**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, analyzing, 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 analyze 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, analyze 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 Experiments**

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 : LEELA SIDDARDHA ANNAPUREDDY

Student Regd. No. : 20BQ1A05C0

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 | 11-14 |  |  |
| 2 | Experiment -2 | 12-10-2021 | 18-10-2021 | 15-17 |  |  |
| 3 | Experiment -3 | 26-10-2021 | 28-10-2021 | 19-22 |  |  |
| 4 | Experiment -4 | 26-10-2021 | 28-10-2021 | 23-26 |  |  |
| 5 | Experiment -5 | 02-11-2021 | 05-11-2021 | 27-28 |  |  |
| 6 | Experiment -6 | 02-11-2021 | 05-11-2021 | 29-32 |  |  |
| 7 | Experiment -7 | 09-11-2021 | 11-11-2021 | 33-38 |  |  |
| 8 | Experiment -8 | 09-11-2021 | 11-11-2021 | 39-42 |  |  |
| 9 | Experiment -9 | 16-11-2021 | 19-11-2021 | 43-49 |  |  |
| 10 | Experiment -10 | 16-11-2021 | 19-11-2021 | 50-51 |  |  |
| 11 | Experiment -11 | 23-11-2021 | 27-11-2021 | 52-53 |  |  |
| 12 | Experiment -12 | 23-11-2021 | 27-11-2021 | 53-54 |  |  |
| 13 | Experiment -13 | 30-11-2021 | 13-12-2021 | 55-57 |  |  |
| 14 | Experiment -14 | 30-11-2021 | 13-12-2021 | 58-60 |  |  |
| 15 | Experiment -15 | 30-11-2021 | 13-12-2021 | 61-64 |  |  |
| 16 | Experiment -16 | 14-12-2021 | 28-12-2021 | 65-79 |  |  |
| 17 | Experiment -17 | 14-12-2021 | 28-12-2021 | 80-111 |  |  |
| 18 | Experiment -18 | 14-12-2021 | 28-12-2021 | 112-115 |  |  |
| 19 | Experiment -19 | 04-01-2022 | 06-01-2022 | 112-115 |  |  |
| 20 | Experiment -20 | 04-01-2022 | 06-01-2022 | 116-117 |  |  |
| **Additional Experiments** | | | | | | |  |  |  |  |  |  |
| 21 | Experiment -1 | 16-11-2021 | 19-11-2021 | 118-120 |  |  |
| 22 | Experiment -2 | 16-11-2021 | 19-11-2021 | 121-122 |  |  |
| 23 | Experiment -3 | 16-11-2021 | 19-11-2021 | 123-125 |  |  |
| 24 | Experiment -4 | 30-11-2021 | 13-12-2021 | 126-128 |  |  |
| 25 | Experiment -5 | 30-11-2021 | 13-12-2021 | 129-130 |  |  |
| 26 | Experiment -6 | 30-11-2021 | 13-12-2021 | 131-132 |  |  |
| 27 | Experiment -7 | 30-11-2021 | 13-12-2021 | 133-134 |  |  |

26

**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:**

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

* **Fields**
* **Methods**
* **Constructors**
* **Blocks**
* **Nested class and interface**

**Getter method**

A getter method in Java enables us to retrieve or obtain the data of a variable. It returns the value of private members without changing the class type. This method is also called the Accessor method as it accesses the file of classes representing an object. We should always create getter methods for every private property of a class. We can set the access modifier of a variable’s getter method, depending on the level of excess given to a variable. If we declare the variables within class as private, we will have to add public getter methods for every member.

**Setter method**

A setter method in Java enables us to update or set the data of a variable. It modifies the value of private members without changing the class type. This method is also called the Mutator method as it modifies or we can say mutates the number of classes representing an object. Like the getter method we should always create setter methods for every private property of a class. And again if we declare the variables within class as private, we will have to add public setter methods for every member.

**SYNTAX:**

For class:

**class** {

field;

  method;

}

For getters and setters:

**public** **return** type getPropertyName()

**public** **void** setPropertyName( dataType property value)

**PROGRAM:**

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

**class** happy {

String partno;

String description;

**int** quantity;

**double** price;

happy(String partno,String description,

**int** quantity,**double** price){

**this**. partno=partno;

**this**.description=description;

**this**.quantity=quantity;

**this**.price=price;}

**public** String getPartno() {

**return** partno;}

**public** **void** setPartno(String partno) {

**this**.partno = partno;}

**public** String getDescription() {

**return** description;}

**public** **void** setDescription(String description) {

**this**.description = description;}

**public** **int** getQuantity() {

**if**(quantity<0)

**return** 0;

**else**

**return** quantity;}

**public** **void** setQuantity(**int** quantity) {

**this**.quantity = quantity;}

**public** **double** getPrice() {

**if**(price<0)

**return** 0.0;

**else**

**return** price;}

**public** **void** setPrice(**double** price) {

**this**.price = price;}

**double** getInvoiceAmount() {

**return** getPrice()\*getQuantity();}}

**public** **class** Invoice1 {

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

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

**double** a,a1;

String c,c1,b,b1;

**int** x,y,z;

System.***out***.println("\nEnter no of products");

y=sc.nextInt();

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

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

c1=sc.next();

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

b1=sc.next();

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

z=sc.nextInt();

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

a1=sc.nextDouble();

happy in= **new** happy(c1,b1,z,a1);

a=in.getInvoiceAmount();

c=in.getPartno();

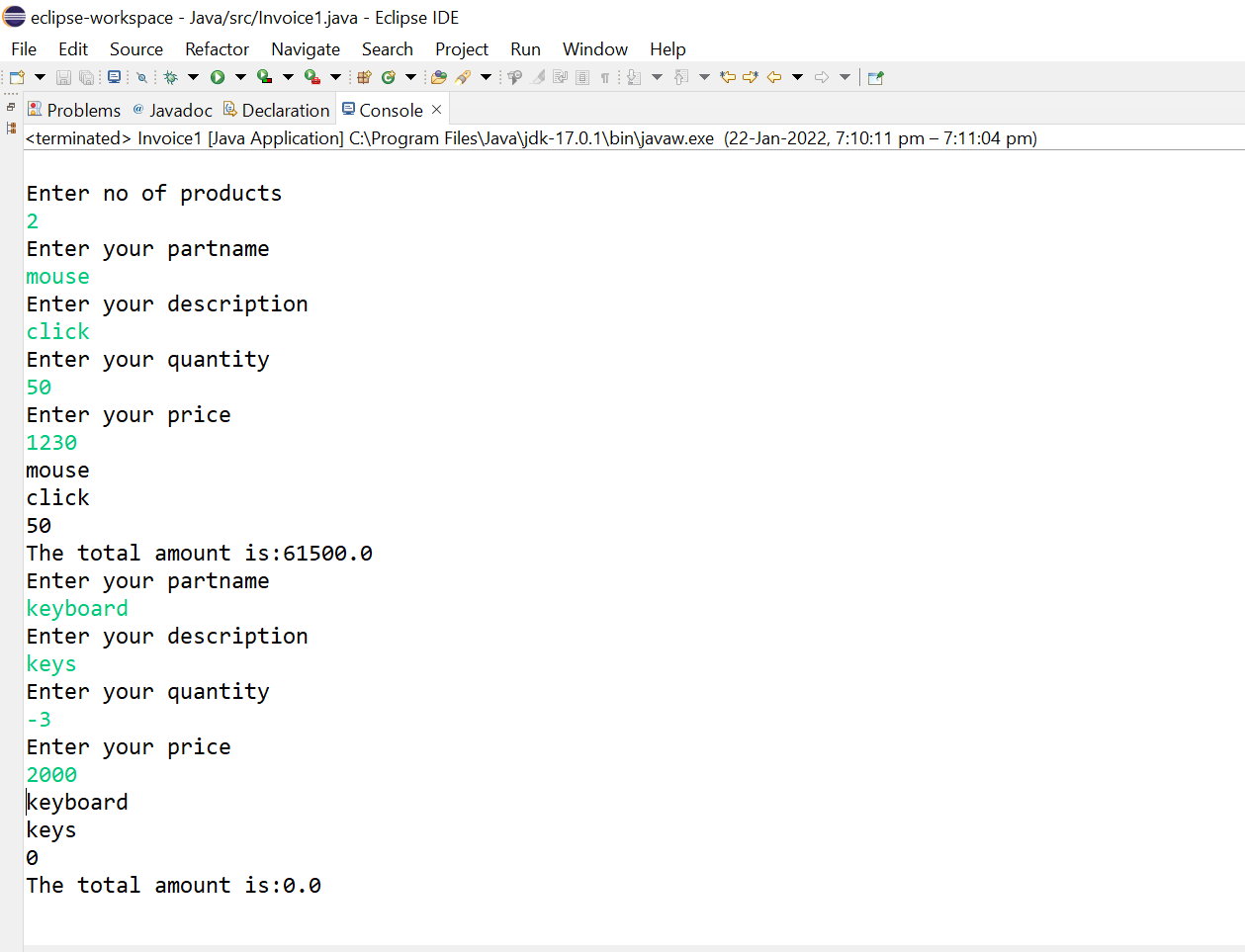
b=in.getDescription();

x=in.getQuantity();

System.***out***.println(c+"\n"+b+"\n"+x);

System.***out***.println("The total amount is:"+a);}}}

**OUTPUT:**

****

**EXPERIMENT NO: 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:** Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear. However, [Java](https://www.javatpoint.com/java-tutorial) provides statements that can be used to control the flow of Java code. Such statements are called control flow statements.

decision-making statements decide which statement to execute and when. Decision-making statements evaluate the Boolean expression and control the program flow depending upon the result of the condition provided. There are two types of decision-making statements in Java, i.e., If statement and switch statement.

## **If..else statement**

In this statement, if the condition specified is true, the if block is executed. Otherwise, the else block is executed.

**SYNTAX:**

**if**(condition 1) {

statement 1;

}

**else** **if**(condition 2) {

statement 2;

}

**else** {

statement 2;

}

**PROGRAM:**

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

**class** Bill1{

**long** cno;

String cname;

**double** pr;

**double** cr;

String connection;

Bill1(**long** cno,String cname,**double** pr,**double** cr,String connection){

**this**.cno=cno;

**this**.cname=cname;

**this**.pr=pr;

**this**.cr=cr;

**this**.connection=connection;}

**void** eb() {

**double** sum=0;**double** i=cr-pr;

**if**(connection.equals("domestic")) {

**if** (i<=100) {System.***out***.println(i);}

**else** **if** (i<=200&&i>101) {

sum+=100;

System.***out***.println( sum+=(i-100)\*2.5);}

**else** **if** (i<=500&&i>201) {

sum+=100;

sum+=100\*2.5;

System.***out***.println(sum+=(i-200)\*4);}

**else** {

sum+=100;

sum+=100\*2.5;

sum+=300\*4;

System.***out***.println(sum+=(i-500)\*6);}}

**else** {

**if** (i<=100) {System.***out***.println(i\*2);}

**else** **if** (i<=200&&i>101) {

sum+=200;

System.***out***.println( sum+=(i-100)\*4.5);

}

**else** **if** (i<=500&&i>201) {

sum+=200;

sum+=100\*4.5;

System.***out***.println(sum+=(i-200)\*6);}

**else** {

sum+=200;

sum+=100\*4.5;

sum+=300\*6;

System.***out***.println(sum+=(i-500)\*7);}}}}

**public** **class** Bill {

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

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

**long** a;

String b,b1;

**double** d,c;

System.***out***.println("Enter your EB connection");

b1=sc.next();

System.***out***.println("Enter your cusumer no");

a=sc.nextInt();

System.***out***.println("Enter your consumer name ");

b=sc.next();

System.***out***.println("Enter your previous month reading ");

d=sc.nextDouble();

System.***out***.println("Enter your current month reading");

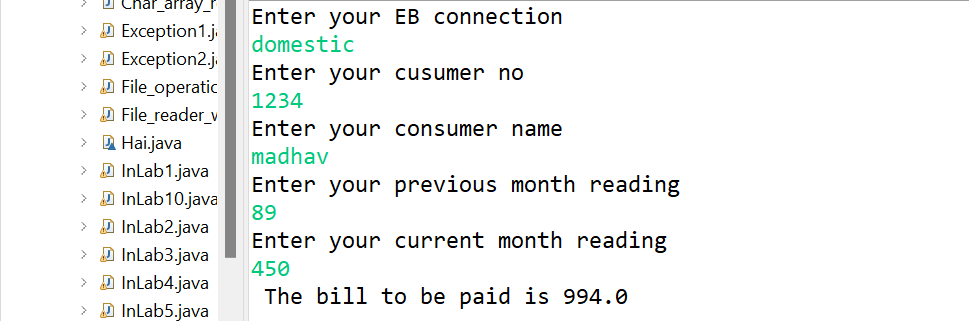
c=sc.nextDouble();

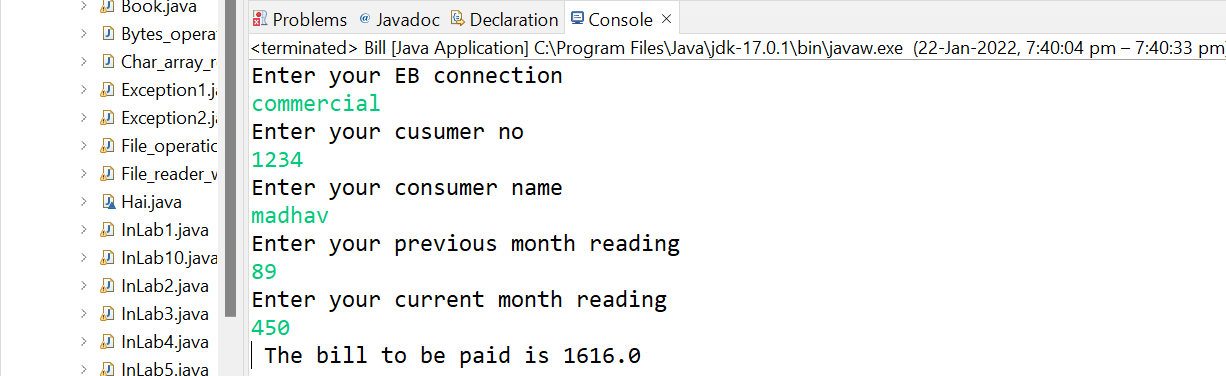
Bill1 d1=**new** Bill1(a,b,d,c,b1);

System.***out***.print(" The bill to be paid is ");

d1.eb();}}

**OUTPUT:**





**EXPERIMENT NO: 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:**

The **static keyword** in [Java](https://www.javatpoint.com/java-tutorial) is used for memory management mainly. We can apply static keyword with [variables](https://www.javatpoint.com/java-variables), methods, blocks and [nested classes](https://www.javatpoint.com/java-inner-class). The static keyword belongs to the class than an instance of the class.

