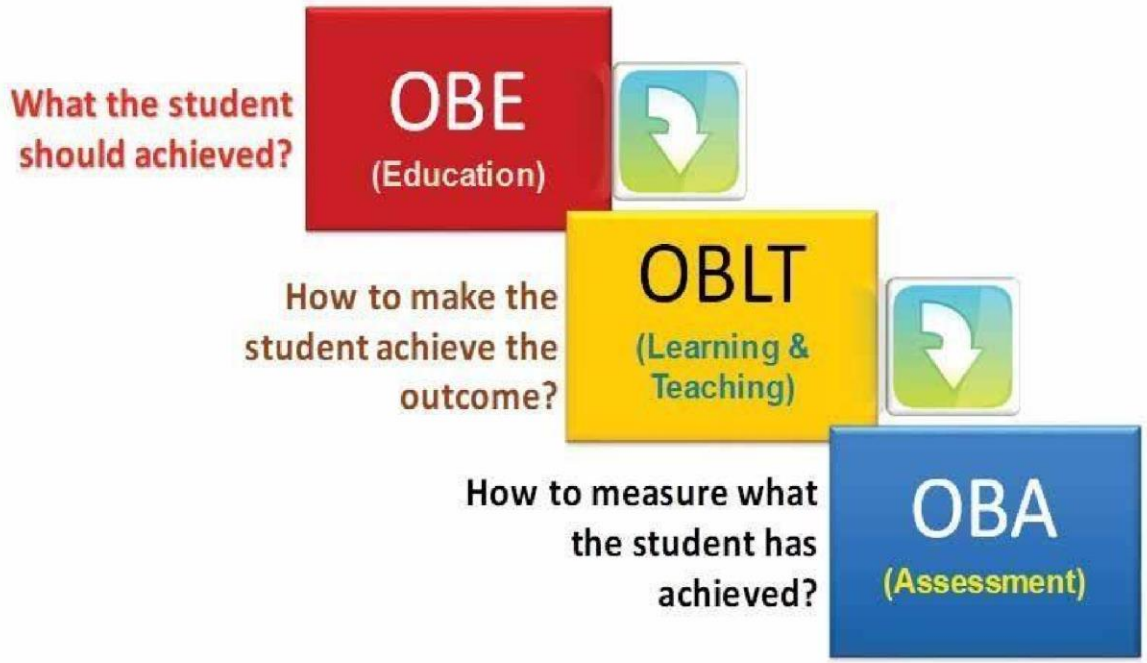
GALGOTIAS UNIVERSITY

GUSOP-01

Course-Pack Framework

A comprehensive instructional delivery document



Prepared by:

VCO

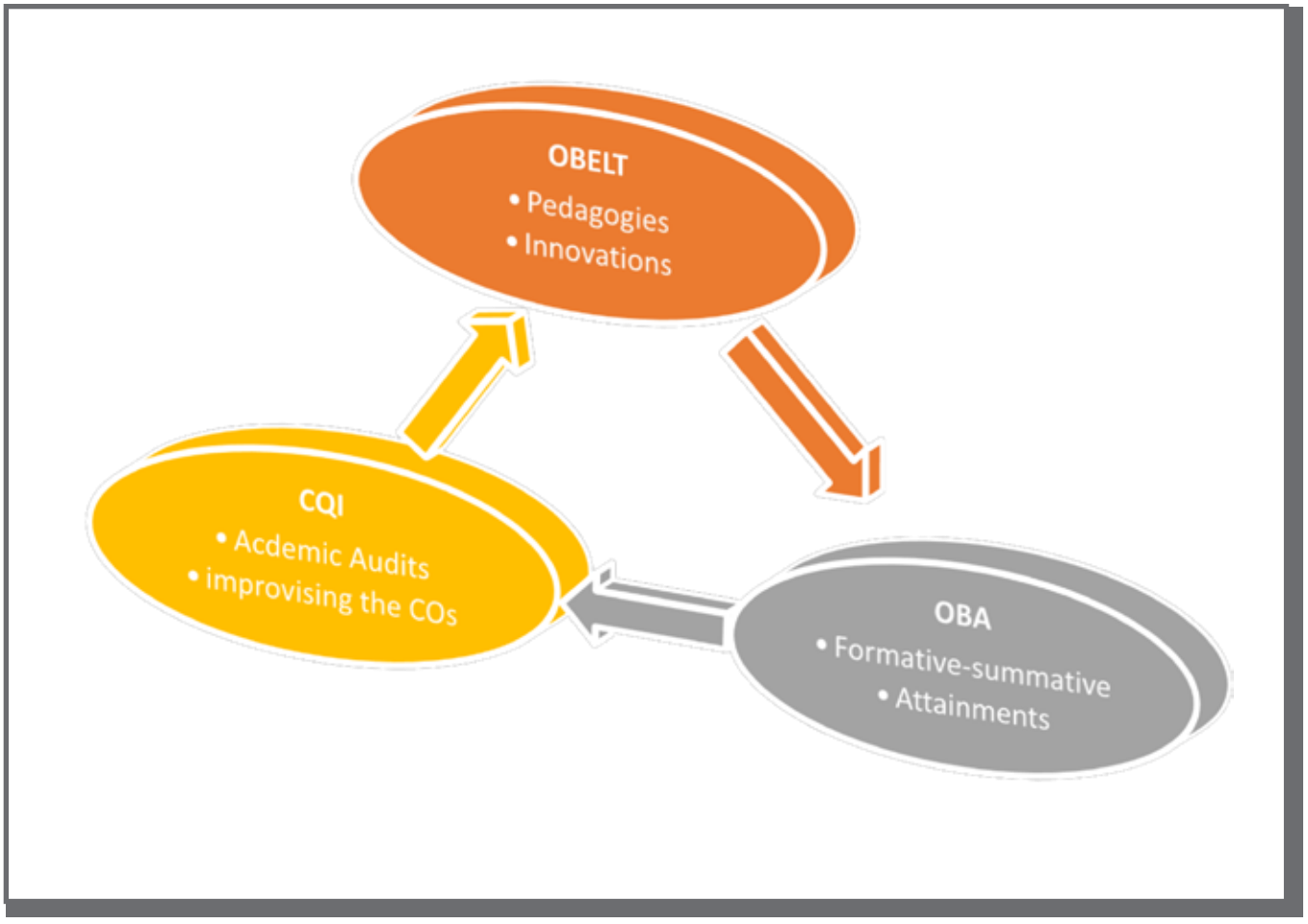
(Revised on Aug-2023)

amuna Expressway, Opposite
autam Buddha Nagar, Uttar F
)0 9717300418, 9582847(
:university.edu.in | www.gal

The Course Pack is a comprehensive and complete pedagogical guideline document that describes the components of instruction delivery by a faculty member. It consists of the scheme of the course, Course Overview, Course Objectives, Prerequisite course, Program-specific Outcomes (PSOs), Course outcomes (COs), Bloom's taxonomy (Knowledge Levels), Types of Courses, Course articulation matrix, Course assessment patterns, Course content, Lesson Plan, Bibliography, Problem-based learning/case-studies/clinical, and Student-Centered learning (self- learning towards life-long-learning). It not only provides a uniform design of Course delivery across the University but also ensures freedom and flexibility to introduce innovations in learning and teaching and create vivid kinds of assessment tools (alternate assessment tools) by a faculty member.

The course pack is developed by the faculty member teaching a course. If more than one faculty teaches the same course, all the faculty members teaching the course shall be formed as a cluster, and a senior faculty member (Course-lead) lead the Course delivery design in a team effort. The Course Pack provides ample scope and opportunity to bring innovations in teaching pedagogies in a school/department.

Hence, the Course pack is a comprehensive learning-teaching strategy framework to be followed by all the faculty members in schools/departments in the university. It is not only a tool for measuring the learning of a class but also analyses the achievement levels (learning outcomes of the course) of all the students in a class in a continuous manner.



COURSEPACK SCHEME

<table>
<tr><td>Course Title</td><td colspan="3">Java Programming</td><td colspan="3">Course Type</td><td colspan="3">Comprehensive</td></tr>
<tr><td>Course Code</td><td colspan="3">E1UA307C</td><td colspan="3">Class</td><td colspan="3">B.Tech(CSE)</td></tr>
<tr><td rowspan="6">Instruction delivery</td><td>Activity</td><td>Credits</td><td>Weekly Hours</td><td colspan="3" rowspan="2">Total Number of Classes per Semester</td><td colspan="2" rowspan="2">Assessment in Weightage</td></tr>
<tr><td>Lecture</td><td>3</td><td>3</td></tr>
<tr><td>Tutorial</td><td>0</td><td>0</td><td rowspan="3">Theory</td><td rowspan="3">Tutorial</td><td rowspan="3">Practical</td><td rowspan="3">Self-study</td><td rowspan="3">CIE</td><td rowspan="3">SEE</td></tr>
<tr><td>Practical</td><td>2</td><td>4</td></tr>
<tr><td>Self-study</td><td>0</td><td>0</td></tr>
<tr><td>Total</td><td>5</td><td>7</td><td>45</td><td>0</td><td>30</td><td>0</td><td>50%</td><td>50%</td></tr>
<tr><td rowspan="3">Names Course Instructors</td><td>Course Lead</td><td colspan="3">Dr. Jitender Tanwar</td><td>Course Coordinator</td><td colspan="3">Mr. Rakesh Bharati</td></tr>
<tr><td colspan="4">Theory</td><td></td><td colspan="4">Practical</td></tr>
<tr><td colspan="4">• Avinash Dwivedi
• C. Rajesh Babu
• Deepa Joshi
• Dhirendra Kumar Shukla
• Jitender Tanwar
• Ravi Sharma (231893)
• Ajay Shankar
• Basu Dev Shivahare
• Kushal Gupta
• Murari Krishna Saha
• Neha Bagwari
• Rakesh Bharati
• V. Janakiraman
• R Nagendran
• Ruchi Sharma
• P. Rajakumar
• Abhishek Kumar Agrahari
• Gaurav Kumar
• Smita
• Neha Kumari
• R. Muthuganesh
• Arpesh Singh
• Chandra Bhushan Kumar Yadav</td><td></td><td colspan="4">• Avinash Dwivedi
• C. Rajesh Babu
• Deepa Joshi
• Dhirendra Kumar Shukla
• Jitender Tanwar
• Ravi Sharma (231893)
• Ajay Shankar
• Basu Dev Shivahare
• Kushal Gupta
• Murari Krishna Saha
• Neha Bagwari
• Rakesh Bharati
• V. Janakiraman
• R Nagendran
• Ruchi Sharma
• P. Rajakumar
• Abhishek Kumar Agrahari
• Gaurav Kumar
• Smita
• Neha Kumari
• R. Muthuganesh
• Arpesh Singh
• Chandra Bhushan Kumar Yadav</td></tr>
</table>

