Course Code CSE 3002	Programming in Java	Course Type LTP	Credits 4
Prerequisite	Object oriented programming with C++		

Course Objectives:

- Creating high-performing multi-threaded applications
- Creating Java technology applications that leverage the object-oriented features of the Java language, such as encapsulation, inheritance, and polymorphism
- Implementing input/output (I/O) functionality to read from and write to data and text files and understand advanced I/O streams
- Executing a Java technology application from the command line
- Manipulating files, directories and file systems using the JDK NIO.2 specification
- Creating applications that use the Java Collections framework
- Performing multiple operations on database tables, including creating, reading, updating and deleting using both JDBC and JPA technology
- Searching and filter collections using Lambda Expressions
- Implementing error-handling techniques using exception handling
- Using Lambda Expression concurrency features

Course Outcomes:

At the end of the course, students should able to

- Create Java programs that solve simple businessproblems using object oriented approach
- Demonstrate synchronization among different processes using multithreading approach
- To develop and create real time applications using JDBC and JPA technology

Student Outcomes (SO): a,b, c,l

a. An ability to apply the knowledge of mathematics, science and computing appropriate to the discipline

b. An ability to analyze a problem, identify and define the computing requirements appropriate to its solution.

c. An ability to design, implement and evaluate a system / computer-based system, process, component or program to meet desired needs

l. An ability to apply mathematical foundations, algorithmic principles and computer science theory in the modelling and design of computer-based systems (CS)

Unit No	Unit Content	No. of hours	SOs
1	**Java Platform Overview:** Defining how the Java language achieves platform independence - Differentiating between the Java ME, Java SE, and Java EE Platforms Evaluating Java libraries, middle-ware, and database options - Defining how the Java language continues to evolve - **Java Syntax and Class Review:** Creating simple Java classes -Creating primitive variables Using operators - Creating and manipulate strings - Using if-else and switch statements - Iterating with loops: while, do-while, for, enhanced for Creating arrays Using Java fields - constructors, and methods. **Encapsulation and Subclassing:** Using encapsulation in Java class design - Modeling business problems using Java classes - Making classes	**10**	**a,b,c**

	immutable - Creating and use Java subclasses - Overloading methods.		
2	**Overriding Methods, Polymorphism, and Static Classes:** Using access levels: private, protected, default, and public - Overriding methods - Using virtual method invocation- Using varargs to specify variable arguments - Using the instanceof operator to compare object types - Using upward and downward casts - Modeling business problems by using the static keyword - Implementing the singleton design pattern. **Abstract and Nested Classes**: Designing general-purpose base classes by using abstract classes - Constructing abstract Java classes and subclasses - Applying final keyword in Java- Distinguish between top-level and nested classes. **Interfaces and Lambda Expressions**: Defining a Java interface- Choosing between interface inheritance and class inheritance- Extending an interface - Defaulting methods - Anonymous inner classes - Defining a Lambda Expression.	**12**	**a,b,c**
3	**Collections and Generics:** Creating a custom generic class - Using the type inference diamond to create an object - Creating a collection by using generics - Implementing an ArrayList - Implementing a TreeSet - Implementing a HashMap - Implementing a Deque - Ordering collections. **Collections Streams, and Filters**: Describing the Builder pattern - Iterating through a collection using lambda syntax - Describing the Stream interface - Filtering a collection using lambda expressions - Calling an existing method using a method reference - Chaining multiple methods together - Defining pipelines in terms of lambdas and collections. **Lambda Built-in Functional Interfaces**: Listing the built-in interfaces included in java.util.function - Core interfaces - Predicate, Consumer, Function, Supplier - Using primitive versions of base interfaces - Using binary versions of base interfaces - **Lambda Operations:** Extracting data from an object using map - Describing the types of stream operations - Describing the Optional class - Describing lazy processing - Sorting a stream - Saving results to a collection using the collect method - Grouping and partition data using the Collectors class**.**	**12**	**a,b,c**
4	**Exceptions and Assertions:** Defining the purpose of Java exceptions - Using the try and throw statements - Using the catch, multi-catch, and finally clauses – Auto close resources with a try-with-resources statement - Recognizing common exception classes and categories - Creating custom exceptions - Testing invariants by using assertions. **Java Date/Time API**: Creating and manage date-based events - Creating and manage time-based events - Combining date and time into a single object - Working with dates and times across time zones - Managing changes resulting from daylight savings - Defining and create timestamps, periods and durations - Applying formatting to local and zoned dates and times. **I/O Fundamentals:** Describing the basics of input and output in Java -	**12**	**a,b,c**