The static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)
3. Block
4. Nested class

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

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

### **Advantages of static variable**

It makes your program **memory efficient** (i.e., it saves memory).

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

There are many non-access modifiers, such as static, abstract, synchronized, native, volatile, transient, etc. Here, we are going to learn the access modifiers only.

**SYNTAX:**

**private** **void** msg(){System.out.println("Hello java");}

**static** String college

**PROGRAM:**

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

**class** Interst{

**static** **double** *annualIntersetRate*;

**private** **double** savingsBalance;

Interst( **double** annualIntersetRate,**double** savingsBalance){

**this**. *annualIntersetRate* =annualIntersetRate;

**this**. savingsBalance= savingsBalance;}

**double** calculateMonthlyInterset() {

**return** savingsBalance+=(savingsBalance\**annualIntersetRate*)/12;}

**static** **void** modifyInterest(**double** c) {

*annualIntersetRate*=c;}}

**public** **class** Saving\_Account {

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

**double** a,b1,c,d;

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

System.***out***.println("Enter your first account balance ");

a=sc.nextDouble();

System.***out***.println("Enter your second account balance ");

b1=sc.nextDouble();

System.***out***.println("Enter your annualConcentration rate");

c=sc.nextDouble();

Interst saver1=**new** Interst(c,a );

System.***out***.println(saver1.calculateMonthlyInterset());

Interst saver2=**new** Interst(c,b1);

System.***out***.println(saver2.calculateMonthlyInterset());

System.***out***.println("Enter your annualConcentration rate");

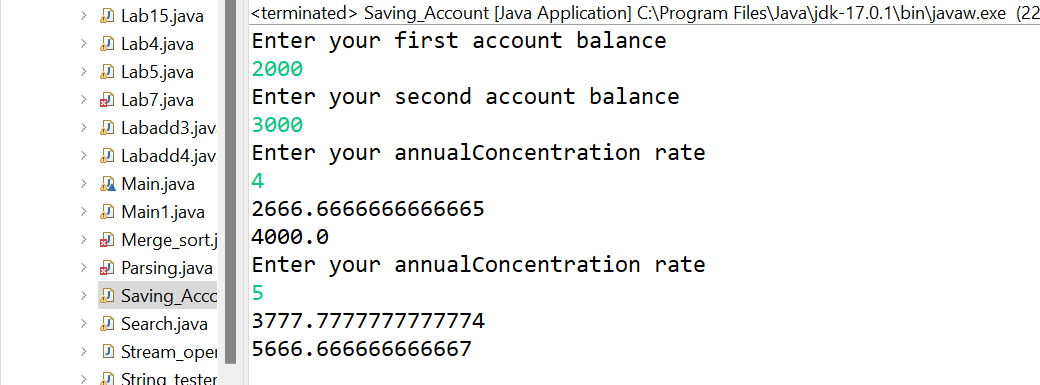
d=sc.nextDouble();

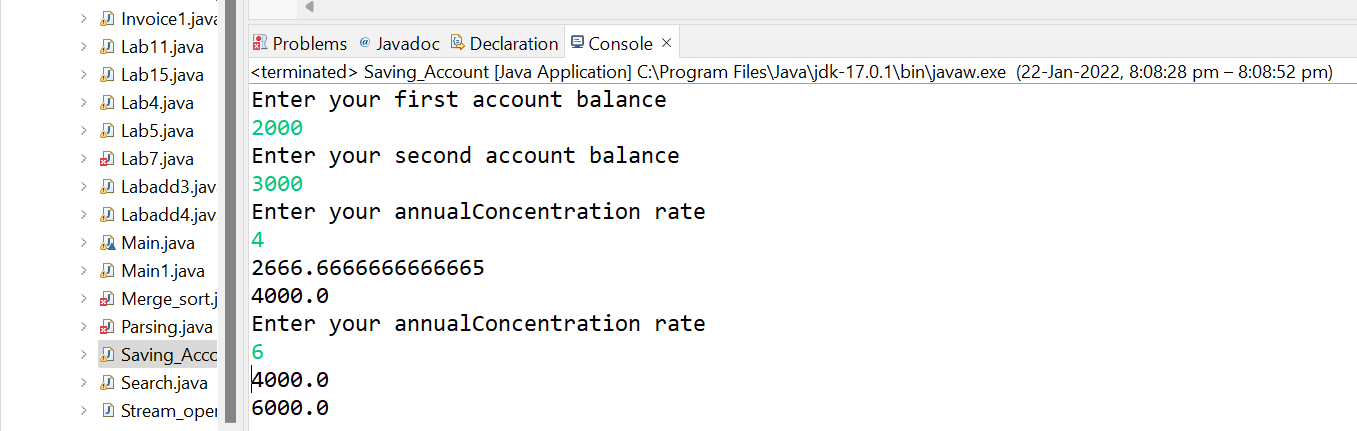
Interst.*modifyInterest*(d);

System.***out***.println(saver1.calculateMonthlyInterset());

System.***out***.println(saver2.calculateMonthlyInterset());}}

**OUTPUT:**





**EXPERIMENT NO: 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:** Java is an object-oriented programming language. Most of the work done with the help of **objects**. We know that an array is a collection of the same data type that dynamically creates objects and can have elements of primitive types. Java allows us to store objects in an array. In [Java](https://www.javatpoint.com/java-tutorial), the class is also a user-defined data type. An array that conations **class type elements** are known as an **array of objects**. It stores the reference variable of the object.

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.

**SYNTAX:**

Class\_Name objectArrayReference[ ];

ClassName obj[]=**new** ClassName[array\_length];

**PROGRAM:**

**import** java.util.Scanner;

**class** Book1{

String bookname;

**long** isbnno;

String authorname;

String publisher;

**public** String getBookname() {

**return** bookname;}

**public** **void** setBookname(String bookname) {

**this**.bookname = bookname;}

**public** **long** getIsbnno() {

**return** isbnno;}

**public** **void** setIsbnno(**long** isbnno) {

**this**.isbnno = isbnno;}

**public** String getAuthorname() {

**return** authorname;}

**public** **void** setAuthorname(String authorname) {

**this**.authorname = authorname;}

**public** String getPublisher() {

**return** publisher;}

**public** **void** setPublisher(String publisher) {

**this**.publisher = publisher;}

Book1(String bookname,**long** isbnno,String authorname,String publisher){

**this**.bookname=bookname;

**this**.isbnno=isbnno;

**this**. authorname=authorname;

**this**. publisher=publisher;}

**public** Book1() {}

**void** getBookInfo() {

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

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

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

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

**public** **class** Book {

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

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

String bn;

**long** is;

String an;

String pu;

Book1 a[]=**new** Book1[30];

**for**(**int** i=0;i<30;i++) {

a[i]=**new** Book1();}

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

**int** i;

i=sc.nextInt();

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

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

bn=sc.next();

a[i].setBookname(bn);

System.***out***.println("Enter your isbn number");

is=sc.nextLong();

a[i].setIsbnno(is);

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

an=sc.next();

a[i].setAuthorname(an);

System.***out***.println("Enter your publisher name");

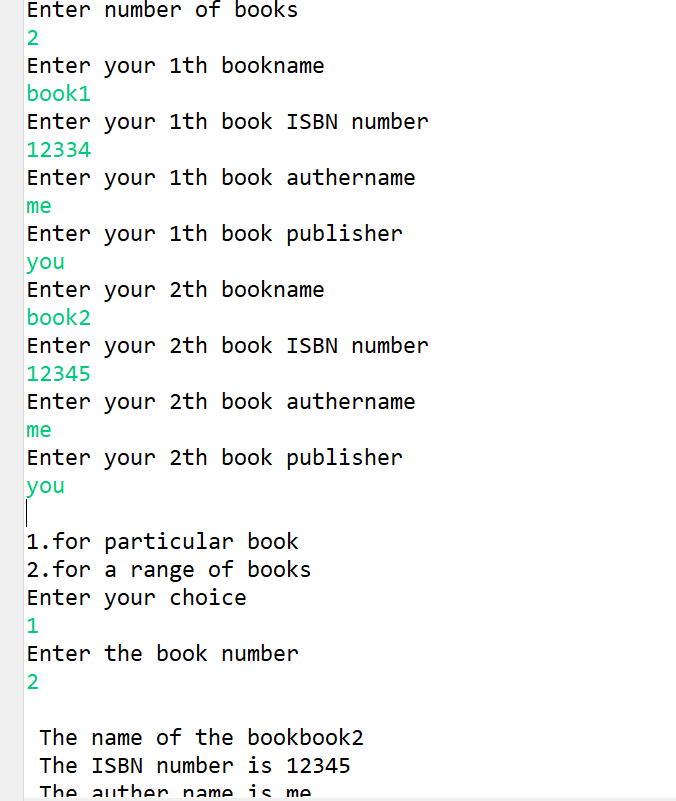
pu=sc.next();

a[i].setPublisher(pu);}

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

a[j].getBookInfo() ;}}}

**OUTPUT:**



**EXPERIMENT NO: 5**

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

**DESCRIPTION:** A simple approach is to do a [**linear search**](https://www.geeksforgeeks.org/linear-search/)**.** The time complexity of the above algorithm is O(n). Another approach to perform the same task is using Binary Search.   
**Binary Search:** Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise, narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.

The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(Log n).

**Java while loop** is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.

**SYNTAX:**

while (test\_expression)

{

update\_expression;

}

**PROGRAM:**

**import** java.util.Scanner;

**class** BinarySearch{

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

**int** key,n;

BinarySearch(**int** a[],**int** b,**int** c){

**this**.a=a;

n=b;

key=c;}

**void** binarysearch() {

**int** mid,lb=0,ub=n-1;

**while**(lb<=ub) {

mid=(ub+lb)/2;

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

System.***out***.println("Key found at:"+mid);

System.*exit*(0);}

**else**

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

ub=mid-1;

**else**

lb=mid+1;}

System.***out***.println("Key not found");}}

**public** **class** LabProgram5 {

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

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

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

**int** n,i,key;

System.***out***.println("Enter no of elements in the array:");

n=sc.nextInt();

System.***out***.println("Enter elements in the array:");

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

a[i]=sc.nextInt(); }

System.***out***.println("Enter the key:");

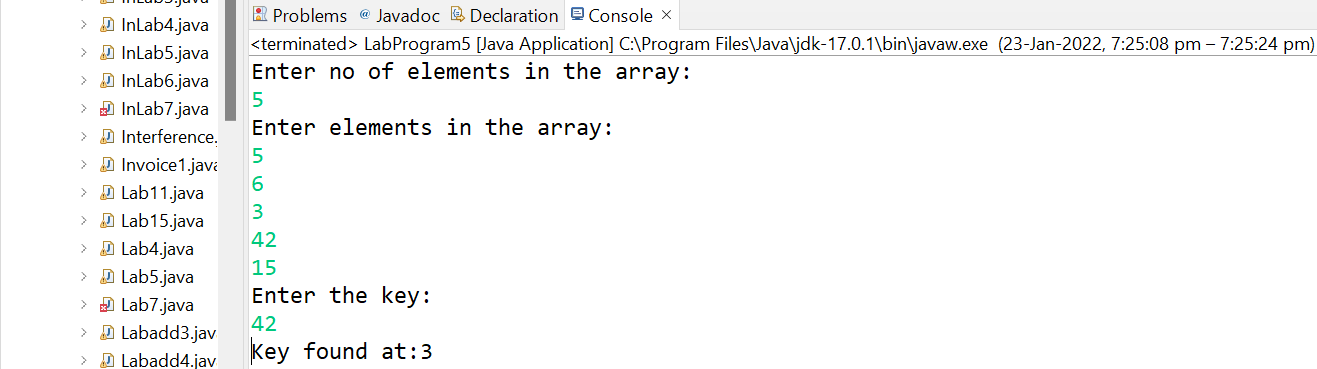
key=sc.nextInt();

BinarySearch b1=**new** BinarySearch(a,n,key);

b1.binarysearch();

sc.close(); }}

**OUTPUT:**



**EXPERIMENT NO: 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 is a [Divide and Conquer](https://www.geeksforgeeks.org/divide-and-conquer-introduction/) 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)

A recursive function must have a condition to stop calling itself. Otherwise, the function is called indefinitely.

Once the condition is met, the function stops calling itself. This is called a base condition.

