Course code	Advanced Java Programming	L	T	P	J	C
ITE2005		3	0	2	0	4
Pre-requisite	ITE1002	Syllabus version				
		1.00				

Course Objectives:

- To learn the Advanced concepts in J2SE
- To understand Web Application Development, Database Connectivity and its implementation using Servlets, JSP and JDBC
- To introduce advanced Java frameworks for improving the web application design

Expected Course Outcome:

- Understand and implement advanced Java concepts
- Develop Java based Web applications using Servlets and JSP
- Incorporate cutting-edge frameworks in web application development

Student Learning Outcomes (SLO): 2,5,17

Module:1	Introduction to Java Programming:	6 hours	SLO:2

Features of Java, Data Types, Variables, Operators, Arrays, Control Statements. Introducing Classes and Objects, Methods, Inheritance, Packages and Interfaces, Exception Handling, Inner classes, String Handling

Module:2	Exploring Core Java	6 hours	SLO:5

Multithreaded Programming, Files and IO Streams, Object Serialization ,Applets , Java GUI Programming and Event Handling, Java Networking, RMI, Reflection, Collections, Generics, Java Autoboxing and Annotations

Module:3	Introducing JavaEE	6 hours	SLO: 17

Enterprise Java, Basic Application Structure, Using Web Containers, Creating Servlets, Configuring Servlets, Understanding HTTP methods, Using Parameters and Accepting Form Submissions, Using Init parameters, File Uploading, JDBC

Module:4	Java Server Pages	6 hours	SLO: 17

Creating JSPs, Using Java within JSP, Combining Servlets and JSPs, Maintaining State using Sessions, JSP 2.0 EL, Using Javabeans components in JSP Documents, JSP Custom Tag Library, Integrating Servlets and JSP: Model View Controller Architecture

Module:5	Struts Framework	6 hours	SLO: 17

Introduction to Struts – Building a Simple Struts Application – Understanding Model, View and Controller Layer- Overview of Tiles

Module:6	Java Server Faces(JSF)	7 hours	SLO: 17

Introduction to Java Server Faces (JSF)- JSF Application Architecture – Building a simple JSF Application - JSF Request Processing Lifecycle – The Facelets View Declaration Language – User Interface Component Model- JSF Event Model

Module:7	Spring Framework and Hibernate	6 hours	SLO: 17

Understanding Inversion of Control (IoC), Aspect Oriented Programming (AOP) and Dependency Injection, MVC pattern for Web Applications, Spring Framework, Understanding Application Context, Bootstrapping Spring framework, Configuring Spring framework, Data Persistence, Object/relational Mapping, Hibernate ORM, Mapping Entities to Tables

Module:8	**Contemporary issues:**	**2 hours**	
	Total Lecture hours:	**45 hours**	

Text Book(s)

1. Herbert Schildt, The Complete Reference-Java, Tata Mcgraw-Hill Edition, Eighth Edition, 2014.

Reference Books

1. Nicholas S. Williams, Professional Java for Web Applications, Wrox Press, 2014.
2. Ed Burns, Chris Schalk, JavaServer Faces 2.0, The Complete Reference, McGraw-Hill Publishers, 2010.
3. Christian Bauer, Gavin King, Gary Gregory, Java Persistence with Hibernate, 2015.
4. Craig Walls, Spring in Action Paperback, Manning Publications, 2014.

List of Challenging Experiments (Indicative) **SLO: 14,17**

1. Write a program to read the First name and Last name of a person, his weight and height using command line arguments. Calculate the BMI Index which is defined as the individual's body mass divided by the square of their height.

Category	BMI Range-Kg/m^2
Underweight	<18.5
Normal (healthy weight)	18.5 to 25
Overweight	25 to 30
Obese Class	Over 30

Display the name and display his category based on the BMI value thus calculated.
Low Level: Implement it for a single person
Medium Level: Read all the above mentioned details for ‘n’ persons, calculate their BMI index and hence display their category
High Level: Read ‘n’ as the first command line input and then read the above mentioned details for ‘n’ persons as the remaining command line inputs, calculate their BMI index and hence display their category.

2. If there are 4 batches in BTech(IT) learning ‘ITE2005’ course, read the count of the slow learners (who have scored <25) in each batch. Tutors should be assigned in the ratio of 1:4 (For every 4 slow learners, there should be one tutor). Determine the number of tutors for each batch. Create a 2-D jagged array with 4 rows to store the count of slow learners in the 4 batches. The number of columns in each row should be equal to the number of groups formed for that particular batch (Eg., If there are 23 slow learners in a batch, then there should be 6 tutors and in the jagged array, the corresponding row should store 4, 4, 4, 4,