COURSE OVERVIEW

This course provides a comprehensive introduction to Java programming, focusing on both fundamental and advanced concepts. Starting with the basics of Object-Oriented Programming (OOP), students will learn how to write efficient Java programs using variables, data types, control structures, and expressions. The course progresses into more advanced topics, including classes, objects, method overloading, and string handling, laying a strong foundation for understanding Java's core features.

Students will explore advanced object-oriented concepts such as inheritance, polymorphism, and exception handling, equipping them with the skills to create robust and maintainable applications. The course also covers the Java Collections Framework, providing insight into data management through sets, lists, queues, and maps, along with file I/O operations and multithreading, including thread synchronization and concurrency.

In the latter part of the course, students will deep dive into graphical user interface (GUI) development using AWT and Swing, event handling, and the basics of Java networking. The course concludes with an introduction to web development in Java, covering JDBC for database connectivity, Servlets, JSP, and JSTL for simplifying web application development.

By the end of the course, students will have gained a well-rounded understanding of Java, enabling them to design, develop, and deploy Java applications ranging from simple programs to complex, multithreaded, and web-based applications.

COURSE OBJECTIVE

1. **Fundamentals of Java and Object-Oriented Programming**
   To introduce students to the core concepts of Object-Oriented Programming (OOP) and the fundamental features of Java, enabling them to write basic Java programs using essential constructs like variables, data types, operators, and control structures.
2. **Advanced Object-Oriented Concepts and Exception Handling:**
   To deepen students' understanding of advanced OOP concepts such as inheritance, polymorphism, and exception handling, and to equip them with the skills to develop robust Java applications using packages, abstract classes, interfaces, and custom exceptions.
3. **Data Management and Multithreading:**
   To provide students with practical knowledge of the Java Collections Framework, file I/O operations, and multithreading, including thread synchronization and the development of concurrent applications.
4. **GUI, Networking, and Web Development in Java:**
   To introduce students to graphical user interface (GUI) development using AWT and Swing, basic Java networking, and web development using JDBC, Servlets, and JSP, including the use of JSTL for simplifying JSP code.

COURSE OUTCOMES (COs)

Course Outcomes	Upon successful completion of this course, the student will be able to:
CO.1.	Apply the concepts of packages, OOPS such as inheritance, polymorphism, and exception handling in Java, and create robust applications using abstract classes, interfaces, and user-defined exceptions.
CO.2.	Utilize the Java Collections Framework, interfaces to manage data, and perform file operations using I/O streams and demonstrate the creation of multithreaded applications using generic collections, classes, and methods.
CO.3.	Design and develop graphical user interfaces (GUIs) using AWT and Swing, handle events using the Delegation Event Model, and understand the basics of Java networking.
CO.4.	Develop database-driven applications using JDBC, perform CRUD operations, and design the architecture and lifecycle of servlets and JSPs, including the use of JSTL to simplify JSP code.

PROGRAM OUTCOMES (POs)

PO1	**Computing Science knowledge:** Apply the knowledge of mathematics, statistics, computing science and information science fundamentals to the solution of complex computer application problems.
PO2	**Problem analysis:** Identify, formulate, review research literature, and analyze complex computing science problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and computer sciences**.**
PO3	**Design/development of solutions:** Design solutions for complex computing 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.
PO4	**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.
PO5	**Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern computing science and IT tools including prediction and modeling to complex computing activities with an understanding of the limitations.
PO6	**IT specialist 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 computing science and information science practice.
PO7	**Environment and sustainability**: Understand the impact of the professional computing science solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8	**Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the computing science practice.
PO9	**Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	**Communication:** Communicate effectively on complex engineering activities with the IT analyst 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.
PO11	**Project management and finance:** Demonstrate knowledge and understanding of the computing science 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.
PO12	**Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAMME SPECIFIC OUTCOME (PSO):

The students of Computer Science and Engineering shall:

PSO1: Have the ability to work with emerging technologies in computing requisite to Industry 4.0.

PSO2: Demonstrate Engineering Practice learned through industry internship and research project to solve live problems in various domains

BLOOM'S LEVEL OF THE COURSE OUTCOMES

CO No.	Remember **BTL1**	Understand **BTL2**	Apply **BTL3**	Analyse **BTL4**	Evaluate **BTL5**	Create **BTL6**
CO1			√			√
CO2			√			√
CO3			√			√
CO4			√			√

COURSE ARTICULATION MATRIX

CO/PO Mapping (1 / 2 / 3 indicates strength of correlation) 3 - High, 2 - Medium, 1 – Low														
COs	**Program Outcomes (POs)**													
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**
CO 1	2	1	1	-	2	-	-	-	-	-	-	-	-	-
CO 2	2	-	1	-	2	-	-	-	-	-		1	-	-
CO 3	2	-	1	-	2	-	-	-	1	1		1	1	-
CO 4	2	-	1	-	2	-	-	-	1	1		1	1	-

COURSE CONTENT – THEORY

BASICS OF JAVA

Introduction to object-oriented programming - Features of Java – JDK-JVM- Keywords- Variables - Data types – Operators-Expression- Control structures - - Type Casting -Classes and Objects -Methods -Access specifiers & modifiers- Constructors - Method Overloading- this Keyword - Static – Arrays String - String Buffer-String handling mechanism-Command line arguments. Wrapper Classes.

PACKAGE, INHERITANCE, POLYMORPHISM AND EXCEPTION HANDLING

Java API Packages - Package Access - Basics of Inheritance - Forms of Inheritance - Super keyword – Final - Method Overriding - Abstract Class – Interface- Inner Classes-Exception Handling- Exception Hierarchy - Exception Types - Exception handling strategies -User defined Exception.

COLLECTION, I/O STREAMS AND Threads

Collection Class -Collections Framework- Interface & Classes Implementation: Set- List- Queue-Map-File- I/O Stream- Character Streams – Byte Streams- Object Serialization –Threads - Thread states - Thread priority - Thread operations -– Thread Synchronization – Multithreading-Generic Collections - Generic Classes and Methods.

AWT, SWING AND NETWORKING

Applets: Basics of applets – Applet, Architecture - Life cycle of an Applet – AWT, Layout Manager: Event Handling-Delegation event Model, - Swing overview- Swing Components -Basics of Java Networking.

JDBC, SERVLET and JSP

Introduction to JDBC API-JDBC Architecture - JDBC Drivers- Database connectivity in Java - CRUD Operations-java. sql methods and interfaces -Introduction to Servlet, Servlet Architecture, Lifecycle of a SERVLET, Introduction to JSP, JSP Architecture, Introduction to JSTL and EL. Use JSTL to simplify JSP code.

PRACTICAL

Java Data Type, Expression, and Operators; Conditional Statements; Control statements; Strings and StringBuffer Class. Array, Implementation of OOPS properties, Constructor overloading, Abstract class and interface. Wrapper class and Linked Structure of Java. Multithreading, Exception handling, Collection, CRUD operation and database connectivity, Networking with java. Servlet and JSP.

COURSE ASSESSMENT

The course assessment patterns are the assessment tools used both in formative and summative examinations.

