GUJARAT TECHNOLOGICAL UNIVERSITY

Program Name: Bachelor of Engineering
Level: UG
Subject Code: BE04000231
Subject Name: Object Oriented Programming

w. e. f. Academic Year:	2024-25
Semester:	4
Category of the Course:	Professional Core Course

Prerequisite:	Basics of Programming in C
Rationale:	The course Object-Oriented Programming using Java introduces students to the fundamental principles of object-oriented design such as encapsulation, inheritance, polymorphism, and abstraction, while providing hands-on experience in implementing them using Java. Building on their prior knowledge of procedural programming, this course helps students transition to object-oriented thinking and equips them with the skills to design modular, reusable, and maintainable applications. With Java's wide industry adoption, platform independence, and rich libraries, the syllabus ensures that learners not only gain theoretical understanding but also develop practical, industry-relevant coding skills essential for real-world software development.

Course Outcomes:

Sr. No.	**CO statement**	**Marks% weightage**
CO-1	Describe the structure of Java programs and the foundational elements of the Java programming language, including data types, operators, control structures, memory model, and execution environment.	10
CO-2	Apply Object-Oriented Programming principles such as encapsulation, inheritance, abstraction, and polymorphism to design modular and reusable Java applications using classes, constructors, interfaces, and packages.	40
CO-3	Implement exception handling, multithreading, and file I/O operations to develop robust and efficient Java applications.	30
CO-4	Develop GUI-based applications using JavaFX with proper event handling and layout management.	10
CO-5	Utilize Java Collections and Generics to store, retrieve, and process data efficiently in Java programs.	10

Teaching and Examination Scheme:

Teaching - Learning scheme (in Hours per Semester)					**Total Credits = TH/30**	**Assessment Pattern and Marks**					**Total Marks**
						Theory		**Tutorial / Practical**			
L	**T**	**P**	**PBL***	**TH**		**ESE (E)**	**PA (M)**	**PA (I)**	**PBL (I)**	**ESE (V)**	
45	0	30	45	120	04	70	30	20	30	50	200

*** Problem Based Learning (PBL) aims to accommodate learning beyond syllabus as per clause 9.4 of NBA manual.**

Course Content:

Sr. No.	Content	Total Hrs	% of Weightage
1	**Basic of Java:** Java Platform and Architecture (JVM, JDK, JRE), Java Program Structure, Compilation, and Execution Java program, Data types (byte, short, int, long, float, double, char, boolean, String, Arrays), variables, keywords, literal, operators, operator precedence and associativity, type conversion	3	5
2	**Conditional and looping statements:** simple *if*, *if-else* statement, *else-if* ladder, *switch-case* statement, *for* loop, *while* loop, *do-while* loop, Enhanced for Loop (for-each loop), *break*, *continue*, labelled *break*, labelled *continue*	3	5
3	**Basics of Object Oriented Programming:** Data Encapsulation by defining classes, data members, member functions, Data Abstraction, access specifiers, static data member and static member function, defining constructors, objects and references, *this* keyword, returning and passing objects, array of objects, static block, instance block, inner class	7	20
4	**Inheritance, Polymorphism and Wrapper classes:** Inheritance, types of inheritance, *super*, *this* and *final* keywords, method overloading, method overriding, dynamic binding, casting objects, instanceof operator, protected data members, String, StringBuffer and StringBuilder class, Wrapper classes for Primitive types, autoboxing/unboxing, Collection framework, Java API Packages, User-defined packages	8	20
5	**Interface, Abstract class and Exception Handling:** Interface, Partial interface, Multiple inheritance using interface, Abstract class, Abstract methods, Handling Exception using *try*, *catch* and *finally* keywords, Throwing exception using *throw*, checked and unchecked exception, *throws* keyword, Creating user-defined exceptions	7	20
6	**Concurrency control:** Thread life cycle, Creating threads using *Thread* class and *runnable* interface, Thread methods, Thread synchronization using *synchronized* keyword	5	10
7	**I/O Management and Generics:**, *File* class, Writing, reading and Random Access files, Purpose of Generics, Defining Generic classes and methods, Generics with Collections like ArrayList, HashMap, etc	6	10
8	**Designing GUI Applications using JavaFx:** Basic structure of JavaFx application, Layout Panes, UI Components, *Color* and *Font* classes, Event Handling	6	10
		45	100

Suggested Specification table with Marks (Theory): (For B.E. only)

Distribution of Theory Marks					
R Level	U Level	A Level	N Level	E Level	C Level
10	**20**	**40**	**00**	**00**	**00**

R: Remembrance; U: Understanding; A: Application, N: Analyze and E: Evaluate C: Create and above Levels (Revised Bloom's Taxonomy)

Note: This specification table shall be treated as a general guideline for students and teachers. The actual distribution of marks in the question paper may vary slightly from above table.

