	School of Engineering & Technology	
	Department: SOET	**Session: 2025-26**
	Programme: B.Tech. , BCA, BSc.	**Semester: 3rd**
	Course Code: ENCS201, ENCA 203, ENBC205	**Number of students:**
	Course Name: Java Programming	**Faculty: Ms. Radhika Gupta**

Capstone Assignment

Instructions:

- Assignment needs to be submitted by given date
- Assignment must be submitted on (https://lms.krmangalam.edu.in/)
- Use of ChatGPT and similar tools is strictly prohibited.
- Assignment must be written on dedicated assignment copy for Java Programming subject.
- All assignments must be prepared as per the format shared in classes
- Assignment needs to be submitted by each individual.
- Total marks:10
- This assignment contributes to total 10% of internal evaluation
- Assignments will be evaluated on the basis of the following metrics.
  - Originality
  - Correctness
  - Completeness

Sr.no	Assignment Details	COs
1.	**Project Title: Banking Application for Account Management** **Problem Statement:** Design and implement a banking application that allows users to manage their bank accounts through various operations such as creating accounts, depositing money, withdrawing money, and viewing account details. The application should incorporate the concepts of Java features, control structures, arrays, and strings as per the syllabus. **Project Objectives:** • Apply the basics of Java programming including data types, operators, control structures, and type casting. • Utilize Java's control structures for decision making and looping. • Implement basic I/O operations using the Scanner class. • Handle single and multi-dimensional arrays. • Manipulate strings using Java's String class and methods. **Learning Outcomes:** • Develop a foundational understanding of Java programming. • Gain practical experience with Java control structures and data handling. • Implement real-world applications using Java arrays and strings. • Learn to manage and share code using GitHub. **Project Instructions:** 1. Account Class Design:	CO1

- Attributes:
  - accountNumber: Integer, Unique account number.
  - accountHolderName: String, Name of the account holder.
  - balance: Double, Current balance in the account.
  - email: String, Email address of the account holder.
  - phoneNumber: String, Contact number of the account holder.
- Methods:
  - deposit(double amount): Method to deposit money into the account. It should validate the amount to be positive.
  - withdraw(double amount): Method to withdraw money from the account. It should validate the amount to be positive and ensure sufficient balance.
  - displayAccountDetails(): Method to display the current account details.
  - updateContactDetails(String email, String phoneNumber): Method to update the contact details of the account holder.

2. User Interface Class:
   - Attributes:
     - Array to store multiple Account objects.
     - Scanner object for input.
   - Methods:
     - createAccount(): Method to create a new account.
     - performDeposit(): Method to handle deposit operations.
     - performWithdrawal(): Method to handle withdrawal operations.
     - showAccountDetails(): Method to display account details.
     - updateContact(): Method to update contact details.
     - mainMenu(): Method to display the main menu and handle user choices.

Sample Interaction:

Welcome to the Banking Application!
1. Create a new account
2. Deposit money
3. Withdraw money
4. View account details
5. Update contact details
6. Exit
Enter your choice: 1
Enter account holder name: John Doe
Enter initial deposit amount: 1000.0
Enter email address: john.doe@example.com
Enter phone number: 1234567890
Account created successfully with Account Number: 1001