Type of Course (C)	CIE			Total Marks		Grand Total Marks	Weightage (CIE-SEE)
	LAB*	MTE	Course-based Project*	CIE	SEE		
COMPREHENSIVE	25*	50	25*	100	100	200	50-50

***Rubric for the course-based project (Scaled down to 25)**

Type of Assessment Tools	Preliminary Project Plan	Review 1	Review 2	Review 3	Final Viva Voce
Course-based Project Work	10	10	10	20	10

***Rubric for the Laboratory (Scaled down to 25)**

Type of Assessment Tools	Codekata (Lab Internal)	Self-paced Certification	Lab File + Viva	Final Lab Exam
Laboratory	15	10	10	15

PPP (Preliminary Project Plan): *The preliminary project plan (PPP) provides an initial, overview of the project and all of its known parameters. It outlines the project's objectives, relevance to the program, merit, and conformity to current industry/government policy, proposed methodology, and expected outcomes. It should also include any known constraints related to the time frame (Gantt Chart), budget, and, etc.*

LESSON PLAN FOR INTEGRATED COURSES of 3 CREDITS

FOR THEORY 15 weeks * 3 Hours = 45 Classes) (1credit = 1Lecture Hour)
FOR PRACTICAL 15 weeks * 2Hours = 30 Hours lab sessions (1 credit = 2 lab hours)

L-No	Topic for Delivery	Theory / Tutorial / Practical Plan	Skills	Competency
1	Introduction to object-oriented programming - Features of Java – JDK-JVM- Keywords- Variables - Data types	Theory	Use JDK to execute java programs including data types. Operators and control statements	CO1
2	Operators-Expression- Control structures - - Type Casting	Theory		
3	Java Data Type, Expression, and Operators: Pseudo code & Program	Practical		
4	Programs of loops, patterns and util package	Practical		

5	Classes and Objects -Methods -Access specifiers & modifiers	Theory	Modeling of real world problem and use of overloading	
6	Method Overloading- this Keyword – Static, Constructors, constructor overloading.	Theory		
7	Program to create classes, use of access specifiers, this and final keyword	Practical		
8	Implementation of constructor and constructor overloading	Practical		
9	Wrapper Classes, Command line arguments	Theory	String handling and interchange of primitive types and objects	
10	Array, String & StringBuffer class, String Tokenizer	Theory		
11	Write programs to convert primitive data types to object and vice-versa	Practical		
12	Array, String and Stringbuffer implementation Use of String Tokenizer in fragmenting string	Practical		
13	Java API Packages - Package Access, Basics of Inheritance - Forms of Inheritance - Super keyword – Final, Method Overriding,	Theory	Use packages and apply inheritance and abstraction	
14	Abstract Class – Interface	Theory		
15	Creation of package, use of built-in packages, final and super keywords	Practical		
16	Implementation of Inheritance, Abstract class and Interface	Practical		
17	Inner classes, Exception Handling- Exception Hierarchy - Exception Types	Theory	Use inner class and will able to stop abrupt program termination using Exception handling	
18	Exception handling strategies -User defined Exception	Theory		
19	Program to create inner classes	Practical		
20	Program implementing exception handling Development of user defined exceptions	Practical		
21	Threads - Thread states -Thread priority	Theory	apply multitasking with concurrency	**CO 2**
22	Thread operations -– Thread Synchronization – Multithreading	Theory		
23	Program to create threads using Thread class and Runnable interface	Practical		
24	Program to set priority, handle shared resource	Practical		
25	Applets: Basics of applets - Applet Architecture. Life cycle of an Applet	Theory	Create user interface with and without browser with interaction	
26	AWT: Event Handling-Delegation event Model	Theory		
27	Program to create Applet with user interaction	Practical		
28	Creation of user interfaces with event	Practical		

	handling			
29	Swing overview- Swing Components	Theory		
30	Different types of Layout managers	Theory		
31	Program to create UI using Swing	Practical		
32	Program to use different layout managers	Practical		
33	I/O Stream- Character Streams – Byte Streams	Theory	use different streams in reading- writing data and develop client server programs	**CO 3**
34	Object Serialization, keyword-transient	Theory		
35	Program to create streams, reading and writing data	Practical		
36	Program to convert object to store data in file and reading. Program to use transient keyword	Practical		
37	Basics of Java Networking: URL, IP Address,	Theory		
38	Client- Server communication	Theory		
39	Program to use URL, IP Address and protocols	Practical		
40	Implementation of client-server communication	Practical		
41	Introduction to JDBC API-JDBC Architecture	Theory	CRUD operations using database and use collection in keeping data in different data structures	
42	JDBC Drivers- Database connectivity in Java	Theory		
43	Program to perform CRUD operation using JDBC and oracle using Statement	Practical		
44	Program to perform CRUD operation using JDBC and oracle using Prepared Statement	Practical		
45	Collection Class -Collections Framework- Interface & Classes	Theory		
46	Iterator, Generic Collections - Generic Classes and Methods	Theory		
47	Program to use different collection data structures in keeping objects	Practical		
48	Program to use iterator in Java collection	Practical		
49	CRUD Operations-java.sql methods and interfaces	Theory	Use different objects in developing web pages	**CO 4**
50	Implementation: Set- List- Queue-Map	Theory		
51	WAP in Java to implement the addition of elements in the List	Practical		
52	A Java program to illustrate a Map. and add elements using Map	Practical		
53	Introduction to JDBC API	Theory		
54	JDBC Architecture - JDBC Drivers	Theory		
55	A Java program to illustrate a Set. And add Elements using Set.	Practical		
56	Java Program to Get the Size of Collection and Verify that Collection is Empty	Practical		

57	Database connectivity in Java	Theory	Develop dynamic web pages that interact with the Document Object Model (DOM). Develop dynamic web pages using Servlets.	**CO 4**
58	CRUD Operations-java	Theory		
59	Write program for connecting an application to the database.	Practical		
60	Sql methods and interface	Theory		
61	Architecture of Servlet, Lifecycle of a SERVLET	Theory		
62	Write a Servlet program for sending e-mail	Practical		
63	The SERVLET API, Reading SERVLET parameters	Theory		
64	Reading initialization parameters	Theory		
65	Write a program using Servlet for reading different parameters using Get and Post Methods	Practical		
66	Handling Http Request & Responses	Theory		
67	Session management.	Theory		
68	Create a Servlet page for login system using session	Practical		
69	Write simple Servlet program to set cookies and read it	Practical		
70	Revision	Theory		
71	Revision	Theory		
72	Revision	Theory		
73	Revision	Practical		
74	Revision	Practical		
75	Revision	Practical		

BIBLIOGRAPHY

Text Book:

1. **Herbert Schildt**, "**Java the Complete Reference**", Ninth edition, Tata Mc-Graw Hill ,2014.
2. Database Programming with JDBC & Java (Java (O'Reilly)).
3. Kathy Sierra, and Bates Bert. Head First Java: A Brainc-Friendly Guide. " O'Reilly Media, Inc.", Second Edition, 2009

Reference Books:

1. H.M. Deitel and P.J. Deitel," Java How to Program", Pearson Prentice Hall Seventh Edition.
2. Kathy Sierra, and Bates Bert. Sun Certified Programmer for Java. McGraw Hill Publications, 2008.
3. Keyur Shah, Gateway to Java Programmer Sun Certification, Tata McGraw Hill, 2002.
4. Joshua Bloch, ―Effective Java: A Programming Language Guide, Second Edition, Pearson, 2008.
5. Bruce Eckel – "**Thinking in Java**" Pearson Prentice Hall Third Edition-2006
6. Kogent Learning Solutions Inc,"JAVA 7 Programming Black Book", DreamTech Press, 2010.

SWAYAM/NPTEL/MOOCs Certification

1. https://onlinecourses.swayam2.ac.in/cec22_cs20/preview
2. https://onlinecourses.nptel.ac.in/noc22_cs32/preview
3. https://onlinecourses.nptel.ac.in/noc19_cs42/preview

Optional certifications and online platforms

1. https://www.codechef.com/practice/
2. https://skillsforall.com/course/

PRACTICE PROBLEMS

S. No	Program
1	Write a program to find factorial of a number with recursion and iteration
2	Java IO program to take input through keyboard at runtime.
3	Write a program to reverse a number
4	Write a Java program to Print Sum of Series 1+x+x2+x3+.....+xn in java
5	Write a program to display the following pattern: 1 1 2 1 2 3 1 2 3 4 1 2 3 4 5
6	
7	Write a java program to create patterns * * * * * * * * * *
8	Write a program to display the following pattern: *
9	Write a Java Program to Make a Simple Calculator Using switch...case and Scanner class.
10	Write a program to Search an Element in an Array in java.
11	Write a program to add two matrices of order 3X3
12	Create a class Student (name, roll_no, marks) with one method show() and initialize instance variables using all the ways: reference, method and constructor.

13	WAP showing importance of this keyword
14	WAP implementing Constructor overloading and use this key word in overloading.
15	WAP to access static block, static variable and static method.
16	WAP in java to convert primitive data type to object and vice versa using Wrapper classes both the ways: implicitly and explicitly. WAP to show that String is immutable
17	WAP to make String mutable using StringBuffer class
18	Program to Illustrate the use of Methods of String Class Write a java program with menu to solve following queries 1. Count Vowels and Consonants in a String 2. Count the Number of Duplicate Words in a String 3. Count Number of Words in Given String 4. Count the Number of Occurrences of Substring in a String 5. Count the Occurrences of Each Character in String 6. Java Program to Remove Duplicate Words from String
19	Using String Tokenizer separate any string in token using space delimiter
20	Create 2 packages P1 & P2 and create classes Student and BTech in P1 and P2 respectively. Check the accessibility using all the access modifiers.
21	WAP to implement 3 interfaces and extend one class.
22	Implement multiple inheritance in java using interface.
23	WAP to show the use of Final keyword also initialize a final instance variable which is declared but not initialized.
24	Write a Java program to create an abstract class named Shape that contains two integers and an empty method named print Area ().
25	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.
26	Write a Java program that reads a list of integers from the user and throws an exception if any numbers are duplicates.
27	WAP to access method of inner class in java.
28	WAP to stop access any method of inner class in java.
29	Write an user defined exception “low amount” and it should be raised when amount goes below 10000.