To prevent infinite recursion, you can use [if...else statement](https://www.programiz.com/javascript/if-else) (or similar approach) where one branch makes the recursive call, and the other doesn't

**SYNTAX:**

function recurse() {

**if**(condition) {

recurse();

}

**else** {

// stop calling recurse()

}

}

recurse();

**PROGRAM:**

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

**public** **class** MergeLab6 {

**static** **void** mergePass(**int** b[],**int** lb,**int** ub) {

**int** mid;

**if**(lb!=ub) {

mid=(lb+ub)/2;

*mergePass*(b,lb,mid);

*mergePass*(b,mid+1,ub);

*mergeSort*(b,lb,mid,ub);}}

**static** **void** mergeSort(**int** c[],**int** lb,**int** mid,**int** ub) {

**int** i,j,k;

**int** temp[]=**new** **int**[20];

i=lb;j=mid+1; k=lb;

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

**if**(c[i]<c[j])

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

**else**

temp[k++]=c[j++]; }

**while**(i<=mid)

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

**while**(j<=ub)

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

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

c[i]=temp[i];}

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

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

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

**int** n;

System.***out***.println("Enter the number of elements in the array:");

n=sc.nextInt();

System.***out***.println("Enter the elements in thearray:");

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

a[i]=sc.nextInt();

System.***out***.println("Before Sorting:");

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

System.***out***.printf("%d\n",a[i]);

*mergePass*(a,0,n-1);

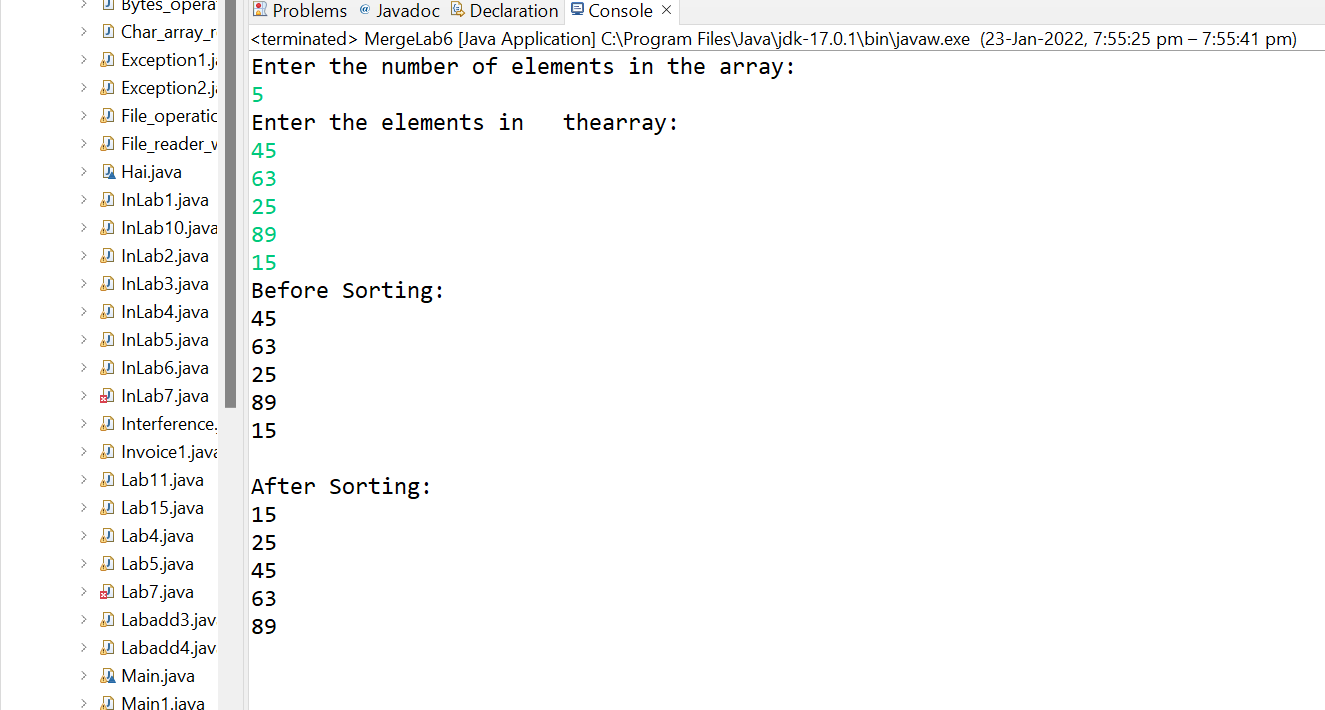
System.***out***.println("\nAfter Sorting:");

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

System.***out***.printf("%d\n",a[i]);

sc.close(); }}

**OUTPUT:**



**EXPERIMENT NO: 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:** The **java.lang.Math.random()** is used to return a pseudorandom double type number greater than or equal to 0.0 and less than 1.0. The default random number always generated between 0 and 1.

If you want to specific range of values, you have to multiply the returned value with the magnitude of the range. For example, if you want to get the random number between 0 to 20, the resultant address has to be multiplied by 20 to get the desired result.

**SYNTAX:**

**public** **static** **double** random( )

**PROGRAM:**

**import** java.util.Random;

**public** **class** Randfunc{

**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;

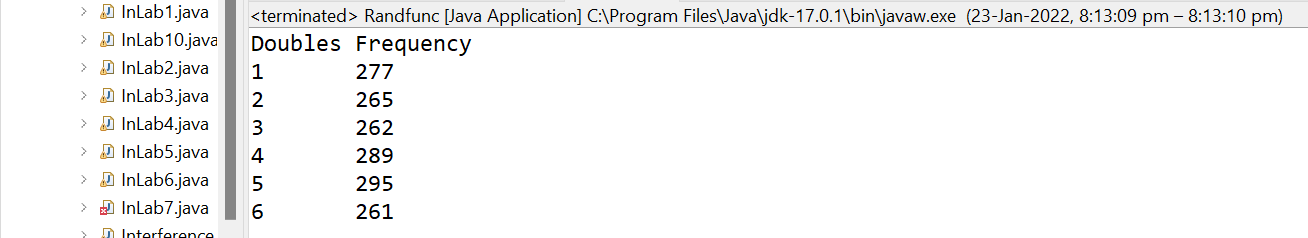
**break**;}}}

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 NO: 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** is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of [OOPs](https://www.javatpoint.com/java-oops-concepts) (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new [classes](https://www.javatpoint.com/object-and-class-in-java) 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

* **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
* **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
* **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
* **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

**SYNTAX:**

**class** Subclass-name **extends** Superclass-name

{

   //methods and fields

}

**PROGRAM:**

**import** java.util.Scanner;

**class** Employee111{

String Emp\_name, Mail\_id, address;

**long** Emp\_id, Mobile\_no;

**double** hra,da,basicpay,pf,staffclub,gross,net;

**public** Employee111(String name, **long** id, String add, String mail, **long** mobile) {

// **TODO** Auto-generated constructor stub

Emp\_name=name;

Emp\_id=id;

address=add;

Mail\_id=mail;

Mobile\_no=mobile;

}

**void** disp() {

System.***out***.println("EMP NAME:"+Emp\_name);

System.***out***.println("EMP ID:"+Emp\_id);

System.***out***.println("EMP ADDRESS:"+address);

System.***out***.println("EMP MAIL ID:"+Mail\_id);

System.***out***.println("EMP MOBILE NUMBER:"+Mobile\_no);

}

**void** show() {

System.***out***.println("BASIC PAY:"+basicpay);

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

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

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

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

}

}

**class** Programmer **extends** Employee111{

Programmer(String name, **long** id,String add,String mail,**long** mobile){

**super**(name,id,add,mail,mobile);

}

**void** setBasicPay(**double** p) {

basicpay=p;

}

**void** bill() {

da=basicpay\*0.97;

hra=basicpay\*0.1;

pf=basicpay\*0.12;

staffclub=basicpay\*(0.1/100);

gross=basicpay+da+hra+pf;

net=gross-pf-staffclub;

disp();

show();

} }

**class** Assistant\_Professor **extends** Employee111{

Assistant\_Professor(String name, **long** id,String add,String mail,**long** mobile){

**super**(name,id,add,mail,mobile);

}

**void** setBasicPay(**double** p) {

basicpay=p;

}

**void** bill() {

da=basicpay\*0.94;

hra=basicpay\*0.09;

pf=basicpay\*0.13;

staffclub=basicpay\*(0.09/100);

gross=basicpay+da+hra+pf;

net=gross-pf-staffclub;

disp();

show();}}

**class** Associate\_Professor **extends** Employee111{

Associate\_Professor(String name, **long** id,String add,String mail,**long** mobile){

**super**(name,id,add,mail,mobile); }

**void** setBasicPay(**double** p) {

basicpay=p; }

**void** bill() {

da=basicpay\*0.96;

hra=basicpay\*0.2;

pf=basicpay\*0.11;

staffclub=basicpay\*(0.1/100);

gross=basicpay+da+hra+pf;

net=gross-pf-staffclub;

disp();

show(); }}

**class** Professor **extends** Employee111{

Professor(String name, **long** id,String add,String mail,**long** mobile){

**super**(name,id,add,mail,mobile);}

**void** setBasicPay(**double** p) {

basicpay=p; }

**void** bill() {

da=basicpay\*0.97;

hra=basicpay\*0.1;

pf=basicpay\*0.12;

staffclub=basicpay\*(0.1/100);

gross=basicpay+da+hra+pf;

net=gross-pf-staffclub;

disp();

show();}}

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

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

System.***out***.println("ENTER THE NAME:");

String name=sc.nextLine();

System.***out***.println("ENTER THE ID:");

**long** id=sc.nextLong();

sc.nextLine();

System.***out***.println("ENTER THE ADDRESS:");

String add=sc.nextLine();

System.***out***.println("ENTER THE MAIL ID:");

String mail=sc.nextLine();

System.***out***.println("ENTER MOBILE NO:"); **long** mob=sc.nextLong();

System.***out***.println("\tEMPLOYEE TYPE");

System.***out***.println("1.PROGRAMMER");

System.***out***.println("2.ASSISTANT PROFESSOR");

System.***out***.println("3.ASSOCIATE PROFESSOR");

System.***out***.println("4.PROFESSOR"); **int** choice;

System.***out***.println("\nENTER YOUR CHOICE");

choice=sc.nextInt();

**switch**(choice) {

**case** 1: Programmer e1=**new** Programmer(name,id,add,mail,mob);

System.***out***.println("ENTER THE BASIC PAY:");

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

e1.setBasicPay(m);

e1.bill();

**break**;

**case** 2:

Assistant\_Professor e2= **new** Assistant\_Professor(name,id,add,mail,mob);

System.***out***.println("ENTER THE BASIC PAY:");

**double** m1=sc.nextDouble();

e2.setBasicPay(m1);

e2.bill();

**break**;

**case** 3:

Associate\_Professor e3=**new** Associate\_Professor(name,id,add,mail,mob);

System.***out***.println("ENTER THE BASIC PAY:");

**double** m2=sc.nextDouble();

e3.setBasicPay(m2);

e3.bill();

**break**;

**case** 4:

Professor e4=**new** Professor(name,id,add,mail,mob);

System.***out***.println("ENTER THE BASIC PAY:");

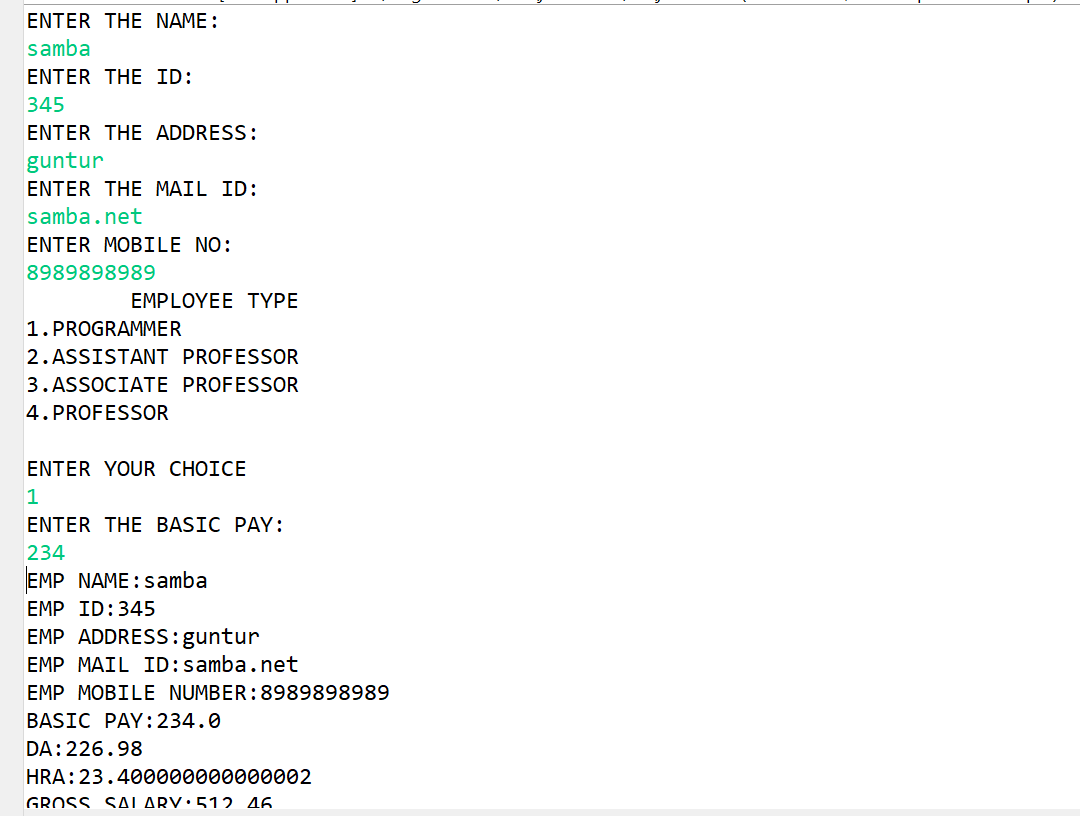
**double** m3=sc.nextDouble();

e4.setBasicPay(m3);

e4.bill();

**break**;}}}

**OUTPUT:**

****

**EXPERIMENT NO: 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:** If a [class](https://www.javatpoint.com/object-and-class-in-java) has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

If we have to perform only one operation, having same name of the methods increases the readability of the [program](https://www.javatpoint.com/java-programs).

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

So, we perform method overloading to figure out the program quickly.

## **Advantage of method overloading**

Method overloading increases the readability of the program.