2.	**Project Title: Employee Management System using Inheritance and Polymorphism** **Problem Statement:** Design and implement an employee management system that showcases inheritance and polymorphism. The system should define a base class for generic employees and use derived classes for specialized roles like "Manager" and "Developer." The application must demonstrate method overriding for role-specific logic and method overloading for flexible data input. **Project Objectives:** • Understand and apply the concepts of inheritance and polymorphism in Java. • Implement super keyword to access superclass members. • Practice declaring and implementing different types of inheritance. • Use method overriding for dynamic polymorphism. • Develop a well-structured application using classes and objects. **Learning Outcomes:** • Gain a deeper understanding of object-oriented programming concepts. • Implement real-world applications using inheritance and polymorphism. • Learn to create and manage a class hierarchy. **Project Instructions:** 1. Employee Class Design: o Create a base class Employee with attributes: employeeId (Integer), name (String), and salary (Double). o Include a calculateBonus() method that returns a bonus based on the base salary. o Provide a displayDetails() method to show all employee information. 2. Derived Classes: o Create Manager and Developer classes that inherit from Employee. o Manager should have an additional attribute department (String) and override calculateBonus() to provide a higher bonus. o Developer should have an additional attribute programmingLanguage (String) and override calculateBonus() to provide a different bonus structure. 3. Main Class (ManagementSystem): o Use an array to store multiple Employee objects (polymorphism). o Implement a menu-driven interface to create and add new employees (both Manager and Developer), display details of a specific employee, and display the details of all employees. **Sample Interaction:** Welcome to the Employee Management System! 1. Add Manager 2. Add Developer 3. Display Employee Details 4. Display All Employees 5. Exit Enter your choice: 1 Enter Manager details... Manager added successfully. Enter your choice: 3 Enter Employee ID to search: 101	CO2

	Employee ID: 101, Name: Jane Doe, Department: HR, Salary: 75000.0, Bonus: 7500.0	
3.	**Project Title: Student Result Management System** **Problem Statement:** Design and implement a student result management system that collects student details and their subject marks, calculates results, and displays them. The system must handle various types of errors and exceptions such as invalid marks, null values, input mismatches, and missing data. The application should demonstrate exception handling in Java, including the use of built-in exceptions, custom exceptions, try-catch blocks, throw, throws, and finally clauses. **Project Objectives:** • Understand and implement Java exception handling. • Classify exceptions into checked and unchecked types. • Practice declaring and throwing exceptions using throw and throws. • Design and use custom exception classes for domain-specific validation. • Utilize try-catch-finally blocks for robust program execution. **Learning Outcomes:** • Identify and handle runtime errors using exception handling. • Distinguish between checked and unchecked exceptions in Java. • Write clean, modular code that handles exceptions gracefully. • Develop real-world Java applications using robust error-handling logic. **Project Instructions:** 1. Student Class Design: o Attributes: rollNumber (Integer), studentName (String), marks (Integer array for 3 subjects). o Methods: ▪ validateMarks(): Validates each subject's marks to be in the range 0-100, throwing a custom exception if invalid. ▪ calculateAverage(): Calculates average marks. ▪ displayResult(): Displays roll number, name, marks, and result status (Pass/Fail). 2. Custom Exception Class: o Create a class InvalidMarksException that extends Exception. o It should have a constructor with a custom error message. 3. User Interface Class (ResultManager): o Attributes: An array to store multiple Student objects and a Scanner object for input. o Methods: ▪ addStudent(): Accepts student data and validates marks, throwing InvalidMarksException. ▪ showStudentDetails(): Displays details for a specific student. ▪ mainMenu(): Provides options to perform various operations, using try-catch to handle exceptions and finally to release resources or display a closing message. **Sample Interaction:** ===== Student Result Management System =====	CO4

	1. Add Student 2. Show Student Details 3. Exit Enter your choice: 1 Enter Roll Number: 102 Enter Student Name: Bob Enter marks for subject 1: -10 Error: Invalid marks for subject 1: -10. Returning to main menu...	
4.	**Project Title: Contact List Application** **Problem Statement:** Design and implement a contact list application that can store, manage, and retrieve contact information. The application should use Java's Collections Framework to efficiently manage a list of contacts in memory. Additionally, it must utilize File Handling to persist the contact data to a file, allowing the user to save their contacts and load them back into the application. **Project Objectives:** • Understand and implement Java's File Handling capabilities. • Use File class methods to manage files and directories. • Implement reading from and writing to files using character streams. • Apply the Java Collections Framework to store and manage data. • Demonstrate the use of HashMap for efficient data storage and retrieval. **Learning Outcomes:** • Gain practical experience with Java I/O streams and file handling. • Implement real-world applications using the Java Collections Framework. • Develop robust applications that can persist data between sessions. **Project Instructions:** 1. Contact Class: o Create a Contact class with attributes: name (String), phoneNumber (String), and email (String). 2. ContactManager Class: o Use a HashMap<String, Contact> to store contacts, where the key is the contact's name. o Implement methods to add, remove, and search for contacts. o Implement methods to saveContactsToFile(String filename) and loadContactsFromFile(String filename) using FileWriter and FileReader (or BufferedWriter and BufferedReader). 3. Main Class (ContactListApp): o Implement a menu-driven interface that allows the user to: ▪ Add a new contact. ▪ Remove an existing contact. ▪ Search for a contact by name. ▪ Display all contacts. ▪ Save contacts to a file. ▪ Load contacts from a file. ▪ Exit the application. **Sample Interaction:** Welcome to the Contact List Application! 1. Add Contact 2. Search Contact 3. Display All Contacts	CO6