	Read and write data from the console - Using streams to read and write files - Writing and read objects using serialization. **File I/O (NIO.2)**: Using the Path interface to operate on file and directory paths - Using the Files class to check, delete, copy, or move a file or directory - Using Stream API with NIO2.		
5	**Concurrency**: Describing operating system task scheduling - Creating worker threads using Runnable and Callable - Using an ExecutorService to concurrently execute tasks - Identifying potential threading problems - Using synchronized and concurrent atomic to manage atomicity - Using monitor locks to control the order of thread execution - Using the java.util.concurrent collections - **The Fork-Join Framework**: Parallelism - The need for Fork-Join Work stealing - Recursive Task. **Parallel Streams**: Reviewing the key characteristics of streams - Describing how to make a stream pipeline execute in parallel - List the key assumptions needed to use a parallel pipeline - Defining reduction - Describing why reduction requires an associative function - Calculating a value using reduce - Describing the process for decomposing and then merging work - Listing the key performance considerations for parallel streams. **Database Applications with JDBC**: Defining the layout of the JDBC API - Connecting to a database by using a JDBC driver - Submitting queries and get results from the database - Specifying JDBC driver information externally - Performing CRUD operations using the JDBC API. **Localization**: Describing the advantages of localizing an application - Defining what a locale represents - Read and set the locale by using the Locale object - Building a resource bundle for each locale - Calling a resource bundle from an application - Changing the locale for a resource bundle.	**12**	**a,b,c**
6	**Guest Lecture on Contemporary Topics**	**2**	
	Total Hours:	**60**	

Mode of Teaching and Learning: *Flipped Class Room, Activity Based Teaching/Learning, Digital/Computer based models, wherever possible to augment lecture for practice/tutorial and minimum 2 hours lectures by industry experts on contemporary topics*

Mode of Evaluation and assessment:
The assessment and evaluation components may consist of unannounced open book examinations, quizzes, student's portfolio generation and assessment, and any other innovative assessment practices followed by faculty, in addition to the Continuous Assessment Tests and Final Examinations.

Text Books:

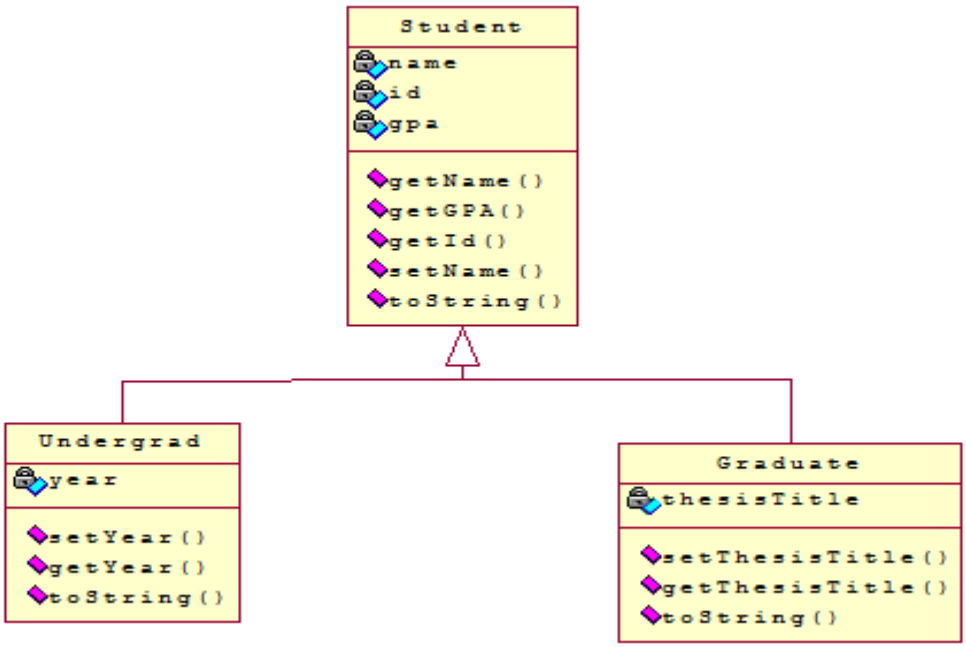
1. **Herbert Schildt, "Java The complete reference", 11th edition, Oracle press , 2018**