### **Different ways to overload the method**

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

**SYNTAX:**

1. **static** **int** add(**int** a,**int** b){**return** a+b;
2. **static** **int** add(**int** a,**int** b,**int** c){**return** a+b+c;

**PROGRAM:**

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

**abstract** **class** Shape{

**double** dim1,dim2;

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

dim1=a;

dim2=b; }

**abstract** **void** Area();}

**class** Rectangle1 **extends** Shape{

Rectangle1(**double** a,**double** b){

**super**(a,b); }

**void** Area() {

System.***out***.println("The area of the Rectangle:"+(dim1\*dim2));}}

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

Triangle(**double** a,**double** b){

**super**(a,b);}

**void** Area() {

System.***out***.println("The area of the tringle:"+(0.5\*dim1\*dim2)); }}

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

Circle(**double** a,**double** b){

**super**(a,b);}

**void** Area() {

System.***out***.println("The area of the circle :"+(3.14\*dim1\*dim2)); } }

**public** **class** Randfunc {

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

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

System.***out***.println("\t\tFind:\n1.Area of Rectangle\n2.Area of triangle"

+ "\n3.Area of circle");

**int** choice;

**double** a,b;

System.***out***.print("Enter your choice:");

choice=sc.nextInt();

**switch**(choice) {

**case** 1:

System.***out***.println("Enter the length:");

a=sc.nextDouble();

System.***out***.println("Enter the breath:");

b=sc.nextDouble();

Rectangle1 r=**new** Rectangle1(a,b);

r.Area();

**break**;

**case** 2:

System.***out***.println("Enter the base:");

a=sc.nextDouble();

System.***out***.println("Enter the height:");

b=sc.nextDouble();

Triangle t=**new** Triangle(a,b);

t.Area();

**break**;

**case** 3:

System.***out***.println("Enter the radius of the circle:");

a=sc.nextDouble();

Circle c=**new** Circle(a,a);

c.Area();

**break**;

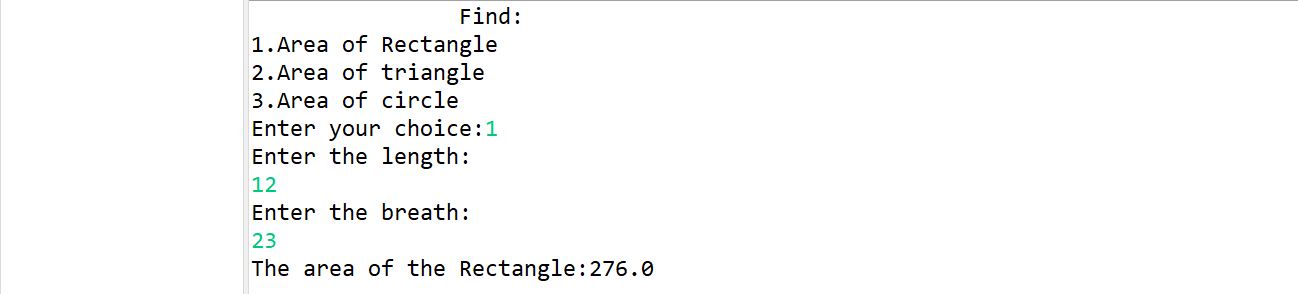
**default**:

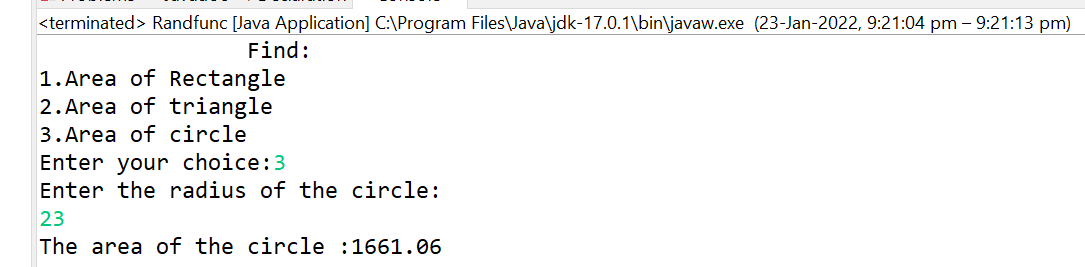
System.***out***.println("Wrong Choice");

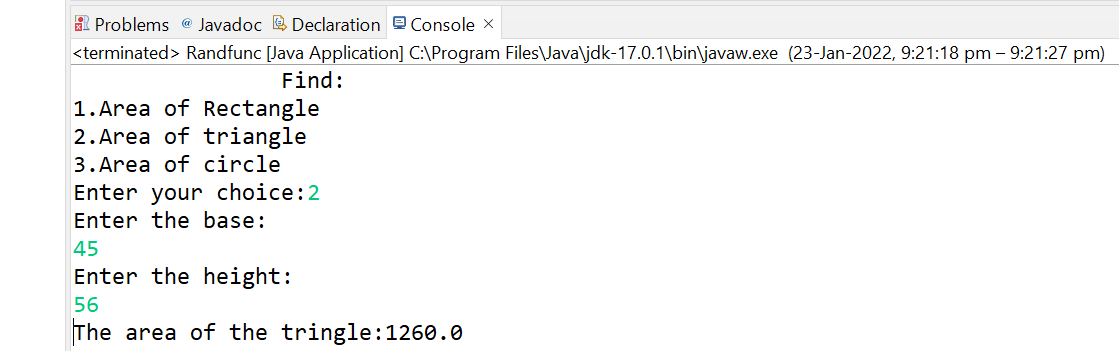
**break**;}

sc.close();}}

**OUTPUT:**







**EXPERIMENT NO: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](https://www.geeksforgeeks.org/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).

Package names and directory structure are closely related. For example if a package name is college.staff.cse, then there are three directories, college, staff and cse such that cse is present in staff and staff is present college.

**SYNTAX:**

import package.subpackage;

**PROGRAM:**

**package** Timeconverter;

**public** **class** Timeconversion {

**double** hr,min,sec;

**public** **void** hrtomin(**double** hr) {

**this**.hr=hr;

System.***out***.println(" min = "+hr\*60); }

**public** **void** mintohr(**double** min){

**this**.min=min;

System.***out***.println(" hrs = "+min/60); }

**public** **void** hrtosec(**double** hr) {

**this**.hr=hr;

System.***out***.println(" sec = "+hr\*60\*60);}

**public** **void** sectohr(**double** sec) {

**this**.sec=sec;

System.***out***.println("hours = "+sec/(60\*60));}}

**package** Distanceconverter;

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

**double** km,m,mile;

**public** **void** kmtom(**double** km){

**this**.km=km;

System.***out***.println("in meters "+km\*1000);}

**public** **void** mtokm(**double** m) {

**this**.m=m;

System.***out***.println("in kilometers "+m/1000); }

**public** **void** miletom(**double** mile) {

**this**.mile=mile;

System.***out***.println("in meters "+mile/1.6); }

**public** **void** mtomile(**double** m){

**this**.m=m;

System.***out***.println("in miles "+m\*1.6);}}

**package** currencyconverter;

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

**double** i,j;

**public** **void** dti( **double** i) {

System.***out***.println("The rupees are"+(i\*74)); }

**public** **void** itd(**double** i) {

System.***out***.println("The dollars are"+(i/74));}

**public** **void** eti( **double** i) {

System.***out***.println("The rupees are"+(i\*87));}

**public** **void** ite( **double** i) {

System.***out***.println("The euros are"+(i/87));}

**public** **void** yti( **double** i) {

System.***out***.println("The rupees are"+(i\*0.65));}

**public** **void** ity( **double** i) {

System.***out***.println("The yens are"+(i/0.65));}}

**import** currencyconverter.Currency;

**import** Distanceconverter.Distance;

**import** Timeconverter.Timeconversion;

**import** java.util.Scanner;

**public** **class** Tester {

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

**int** i;

Currency c1 = **new** Currency();

Distance d1 = **new** Distance();

Timeconversion tc = **new** Timeconversion();

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

System.***out***.println("1.kilometers to meters "

+ "\n2.meters to kilometers\n3.meters to miles"

+ "\n4.miles to meters");

System.***out***.println("5.ruppees to dollor\n6.dollors to ruppes\n7.yen to ruppee"

+ "\n8.ruppees to yen\n9.ruppees to euros");

System.***out***.println("10.euro to ruppe\n11.hours to min"

+ "\n12.min to hours\n13.hr to sec\n14.sec to hr");

System.***out***.println("enter tour choice");

i = sc.nextInt();

**switch**(i) {

**case** 1:

System.***out***.println("enter kilometers");

d1.kmtom(sc.nextDouble());

**break**;

**case** 2:

System.***out***.println("enter meters");

d1.mtokm(sc.nextDouble());

**break**;

**case** 3:

System.***out***.println("enter miles");

d1.miletom(sc.nextDouble());

**break** ;

**case** 4:

System.***out***.println("enter meters");

d1.mtomile(sc.nextDouble());

**break**;

**case** 5:

System.***out***.println("enter ruppees");

c1.itd(sc.nextDouble());

**break** ;

**case** 6:

System.***out***.println("enter dollors");

c1.dti(sc.nextDouble());

**break**;

**case** 7:

System.***out***.println("enter yen");

c1.yti(sc.nextDouble());

**break**;

**case** 8:

System.***out***.println("enter ruppees");

c1.ity(sc.nextDouble());

**break**;

**case** 9:

System.***out***.println("enter ruppees");

c1.ite(sc.nextDouble());

**break**;

**case** 10:

System.***out***.println("enter euros");

c1.eti(sc.nextDouble());

**break**;

**case** 11:

System.***out***.println("enter hours");

tc.hrtomin(sc.nextDouble());

**break**;

**case** 12:

System.***out***.println("enter min");

tc.mintohr(sc.nextDouble());

**break**;

**case** 13:

System.***out***.println("enter hrs");

tc.hrtosec(sc.nextDouble());

**break**;

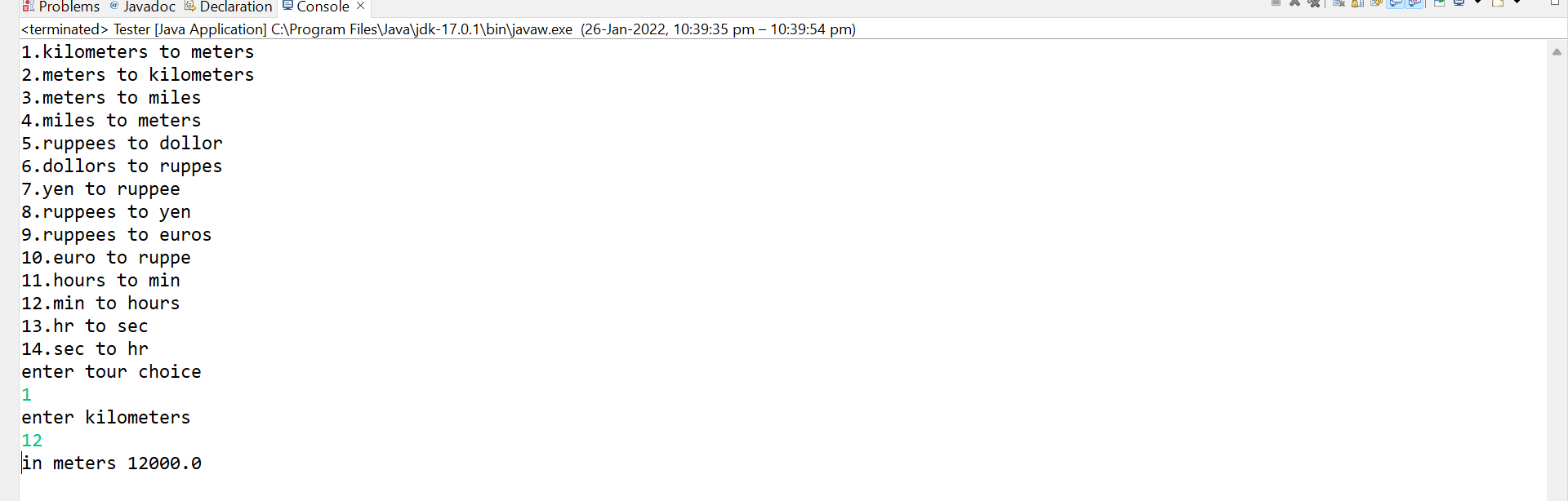
**case** 14:

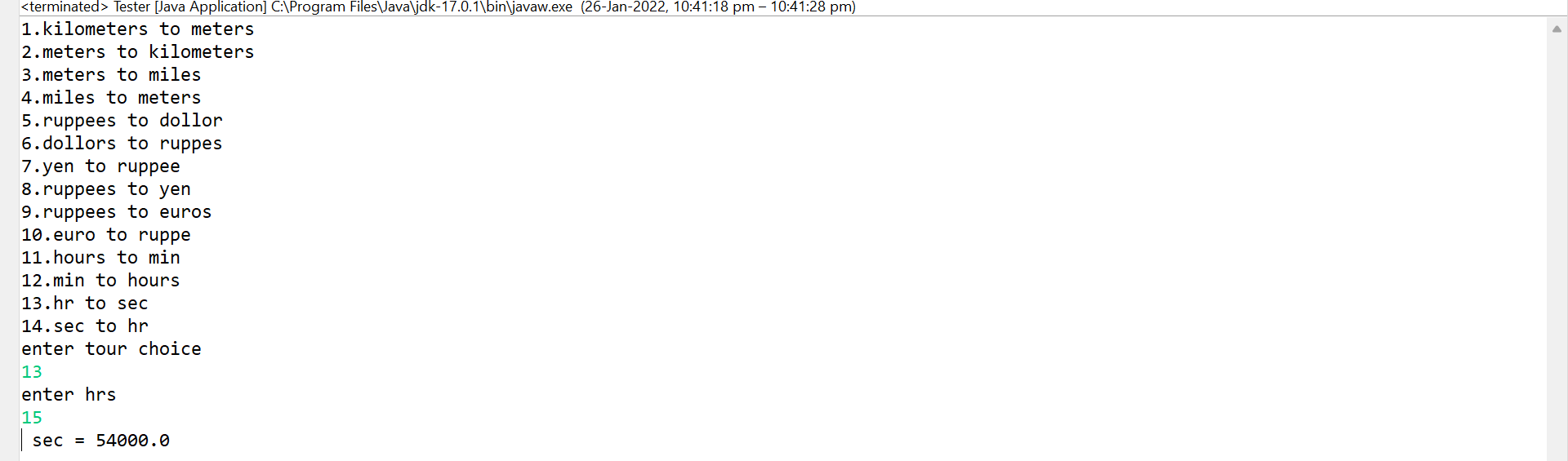
System.***out***.println("enter seconds");

tc.sectohr(sc.nextDouble());

**break**; }}}

**OUTPUT:**







**EXPERIMENT NO: 11**

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

**DESCRIPTION:**

The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that the normal flow of the application can be maintained.