Reference Books:

1. **Java: The Complete Reference** – Herbert Schildt, McGraw Hill Education
2. **Head First Java** – Kathy Sierra, Bert Bates, O'Reilly Media.
3. **Core Java: An Integrated Approach** – R. Nageswara Rao, Dreamtech Press
4. **Thinking in Java** – Bruce Eckel, Pearson Education
5. **Object-Oriented Programming with Java** – C. Thomas Wu, Tata McGraw Hill
6. **Programming with Java** – E. Balagurusamy, McGraw Hill Education

Suggested MOOC Courses:

1. (https://onlinecourses.nptel.ac.in/noc22_cs47/preview)
2. Java by Prof. Kannan Moudgalya, Indian Institute of Technology Bombay (https://onlinecourses.swayam2.ac.in/aic20_sp13/preview)
3. Object oriented Programming in Java by Mrs. Priyanka Sharma, Manav Rachna International Institute of Research and Studies (https://online-degree.swayam2.ac.in/mri22_01_d03_s1_cc3/preview)

List of Experiments:

Sr. No	**Practical**	**CO**
	Practical Set-1 (Basics of JAVA)	CO1
1.1	Develop a Java program that prompts the user to enter a distance in meters. Your program should then convert this distance to feet (1 meter = 3.28084 feet) and display the result formatted to two decimal places.	
1.2	Write a Java program to solve a system of two linear equations with two variables (e.g., ax + by = e and cx + dy = f). Prompt the user to enter the coefficients a, b, c, d, e, f. Calculate and display the values of x and y using Cramer's rule. Include error handling for cases where the denominator is zero.	

	Cramer's rule : D=ad−bc, D_x=ed−bf, D_y=af−ce, x=D_x/D, y=D_y/D	
1.3	Write a Java program that prompts the user to enter a single letter (character). Determine whether the entered character is a vowel (a, e, i, o, u, case-insensitive) or a consonant, and display the result.	
1.4	Develop a Java application that calculates a person's Body Mass Index (BMI). The program should ask the user for their weight in pounds and height in inches. Convert these values to kilograms and meters respectively (1 pound = 0.45359237 kg, 1 inch = 0.0254 meters) and then calculate BMI (weight in kg / (height in meters)^2). Display the calculated BMI.	
1.5	Simulate a simple ATM or cashier. Given an integer amount to be dispensed (e.g., 787), calculate and display the minimum number of currency notes of denominations 100, 50, 10, 5, 2, and 1 that would be given to the user.	
1.6	Write a Java program that accepts a five-digit integer from the keyboard. Your program should then create a new number by adding one to each digit of the input number. For example, if the input is 12391, the output should be 23402 (note: 9+1=10, so it becomes 0 with a carry)	
1.7	Write a program that takes the lengths of three sides of a triangle as input. Calculate and print the area of the triangle. Ensure that the program validates if the given side lengths can actually form a triangle (sum of any two sides must be greater than the third side) before calculating the area (use Heron's formula: Area = sqrt(s*(s-a)*(s-b)*(s-c)) where s = (a+b+c)/2).	
1.8	Write a Java program that accepts two numbers as command-line arguments. Convert these arguments to appropriate numeric types (e.g., int or double), perform a simple calculation (e.g., sum or product), and print the result to the console.	
	Practical Set-2 (Basic OOP concepts-1)	**CO2**
2.1	Define a Java class named *Rectangle*. It should have two double data fields: *width* and *height*, both with a default value of 1. Implement a no-argument constructor and a constructor that takes *width* and *height* as parameters. Include methods *getArea()* and *getPerimeter()* that return the calculated area and perimeter respectively.	
2.2	Create an *Employee* class with private instance variables for *employeeName* (String) and *employeeSalary* (double). Implement public methods *readEmployeeData()* (to take input from the user) and *displayEmployeeData()* (to print the employee's name and salary). Demonstrate object creation and method invocation in a *main* method.	
2.3	Design a class *Time* with *hours* (int) and *minutes* (int) as data members. Include method *setTime(int h, int m)* to initialize the time and *displayTime()* to display the time. Implement a method *addTime(Time t1, Time t2)* that takes two Time objects as arguments, adds their hours and minutes, and stores the result in the calling object. Do not use constructors for initialization in this specific practical.	
2.4	Create a *Point* class representing a 2D point (x, y). Implement a default constructor that initializes both *x* and *y* to 5. Provide a parameterized constructor to initialize *x* and *y* with user-supplied values. Also, implement a copy constructor to create a new *Point* object as a copy of an existing Point object. Include a *display()* method to show the point's coordinates and write a *main* method to test all constructors and the display functionality.	
2.5	Define a Java class named *Rectangle*. It should have two double data fields: *width* and	