Reference Books:

1. **Oracle University Reference E-Kit**
2. **Deitel and Deitel, "Java How to Program (Early objects)",10thedition,Pearson, 2015**
3. **Cay S.Horstmann and Gary Cornell, "Core Java Vol I–Fundamentals",8thedition,Pearson, 2011**
4. **Steven Holzner et al., "Java 2 Black Book", Dreamtech press, Reprint edition 2010**

Indicative List of Experiments

No.	Description of Experiment	SO
1	Write an application that displays a box, an oval, an arrow and a diamond using asterisks(*).	l
2	One interesting application of computers is to display graphs and bar charts. Write an application that reads five numbers between 1 and 30. For each number that is read, your program should display the same number of adjacent asterisks. For example, if your program reads the number 7, it should display *******. Display the bars of asterisks after your read all five numbers.	l
3	Write an application with following method to resolve two player game of Rock-Paper-Scissors. rockPaperScissorsmethod takes two parameters: a string representing a first player's move in a game of Rock-Paper-Scissors and a string representing a second player's move. Depending on each player's move, the function should output to the console the winner of the round if there was a winner, a tie if both players had the same move, or whether a player gave an invalid move. When comparing moves, letter-casing does not matter ("rock" should be considered the same move as "rOcK"). The two-player game of Rock-Paper-Scissors works as following: Each player chooses one of three moves: "rock", "paper", or "scissors" A move of "rock" wins over another player's move of "scissors" A move of "scissors" wins over another player's move of "paper" A move of "paper" wins over another player's move of "rock" If two players use the same move, the result is a tie For example, a call of rockPaperScissors("rock", "scissors") would result in console output of "Player 1 wins!". A call of rockPaperScissors("ROCK", "Paper") would result in console output of "Player 2 wins!". A call of rockPaperScissors("SCISSORS", "Scissors") would result in console output of "TIE!". If one player gives a move other than "rock", "paper", or "scissors", the console output should be "Invalid move of [player's invalid move]!", where you should replace [player's invalid move] with the exact value passed by that player. If both players pass invalid moves, the console output should be "Invalid moves of [player 1's invalid move] and [player 2's invalid move]!".	l
4	Define a class named TimeSpan. A TimeSpan object stores a span of time in hours and minutes (for example, the time span between 6:00am and 8:30am is 2 hours, 30 minutes). Each TimeSpan object should have the following public methods: TimeSpan(hours, minutes) Constructs a TimeSpan object storing the given time span of hours and minutes. getHours() Returns the number of hours in this time span. getMinutes() Returns the number of minutes in this time span, between 0 and 59. add(hours, minutes) Adds the given amount of time to the span. For example, (1 hours, 15 min) + (2 hour,	l