30	Write a Java program to iterate through all elements in a linked list starting at the specified position.
31	Write a Java program to insert the specified element at the specified position in the linked list (Without using collection).
32	Program to delete duplicate object from an arraylist
33	Write a Java program that implements a multi-thread application that has three threads. First thread generates 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.
34	Install a database (Mysql or Oracle). Create a table which should contain at least the following fields: name, password, email-id, phone number Write a java program to connect to that database and extract data from the tables and display them. Insert the details of the users who registers with the UI, whenever a new user clicks the submit button in the registration page.
35	Write a Java Program to a. Develop an applet in Java that displays a simple message. b. Develop an applet in Java that receives an integer in one text field, and computes its factorial Value and returns it in another text field, when the button named “Compute” is clicked.
36	Write a Java program that creates a user interface to perform integer divisions. The user enters two numbers in the text fields, Num1 and Num2. The division of Num1 and Num 2 is displayed in the Result field when the Divide button is clicked. If Num1 or Num2 were not an integer, the program would throw a Number Format Exception. If Num2 were Zero, the program would throw an Arithmetic Exception. Display the exception in a message dialog box.
37	Create an user interface using Swing and apply insert, delete, update operations. Also show the students details
38	Write a Java program that simulates a traffic light. The program lets the user select one of three lights: red, yellow, or green with radio buttons. On selecting a button, an appropriate message with “Stop” or “Ready” or “Go” should appear above the buttons in selected color. Initially, there is no message shown
39	Write a Java IO program to read File.
40	Write a Java IO program to write in a File.

41	Write a Java IO program to read & write in a File.
42	Using Serialization, convert objects into stream of bytes so that it can be written into a file also complete deserialization.
43	WAP using transient keyword
44	Program to create sockets and establishment connection between client and server.
45	Program to create sockets and establishment connection between client and server with read and write in both the way (Chat application).
46	WAP in Java to implement the addition of elements in the List
47	Java Program to Get the Size of Collection and Verify that Collection is Empty
48	Develop a Web Page to display browsers information and print the information of the page.
49	A Java program to illustrate a Set and add Elements using Set.
50	Create a form using form elements and stop user to submit empty field. To validate an email id
51	Create a Servlet page for login system using session
52	Write a Servlet program for sending e-mail
53	Write simple Servlet program to set cookies and read it.

PROBLEM-BASED LEARNING

Exercises in Problem-based Learning (Assignments)

S. No.	Project Name	Description
1	Student Management System	Develop a console-based application for managing student records, including features for adding, updating, and displaying student information.
2	Employee Payroll System	Develop an application for calculating and managing employee payroll, including salary calculations, tax deductions, and paystub generation.
3	Inventory Tracking System	Create an inventory management system for tracking products, restocking items, and generating reports on inventory levels.
4	Contact Management Application	Build a Java program for managing contacts and organizing them into categories. Include features for adding, editing, and searching for contacts.
5	Music Player	Create a simple music player that can load and play audio files. Implement features like playlist creation, playback controls, and audio visualization.
6	Text-Based Adventure Game	Design a text-based adventure game where players make choices to navigate through a story, solve puzzles, and complete quests
7	Hotel Reservation System	Design a program that allows users to make hotel room reservations, view availability, and manage bookings.
8	Personal Finance Tracker	Design a personal finance tracker that allows users to log and categorize their income and expenses. Provide insights into spending patterns
9	File Explorer	Create a simple file explorer application that allows users to navigate through directories, view files, and perform file operations such as copying and deleting
10	Student Gradebook	Build a program for teachers to manage student grades, calculate averages, and generate report cards.
11	Online Quiz Application	Build an interactive quiz application where users can take quizzes on various topics. The program should keep track of scores and provide feedback on quiz performance
12	Employee Management System	Develop an application for managing employee records, including features like adding new employees, updating details, and generating reports.

13	Online Voting System	Create a console-based application that simulates an online voting system for conducting polls and elections. Users can vote for candidates, and the system tallies the results.
14	Banking System Simulation	Create a banking system simulation where users can open accounts, make transactions, and view their account balances. Implement features like interest calculation and account types.
15	Mini Social Media Platform	Build a simplified social media platform that allows users to create profiles, post messages, and connect with others. You can implement features like timelines and user interactions.

GUVI CODEKATA LAB PROBLEMS

1. **Complex Variable Initialization and Arithmetic Operations**

   Problem Statement

   Write a program that takes four integers as input, performs a series of arithmetic operations using these integers, and outputs the results in a specific format.

   Description

   You need to declare four integer variables, perform the following operations, and print the results:

   Add the first and the second integer, then multiply the result by the third integer.
   Subtract the fourth integer from the second integer, then divide the result by the first integer.
   Multiply the first integer by the fourth integer, then add the third integer to the result.
   Add all four integers together and divide by two.

   Input Format

   Four integers, each on a new line.

   Output Format

   Four lines of output, each showing the result of the respective operations in the following format:
   Result of operation 1: <result>
   Result of operation 2: <result>
   Result of operation 3: <result>
   Result of operation 4: <result>

   Sample Input:
   3
   4
   2
   1
   Sample output:
   Result of operation 1: 14
   Result of operation 2: 1
   Result of operation 3: 5
   Result of operation 4: 5

2. **Bitwise Operations with Floating Point Variables**

   Problem Statement

   Write a program that takes two floating-point numbers as input, performs bitwise operations by converting them to integers, and outputs the results in a specific format.

   Description

   You need to declare two floating-point variables, convert them to integers, perform the following bitwise operations, and print the results:

   Perform bitwise AND on the two integers.

Perform bitwise OR on the two integers.
Perform bitwise XOR on the two integers.
Perform bitwise NOT on the first integer and AND it with the second integer.
Input Format
Two floating-point numbers, each on a new line.
Output Format
Four lines of output, each showing the result of the respective operations in the following format:
Bitwise AND result: <result>
Bitwise OR result: <result>
Bitwise XOR result: <result>
Bitwise NOT and AND result: <result>

Sample Input:
5.5
3.3
Sample Output:
Bitwise AND result: 1
Bitwise OR result: 7
Bitwise XOR result: 6
Bitwise NOT and AND result: 2

3. Variable Transformation Challenge

Problem Statement:
In a parallel universe, the concept of variable transformation is quite different. You need to write a Java program that performs a series of operations on variables based on specific conditions. Given three integers a, b, and c, your task is to transform these variables according to the following rules and output the final values.

Description:
If
a is even, add b toa.
If
b is odd, multiply c by 2.
If c is a multiple of 3, add a to c.
If the sum of a, b, and c is greater than 100, subtract 100 from each of a, b, and c.
Your program should then print the transformed values of
a, b, and c in the format "a: [value], b: [value], c: [value]".

Input Format:
Three integers
a, b, and c are given as input from the user, each on a new line.
Output Format:
Print the transformed values of
a, b, and c in the specified format.

Sample input:
10
15
9
Sample output:
a: 25, b: 15, c: 43

4. **Odd-Even Sum Checker**

Problem Statement:
Write a program that reads a list of integers from the user and checks if the sum of odd numbers is greater than the sum of even numbers in the list. If the sum of odd numbers is greater, print "Odd Sum Greater"; otherwise, print "Even Sum Greater".

Description:
The program should continuously read integers until the user inputs a specific stop value. You need to handle edge cases where there are no odd or even numbers.

Input Format:
The first line contains an integer n (number of elements).
The second line contains n integers separated by spaces.
The input stops when the user enters -1.
Output Format:
Print "Odd Sum Greater" if the sum of odd numbers is greater.
Print "Even Sum Greater" otherwise.

Sample input:
4 2 4 6 8 -1
Sample Output:
Even Sum Greater

5. Grade Classification

Problem Statement:
Write a program that classifies grades based on the score input by the user. The program should classify the score into "Excellent", "Good", "Average", "Pass", or "Fail" using if-else statements.

Description:
The program should read the scores continuously until the user inputs a stop value. The classification is based on the following criteria:

90-100: Excellent
75-89: Good
50-74: Average
35-49: Pass
0-34: Fail
Input Format:
The first line contains an integer n (number of scores).
The second line contains n integers separated by spaces.
The input stops when the user enters -1.
Output Format:
For each score, print its classification.

6. Month Days Finder

Problem Statement:
Write a program that takes a month (as a number) and a year as input and prints the number of days in that month, considering leap years.

Description:

The program should read the month and year continuously until the user inputs a stop value. You need to handle leap years and invalid inputs.

Input Format:
The first line contains an integer n (number of entries).
The next n lines contain two integers each: the month and the year.
The input stops when the user enters -1 -1.
Output Format:
Print the number of days in the given month for each input.

Sample Input:
2 2020 -1 -1
Sample Output:
29
Sample Input:
5
95 85 65 45 30 -1
Sample Output:
Excellent
Good
Average
Pass
Fail

7. **"The Voting System"**

Problem Statement:
You are designing a voting system where voters can cast their votes for various candidates. Each voter can vote only once, and the system needs to validate each vote and ensure the following rules are adhered to:

A vote must be cast for a valid candidate number.
The total number of votes should not exceed the number of registered voters.
A vote for a candidate should only be allowed if the candidate's ID is within the valid range.
Description:
You need to implement a Java program that:

Takes the number of registered voters as input.
Takes the number of candidates as input.
Takes each vote (candidate ID) as input.
Outputs whether each vote is valid or invalid based on the rules above.
Input Format:

First line: Integer
N (Number of registered voters)
Second line: Integer
C (Number of candidates)
Third line: Integer
V (Number of votes)
Next
V lines: Each line contains an integer
ID (candidate ID)

Output Format:

For each vote, print "VALID" if the vote is valid, otherwise print "INVALID".

Sample Input:
100
5
4
1
6
3
2
Sample Output:
VALID
INVALID
VALID
VALID

8. **"Nested Loop Patterns"**

Problem Statement:
Write a program that generates a pattern based on user input using nested loops. The pattern to be generated is a pyramid where each level contains a specific number of symbols. Each level's pattern should follow a unique rule:

The number of symbols on each level is equal to the level number.
The symbol for each level is based on the level number where odd levels use * and even levels use #.
Description:
You need to implement a Java program that:

Takes an integer
N (number of levels) as input.
Generates and prints the pyramid pattern according to the rules described.
Input Format:

Single line: Integer
N (Number of levels)
Output Format:

Pyramid pattern with
N levels, where odd levels use * and even levels use #.