	height. In your main method, create two Rectangle objects: one with *width* 4 and *height* 40, and another with *width* 3.5 and *height* 35.9. For each rectangle, display its *width*, *height*, calculated area, and perimeter. Then, compare the two rectangles based on their areas and print which one has a larger area.	
2.6	Create a class *BankAccount* with *accountId* (String), *accountHolderName* (String), and *balance* (double) as instance variables. Include methods *assignValues()* (for initialization) and *displayValues()*. Implement a search function that takes an *accountId* as input and, if found within an array of *BankAccount* objects, displays the details of that specific account. In your main method, create an array of at least five BankAccount objects and demonstrate adding, displaying, and searching for accounts.	
2.7	Write a program for billing system for a shopping mall. Create a class *BillGenerator* that uses method overloading to generate bills based on customer type: *generateBill(int itemTotal)*: For regular customers, apply no discount. *generateBill(int itemTotal, int discount)*: For privileged customers, apply flat discount in rupees. *generateBill(int itemTotal, double discountPercent)*: For festive offers, apply percentage discount. Write a program to display the final bill amount using appropriate overloaded method based on customer category.	
2.8	A bank wants to offer a facility to calculate EMI (Equated Monthly Installment) for different types of loans. Design a class *LoanCalculator* with the following overloaded methods: *calculateEMI(int principal, int time, float rate)*: For home loans *calculateEMI(double principal, int time, double rate)*: For vehicle loans *calculateEMI(int principal, int time)*: For short-term personal loans with a fixed interest rate of 10% Demonstrate the use of all three methods in the main method by calculating EMIs for different loan types.	
	Practical Set-3 (Basic OOP concepts-1)	**CO2**
3.1	Create a Java class named *University* with a static data member *totalStudents* to keep track of the number of student objects created. Implement a static method *getTotalStudents()*. Also, include a static block to initialize a static variable (e.g., universityName) and an instance block to print a message when an object is created. Demonstrate their execution order.	
3.2	Design a class *BankAccount* with *account_holder_name* and *balance*. Use a static variable *interest_rate* (same for all accounts). Include methods to calculate and display the interest earned. Update interest rate using a static method.	
3.3	Write a Java program to model a college admission system using the concept of inner classes. Create an outer class named *College* that stores the *collegeName* as a data member and initializes it through a constructor. Within the *College* class, define a non-static inner class named *Admission* that contains student-specific details such as the *studentName* and the course they are enrolling in. The inner class should have methods to accept and display student information, and it should also be able to access and display the *collegeName* stored in the outer class. In the *main* method, create an object of the *College* class by passing the	