In this tutorial, we will learn about Java exceptions, it's types, and the difference between checked and unchecked exceptions.

In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

### **Advantage of Exception Handling**

The core advantage of exception handling is **to maintain the normal flow of the application**. An exception normally disrupts the normal flow of the application; that is why we need to handle exceptions.

**SYNTAX:**

* + - 1. **try**{

//code that may throw an exception

}**catch**(Exception\_class\_Name ref){}

* + - 1. **try**{

//code that may throw an exception

}**finally**{}

**PROGRAM:**

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

**class** Exception\_handler{

**int** a,c,d; String s1;

**public** Scanner s =**new** Scanner(System.***in***);

**void** arthimatic() **throws** ArithmeticException {

**try** {

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

a = s.nextInt();

c=a/0;}

**catch** (ArithmeticException e) {

System.***out***.println("Arithmetic Exception \t " +e);

**throw** **new** ArithmeticException() ;}}

**void** mismatch() **throws** InputMismatchException {

**try** {

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

a= s.nextInt();}

**catch** (InputMismatchException e) {

System.***out***.println("InputMismatch Exception \t " +e);

**throw** **new** InputMismatchException() ;}}}

**public** **class** Lab11 {

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

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

Exception\_handler k =**new** Exception\_handler();

**try** {

k.arthimatic();}

**catch** (ArithmeticException e) {

System.***out***.println("recaught \t"+e);}

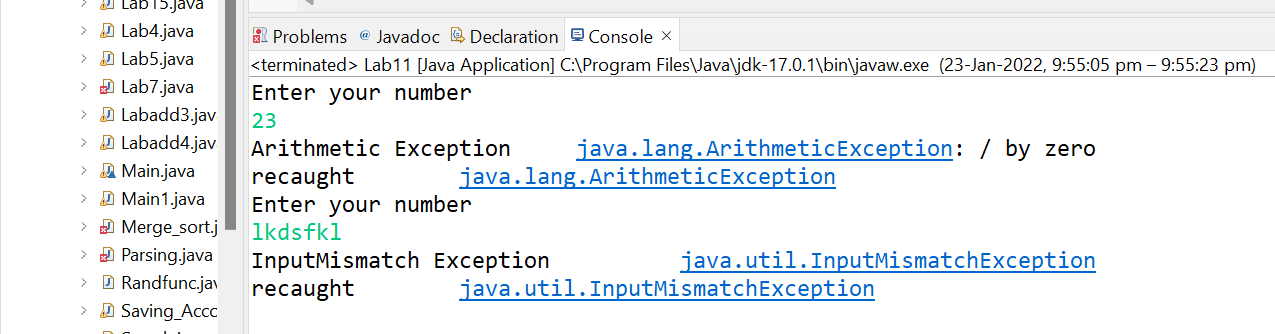
**try** {

k.mismatch();}

**catch** (InputMismatchException e) {

System.***out***.println("recaught \t"+e);}}}

**OUTPUT:**



**EXPERIMENT NO: 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:**

**Multithreading in**[**Java**](https://www.javatpoint.com/java-tutorial) 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.

However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation, etc.

### **Advantages of Java Multithreading**

1) It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.

2) You **can perform many operations together, so it saves time**.

3) Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.

**SYNTAX:**

* Thread()
* Thread(String name)
* Thread(Runnable r)
* Thread(Runnable r,String name)

**PROGRAM:**

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

**import** java.io.PipedWriter;

**import** java.io.PipedReader;

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

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** Randfunc{

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

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

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

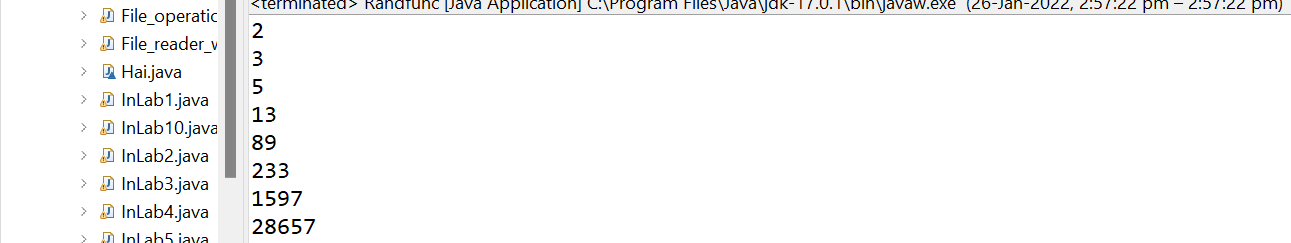
receiver r=**new** receiver(fi,pri);

fi.start();

pri.start();

r.start();}}

**OUTPUT:**



**EXPERIMENT NO: 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:**

A thread is also known as lightweight process. The idea is to achieve parallelism by dividing a process into multiple threads. For example, in a browser, multiple tabs can be different threads. MS Word uses multiple threads: one thread to format the text, another thread to process inputs, etc.

The primary difference is that threads within the same process run in a shared memory space, while processes run in separate memory spaces.  
Threads are not independent of one another like processes are, and as a result threads share with other threads their code section, data section, and OS resources (like open files and signals). But, like process, a thread has its own program counter (PC), register set, and stack space.

 Effective utilization of multiprocessor system: If we have multiple threads in a single process, then we can schedule multiple threads on multiple processor. This will make process execution faster.

**SYNTAX:**

1. **public void run():** is used to perform action for a thread.
2. **public void start():** starts the execution of the thread.JVM calls the run() method on the thread.
3. **public void sleep(long miliseconds):** Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
4. **public void join():** waits for a thread to die.
5. **public void join(long miliseconds):** waits for a thread to die for the specified milisecond

**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);}}}}

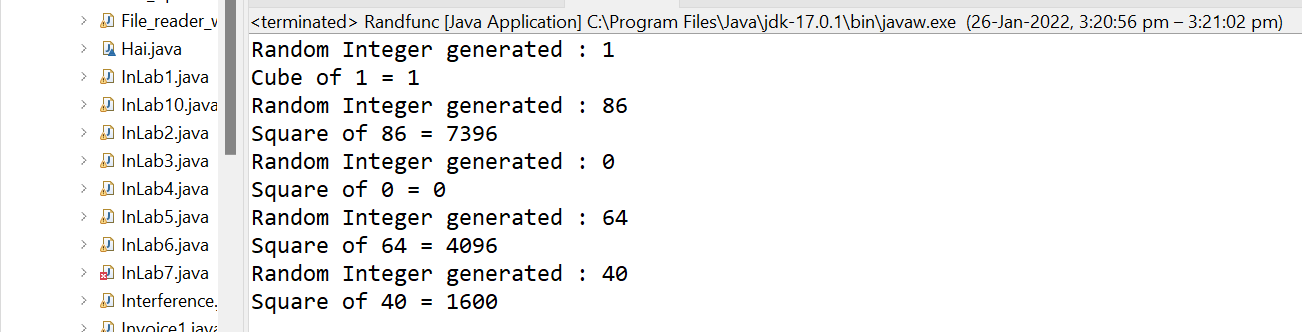
**public** **class** Randfunc {

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

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

n.start();}}

**OUTPUT:**



**EXPERIMENT NO: 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**:

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 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.

**SYNTAX:**

public final void wait(long timeout)

**public** **final** **void** notify()

**public** **final** **void** notifyAll()

**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** Randfunc {

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

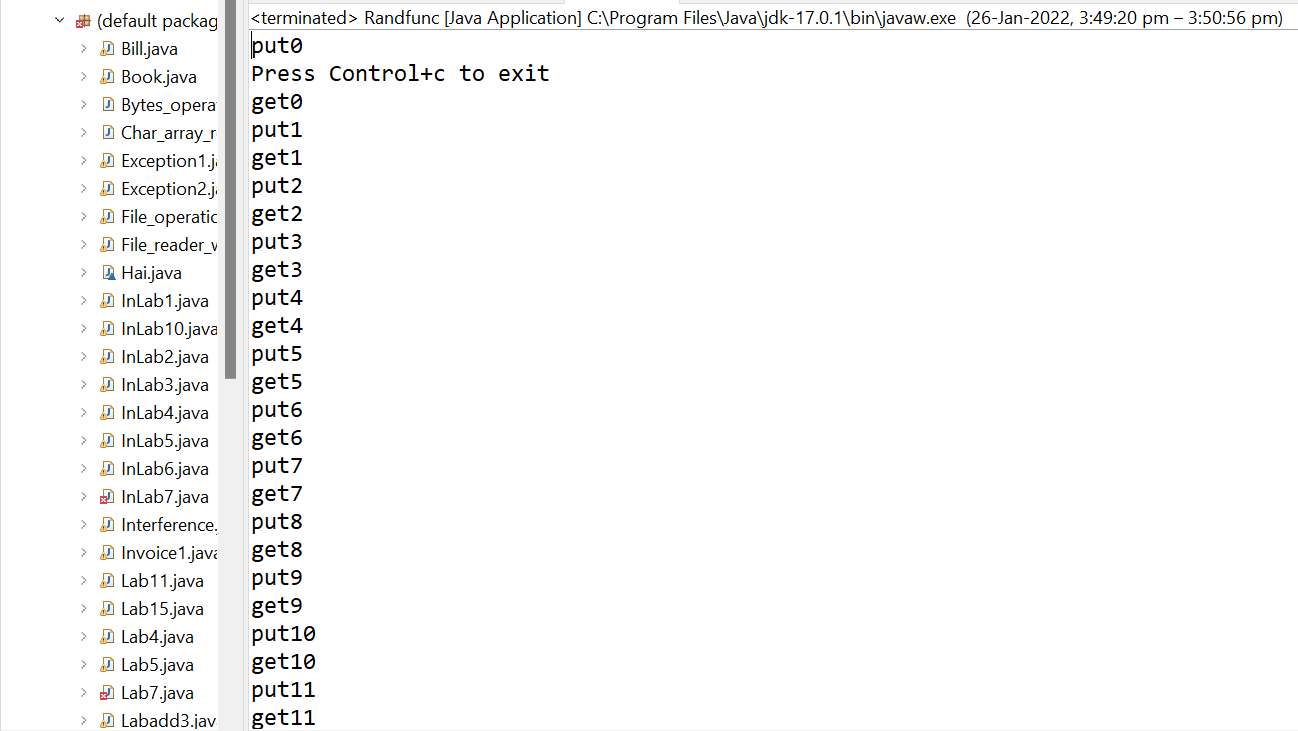
Thread1 t=**new** Thread1();

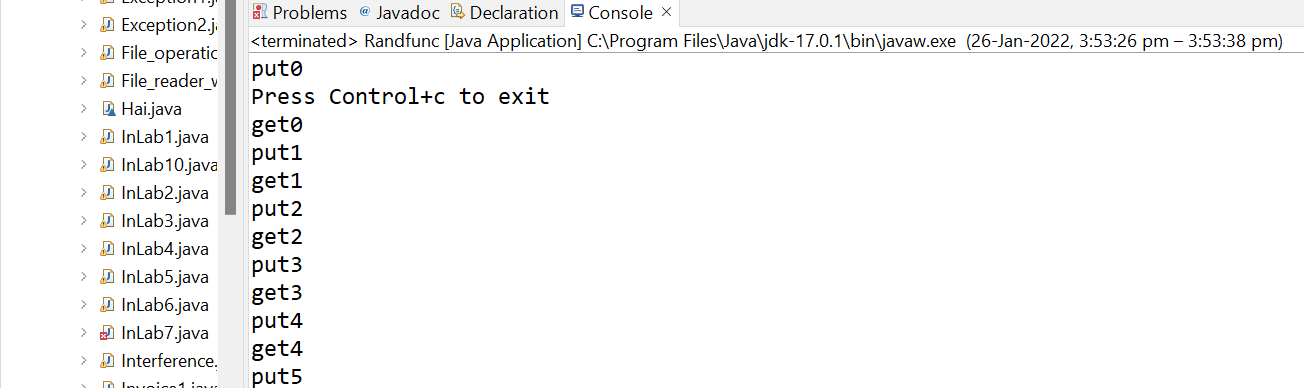
**new** Producer(t);

**new** Consumer(t);

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

**OUTPUT:**

****



**EXPERIMENT NO: 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 FileInputStream class obtains input bytes from a [file](https://www.javatpoint.com/java-file-class)

. It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc. You can also read character-stream data. But, for reading streams of characters, it is recommended to use [FileReader](https://www.javatpoint.com/java-filereader-class)

class.

|  |  |
| --- | --- |
| int available() | It is used to return the estimated number of bytes that can be read from the input stream. |
| int read() | It is used to read the byte of data from the input stream. |
| int read(byte[] b) | It is used to read up to **b.length** bytes of data from the input stream. |
| int read(byte[] b, int off, int len) | It is used to read up to **len** bytes of data from the input stream. |
| long skip(long x) | It is used to skip over and discards x bytes of data from the input stream. |
| FileChannel getChannel() | It is used to return the unique FileChannel object associated with the file input stream. |
| FileDescriptor getFD() | It is used to return the [FileDescriptor](https://www.javatpoint.com/java-filedescriptor-class)  object. |
| protected void finalize() | It is used to ensure that the close method is call when there is no more reference to the file input stream. |
| void close() | It is used to closes the [stream](https://www.javatpoint.com/java-8-stream) |

**SYNTAX:**

**public** **class** FileInputStream **extends** InputStream

**PROGRAM:**

import java.io.\*;

import java.util.\*;

class Hai1 implements FilenameFilter{

String s;

Hai1 (String s){

this.s=s;}

public boolean accept(File dir,String name) {

return name.endsWith(s);}}

public class File\_operations {

public static void main(String [] args) {

Scanner sc= new Scanner(System.in);

String st;

st=sc.nextLine();

File f1 = new File ("C:\\Users\\Siddardha\\OneDrive\\Documents\\code blocks");

if(f1.isDirectory()) {

String s1[]=f1.list();

for(int i=0;i<s1.length;i++)

System.out.println(s1[i]);

Hai1 s2=new Hai1(st);