	4,3). Use for-each loop to traverse the array and print the details. Also print the number of batches in which all tutors have exactly 4 students. Low Level: Implement using a 2D-regular array with sufficient and fixed number of columns to accommodate maximum number of groups and print using regular 'for' loop. Medium Level: Implement with 2D-ragged (jagged) array and enhanced-for loop. High Level: Implement it for 'n' such courses by employing a 3D-ragged array.
3.	Write a program to read a chemical equation and find out the count of the reactants and the products. Also display the count of the number of molecules of each reactant and product.Eg., For the equation, 2NaOH + H2SO4 -> Na2SO4+ 2H2O, the O/P should be as follows. Reactants are 2 moles of NaOH, 1 mole of H2SO4. Products are 1 mole of Na2SO4 and 2 moles of H2O. Low Level: Display the count of the number of molecules of each reactant and product. Medium Level: Display the individual elements in the reactants side. Eg., Na, O, H, S High Level: Count the number of atoms of each element both in the reactants side and the products side and hence check if the equation is balanced or not. Eg., In the products side, there are 2 atoms of Na, 1 atom of S, 6 atoms of O and 4 atoms of H. In the same way, count in the reactants side and check if both the counts are equal.
4.	(Bioinformatics: finding genes) Biologists use a sequence of letters A, C, T, and G to model a genome. A gene is a substring of a genome that starts after a triplet ATG and ends before a triplet TAG, TAA, or TGA. Furthermore, the length of a gene string is a multiple of 3 and the gene does not contain any of the triplets ATG, TAG, TAA, and TGA. Write a program that prompts the user to enter a genome and displays all genes in the genome. If no gene is found in the input sequence, displays no gene. Here are the sample runs: Enter a genome string: TTATGTTTTAAGGATGGGGCGTTAGTT O/P: TTT GGGCGT Low Level: Extract the genes present in between the starting triplet ATG and any of the 3 ending triplets Medium Level: Check if the length of the extracted gene is a multiple of 3 and that it doesn't contain any of the above mentioned triplets High Level: Define your own method to implement the functionality of indexOf() method and use it in your program instead of the built-in indexOf() method.
5.	Create a class Film with string objects which stores name, language and lead_actor and category (action/drama/fiction/comedy). Also include an integer data member that stores the duration of the film. Include parameterized constructor, default constructor and accessory functions to film class. Film objects can be initialized either using a constructor or accessor functions. Create a class FilmMain that includes a main function. In the main function create an array of Film objects. Define methods to display the following a. The English film(s) that has Arnold as its lead actor and that runs for shortest duration. b. The Tamil film(s) with Rajini as lead actor. c. All the comedy movies. Low Level: Implement the above said methods Medium Level: Overload the method that displays comedy movies to display only the

	comedy movies of a particular actor. High Level: For every Film, read also the year-of-release and thus create a 2D array of objects that stores the Film objects as follows: Films released before 1971 – stored in the 0^{th} row Films released within the period 1971-1980 - stored in the 1^{st} row, Films released within the period 1981-90 - stored in the 2^{nd} row, Films released within the period 1991-2000 - stored in the 3^{rd} row, Films released within the period 2001-2010 –stored in the 4^{th} row Films released after 2010 – stored in the 5^{th} row Implement the above said methods for the 2D array of objects.
6.	Define an abstract class 'Themepark' and inherit 2 classes 'Queensland' and 'Veegaland' from the abstract class. In both the theme parks, the entrance fee for adults is Rs. 500 and for children it is Rs. 300. If a family buys 'n' adult tickets and 'm' children tickets, define a method in the abstract class to calculate the total cost. Also, declare an abstract method playGame() which must be redefined in the subclasses. In Queensland, there are a total of 30 games. Hence create a Boolean array named 'Games' of size 30 which initially stores false values for all the elements. If the player enters any game code that has already been played, a warning message should be displayed and the user should be asked for another choice. In Veegaland, there are a total of 40 different games. Thus create an integer array with 40 elements. Here, the games can be replayed, until the user wants to quit. Finally display the total count of games that were repeated and count of the games which were not played at all. Low Level: Implement the game play for a single person Medium Level: Simulate the game play for 'n'+'m' family members at the theme park of their choice. High Level: Make necessary modifications in your code to have the top level structure 'ThemePark' defined as an interface.
7.	Read the Register Number and Mobile Number of a student. If the Register Number does not contain exactly 9 characters or if the Mobile Number does not contain exactly 10 characters, throw an IllegalArgumentException. If the Mobile Number contains any character other than a digit, raise a NumberFormatException. If the Register Number contains any character other than digits and alphabets, throw a NoSuchElementException. If they are valid, print the message 'valid' else 'invalid' Low Level: Implement the code to perform the above said operations Medium Level: Derive 3 different user-defined exceptions from Exception class and throw them instead of the given built-in exceptions. Provide appropriate code to handle them. High Level: Write the code for checking the mobile number in a different method and let the exception thrown in that method, be handled in the calling method. Design the try-catch blocks in such a way that the catch block for handling IllegalArgumentException is written only once.
8.	Within the package named 'primespackage', define a class Primes which includes a method checkForPrime() for checking if the given number is prime or not. Define another class named TwinPrimes outside of this package which will display all the pairs of prime numbers whose difference is 2. (Eg, within the range 1 to 10, all possible twin prime numbers are (3,5), (5,7)). The TwinPrimes class should make use of the checkForPrime() method in the Primes class.