	collegeName, and then use this object to create an instance of the inner *Admission* class. Invoke appropriate methods to input the student's name and course, and then display the complete admission details, including the college name.	
3.4	Create a Java method *isValidPassword*(String password) that checks if a given string is a valid password based on the following rules: ● It must have at least eight characters. ● It must consist only of letters and digits. ● It must contain at least two digits. The program should prompt the user to enter a password and display "Valid Password" or "Invalid Password" accordingly.	
3.5	Write a Java program that demonstrates method overloading to calculate the volume of different 3D shapes. Implement overloaded methods named *calculateVolume()* for a Cube (takes one side length), a RectangularCube (takes length, width, height), and a Sphere (takes radius).	
	Practical Set-4 (Inheritance & Polymorphism)	**CO2**
4.1	Design a base class *Shape* with two double data members *d1* and *d2* to store dimensions. Include a method *getData(double d1, double d2)* to initialize these dimensions. Create two derived classes, *Triangle* and *Rectangle*, which inherit from *Shape*. Each derived class should have its own method to calculate its specific area.	
4.2	Define a base class *BankAccount* with common attributes like *accountNumber*, *accountHolderName*, and *balance*. Then, define two subclasses: *SavingAccount* and *FixedDepositAccount*, which inherit from *BankAccount*. Implement basic operations like *openAccount()*, *deposit()*, *checkBalance()*, and *withdraw()* in *BankAccount*. The *SavingAccount* should include a *calculateInterest()* method specific to savings accounts, and *FixedDepositAccount* should have a *maturityAmount()* method considering fixed deposit terms.	
4.3	Create a base class named *Employee* that contains a method *displayDetails()* which prints general employee details such as *name* and *department*. Now create a subclass *Manager* that inherits from *Employee* and overrides the *displayDetails()* method to include additional information such as the manager's team size or project name. In the *main* method, create objects of both *Employee* and *Manager* classes and call the *displayDetails()* method using each object to show how Java determines which version of the method to execute at runtime.	
4.4	Create a base class named *Vehicle* that contains common attributes such as *vehicleNumber*, *brand*, and *fuelType*. Include a constructor to initialize these fields and a method *displayDetails()* to print them. Derive a subclass *Car* from *Vehicle* which adds attributes such as *numberOfSeats* and *ACavailable* (boolean). Override the *displayDetails()* method to include the car-specific details, and use the *super* keyword to invoke the parent class constructor and methods. Further, derive another subclass *ElectricCar* from *Car* that includes attributes such as *batteryCapacity* and *chargingTime*, and again override the *displayDetails()* method to include electric car-specific details. Demonstrate constructor chaining, method overriding, use of protected access specifier for inherited members, and *instanceof* operator to check object type at runtime. In the *main()* method, create objects of all three classes and display their details using overridden methods. Also, use upcasting (Vehicle v = new Car(...)) and downcasting with *instanceof* check to access subclass-	

	specific features.	
4.5	Write a Java program to demonstrate method overriding in an online payment system. Create a superclass *Payment* with a method *processPayment(int amount)*. In the subclass *CreditCardPayment*, override the method to print "Payment of Rs.<amount> done using Credit Card". In the subclass *UPIPayment*, override the method to print "Payment of Rs.<amount> done using UPI". Accept user choice and amount, and display the payment result using method overriding.	
Practical Set-5 (Interface, Abstract Class and Package)		**CO2**
5.1	Given an interface *Classify* with a method *String getDivision(double average)*. Implement this *getDivision* method in a class *Result* such that it returns "First Division" if the average is 60 or more.	
5.2	Write the Java code for an interface named *Exam* which declares a single abstract method *boolean isPassed(int mark)*. This method should take an integer *mark* as an argument. Write the Java code for another interface named *Classify* which declares a single abstract method *String getDivision(double average)*. This method should take a double *average* as an argument. Create a class named *Result* that implements both the *Exam* and *Classify* interfaces. Provide concrete implementations for *isPassed()* and *getDivision()* methods. In your *main* method, create an instance of *Result*, set some *marks* and *average*, and demonstrate the use of both interface methods.	
5.3	Design a Java program for an Online Order Processing System using partial interface implementation. Create an interface *Order* with three methods: *placeOrder(String item, int qty)* *cancelOrder(int orderId)* *generateBill()* Create an abstract class *PartialOrder* that implements the *Order* interface but provides implementation only for *placeOrder()* (storing order details). Create a concrete class *FinalOrder* that extends *PartialOrder* and implements the remaining methods *cancelOrder()* and *generateBill()*. Accept user input for order details and allow user to either generate a bill or cancel the order.	
5.4	Write a Java program to create an abstract class Vehicle with: ● An abstract method fuelType() that returns the type of fuel used. ● An abstract method noOfWheels() that returns the number of wheels. Create two subclasses: ● Car that uses Petrol/Diesel and has 4 wheels. ● Bike that uses Petrol and has 2 wheels. In the main method, create objects of Car and Bike, and display their fuel type and number of wheels.	
5.5	Write a Java program using packages to generate a mark sheet for students. Create a package *student* that contains a class *Student* with the following: ● Data members: *rollNo, name*. ● A constructor to initialize student details. ● A method *displayStudent()* to display student information. Create another package *exam* that contains a class *Result* which: ● Extends the *Student* class.	