String s3[]=f1.list(s2);

for(int i=0;i<s3.length;i++)

System.out.println(s3[i]);

File f[]=f1.listFiles(s2);

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

if(f[i].exists()) {

System.out.println("The "+f[i].getName()+"exists");

if(f[i].isDirectory()) {

System.out.println(f[i].getParent());

System.out.println(f[i].getAbsolutePath());

System.out.println(f[i].lastModified());

System.out.println(f[i].isAbsolute());

System.out.println(f[i].getName());}

else {

System.out.println(f[i].canRead());

System.out.println(f[i].canWrite());

System.out.println(f[i].getName());

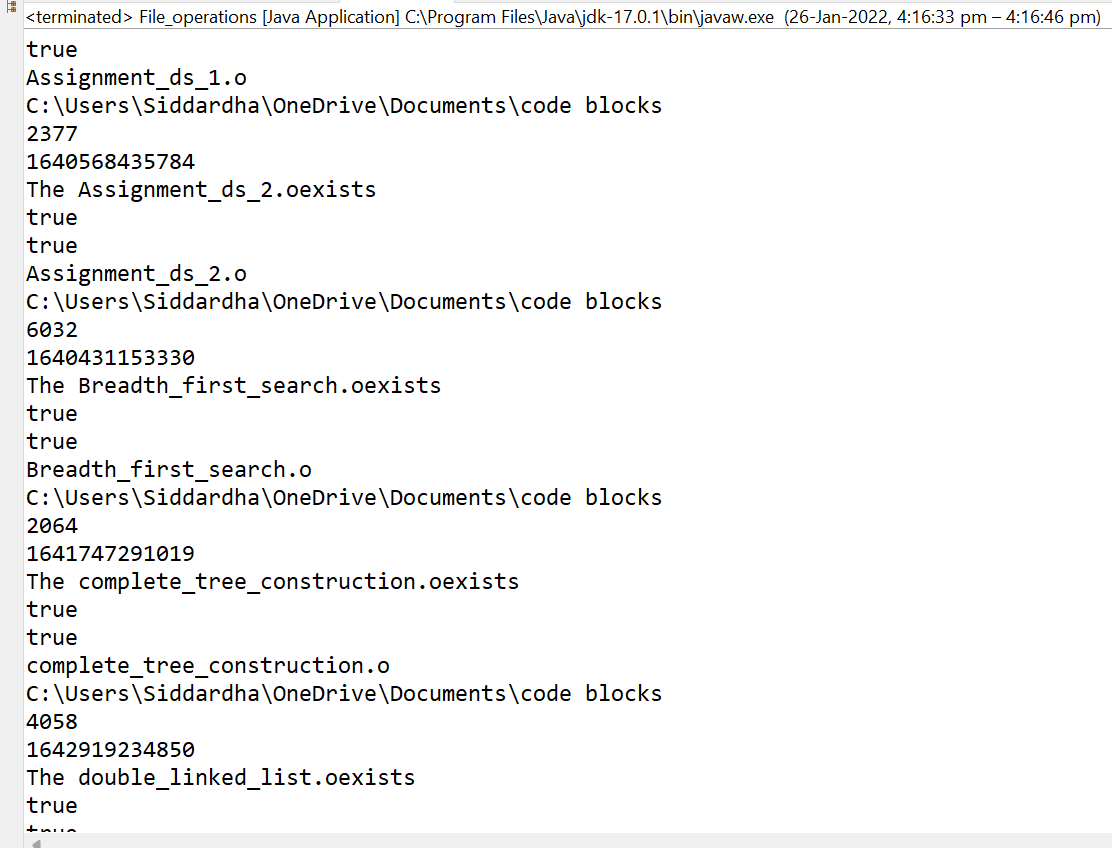
System.out.println(f[i].getParent());

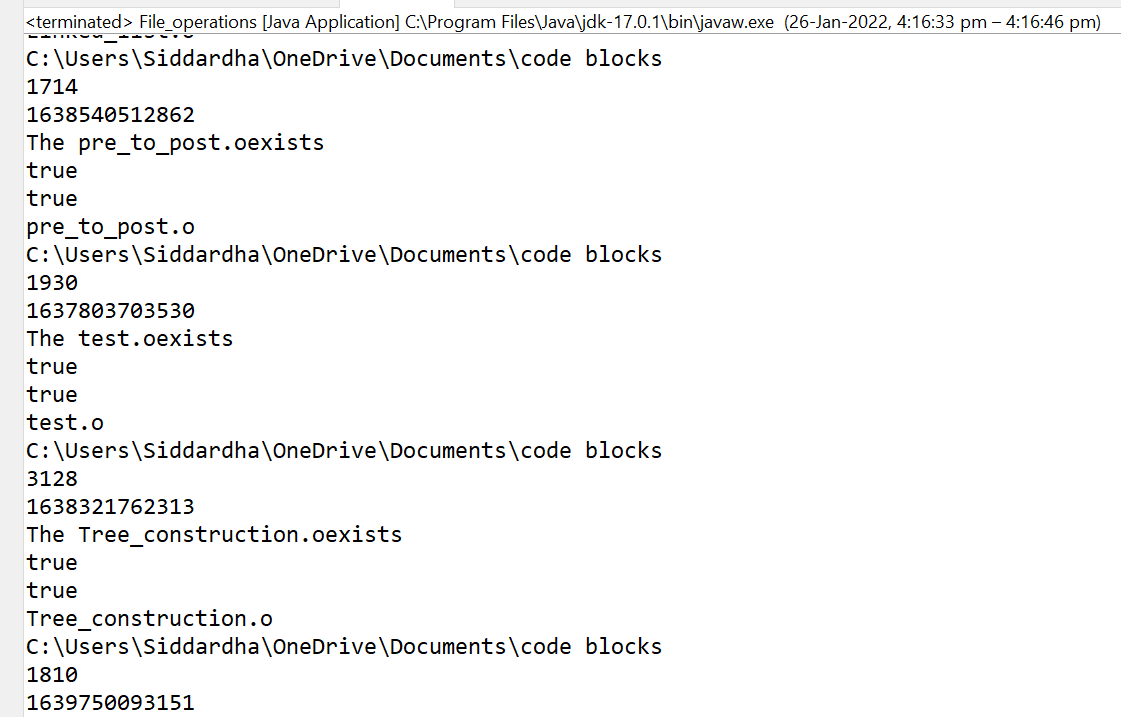
System.out.println(f[i].length());

System.out.println(f[i].lastModified());}}

else {

System.out.println("The file does not exists");}}}}}





**EXPERIMENT NO: 16**

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

**DESCRIPTION:** The Java ActionListener is notified whenever you click on the button or menu item. It is notified against ActionEvent. The ActionListener interface is found in java.awt.event [package](https://www.javatpoint.com/package). It has only one method: actionPerformed().

The actionPerformed() method is invoked automatically whenever you click on the registered component.

The listener interface for receiving action events. The class that is interested in processing an action event implements this interface, and the object created with that class is registered with a component, using the component's addActionListener method. When the action event occurs, that object's actionPerformed method is invoked.

In general, to detect when the user clicks an onscreen button (or does the keyboard equivalent), a program must have an object that implements the ActionListener interface. The program must register this object as an action listener on the button (the event source), using the addActionListener method. When the user clicks the onscreen button, the button fires an action event. This results in the invocation of the action listener's actionPerformed method (the only method in the ActionListener interface). The single argument to the method is an ActionEvent object that gives information about the event and its source.

**SYNTAX:**

**public** **abstract** **void** actionPerformed(ActionEvent e);

**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

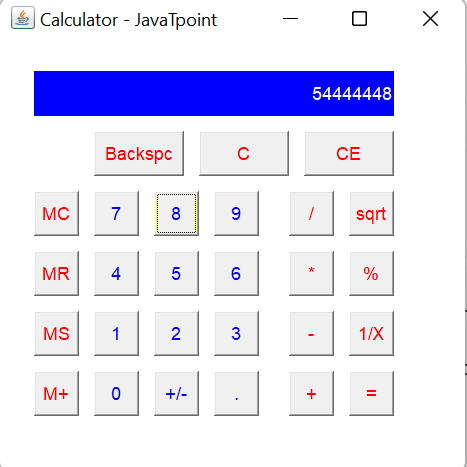
due to the representation of Floating point numbers in computer

is an approximation to the given value that can be stored

physically in memory.

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

**OUTPUT:**



**EXPERIMENT NO: 17**

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

**DESCRIPTION:**

The JMenuBar class is used to display menubar on the window or frame. It may have several menus.

The object of JMenu class is a pull down menu component which is displayed from the menu bar. It inherits the JMenuItem class.

The object of JMenuItem class adds a simple labeled menu item. The items used in a menu must belong to the JMenuItem or any of its subclass.

JMenuBar, JMenu and JMenuItems are a part of Java Swing package. JMenuBar is an implementation of menu bar . the JMenuBar contains one or more JMenu objects, when the JMenu objects are selected they display a popup showing one or more JMenuItems .  
JMenu basically represents a menu . It contains several JMenuItem Object . It may also contain JMenu Objects (or submenu).

1. **JMenuBar() :** Creates a new MenuBar.
2. **JMenu() :** Creates a new Menu with no text.
3. **JMenu(String name) :** Creates a new Menu with a specified name.
4. **JMenu(String name, boolean b) :** Creates a new Menu with a specified name and boolean  
   value specifies it as a tear-off menu or not. A tear-off menu can be opened and dragged away from its parent menu bar or menu.

**SYNTAX:**

**public** **class** JMenuBar **extends** JComponent **implements** MenuElement, Accessible

**public** **class** JMenu **extends** JMenuItem **implements** MenuElement, Accessible

**public** **class** JMenuItem **extends** AbstractButton **implements** Accessible, MenuElement

**PROGRAM:**

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

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

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

**public class Menu 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 Menu();**

**}**

**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 Menu() { // 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**

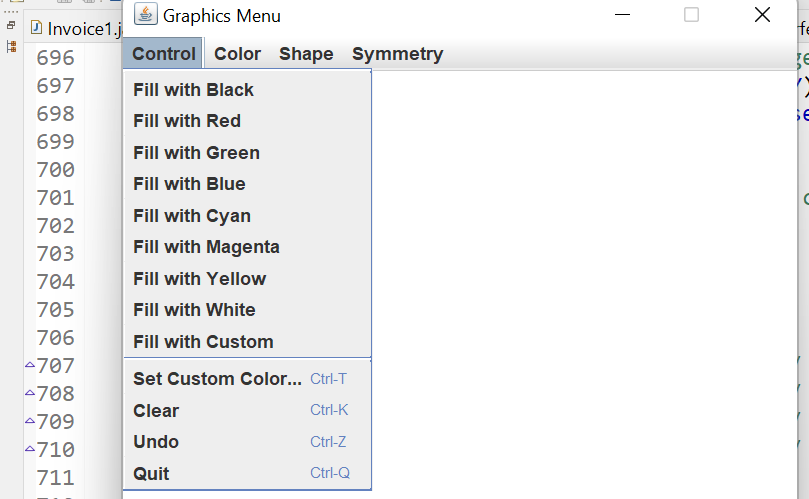
**public void mouseClicked(MouseEvent evt) { } // and MouseMotionListener**

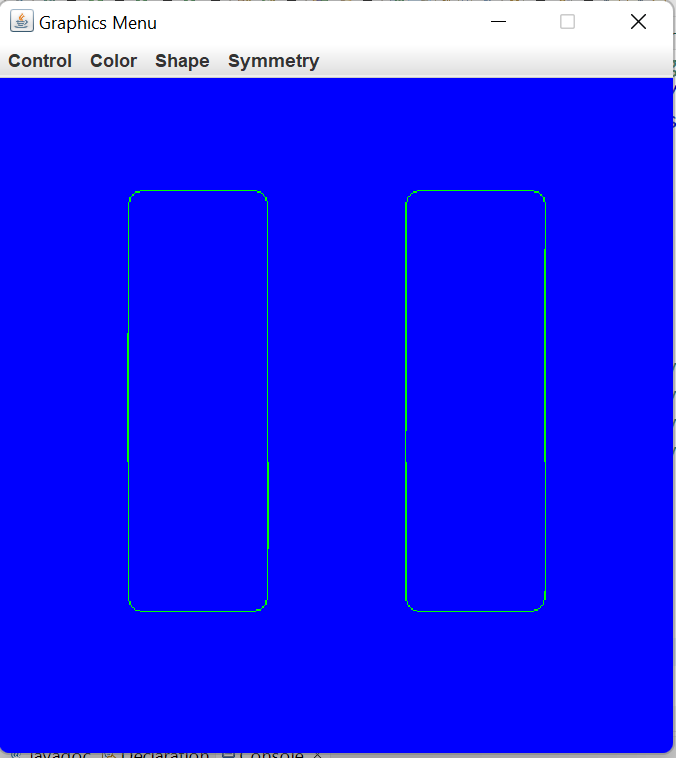
**public void mouseMoved(MouseEvent evt) { } // interfaces).**

**} // end nested class Display**

**} // end class Menu**

**OUTPUT:**

****

****

**EXPERIMENT NO: 18,19**

**AIM:** Write a Java program to implement JTable ,Jtabbed Pane and JTree . [CO4]

**DESCRIPTION:**

The JTable class is a part of Java Swing Package and is generally used to display or edit two-dimensional data that is having both rows and columns. It is similar to a spreadsheet. This arranges data in a tabular form.  
**Constructors in JTable**:

1. **JTable():**A table is created with empty cells.
2. **JTable(int rows, int cols):**Creates a table of size rows \* cols.
3. **JTable(Object[][] data, Object []Column):**A table is created with the specified name where []Column defines the column names.

The JTree class is used to display the tree structured data or hierarchical data. JTree is a complex component. It has a 'root node' at the top most which is a parent for all nodes in the tree. It inherits JComponent class.