	4. Save to File 5. Load from File 6. Exit Enter your choice: 1 Enter contact name: Alice Enter phone number: 9876543210 Enter email: alice@example.com Contact added successfully.	
5.	**Project Title: Library Management System** **Problem Statement:** Design and implement a library management system that can manage a collection of books. The system should use Object-Oriented Programming principles by creating a Book class and a Library class to manage the books. It must demonstrate the concept of method overloading by providing different ways to search for books (e.g., by title and by author). The application should also allow for basic operations like adding, borrowing, and returning books. **Project Objectives:** • Apply fundamental Java concepts, including classes, objects, and control structures. • Demonstrate the use of method overloading for different search functionalities. • Implement a class-based structure for a real-world application. • Practice manipulating strings for data management. **Learning Outcomes:** • Gain a practical understanding of object-oriented principles. • Develop a foundational understanding of method overloading. • Write clean, modular code for a functional application. • Implement real-world applications using Java arrays and strings. **Project Instructions:** 1. Book Class Design: o Create a Book class with attributes: title (String), author (String), and isAvailable (boolean). o Include a constructor to initialize these attributes. 2. Library Class: o Use an array of Book objects to store the library's collection. o Implement overloaded searchBook() methods: ▪ searchBook(String title): Searches for a book by its title. ▪ searchBook(String author): Searches for all books by a specific author. o Implement methods to addBook(), borrowBook(), returnBook(), and displayAllBooks(). 3. Main Class (LibraryApp): o Create a menu-driven user interface with options to perform all the operations defined in the Library class. o Use control structures to navigate the menu and handle user input. **Sample Interaction:** Welcome to the Library Management System! 1. Add a new book 2. Search for a book by title 3. Search for books by author 4. Borrow a book 5. Return a book 6. Display all books	CO2

7. Exit
Enter your choice: 2
Enter book title: The Hitchhiker's Guide to the Galaxy
Book found: The Hitchhiker's Guide to the Galaxy by Douglas Adams.
Status: Available

Evaluation Highlight Rubrics (10 points):

Criterion	Points	Description
Correct Implementation of Core Classes	2	All required classes, attributes, and methods implemented as per project requirements, with proper encapsulation and data handling.
Functionality of User Interface & Features	2	All menu options and required operations work correctly, handlin multiple objects as per project scope.
Use of Java Programming Constructs	1	Appropriate use of loops, decisior making, arrays/collections, strings and Java-specific features.
Data Handling & Validation	1	Correct use of arrays, collections, and/or file handling; proper validation of inputs.
Exception Handling	1	Effective handling of exceptions with try-catch and custom exceptions if required; no progran crashes.
Code Structure & Modularity	1	Logical organization into classes and methods, following OOP principles (encapsulation, inheritance, polymorphism where applicable).
String Handling & Formatting	0.5	Correct manipulation and validation of string inputs (names, emails, identifiers).
Documentation & Readability	0.5	Meaningful comments, descriptive names, consistent indentation, an readable code style.
Originality & Completeness	0.5	Code is original, meets all requirements, and implements all specified features.
Correctness & Output Accuracy	0.5	Program executes without errors and produces correct outputs for various test cases.