	● Has data members: *marks1, marks2, marks3*. ● A method *displayResult()* that prints the student's mark sheet including total and average marks. In the main method (inside the exam package), create a student with marks and display the mark sheet.	
5.6	Write a Java program using four different packages to demonstrate the use of access specifiers. ● Package *apack*: ○ Define a class *A* with three variables: ▪ *public int pubVar* ▪ *protected int protVar* ▪ *private int privVar* ○ Provide a constructor to initialize them. ● Package *bpack*: ○ Define a class *B* that extends *A*. ○ Create a *display()* method that tries to access variables of *A* using inheritance. ● Package *cpack*: ○ Define a class *C* with a *display()* method. ○ Inside *display()*, create an object of class *A* and try to access its variables. ● Package *dpack*: ○ Define a class *ProtectedDemo* with *main()*. ○ Create objects of class *B* and class *C*. ○ Call their respective *display()* methods to show which variables are accessible and which are not.	
	Practical Set-6 (Exception Handling)	**CO3**
6.1	Take the value of denominator and numerator from user using command-line argument. Implement the concept of exception handling to manage all possible run-time error.	
6.2	Write a Java program to create a class *VotingApp* where: ● The method *checkEligibility(int age)* checks if a person is eligible to vote. ● If age < 18, explicitly throw the predefined exception IllegalArgumentException with the message "Age must be 18 or above to vote". In the main method, test the method with different age inputs. ● Use a try-catch-finally block to handle exceptions. ● The finally block should always print "Validation process completed"	
6.3	Write a Java application that defines a method *average(String[] values)* which: ● Accepts an array of strings as an argument. ● Converts each string element into a double and computes the average. ● If any array element is null, the method should throw a *NullPointerException*. ● If any element is not a valid number (e.g., "abc"), it should throw a *NumberFormatException*. ● Include the throws clause in the method declaration. ● In the main method, demonstrate the working of this program with valid and invalid inputs using *try-catch-finally*	

6.4	Write a Java program to create a Banking Application using classes and exception handling. Create a *class BankAccount* with: ● A constructor to initialize the balance with 1000.00. ● A method *deposit(double amount)* to add money to the account. ● A method *withdraw(double amount)* that subtracts money from the account. ● If withdrawal is more than the available balance, it should throw a custom exception called *NotSufficientFundException*. In the main method: ● Deposit 1000.00. ● Perform three withdrawals: 400.00, 300.00, and 500.00. ● The last withdrawal should throw the exception with the message "Not Sufficient Fund".	
6.5	Define a custom exception class *BookNotAvailableException* that extends *Exception*. ● Create a class *Library* with: ● An instance variable *availableBooks* (integer). ● A method *issueBook(int count)* that: ○ If *count <= availableBooks*, reduce the number of books and display "Book issued successfully". ○ Otherwise, throw *BookNotAvailableException* with the message "Requested books not available". In the *main()* method: ● Initialize the library with 3 available books. ● Try issuing 2 books (valid). ● Then try issuing 2 more books (should throw the custom exception).	
	Practical Set – 7 (Concurrency Control – Threading)	**CO3**
7.1	Write a Java program that creates two threads: ● First thread prints numbers from 1 to 10 at the interval of 1 second. ● Second thread prints numbers from 11 to 20 at the interval of 500 ms. Run both threads and display the output.	
7.2	Write a Java program where Thread T1 prints 1 to 100, T2 prints 101 to 200 and T3 prints 201 to 300. Initiate execution of all the three threads but ensure that numbers should be printed sequentially.	
7.3	Write a Java program where two threads print multiplication tables (e.g., Table of 5 and Table of 7). Use a synchronized method so that table outputs do not mix and remain consistent.	
7.4	Write a Java program where: ● **Thread 1** computes the sum of numbers from 1 to 1000. ● **Thread 2** computes the sum of numbers from 1001 to 2000. ● Both threads should run **in parallel**. ● Each thread should **return its computed sum** to the main method (e.g., using a getter method or storing result in a shared variable). ● The **main method** should wait for both threads to finish using join(), then combine the two partial sums and print the **final result**.	
	Practical Set – 8 (File I/O)	**CO3**