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** JTabbedPane **extends** JComponent **implements** Serializable, Accessible, SwingConstant

**public** **class** JTree **extends** JComponent **implements** Scrollable, Accessible

**JTable(Object[][] data, Object []Column)**

**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);

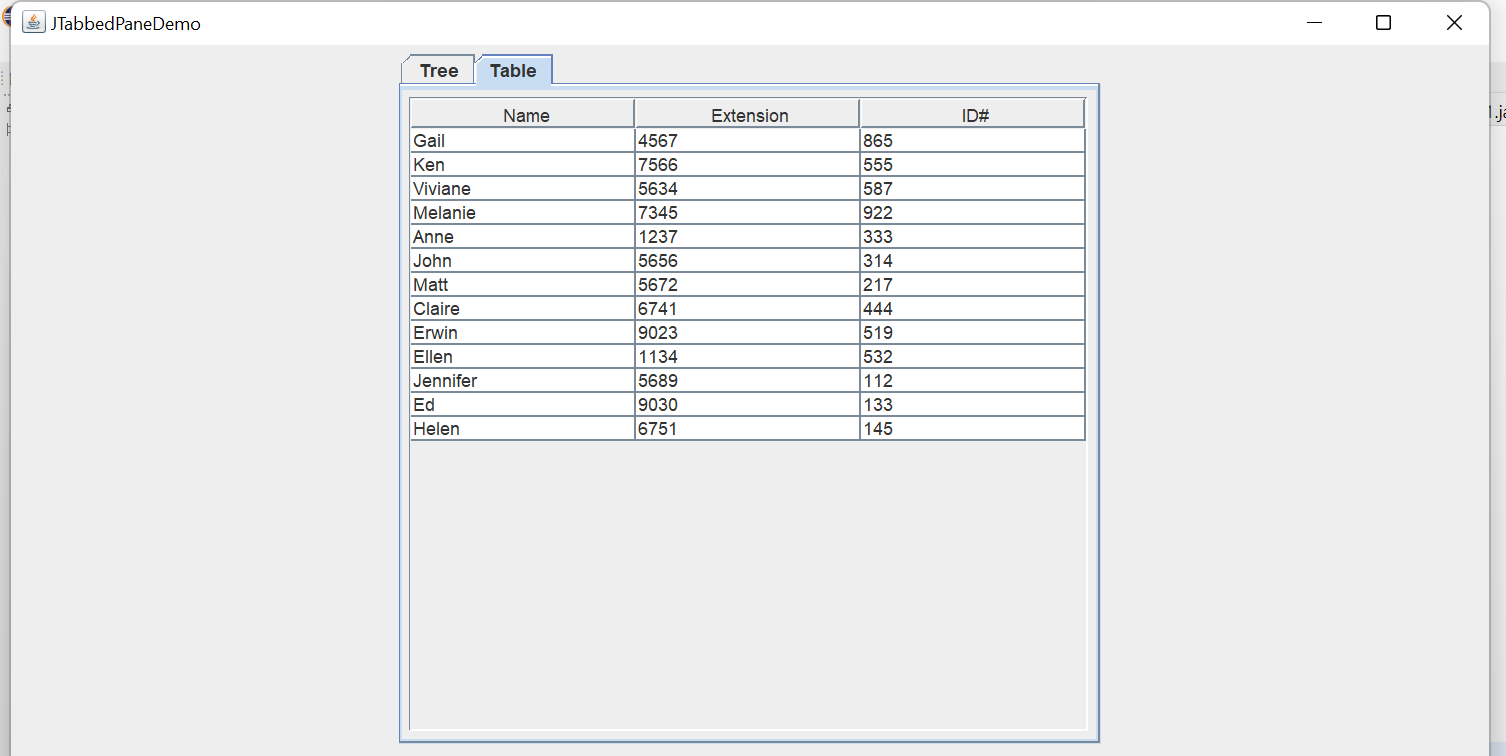
// Add the table to a scroll pane.

JScrollPane jsp = **new** JScrollPane(table);

// Add the scroll pane to the content pane.

add(jsp);}}

**OUTPUT:**



**EXPERIMENT NO: 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()

Public synchronized void close()

**PROGRAM:**

**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);}}}

**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()));

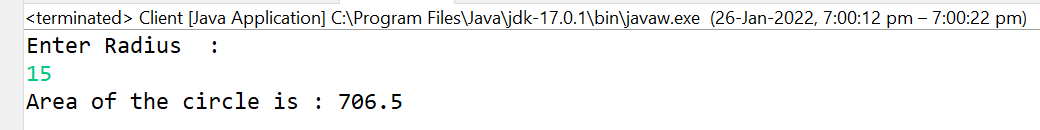
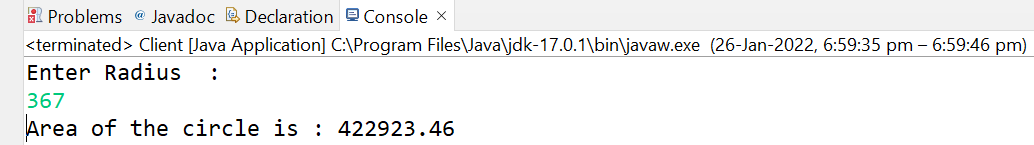
str=br.readLine();

System.***out***.println("Area of the circle is : "+str);

br.close();

ps.close();}}

**OUTPUT:**

****

**Additional Experiments**

**EXPERIMENT NO: 1**

**AIM:** Demonstrate the usage of methods in String

**DESCRIPTION:**

A String that can be modified or changed is known as mutable String. StringBuffer and StringBuilder classes are used for creating mutable strings.

Strings in Java are Objects that are backed internally by a char array. Since arrays are immutable(cannot grow), Strings are immutable as well. Whenever a change to a String is made, an entirely new String is created.

The string can also be declared using **new** operator i.e. dynamically allocated. In case of String are dynamically allocated they are assigned a new memory location in heap. This string will not be added to String constant pool.

* [CharBuffer](https://www.geeksforgeeks.org/tag/java-charbuffer/): This class implements the CharSequence interface. This class is used to allow character buffers to be used in place of CharSequences. An example of such usage is the regular-expression package java.util.regex.
* [String](https://www.geeksforgeeks.org/string-class-in-java/): String is a sequence of characters. In java, objects of String are immutable which means a constant and cannot be changed once created.

**SYNTAX:**

<String\_Type> <string\_variable> = "<sequence\_of\_string>";

**PROGRAM:**

**public** **class** AddExp1 {

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

**char** ch[]= {'x','y','z'};

String s=**new** String(ch);

String s1=**new** String();

**byte** ascii[]= {12,25,45,37,19};

s1="horgwarts";

String s2=**new** String("harry");

String s3=**new** String(ch,1,2);

String s4=**new** String(ascii,2,3);

String s5="grifendor";

String s7="grifendor";

String s6="our house is"+s5+" great";

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

System.***out***.println(s1);

System.***out***.println(s2);

System.***out***.println(s3);

System.***out***.println(s4);

System.***out***.println(s5);

System.***out***.println(s6);

System.***out***.println("length of string VVIT is:"+ s1.length());

Box b=**new** Box();

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

System.***out***.println("Acessing a character at 2 postion in the word VIT :"+"VIT".charAt(2));

**char** ch1[]=**new** **char**[20];

s1.getChars(0,2,ch1,0);

System.***out***.print("Acessing more than one character from string 1:");

System.***out***.println(ch1);

System.***out***.println(s1.toCharArray());

System.***out***.println("Domestic".equals("domestic"));

System.***out***.println("Domestic".equalsIgnoreCase("domestic"));

System.***out***.println("harryportter".startsWith("ha"));

System.***out***.println("madangels".endsWith("ls"));

System.***out***.println(s1==s2);

System.***out***.println(s1==s7);

System.***out***.println("suravamsh".indexOf('v'));

System.***out***.println("vasireddy".lastIndexOf('v'));

System.***out***.println(s2.substring(1,3));

System.***out***.println(s2.concat(s3));

System.***out***.println(s6.replace('w', 's'));

System.***out***.println(s6.toLowerCase());

System.***out***.println(s6.toUpperCase());}}

**class** Box{

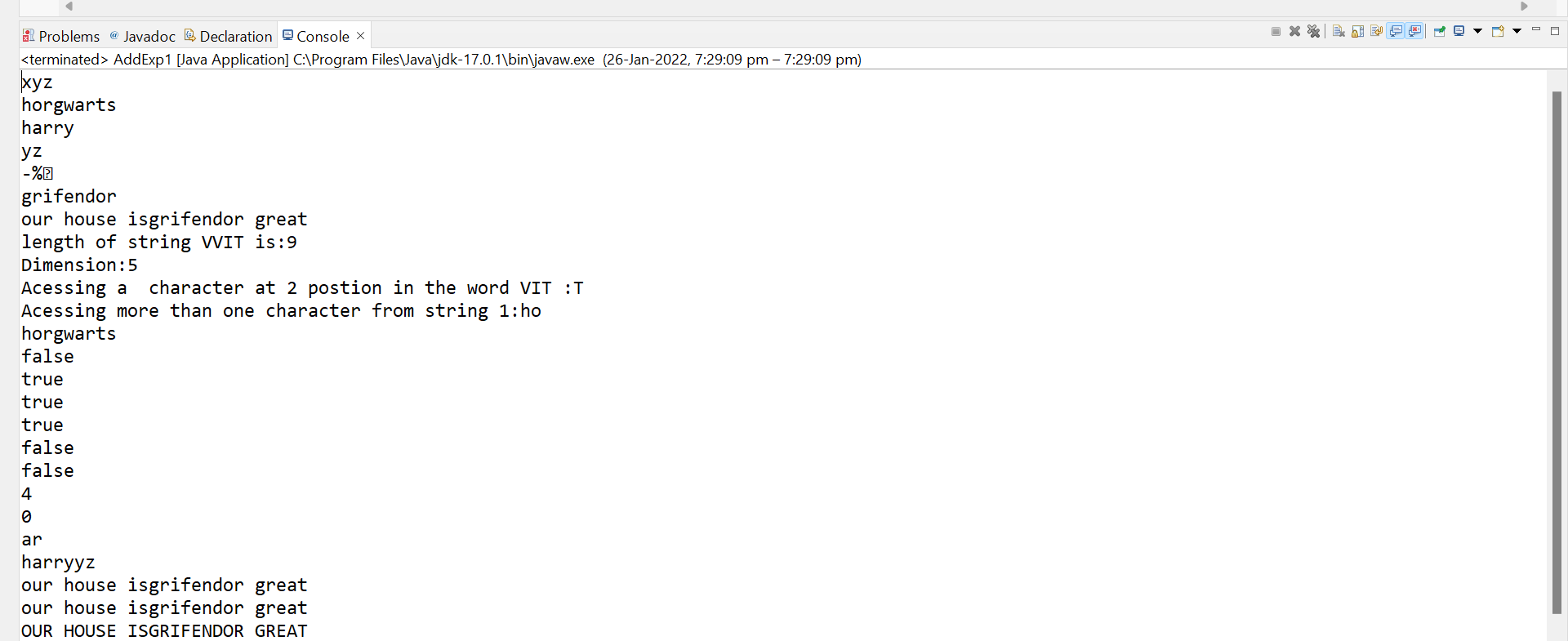
String i="Dimension:";

**int** j=5;

**public** String toString() {

**return** i+ j; }}

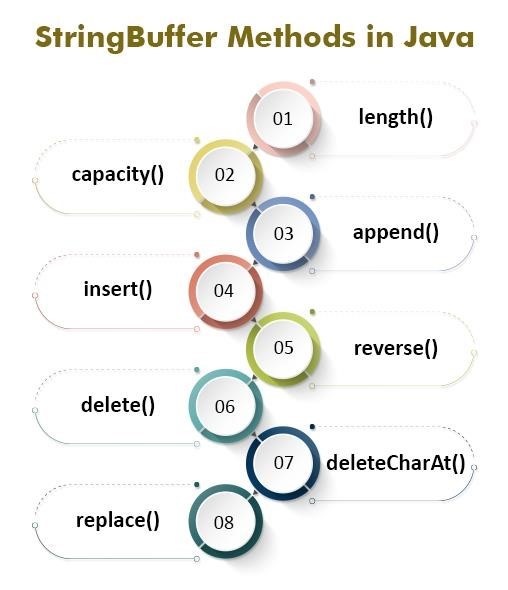
**OUTPUT:**

****

**EXPERIMENT NO: 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** StiringBuffer1 {

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

StringBuffer s1=**new** StringBuffer();

StringBuffer s2=**new** StringBuffer("Harry");

//char[] ch= {'a','c','e'};

//StringBuffer s3=new StringBuffer(ch);

System.***out***.println("length of string harry is:"+s2.length());

System.***out***.println("capacity of string harry is:"+s2.capacity());

s2.ensureCapacity(30);

System.***out***.println("after ensureCapacity capacity of string harry is:"+s2.capacity());

s2.setLength(7);

System.***out***.println("after setting length of string harry is:"+s2.length());

System.***out***.println("Before setting char at 2nd position is:" +s2.charAt(2));

s2.setCharAt(2,'k');

System.***out***.println("after setting char at 2nd position is:" +s2.charAt(2));

**char**[] ch=**new** **char**[10];

s2.getChars(0, 3, ch, 0);

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

s2.append("clg");

System.***out***.println(s2);

s2.insert(5, "study");

System.***out***.println(s2);

System.***out***.println(s2.reverse());

System.***out***.println(s2.delete(3,6));

System.***out***.println(s2.deleteCharAt(8));

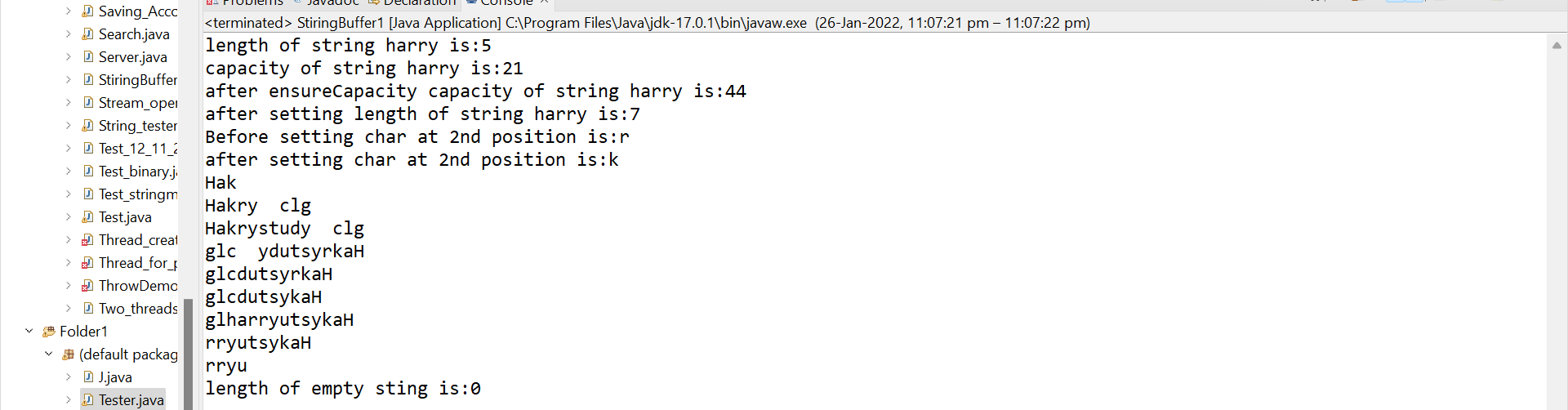
System.***out***.println(s2.replace(2 ,4, "harry"));

System.***out***.println(s2.substring(4));

System.***out***.println(s2.substring(4,8));

System.***out***.println("length of empty sting is:"+s1.length());}}

**OUTPUT:**



**EXPERIMENT NO: 3**

**AIM:** Write a java program to demonstrate Stack implementation.

**DESCRIPTION:**

Stack is a linear data structure that follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out).