Sample Input:
3
Sample Output:
*
##
***

9. **Classes and Objects**

Problem Statement
Create a class Car that represents a car with a unique twist. The Car class should include properties such as

make, model, and year. Additionally, it should have a method getCarAge that calculates the car's age based on the current year input by the user. However, you need to ensure that the car's year of manufacture is validated to be not in the future.

Description
Define a class Car with private properties: make (String), model (String), and year (int).
Include a constructor that initializes these properties.
Create a method getCarAge that calculates the age of the car based on the current year provided by the user.
Ensure the year property is validated to be less than or equal to the current year.
Input Format
String representing the make of the car.
String representing the model of the car.
Integer representing the year of the car.
Integer representing the current year.
Output Format
Integer representing the age of the car.
A message indicating if the year is invalid.

Sample Input:
Toyota
Corolla
2015
2024
Sample Output:
9

10. OOPs Features & Interface

Problem Statement
Create an interface Shape with methods area and perimeter. Implement this interface in two classes: Rectangle and Circle. Each class should have appropriate constructors and methods to calculate the area and perimeter.

Description
Define an interface Shape with methods double area() and double perimeter().
Create a class Rectangle that implements Shape with properties length and width.
Create a class Circle that implements Shape with property radius.
Ensure the Rectangle and Circle classes have constructors to initialize their properties.
Implement the methods area and perimeter in both classes.
Input Format
String representing the shape type (Rectangle or Circle).
If Rectangle, two doubles representing length and width.
If Circle, one double representing radius.
Output Format
Double representing the area of the shape.
Double representing the perimeter of the shape.

Sample Input:
Rectangle
5
10

Sample Output:
Area: 50.0
Perimeter: 30.0

11. Access Modifiers

Problem Statement
Create a class Person with private properties and public methods for accessing these properties. Ensure that the age of the person is always a positive value.

Description
Define a class Person with private properties: name (String) and age (int).
Include public methods setName and setAge for setting the properties.
Include public methods getName and getAge for accessing the properties.
Ensure that the setAge method only allows positive values.
Input Format
String representing the name of the person.
Integer representing the age of the person.
Output Format
String representing the name of the person.
Integer representing the age of the person.
A message indicating if the age is invalid.

Sample Input:
Alice
25
Sample Output:
Name: Alice
Age: 25

12. OOPs Features & Interface

Problem Statement
Implement an interface Employee with methods calculateSalary and getDetails. Create two classes FullTimeEmployee and PartTimeEmployee that implement this interface. Each class should have its own way of calculating salary.

Description
Define an interface Employee with methods double calculateSalary() and String getDetails().
Create a class FullTimeEmployee with properties name (String), monthlySalary (double) and implement the interface methods.
Create a class PartTimeEmployee with properties name (String), hourlyRate (double), and hoursWorked (int) and implement the interface methods.
Ensure each class has a constructor to initialize its properties.
Input Format
String representing the employee type (FullTimeEmployee or PartTimeEmployee).
If FullTimeEmployee, a string for name and a double for monthlySalary.
If PartTimeEmployee, a string for name, a double for hourlyRate, and an integer for hoursWorked.
Output Format
String representing the employee details.
Double representing the calculated salary.

Sample Input:

FullTimeEmployee
John
3000.0
Sample Output:
Name: John, Salary: 3000.0

13. Polymorphic Zoo

Problem Statement:
Create a program that models a zoo with animals demonstrating polymorphism. Each animal should have a speak method, but the sound they make depends on the type of animal. Implement a system that allows adding different animals to the zoo and prints their sounds.

Description:
Your task is to design a Java program that uses polymorphism to model a zoo. You need to create an abstract class Animal with a method speak(). Then, create at least three subclasses (Lion, Elephant, Monkey) that override the speak() method to return their respective sounds. The program should allow the user to add animals to the zoo and then print out the sounds of all animals in the zoo.

Input Format:

The first line of input contains an integer N, the number of animals.
The next N lines each contain a string representing the type of animal (Lion, Elephant, Monkey).
Output Format:

The output should be the sounds of all animals in the zoo, each on a new line, in the order they were added.

Sample Input:
3
Lion
Elephant
Monkey
Sample Output:
Roar
Trumpet
Ooh Ooh Aah Aah

14. Library Management System

Problem Statement:
Design a library management system using OOP principles. The system should allow adding books, borrowing books, and returning books. Implement classes for Book and Library.

Description:
Your task is to create a Java program that models a library management system. You need to create a class Book with attributes for the book title, author, and a boolean to indicate if it is borrowed. Create another class Library that contains a list of books and methods to add a book, borrow a book by title, and return a book by title.

Input Format:

The first line contains an integer N, the number of operations.
The next N lines contain operations in the format add <title> <author>, borrow <title>, or return <title>.

Output Format:

Print the result of each operation. For borrow, print "Book borrowed" if successful or "Book not available" if not. For return, print "Book returned". For add, print "Book added".

Sample Input:
3
add HarryPotter J.K.Rowling
borrow HarryPotter
return HarryPotter
Sample Output:
Book added
Book borrowed
Book returned

15. **Student Grading System**

Problem Statement:
Implement a student grading system using OOP principles. The system should handle multiple students and calculate their grades based on different criteria.

Description:
Your task is to create a Java program that models a student grading system. You need to create a class Student with attributes like name, marks, and grade. Create methods to add marks, calculate grades, and display student details. The grade should be calculated based on the average of the marks.

Input Format:

The first line contains an integer N, the number of operations.
The next N lines contain operations in the format add <name> <marks>, calculate <name>, or display <name>.
Output Format:

Print the result of each operation. For add, print "Marks added". For calculate, print "Grade calculated". For display, print the student's details.

Sample Input:
4
add John 85
add John 90
calculate John
display John

Sample Output:
Marks added
Marks added
Grade calculated
John: 87.5 – B

16. **Banking System**

Problem Statement:
Develop a banking system using OOP principles. The system should handle accounts, deposits,

withdrawals, and balance inquiries.

Description:
Your task is to create a Java program that models a banking system. You need to create a class BankAccount with attributes like accountNumber, balance, and methods like deposit(), withdraw(), and getBalance(). The system should allow the user to perform these operations based on the account number.

Input Format:

The first line contains an integer N, the number of operations.
The next N lines contain operations in the format create <accountNumber>, deposit <accountNumber> <amount>, withdraw <accountNumber> <amount>, or balance <accountNumber>.
Output Format:

Print the result of each operation. For create, print "Account created". For deposit, print "Deposit successful". For withdraw, print "Withdrawal successful" or "Insufficient funds". For balance, print the account balance.

Sample Input:
5
create 12345
deposit 12345 1000
withdraw 12345 500
balance 12345
withdraw 12345 600

Sample Output:
Account created
Deposit successful
Withdrawal successful
Balance: 500.0
Insufficient funds

17. **"Dynamic Vehicle Registration System"**
Problem Statement:
You need to design a dynamic Vehicle Registration System where each vehicle can be of different types (Car, Motorcycle, or Truck) and should be able to report its registration details. Implement a base class Vehicle and three subclasses Car, Motorcycle, and Truck.

Description:

Create a base class Vehicle with:

A constructor to initialize the vehicle's registration number and owner’s name.
A method displayDetails() to display the vehicle's details.
Extend this base class with three subclasses:

Car with an additional attribute for the number of doors.
Motorcycle with an additional attribute for engine capacity.
Truck with an additional attribute for cargo capacity.
Input Format:

First line: Integer
N (Number of operations to perform)
Next
N lines: Each line will describe an operation in the format REGISTER <vehicle_type> <registration_number> <owner_name> <specific_attribute>.
Output Format:

For each operation, output the details of the registered vehicle.

Sample Input:
3
REGISTER Car ABC123 John 4
REGISTER Motorcycle XYZ789 Alice 600cc
REGISTER Truck LMN456 Bob 10000kg
Sample Output:
Vehicle: Car
Registration Number: ABC123
Owner: John
Number of Doors: 4
Vehicle: Motorcycle
Registration Number: XYZ789
Owner: Alice
Engine Capacity: 600cc
Vehicle: Truck
Registration Number: LMN456
Owner: Bob
Cargo Capacity: 10000kg

18. **Unique User Activity Tracker**

Problem Statement:
You are tasked with developing a system to track unique user activities in a social media application. Each user can perform various activities, and you need to ensure that each activity is only logged once per user. Your task is to create a class that maintains user activity logs, ensuring that each activity for a user is unique.

Description:

Implement a class UserActivityTracker with methods to add activities and display unique activities.
Use a Map<String, Set<String>> where the key is the userId and the value is a Set of activities performed by that user.
Implement the following methods:
addActivity(String userId, String activity): Adds an activity for the specified user. If the activity already exists for the user, it should not be added again.
displayActivities(String userId): Displays all unique activities performed by the specified user in a sorted order.
Input Format:

The first line contains an integer n, the number of operations.
Each of the next n lines contains a command followed by the necessary details:
ADD_ACTIVITY userId activity

DISPLAY_ACTIVITIES userId
Output Format:

For each DISPLAY_ACTIVITIES command, output the list of unique activities for the specified user, each on a new line in sorted order. If the user has no activities, print "No activities found."