8.1	Write a program that will count the number of characters, words, and lines in a file. Words are separated by whitespace characters. The file name should be passed as a command-line argument.	
8.2	Create a file named *students.txt*. Write at least three student records (roll number, name, marks) into the file. Read the file content and display all student records on the console. Handle exceptions like IOException properly using try-catch-finally.	
8.3	Write a Java program that reads a text file named *data.txt*. The program should count and display: The total number of lines, The total number of words, The total number of characters (excluding spaces and newline characters), Use FileReader / BufferedReader for reading the file. Handle exceptions like FileNotFoundException and IOException.	
8.4	Write a Java program that takes **file name(s) and commands from the command line arguments** and performs the following operations: • **Copy** a file from source to destination. • **Delete** a given file. • **Rename** a file. After performing the operation, the program should print the following **file properties**: • File name • Absolute path • File size (in bytes) • Whether the file is readable/writable • Last modified date Use File class methods for properties, and handle exceptions such as IOException and FileNotFoundException.	
Practical Set – 9 (Collection Framework and Generics)		**CO5**
9.1	Write a Java program that uses an *ArrayList<Integer>* to store marks of students. Perform the following operations: • Add at least 5 marks. • Display all marks. • Find and display the highest and lowest marks using Collections.max() and Collections.min().	
9.2	Write a program that accepts a sentence from the user and counts the frequency of each word using a HashMap<String, Integer>. Display the results in the format: **Input:** "Java is fun and Java is powerful" **Output:** Java -> 2 is -> 2 fun -> 1 and -> 1 powerful -> 1	
9.3	Write a Java program to simulate a Music Playlist using *LinkedList<String>*. Perform the following operations: 1. Add songs to the playlist. 2. Display the full playlist.	

	3. Play the first song (remove from front). 4. Skip the last song (remove from end). 5. Display the updated playlist after each operation.	
9.4	Create a generic class *Box<T>* with a method *addItem(T item)* that stores items in an *ArrayList<T>*. Demonstrate the class by: • Creating a *Box<String>* for names. • Creating a *Box<Integer>* for roll numbers. • Display stored items for both.	
9.5	Write a generic method *searchElement* that accepts a *LinkedList<T>* and an element T to search. Return true if the element exists, otherwise false. • Test with *LinkedList<Integer>* for roll numbers. • Test with *LinkedList<String>* for names.	
9.6	Write a generic method *sortList(List<T> list)* that sorts elements of an *ArrayList<T>*, where T extends Comparable<T>. Demonstrate with: • An *ArrayList<Integer>* of numbers. • An *ArrayList<String>* of names. • Display the list before and after sorting.	
	Practical Set – 10 (JAVA FX)	**CO4**
10.1	Write a JavaFX program that displays five Text nodes vertically (stacked). For each Text: • Use font Times New Roman, bold + italic, size 22 px. • Assign a random color and random opacity (between 0.3 and 1.0) to each text. • Center the texts horizontally in the window and add spacing between them. UI / Classes to use: VBox, Text, Font, Color, Random. On launch the window shows five vertically arranged lines (e.g., “Text 1”, … “Text 5”) each with different color & opacity and same font style.	
10.2	Create a JavaFX app that shows a circle which continuously bounces (left↔right) inside the scene. Provide: • A Start and Stop button. • A Slider to control the ball speed (min: slow, max: fast). • When window is resized, the ball should keep bouncing within new bounds. UI / Classes to use: Pane or AnchorPane, Circle, TranslateTransition or AnimationTimer, Button, Slider. Following behavior is expected: • Press Start → ball moves horizontally and reverses on boundaries. • Move Slider while running → ball speed updates. • Press Stop → animation pauses.	
10.3	Design a registration form UI with fields: *Roll_NO* (numeric), *Name*, *Age* (numeric), *Email* and a Submit button. Requirements: • Validate inputs on submit: ○ *RollNo* and *Age* must be integers. ○ *Email* must contain @ and . (basic check). • On success show a confirmation Alert with entered data. • On validation failure show an error alert describing the issue. UI / Classes to use: GridPane, TextField, Button, Alert, FileChooser,	