Mainly the following three basic operations are performed in the stack:

* **Push:**Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.
* **Pop:** Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.
* **Peek or Top:** Returns the top element of the stack.
* **isEmpty:**Returns true if the stack is empty, else false.

**SYNTAX:**

type var-name[];

OR

type[] var-name;

**PROGRAM:**

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

**class** Stack{

**int** top=-1,y,a;

**int** a1[];

Scanner sc1=**new** Scanner(System.***in***);

Stack(**int** x){

a1=**new** **int**[x];

y=x;}

**void** push() {

//System.out.println(y);

//System.out.println(a1.length);

**if**(isfull()) {

System.***out***.println("\nThe Stack is full and can't push");}

**else** {

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

a=sc1.nextInt();

a1[++top]=a;}}

**void** pop() {

**if**(isempty()) {

System.***out***.println("\n The stack is Empty");}

**else** {

System.***out***.println("\nThe element poped is "+a1[top--]);}}

**boolean** isfull() {

**return** (top==y-1)?**true**:**false**;}

**boolean** isempty() {

**return** (top==-1)?**true**:**false**;}

**void** peek() {

**if**(isempty()) {

System.***out***.println("\nThe stack is Empty");}

**else** {

System.***out***.println("\n The peek value is "+a1[top]);}}

**void** viewstack() {

**if**(isempty()) {

System.***out***.println("\nThe stack is Empty");}

**else** {

**int** t=top;

**for**(t=top;t>=0;) {

System.***out***.println("\t "+a1[t--]);}}}}

**public** **class** Labadd3 {

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

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

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

**int** a,b;

**boolean** c;

System.***out***.println("Enter your stack size");

b=sc.nextInt();

Stack s=**new** Stack(b);

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

System.***out***.println("\n1.push\n2.pop\n3.isfull"

+ "\n4.isempty\n5.peek value\n6.view stack\n7.exit");

System.***out***.println("\n Enter your option");

a=sc.nextInt();

**switch**(a) {

**case** 1:

s.push();

**break**;

**case** 2:

s.pop();

**break**;

**case** 3:

c=s.isfull();

**if**(c) {

System.***out***.println("\nThe stack is full");}

**else** {

System.***out***.println("\nThe stack is not full");}

**break**;

**case** 4:

c=s.isempty();

**if**(c) {

System.***out***.println("\nThe stack is empty");}

**else** {

System.***out***.println("\nThe stack is not empty");}

**break**;

**case** 5:

s.peek();

**break**;

**case** 6:

s.viewstack();

**break**;

**case** 7:

System.*exit*(0);

**break**;}}}}

**OUTPUT:**

****



**EXPERIMENT NO: 4**

**AIM:** Write a java program to demonstrate Queue implementation.

**DESCRIPTION:** A Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO). A good example of a queue is any queue of consumers for a resource where the consumer that came first is served first. The difference between [stacks](https://www.geeksforgeeks.org/stack-data-structure/)and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.

A queue is an object (an abstract data structure - ADT) that allows the following operations:

* **Enqueue**: Add an element to the end of the queue
* **Dequeue**: Remove an element from the front of the queue
* **IsEmpty**: Check if the queue is empty
* **IsFull**: Check if the queue is full
* **Peek**: Get the value of the front of the queue without removing it

**SYNTAX:**

type var-name[];

OR

type[] var-name;

**PROGRAM:**

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

**class** queue1{

**int** front=0,rear=0,a1[],x,a;

queue1(**int** z){

a1=**new** **int** [z];

x=z;}

Scanner sc1=**new** Scanner(System.***in***);

**void** enqueue() {

**if**(isfull()) {

System.***out***.println("\nThe Queue is full and can't enqueue");}

**else** {

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

a=sc1.nextInt();

a1[rear++]=a;

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

**void** dequeue() {

**if**(isempty()) {

System.***out***.println("\n The Queue is Empty");}

**else** {

System.***out***.println("\nThe element dequeued is "+a1[front++]);}}

**boolean** isfull() {

**return** (x-1==rear)?**true**:**false**;}

**boolean** isempty() {

**return** (rear==front)?**true**:**false**;}

**void** viewQueue() {

**if**(isempty()) {

System.***out***.println("\nThe Queue is Empty");}

**else** {

**int** t=front;

**for**(t=front;t<=rear;) {

System.***out***.println("\t "+a1[t++]);}}}}

**public** **class** Labadd4 {

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

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

**int** a,b;

**boolean** c;

System.***out***.println("Enter your Queue size");

b=sc.nextInt();

queue1 s=**new** queue1(b);

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

System.***out***.println("\n1.enqueue\n2.dequeue\n3.isfull"

+ "\n4.isempty\n5.view Queue \n6.exit");

System.***out***.println("\n Enter your option");

a=sc.nextInt();

**switch**(a) {

**case** 1:

s.enqueue();

**break**;

**case** 2:

s.dequeue();

**break**;

**case** 3:

c=s.isfull();

**if**(c) {

System.***out***.println("\nThe queue is full");}

**else** {

System.***out***.println("\nThe queue is not full");}

**break**;

**case** 4:

c=s.isempty();

**if**(c) {

System.***out***.println("\nThe queue is empty");}

**else** {

System.***out***.println("\nThe queue is not empty");}

**break**;

**case** 5:

s.viewQueue();

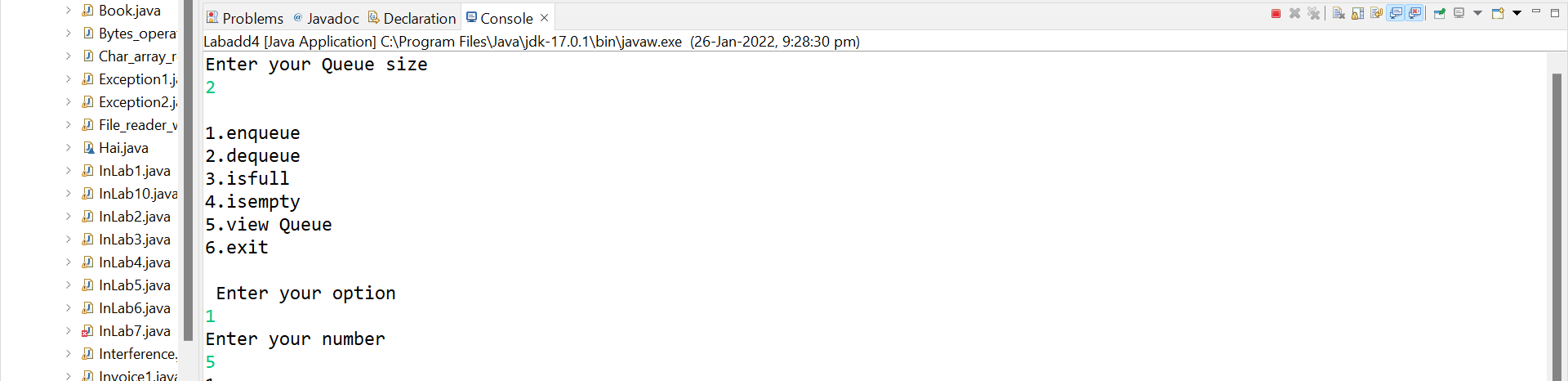
**break**;

**case** 6:

System.*exit*(0);

**break**;}}}}

**OUTPUT:**

****



**EXPERIMENT NO: 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.

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.

The OutputStream is an abstract class that is used to write 8-bit bytes to the stream. It is the superclass of all the output stream classes. This class can't be instantiated; however, it is inherited by various subclasses

**SYNTAX:**

public final class [ByteStreams](https://guava.dev/releases/19.0/api/docs/src-html/com/google/common/io/ByteStreams.html#line.50)

extends [Object](http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html?is-external=true)

**PROGRAM:**

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

**public** **class** AddExp1 {

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

String s="work accomplished is equal to time taken into concentration";

**byte** b1[]=s.getBytes();

**for**(**int** i=0;i<b1.length;i++) {

System.***out***.print((**char**)b1[i]);}

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 bos=**new** ByteArrayOutputStream();

bos.write(b1);

String s1=bos.toString();

System.***out***.printf(s1);

DataOutputStream dos=**new** DataOutputStream(**new** FileOutputStream(

"C:\\Users\\Siddardha\\eclipse-workspace\\Java\\src\\pather"));

dos.writeInt(32);

dos.writeChar('p');

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

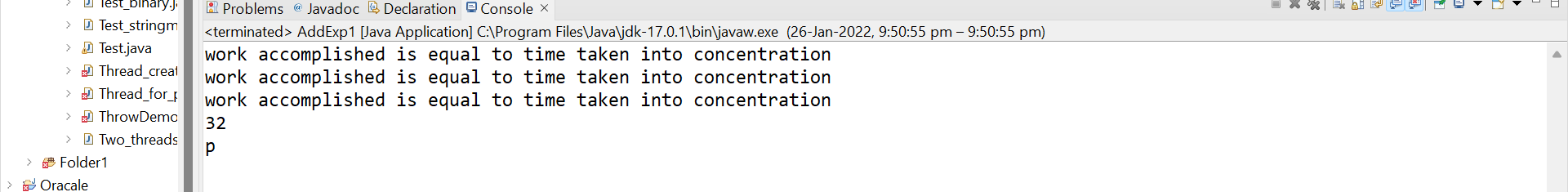
DataInputStream dow=**new** DataInputStream(**new** FileInputStream(

"C:\\Users\\Siddardha\\eclipse-workspace\\Java\\src\\pather"));

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

System.***out***.println(dow.readChar());}}

**OUTPUT:**

****

**EXPERIMENT NO: 6**

**AIM:** Write a java program to demonstrate the usage of CharacterStream 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](https://www.javatpoint.com/java-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.

Writer class is used to write 16-bit Unicode characters to the output stream. The methods of the Writer class generate IOException. Like Reader class, Writer class is also an abstract class that cannot be instantiated; therefore, the subclasses of the Writer class are used to write the characters onto the output stream.

**SYNTAX:**

import java.io.File;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

**AIM:**

work accomplished is equal to time taken into concentration

**PROGRAM:**

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

**public** **class** AddExp1 {

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

File file = **new** File("C:\\Users\\Siddardha\\eclipse-workspace\\Java\\src\\pather");

FileReader reader = **new** FileReader(file);

**char** chars[] = **new** **char**[(**int**) file.length()];

reader.read(chars);

File out = **new** File("C:\\Users\\Siddardha\\eclipse-workspace\\Java\\src\\pather");

FileWriter writer = **new** FileWriter(out);

writer.write(chars);

System.***out***.println("Data successfully written in the specified file");

writer.close();

reader.close();

String s="work accomplished is equal to time taken into concentration";

**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\\Siddardha\\eclipse-workspace\\Java\\src\\pather"));

bw.write(s);

bw.close();

br.close();

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

String s1=" i will win";

**char** c1[]=**new** **char**[s.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;

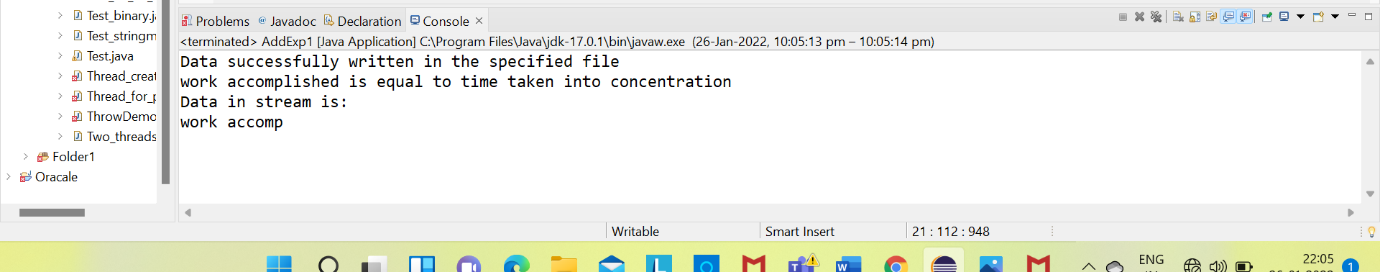
**while**((x1=cr.read())>=0)

System.***out***.print((**char**)x1);

cw.close();

cr.close();}}

**OUTPUT:**



**EXPERIMENT NO: 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)

**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** AddExp1 {

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

DemoofSerialization object = **new** DemoofSerialization(576, "harrypotter");

String filename = "C:\\Users\\Siddardha\\eclipse-workspace\\Java\\src\\pather";

// 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");}

**catch**(ClassNotFoundException ex) {

System.***out***.println("ClassNotFoundException is caught");}}}

**OUTPUT:**