Sample Input:
6
ADD_ACTIVITY user1 login
ADD_ACTIVITY user1 post
ADD_ACTIVITY user1 login
ADD_ACTIVITY user2 login
DISPLAY_ACTIVITIES user1
DISPLAY_ACTIVITIES user2

Sample Output:
login
post
login

19. Exception Handling with Multiple Catch Blocks

Problem Statement:
Write a Java program to handle multiple exceptions. The program should take two integers as input from the user and perform division. If the input is invalid, throw a NumberFormatException. If the second integer is zero, throw an ArithmeticException. Ensure that the program uses multiple catch blocks to handle these exceptions and always prints a final message using the finally block.

Description:
The program should prompt the user to enter two integers. It should then attempt to divide the first integer by the second. If the user enters a non-integer value, handle the NumberFormatException. If the second integer is zero, handle the ArithmeticException. Regardless of whether an exception occurs or not, print "Operation Completed" using the finally block.

Input Format:
The first line contains an integer, a.
The second line contains an integer, b.
Output Format:
If a and b are valid integers and b is not zero, print the result of a / b.
If a or b is not a valid integer, print "Invalid input".
If b is zero, print "Cannot divide by zero".
Always print "Operation Completed" at the end.

Sample Input:
10
2

Sample Output:
5
Operation Completed

20. Custom Exception for Age Validation

Problem Statement:
Create a custom exception called InvalidAgeException. Write a Java program that takes the age of a user as input and throws this custom exception if the age is less than 18. Ensure the program catches the exception and prints an appropriate message.

Description:
Define a custom exception InvalidAgeException that extends the Exception class. The program should prompt the user to enter their age. If the age is less than 18, throw the InvalidAgeException with a message "Age must be 18 or older". Catch this exception and print the message to the user.

Input Format:
The first line contains an integer, age.
Output Format:
If age is 18 or older, print "Age is valid".
If age is less than 18, print "Age must be 18 or older".

Sample Input:
20
Sample Output:
Age is valid

21. Nested Exception Handling

Problem Statement:
Write a Java program that reads a list of integers from the user and calculates their sum. If any non-integer value is entered, handle the exception and continue reading the next value. Additionally, ensure that the program terminates gracefully by printing the final sum, even if an exception occurs during the input process.

Description:
Your task is to implement nested exception handling in Java. The program should continue reading integers from the user until a non-integer value is entered. If a non-integer value is encountered, the program should catch the exception, display an appropriate message, and continue reading the next value. Use nested try-catch blocks to achieve this. Finally, ensure that the program prints the sum of all entered integers using a finally block.

Input Format:
The input consists of a sequence of integers and non-integer values entered by the user.
Output Format:
The output should display the sum of all entered integers.

Sample Input:
1 2 3 a 4
Sample Output:
Sum of entered integers: 10

22. Custom Exception for Age Verification

Problem Statement:
Create a Java program that verifies the age entered by the user. If the age is below 18, throw a custom exception called UnderageException. Handle the exception and display an appropriate message.

Description:

Your task is to implement a custom exception in Java. The program should read the age from the user. If the age is below 18, throw a custom exception called UnderageException. Catch the exception and display a message indicating that the user is underage. Ensure that the program continues to execute gracefully after handling the exception.

Input Format:
The input consists of a single integer representing the age.
Output Format:
The output should display an appropriate message based on the age verification.

Sample Input:
17
Sample output:
UnderageException: Age 17 is below the legal age limit.

23. Exception Handling in Array Operations

Problem Statement:
Write a Java program that performs array operations. The program should read an integer n from the user, create an array of size n, and allow the user to fill the array. If the user tries to access an index outside the array bounds, handle the ArrayIndexOutOfBoundsException and display an appropriate message.

Description:
Your task is to implement exception handling for array operations. The program should read the size of the array from the user and then allow the user to fill the array with integers. If the user tries to access or modify an index outside the bounds of the array, catch the ArrayIndexOutOfBoundsException and display a message indicating the error. Ensure the program continues to run gracefully after handling the exception.

Input Format:
The first input is an integer n representing the size of the array.
The next n inputs are the elements of the array.
The program then attempts to access an index outside the array bounds.
Output Format:
The output should display the elements of the array or an appropriate error message if an out-of-bounds access occurs.

Sample Input:
1
100 2
Sample Output:
ArrayIndexOutOfBoundsException: Index 2 out of bounds for length 1

24. Handling Multiple Exceptions

Problem Statement:
Create a Java program that reads two integers from the user and performs division. Handle ArithmeticException for division by zero and NumberFormatException for invalid integer input. Ensure the program prints a message for each exception and continues execution gracefully.

Description:
Your task is to handle multiple exceptions in a Java program. The program should read two integers from the user and perform division. If the user enters a non-integer value, catch the NumberFormatException and display an appropriate message. If the user attempts to divide by zero, catch the ArithmeticException

and display an appropriate message. Ensure that the program continues to execute gracefully after handling each exception.

Input Format:
The first input is the dividend.
The second input is the divisor.
Output Format:
The output should display the result of the division or an appropriate error message if an exception occurs.

Sample Input:
10
2
Sample Output:
Result: 5

25. Custom Exception for Invalid User Input

Problem Statement:
You are developing a user management system where users can register with a unique username. Implement a system where a custom exception is thrown when a user attempts to register with a username that contains forbidden characters or is too short.

Description:
Create a custom exception named InvalidUsernameException that extends the Exception class. This exception should be thrown if the username is less than 5 characters long or contains any non-alphanumeric characters. The UserManager class should handle this exception and inform the user of the error.

Input Format:

A single line of input, the username to be validated.
Output Format:

If the username is valid, output "Username registered successfully."
If the username is invalid, output "Invalid username: [error details]."

Sample Input:
john_doe
Sample Output:
Invalid username: Contains non-alphanumeric characters.

26. Custom Exception for Bank Account Balance

Problem Statement:
You are designing a banking application where users can withdraw money from their account. Implement a custom exception that is thrown if a withdrawal amount exceeds the account balance.

Description:
Create a custom exception named InsufficientFundsException that extends Exception. This exception should be thrown if a withdrawal attempt exceeds the account balance. The BankAccount class should handle this exception and provide appropriate feedback.

Input Format:

The initial balance of the account (a floating-point number).
The withdrawal amount (a floating-point number).
Output Format:

If the withdrawal amount is valid, output "Withdrawal successful. Remaining balance: [amount]."
If the withdrawal amount exceeds the balance, output "Insufficient funds: [error details]."

Sample Input:
500
600
Sample Output:
Insufficient funds: Withdrawal exceeds balance.

27. Reordering Elements in a List

Problem Statement:
Given a list of integers, reorder the list so that all odd numbers come before all even numbers while maintaining the relative order of odd and even numbers.

Description:
You need to read a list of integers from the user and reorder it such that all odd numbers come before all even numbers. The relative order of odd and even numbers should be preserved. You must use Java's List interface and iterators.

Input Format:

The first line contains an integer,
n
n (1 ≤
n
n ≤ 100), denoting the number of elements in the list.
The second line contains
n
n space-separated integers.
Output Format:

Print the reordered list, with all odd numbers appearing before all even numbers, maintaining their relative order.

Sample Input:
5
4 3 2 1 5

Sample Output:
3 1 5 4 2

28. Counting Unique Words

Problem Statement:
Read a string from the user and count the number of unique words using a Set.

Description:
You need to read a string of text from the user and count how many unique words are present in it. Words

are considered case-insensitive and separated by spaces. Use Java's Set interface to achieve this.

Input Format:

The input consists of a single line containing a string of words.
Output Format:

Print the number of unique words in the string.

Sample Input:
Hello world hello

Sample Output:
2

29. **Intersection of Two Sets**

Problem Statement:
Find the intersection of two sets of integers.

Description:
You need to read two sets of integers from the user and print their intersection. Use Java's Set interface and its operations to achieve this.

Input Format:

The first line contains an integer,
n ($1 \leq n \leq 100$), denoting the number of elements in the first set.
The second line contains
n space-separated integers.
The third line contains an integer,
m ($1 \leq m \leq 100$), denoting the number of elements in the second set.
The fourth line contains
m space-separated integers.
Output Format:

Print the elements in the intersection of the two sets, one per line. If there are no common elements, print "No common elements."

Sample Input:
4
1 2 3 4
3
3 4 5

Sample Output:
3
4

30. **Creating a Frequency Map**

Problem Statement:
Read a list of words and create a frequency map of each word.

Description:
You need to read a list of words from the user and create a map where the keys are the words and the values are their respective frequencies. Use Java's Map interface to store and calculate the frequencies.

Input Format:

The first line contains an integer,
n ($1 \leq n \leq 100$), denoting the number of words.
The second line contains
n space-separated words.
Output Format:

Print each word and its frequency, one per line.

Sample Input:
4
apple banana apple grape

Sample Output:
banana 1
apple 2
grape 1

31. **"Unique Items in a Shopping List"**

Problem Statement:
You are developing a shopping application where users can maintain their shopping lists. The application needs to handle scenarios where users can add, remove, and view items in their shopping list. The unique feature of the application is that it must manage items efficiently while ensuring that no duplicates are allowed and each item is stored in a way that preserves its insertion order.

Description:
You need to implement a Java program that:

Allows users to add items to a shopping list.
Allows users to remove items from the list.
Displays the current shopping list without duplicates, maintaining the order of insertion.
Input Format:

First line: Integer
N (Number of operations to perform)
Next
N lines: Each line will contain an operation in the format "ADD <item>" or "REMOVE <item>". The item is a string with spaces.
Output Format:

After processing all operations, print the final shopping list with items in their insertion order, separated by commas.

Sample Input:
5

ADD Apples
ADD Bananas
ADD Apples
REMOVE Bananas
ADD Grapes

Sample Output:
Apples, Grapes

32. "Frequency of Words in Text"

Problem Statement:
You are tasked with implementing a feature that counts the frequency of each word in a given text. The twist is that the word frequencies need to be sorted in descending order of frequency, and in case of a tie, the words should be sorted lexicographically.