	FileWriter/BufferedWriter.	
10.4	Write a JavaFX program that displays a bar chart to represent the percentage distribution of overall grades using Rectangle shapes. • Projects: 20%, displayed in Red • Quizzes: 10%, displayed in Blue • Midterm Exams: 30%, displayed in Green • Final Exam: 40%, displayed in Orange Requirements: 1. Each category should be displayed with a labeled bar. 2. Bars should be proportional in height to the percentage. 3. Use the Rectangle class to create the bars. 4. Display the labels (e.g., "Projects — 20%") under each bar. 5. Arrange the bars horizontally in the scene using an HBox or Pane.	

Major Equipment:
List of Open Source Software/learning website:

IDEs for Core Java

1. Eclipse IDE – https://eclipse.org/downloads/
2. NetBeans IDE – https://netbeans.apache.org/download/
3. IntelliJ IDEA Community Edition – https://www.jetbrains.com/idea/download/
4. Visual Studio Code - https://code.visualstudio.com/

Libraries & Tools (Core Java Utilities)

1. Apache Commons – https://commons.apache.org/
2. Gson (JSON by Google) – https://github.com/google/gson
3. JUnit (Testing) – https://junit.org/junit5/

JavaFX & GUI

1. OpenJFX (JavaFX official) – https://openjfx.io/
2. Scene Builder (Gluon) – https://gluonhq.com/products/scene-builder/
3. ControlsFX – https://github.com/controlsfx/controlsfx

- **List of suggested activities for Problem Based Learning:**

Sl. No.	**Name of the activity**	**No. of hours**	**Evaluation Criteria**
1	Assignment writing. Numerical based assignment is preferable.	5 assignments of 3h each. Total = 15h	Based on the assignment submitted.

2	Problem solving/Coding using C, C++, Python, SCILAB, MATLAB, MS-EXCEL or any other relevant software	5 small coding-based problems of 3h each. Total = 15h	Based on the coding solution submitted.
3	Technical Video based learning related to the subject	Duration of video = 5h Report preparation & Presentation = 10h Total = 15h	Report /presentation based on the video learning outcomes.
4	Discussion on research paper based on relevant subject	3 research paper = 15h	Summarize research paper and evaluation critical parameters
5	Poster/chart/power point preparation on technical topics	Duration = 10 h	Based on poster/chart preparation and presentation skills
6	Application/Software development	Duration = 15 h	Depending on the complexity of the Application/Software
7	Group Discussion on emerging/trending technical topics based on subject	Duration = 1 h each	Based on performance in group discussion, technical depth, knowledge etc.

8	Seminar / Presentation	Duration for study and preparation=5h
		Report writing=3h
		Presentation=2h
		Total=10h
			Topic can be selected technical content beyond syllabus

9	Real world case studies-based learning	Duration of data collection/study = 5h Report preparation = 10h Total = 15h	Based on in-depth study, technical depth, data collected, fact finding, etc.
10	Working/non-working model on technical topics	Working = 12 h Non- working = 8 h	Based on inter department/external evaluation
11	Self-learning on-line course	Minimum duration of the course should be 15h.	Examination based assessment at the end of course. Based on the certificate produced.
12	Complex problem solving	Maximum 3 problem. Study of the problem and solution finding, Total = 15h	Based on the depth of the solution submitted.
13	Industry/Research laboratory visit	Visit = 5h, Report preparation = 5h Total = 10h	Based on report submitted. Report should contain observations and calculations based on

			industry/ lab data.
14	Videos on Industrial safety aspects based on subject	Duration of video = 5h Report preparation = 5h Total = 10h	Based on quiz/report submitted
15	Industrial exposure for 2-3 days to observe and provide tentative solutions on society/environment /health/any other issue	Duration = 15 h for industrial exposure Problem identification and tentative solution = 10 h Total = 20 h	Based on evaluation of critical problems and solutions

Note:

- All the suggested activity should be related to the subject.
- Min 3 activities must be carried out as per the availability of faculties and students.
- The number of hours is suggestive. Faculty can sub-divide the number of hours based on the activity. However, total number of hours is fixed.
- Rubrics for the evaluation can be prepared by the faculty.

****