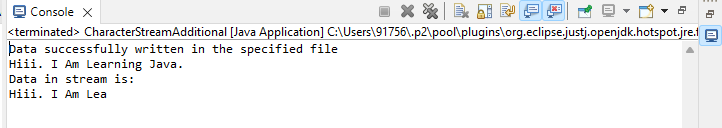
cw.close();

cr.close();

}

}

**OUTPUT:**

****

**Additional Experiment – 7**

**AIM:** Write a java program to demonstrate Serialization and Deserialization.

**DESCRIPTION:** Serialization is a mechanism of converting the state of an object into a byte stream. Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object.

The byte stream created is platform independent. So, the object serialized on one platform can be deserialized on a different platform.

To make a Java object serializable we implement the java.io.Serializable interface.

The ObjectOutputStream class contains writeObject() method for serializing an Object.

**SYNTAX:**

public final void writeObject(Object obj)

throws IOException

**PROGRAM:**

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

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.ObjectInputStream;

**import** java.io.ObjectOutputStream;

**class** DemoofSerialization **implements** java.io.Serializable

{

**public** **int** a;

**public** String b;

// Default constructor

**public** DemoofSerialization(**int** a, String b)

{

**this**.a = a;

**this**.b = b;

}

}

**public** **class** SerializeDeserialize

{

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

{

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

DemoofSerialization object = **new** DemoofSerialization(1, "Eswar");

String filename = "C:\\\\\\\\Users\\\\\\\\91756\\\\\\\\OneDrive - Vasireddy Venkatadri Institute of Technology\\\\\\\\Desktop\\\\\\\\jv2.txt";

// Serialization

**try**

{

//Saving of object in a file

FileOutputStream file = **new** FileOutputStream(filename);

ObjectOutputStream out = **new** ObjectOutputStream(file);

// Method for serialization of object

out.writeObject(object); out.close();

file.close();

System.***out***.println("Object has been serialized");

}

**catch**(IOException ex)

{

System.***out***.println("IOException is caught");

}

DemoofSerialization object1 = **null**;

//Deserialization

**try**

{

// Reading the object from a file

FileInputStream file = **new** FileInputStream(filename);

ObjectInputStream in = **new** ObjectInputStream(file);

// Method for deserialization of object

object1 = (DemoofSerialization)in.readObject();

in.close();

file.close();

System.***out***.println("Object has been deserialized ");

System.***out***.println("a = " + object1.a);

System.***out***.println("b = " + object1.b);

}

**catch**(IOException ex)

{

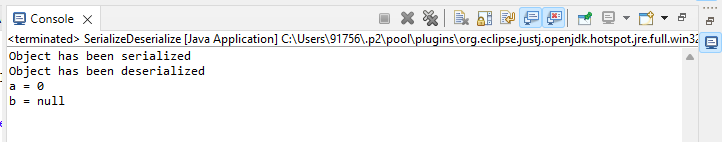
System.***out***.println("IOException is caught");

}

}//main close

}//SerializeDeserialize close

**OUTPUT:**

****