Description:
You need to implement a Java program that:

Reads a block of text from the user.
Counts the frequency of each word in the text.
Outputs the words and their frequencies sorted by frequency (highest first) and lexicographically in case of ties.
Input Format:

A single line of text (can contain multiple words and punctuation).
Output Format:

Each line should contain a word followed by its frequency, sorted by frequency in descending order and lexicographically for tied frequencies.

Sample Input:
apple banana apple fruit banana apple
Sample Output:
apple 3
banana 2
fruit 1

33. Grouping Words by Length

Problem Statement:
Group words by their length from a given list of words.

Description:
You need to read a list of words from the user and group them by their length. Use Java's Map interface where the key is the length of the words and the value is a list of words of that length.

Input Format:

The first line contains an integer,
n ($1 \leq n \leq 100$), denoting the number of words.
The second line contains
n space-separated words.

Output Format:

Print each length and the corresponding list of words.

Sample Input:
5
cat dog elephant rat bat

Sample Output:
3: [cat, dog, rat, bat]
8: [elephant]

34. Finding the Longest Word

Problem Statement:
Given a list of words, find the longest word. If multiple words have the same maximum length, print them all.

Description:
You need to read a list of words from the user and identify the longest word(s). If there are multiple words with the same maximum length, print all of them. Use Java's List interface and iterators to solve this problem.

Input Format:

The first line contains an integer,
n ($1 \leq n \leq 100$), denoting the number of words.
The second line contains
n space-separated words.
Output Format:

Print the longest word(s). If multiple words have the same maximum length, print them all on separate lines.

Sample Input:
5
apple
banana
cherry
date
guva
Sample Output:
banana
cherry

35. Lambda Expressions

Question
Implement a Java program that uses a lambda expression to calculate the sum of all even numbers and the product of all odd numbers in a list provided by the user. The program should then return the difference between the sum of even numbers and the product of odd numbers.

Description

Write a Java program that:

Accepts a list of integers from the user.
Uses a lambda expression to compute the sum of all even numbers.
Uses another lambda expression to compute the product of all odd numbers.
Outputs the difference between the sum of even numbers and the product of odd numbers.
Input Format
An integer n denoting the number of elements in the list.
n space-separated integers representing the list elements.
Output Format
A single integer which is the difference between the sum of even numbers and the product of odd numbers.

Sample Input:
5
1 2 3 4 5
Sample Output:
-9

36. Functional Interfaces

Question
Create a Java program using a custom functional interface to find the longest string in a list provided by the user. The program should then return the length of this longest string.

Description
Write a Java program that:

Accepts a list of strings from the user.
Defines a custom functional interface with a method to find the longest string.
Uses this functional interface to find the longest string in the list.
Outputs the length of the longest string.
Input Format
An integer n denoting the number of strings in the list.
n space-separated strings representing the list elements.
Output Format
A single integer which is the length of the longest string.

Sample Input:
4
apple
banana
strawberry
kiwi
Sample Output:
10

37. Local Variable Type Inference (var)

Question
Implement a Java program that uses var to store a list of user-input integers and then calculates the average of these integers. The program should handle the list and the average calculation using var.

Description

Write a Java program that:

Accepts a list of integers from the user.
Uses var to declare and initialize the list.
Calculates the average of the list elements using var.
Outputs the calculated average.
Input Format
An integer n denoting the number of elements in the list.
n space-separated integers representing the list elements.
Output Format
A single floating-point number which is the average of the list elements.

Sample Input:
4
1 2 3 4
Sample Output:
2.5

38. Stream API

Question
Create a Java program that uses the Stream API to read a list of integers from the user, filter out the prime numbers, sort them in descending order, and then print the sorted list.

Description
Write a Java program that:

Accepts a list of integers from the user.
Uses the Stream API to filter out prime numbers.
Sorts the filtered prime numbers in descending order.
Outputs the sorted list of prime numbers.
Input Format
An integer n denoting the number of elements in the list.
n space-separated integers representing the list elements.
Output Format
The sorted list of prime numbers in descending order, each number on a new line.

Sample Input:
5
2 3 4 5 6

Sample Output:
5
3
2

39. Stream API and Local Variable Type Inference (var)

Question
Develop a Java program that reads a list of words from the user, uses the Stream API to filter out words with more than 5 characters, converts the filtered words to uppercase, and then prints the resulting list. Use var to declare all local variables.

Description
Write a Java program that:

Accepts a list of words from the user.
Uses the Stream API to filter out words with more than 5 characters.
Converts the filtered words to uppercase.
Outputs the resulting list of uppercase words.
Input Format
An integer n denoting the number of words in the list.
n space-separated words representing the list elements.
Output Format
The resulting list of uppercase words, each word on a new line.

Sample Input:
4
apple
banana
pear
fig

Sample output:
PEAR
FIG

40. Lambda Expressions

Question
Write a Java program using a lambda expression to determine if the list of strings provided by the user contains any palindromes. The program should return "Yes" if there is at least one palindrome, otherwise "No".

Description
Write a Java program that:

Accepts a list of strings from the user.
Uses a lambda expression to check if any string in the list is a palindrome.
Outputs "Yes" if there is at least one palindrome, otherwise "No".
Input Format
An integer n denoting the number of strings in the list.
n space-separated strings representing the list elements.
Output Format
"Yes" if there is at least one palindrome in the list, otherwise "No".

Sample Input:
4
level
racecar
hello
world

Sample Output:
Yes

41. Local Variable Type Inference (var)

Question

Write a Java program that uses var to store a list of user-input integers, and then finds the second largest number in the list. The program should use var to handle all variables.

Description

Write a Java program that:

Accepts a list of integers from the user.
Uses var to declare and initialize the list and all other variables.
Finds the second largest number in the list.
Outputs the second largest number.
Input Format
An integer n denoting the number of elements in the list.
n space-separated integers representing the list elements.
Output Format
A single integer which is the second largest number in the list.

Sample Input:
5
1 2 3 4 5

Sample Output:
4

42. Stream API

Question

Create a Java program that uses the Stream API to read a list of integers from the user, squares each number, filters out the squares that are not divisible by 3, and then prints the resulting list.

Description

Write a Java program that:

Accepts a list of integers from the user.
Uses the Stream API to square each number.
Filters out the squares that are not divisible by 3.
Outputs the resulting list.
Input Format
An integer n denoting the number of elements in the list.
n space-separated integers representing the list elements.
Output Format
The resulting list of squares that are divisible by 3, each number on a new line.

Sample Input:
4
1 2 3 4

Sample Output:
9

43. Custom Comparator with Lambda Expression

Problem Statement

Create a custom comparator using a lambda expression to sort a list of Person objects based on their age and then by their name in alphabetical order.

Description

You are given a list of Person objects, where each Person has a name (String) and an age (int). Write a program that sorts the list first by age in descending order. If two people have the same age, sort them by their name in ascending order. Implement this using a lambda expression with a custom comparator.

Input Format

The first line contains an integer n, the number of Person objects.
The next n lines each contain two values: a name (String) and age (int) of a Person.
Output Format
Print the sorted list of Person objects in the specified order.

Sample Input:
5
Alice 30
Bob 25
Charlie 30
Dave 22
Eve 25

Sample Output:
Alice 30
Charlie 30
Bob 25
Eve 25
Dave 22

44. Filtering Even Numbers using Functional Interface

Problem Statement

Filter out even numbers from a list of integers using a lambda expression with a custom functional interface.

Description

You are given a list of integers. Define a functional interface Condition with a method test(int number) that returns a boolean. Use this interface with a lambda expression to filter out even numbers from the list.

Input Format

The first line contains an integer n, the number of integers.
The next line contains n integers separated by spaces.
Output Format
Print the list of odd numbers, each on a new line.

Sample Input:
6
1 2 3 4 5 6

Sample Output:

1
3
5

45. Applying Function to a List

Problem Statement

Use a lambda expression to apply a custom function to a list of strings to transform each string to uppercase and then sort the list in reverse alphabetical order.

Description

You are given a list of strings. Define a functional interface StringFunction with a method apply(String s) that transforms a string. Use this interface with a lambda expression to convert each string to uppercase. Finally, sort the transformed list in reverse alphabetical order.

Input Format

The first line contains an integer n, the number of strings.
The next n lines each contain a string.
Output Format
Print the transformed and sorted list of strings, each on a new line.

Sample Input:
4
hello
world
java
lambda

Sample Output:
WORLD
LAMBDA
JAVA
HELLO

46. Combining Two Lists with Functional Interfaces

Problem Statement

Combine two lists of integers by applying a lambda expression that adds corresponding elements from the two lists.

Description

You are given two lists of integers. Define a functional interface Combiner with a method combine(int a, int b) that combines two integers. Use this interface with a lambda expression to add corresponding elements from the two lists. The lists are guaranteed to be of the same length.

Input Format

The first line contains an integer n, the number of integers in each list.
The next n lines contain integers for the first list.
The next n lines contain integers for the second list.
Output Format
Print the resulting list after combining each corresponding element.

Sample Input:

3
1
2
3
4
5
6

Sample Output:
5
7
9

47. **Wrapper Class Manipulation**

Problem Statement:
Write a Java program that takes a string input representing a list of integers (e.g., "1 2 3 4 5") and performs the following operations:

Convert the string into an array of integers using Integer wrapper class methods.
Compute the sum of the integers and print it.
Print the maximum and minimum values from the array using Integer methods.
Print the average of the integers rounded to two decimal places.
Description:
Your task is to utilize Integer wrapper class methods to parse the input string, perform computations, and display results. The input string may contain both positive and negative integers separated by spaces.