	15 min) = (3 hours 30 min). Assume that the parameters are valid: the hours are non-negative, and the minutes are between 0 and 59. add(timespan) Adds the given amount of time (stored as a time span) to the current time span. getTotalHours() Returns the total time in this time span as the real number of hours, such as 9.75 for (9 hours, 45 min). toString() Returns a string representation of the time span of hours and minutes, such as "28h46m". The minutes should always be reported as being in the range of 0 to 59. That means that you may have to "carry" 60 minutes into a full hour.	
5	Implement the following hierarchy using Java Inheritance. The class Student is the parent class. Note that all the variables are private and hence the child classes can only use them through accessor and mutator methods 	l
6	Consider a superclass PurchaseItem which models customer's purchases. This class has: - two private instance variables name (String) and unitPrice (double). - One constructor to initialize the instance variables. - A default constructor to initialize name to "no item", and unitPrice to 0. use this() - A method getPrice that returns the unitPrice. - Accessor and mutator methods. - A toString method to return the name of the item followed by @ symbol, then the unitPrice. Consider two subclasses WeighedItem and CountedItem. WeighedItem has an additional instance variable weight (double) in Kg while CountedItem has an	l

<table>
<tr><td></td><td>additional variable quantity (int) both private.
- Write an appropriate constructor for each of the classes making use of the constructor of the superclass in defining those of the subclasses.
- Override getPrice method that returns the price of the purchasedItem based on its unit price and weight (WeighedItem), or quantity (CountedItem). Make use of getPrice of the superclass
- Override also toString method for each class making use of the toString method of the superclass in defining those of the subclasses.
toString should return something that can be printed on the receipt.
For example
Banana @ 3.00 1.37Kg 4.11 SR (in case of WeighedItem class)
Pens @ 4.5 10 units 45 SR (in case of CountedItem class)
Write an application class where you construct objects from the two subclasses and print them on the screen.</td><td></td></tr>
<tr><td>7</td><td>Write a program to create a class named shape. In this class we have three sub classes circle, triangle and square each class has two member functionnamed draw () and erase (). Create these using polymorphism concepts</td><td></td></tr>
<tr><td>8</td><td>In computing, the producer–consumer problem (also known as the bounded-buffer problem) is a classic example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer, which share a common, fixed-size buffer used as a queue.
The producer’s job is to generate data, put it into the buffer, and start again.
At the same time, the consumer is consuming the data (i.e. removing it from the buffer), one piece at a time.
Problem:
To make sure that the producer won’t try to add data into the buffer if it’s full and that the consumer won’t try to remove data from an empty buffer.
Solve the problem using Java Multithreading concepts.</td><td>l</td></tr>
<tr><td>9</td><td>Write a method ChangeChar that accepts two parameters: a Scanner representing an input file, and a PrintStream representing an output file. Your method should convert the input file's text to different dialect where various letters are replaced by other letters/numbers. Output the new version of the text to the given output file. Preserve the original line breaks from the input. Also wrap each word of input in parentheses. Perform the following replacements:
<table>
<tr><th>Original character</th><th>Change character</th></tr>
<tr><td>O</td><td>0</td></tr>
<tr><td>l (lowercase L)</td><td>1</td></tr>
<tr><td>E</td><td>6</td></tr>
<tr><td>A</td><td>4</td></tr>
<tr><td>T</td><td>7</td></tr>
<tr><td>s (at the end of a word only)</td><td>5</td></tr>
</table>
For example, if the input file lincoln.txt contains the following text:
four score and
seven years ago our
fathers brought forth on this continent</td><td>l</td></tr>
</table>

	a new nation the output file leet.txt should contain the following text: (f0ur) (sc0r6) (4nd) (s6v6n) (y64rZ) (4g0) (0ur) (f47h6r5) (br0ugh7) (f0r7h) (0n) (7hi5) (c0n7in6n7) (4) (n6w) (n47i0n)	
10	Write a program to illustrate the various Lambda Operations	l
11	Create a simple online quiz application using JDBC and JPA technology	l

Recommendation by the Board of Studies on	
Approval by Academic council on	
Compiled by	