Input Format:

A single line of input containing space-separated integers.
Output Format:

A single line containing the sum of integers.
A second line containing the maximum and minimum values.
A third line containing the average of the integers rounded to two decimal places.

Sample Input:
10 -3 5 7 2
Sample Output:
21
10 -3
4.20

48. **Wrapper Classes and Arithmetic Operations**

Problem Statement:
Create a Java program that performs arithmetic operations on integers represented as String inputs using Integer wrapper class methods. The program should:

Parse two integers from string inputs.
Perform addition, subtraction, multiplication, and division.
Print the results of each operation.
Description:

Use Integer.parseInt() to convert the String inputs into integers and perform the arithmetic operations. Handle division by zero appropriately.

Input Format:

Two lines of input, each containing a string representation of an integer.
Output Format:

The results of addition, subtraction, multiplication, and division.

Sample Input:
20
5
Sample Output:
Addition: 25
Subtraction: 15
Multiplication: 100
Division: 4

49. Wrapper Classes and Type Conversion

Problem Statement:
Create a Java program that converts various types of data (e.g., double, float) to integers using the Integer wrapper class methods. The program should:

Convert a double value to an integer using Double and Integer methods.
Convert a float value to an integer.
Handle any potential issues related to data loss or rounding.
Description:
Use Double.intValue() and Float.intValue() methods to perform the conversions. Print the integer values and any relevant information about rounding or truncation.

Input Format:

Two lines of input: one containing a double value and the other a float value.
Output Format:

The integer values obtained from conversions.
Information on rounding or data loss.

Sample Input:
15.75
3.14
Sample Output:
Converted Integer from Double: 15
Converted Integer from Float: 3
Note: Data was truncated, not rounded.

50. Unique Sum Calculation

Problem Statement:
Write a Java program to calculate the unique sum of integer elements in a list. You need to use Integer wrapper class methods to achieve this. The unique sum is defined as the sum of all distinct integers in the

list.

Description:
The program should read a list of integers from the user and compute the sum of distinct integers. The Integer class methods should be used for operations such as parsing and comparison.

Input Format:

The first line contains an integer N, the number of elements in the list.
The second line contains N space-separated integers.
Output Format:

Print the unique sum of the integers in the list.

Sample Input:
5
1 2 2 3 4

Sample Output:
10

51. Wrapper Class Conversion

Problem Statement:
Write a Java program that converts a list of string representations of numbers into their Integer equivalent and calculates the product of all even integers in the list.

Description:
The program should use Integer.parseInt() for conversion and Integer wrapper class methods to determine if a number is even.

Input Format:

The first line contains an integer N, the number of elements in the list.
The second line contains N space-separated strings representing integers.
Output Format:

Print the product of all even integers in the list. If there are no even integers, print 0.

Sample Input:
5
2 3 4 5 6
Sample Output:
48

52. Sum of Hexadecimal Numbers

Problem Statement:
Write a Java program to sum a list of hexadecimal numbers provided as strings. You must use the Integer.parseInt() method with base 16 for conversion and Integer.toHexString() for the final result.

Description:
Convert each hexadecimal string to an integer, compute the sum, and then output the sum as a hexadecimal

string.

Input Format:

The first line contains an integer N, the number of hexadecimal numbers.
The second line contains N space-separated hexadecimal strings.
Output Format:

Print the sum of the hexadecimal numbers in hexadecimal format (without 0x prefix).

Sample Input:
2
10 f
Sample Output:
1f

53. Integer Frequency

Problem Statement:
Write a Java program to determine the frequency of each integer in a list. Use the Integer wrapper class and appropriate collection classes to achieve this.

Description:
The program should use a Map<Integer, Integer> to count the occurrences of each integer and output the frequency of each integer in ascending order.

Input Format:

The first line contains an integer N, the number of elements in the list.
The second line contains N space-separated integers.
Output Format:

Print each integer and its frequency, sorted by the integer values.

Sample Input:
5
1 2 2 3 3 3

Sample Output:
1: 1
2: 2
3: 2

54. Binary Representation Analysis

Problem Statement:
Write a Java program that reads a list of integers from the user and performs the following tasks using the Integer wrapper class:

Convert each integer to its binary representation.
Count the number of 1s in each binary string.
Calculate the average count of 1s across all integers and print it.
Description:

Use Integer.toBinaryString() to convert integers to their binary forms and analyze the binary strings to count the number of 1s. Compute the average count of 1s for all integers.

Input Format:

The first line contains an integer N, the number of integers.
The second line contains N space-separated integers.
Output Format:

Print the average number of 1s in binary representations, rounded to the nearest integer.

Sample Input:
4
3 5 7 9

Sample Output:
2

55. **Range and Sign Analysis**

Problem Statement:
Write a Java program that reads a list of integers and determines the range and sign of the integer values. Use the Integer wrapper class to perform the following tasks:

Find the minimum and maximum integer values from the list.
Check if each integer is positive, negative, or zero.
Print the results in a formatted output.
Description:
The program should use Integer.MIN_VALUE and Integer.MAX_VALUE to determine the range and manually check the sign of each integer.

Input Format:

The first line contains an integer N, the number of integers.
The second line contains N space-separated integers.
Output Format:

Print the minimum and maximum integer values.
For each integer, print its sign.

Sample Input:
4
-10 0 15 23

Sample Output:
Min: -10
Max: 23
-10: Negative
0: Zero
15: Positive
23: Positive

56. Generic Pair Operations with User Input

Problem Statement:
Write a Java program that defines a generic class Pair<T, U> to hold a pair of values. The class should provide methods to:

Set the values of the pair.
Get the values of the pair.
Swap the values of the pair.
Print the pair in a formatted manner.
The program should prompt the user to input values for the Pair object and demonstrate the following:

Create a Pair object with user-provided values.
Print the original pair.
Swap the values of the pair.
Print the swapped pair.
Description:
The program should involve creating a generic class Pair with two types, T and U. The class should include:

A constructor to initialize the pair.
Methods to set, get, and swap values.
A method to print the pair in a readable format.
In the main method, prompt the user to input the values for a Pair of type String and Integer.

Input Format:

The user first inputs the type of Pair (String or Integer) for the first and second values.
Then, the user inputs the first value followed by the second value.
Output Format:

Print the original pair.
Print the swapped pair.

Sample Input:
Integer
100
String
One Hundred
Sample Output:
Original Pair: (100, One Hundred)
Swapped Pair: (One Hundred, 100)

57. Generic Stack Implementation

Problem Statement:
Create a generic Stack<T> class that supports the following operations:

Push an item onto the stack.
Pop an item from the stack.
Peek at the top item of the stack.
Check if the stack is empty.
Print the stack contents.

Write a main method that allows the user to interact with the stack by providing commands to push, pop, and peek values. The stack should support different data types such as Integer and String.

Description:
The program should involve creating a generic Stack class with basic stack operations. The class should use an ArrayList to store the elements and include methods for stack operations.

Input Format:

The user provides commands in the format push <value>, pop, peek, or print.
The value to push should be provided as part of the command.
Output Format:

Print the result of each operation or the contents of the stack.

Sample Input:
push 10
push Hello
peek
pop
print
exit
Sample Output:
Top of the stack: Hello
Popped value: Hello
Stack contents: [10]

58. Generic Sorting with Comparators

Problem Statement:
Create a generic Sorter<T> class that sorts an array of elements using a Comparator<T>. The Sorter class should have a method sort that accepts an array and a comparator. Demonstrate its usage by sorting arrays of Integer and String.

Description:
The program should involve creating a generic Sorter class with a sort method that uses a Comparator to sort the elements of an array. The main method should prompt the user to choose the type of array to sort and provide the array elements.

Input Format:

The user selects the type of array (Integer or String).
For Integer, input a list of integers.
For String, input a list of strings.
Output Format:

Print the sorted array.

Sample Input:
Integer
5 2 9 1 5
Sample Output:

Sorted Integer Array: [1, 2, 5, 5, 9]

59. Generic Queue Implementation

Problem Statement:
Create a generic Queue<T> class that implements a basic queue with the following operations:

Enqueue an item to the queue.
Dequeue an item from the queue.
Peek at the front item of the queue.
Check if the queue is empty.
Print the queue contents.
Write a main method that interacts with the Queue class by allowing the user to enqueue, dequeue, and peek items, and print the queue contents. The queue should support different data types such as Integer and String.

Description:
The program should involve creating a generic Queue class that supports standard queue operations using a linked list. The class should have methods to enqueue, dequeue, and peek items, and check if the queue is empty.

Input Format:

The user provides commands in the format enqueue <value>, dequeue, peek, or print.
The value to enqueue should be provided as part of the command.
Output Format:

Print the result of each operation or the contents of the queue.

Sample Input:
enqueue 5
enqueue World
print
exit

Sample Output:
Queue contents: [5, World]

60. Generic Pair Comparison

Problem Statement:
Create a generic Pair<T, U> class with methods to:

Compare two pairs to check if they are equal.
Print a detailed comparison result showing if the pairs are equal or not.
Write a main method that:

Prompts the user to input pairs of Integer and String.
Compares the pairs and displays the result.
Description:
The program should involve creating a generic Pair class with a comparison method. The class should implement the equals method to compare pairs based on their values and print a detailed comparison result.

Input Format:

The user inputs two pairs, each with values for Integer and String.
Output Format:

Print whether the pairs are equal or not based on their values.

Sample Input:
10
apple
10
apple
Sample Output:
The pairs are equal.

(Course Lead) **(Program Chair)** **(Dean)**