	Low Level: Perform the above said operations. Medium Level: Create a sub-package 'mersennepackage' for the primespackage and include a class that checks if a given number is a Mersenne prime number or not. [A Mersenne prime is a prime number that is one less than a power of two. That is, it is a prime number of the form $M_n = 2^n - 1$, for some integer n. The exponents *n* which give Mersenne primes are 2, 3, 5, 7, 13, 17, 19, 31, ... and the resulting Mersenne primes are 3, 7, 31, 127, 8191, 131071, 524287, 2147483647, ...] High Level: Define a package 'specialnumbers' with a class named 'MersenneNumbers' that includes the method displayAllMersennePrime() to display all the Mersenne numbers within a given range.
9.	Define a class 'Donor' to store the below mentioned details of a blood donor. - Name, age, Address, Contactnumber, bloodgroup, date of last donation. Create 'n' objects of this class for all the regular donors at Vellore. Write these objects to a file. Read these objects from the file and display only those donors' details whose blood group is 'A+ve' and had not donated for the recent six months. Low Level: Implement the above problem. Medium Level: Include a method that modifies the date of last donation in the file for a Donor immediately after he/she donates blood. High Level: Assume there are 2 blood banks functioning at Vellore. Both of them maintain the details of their donors. Create a single repository of donors by storing the details of all the donors in the 2 files but without any redundancy. If name and contact number are the same for 2 donor objects in the two files, that donor detail should be included only once.
10.	Three students A, B and C of B.Tech-IT II year contest for the PR election. With the total strength of 240 students in II year, simulate the vote casting by generating 240 random numbers (1 for student A, 2 for B and 3 for C) and store them in an array. Create four threads to equally share the task of counting the number of votes cast for all the three candidates. Use synchronized method or synchronized block to update the three count variables. The main thread should receive the final vote count for all three contestants and hence decide the PR based on the values received. Low Level: Let the threads maintain their own set of 3 count variables and return them to the main() method where the individual counts sent by the threads are added and displayed. Medium Level: Implement the above mentioned scenario. High Level: Create an array of 'n' Thread objects and distribute the task of voting evenly among the threads. Also the threads have to fetch the vote details from a file.
11.	Draw a ball, filled with default color. Move the ball from top to bottom of the window continuously with its color changed for every one second. The new color of the ball for the next second should be obtained by adding 20 to the current value of Red component, for the second time by adding 20 to the blue component, and for the third time by adding 20 to the blue component, till all reach the final limit 225, after which the process should be repeated with the default color. Low Level: Implement only the ball movement without the color change. Medium Level: Implement the scenario described above. High Level: Draw another ball in a different frame and implement the horizontal movement of the ball.

12.	Develop a TCP based client-server application to notify the client about the integrity of data sent from its side. Assume the data to have been sent in a single array by the client. Check sum calculation: 1. Add the 16-bit values up. Each time a carry-out (17th bit) is produced, swing that bit around and add it back into the LSB (one's digit). 2. Once all the values are added in this manner, invert all the bits in the result. For example, separate the data into groups of 4 bits only for readability. 1000 0110 0101 1110 1010 1100 0110 0000 0111 0001 0010 1010 First, add the 16-bit values 2 at a time: 1000 0110 0101 1110 First 16-bit value + 1010 1100 0110 0000 Second 16-bit value --- 1 0011 0010 1011 1110 Produced a carry-out, which gets added + \----------> 1 back into LSB --- 0011 0010 1011 1111 + 0111 0001 0010 1010 Third 16-bit value --- 0 1010 0011 1110 1001 No carry to swing around (**) Then take the one's complement of the sum which is 0101 1100 0001 0110, the "one's complement". So the checksum stored in the header should be 0101 1100 0001 0110 Low Level: Perform TCP based client-server communication. Medium Level: Ensure that the server establishes connection with multiple clients and communicate with them simultaneously. High Level: Implement UDP/IP communication protocol to exchange the data and its checksum between the client and the server.
13.	Develop an RMI application to invoke a remote method that takes two numbers and returns 1 if one number is an exact multiple of the other and 0 otherwise. Eg., 5 and 25 -> true 26 and 13 -> true 4 and 18 -> false Low Level: Invoke the remote method from the client side to know if one number is an exact multiple of the other. Medium Level: Pass references of two Student objects from the client to the server and the server should return 0 – if the two objects are identical 1 – if the two references point to the same Student object

	-1 - if the two objects are different High Level: Invoke the remote method from multiple clients.
14.	a)Assume two cookies are created whenever a VIT student visits the VIT webpage-one for his/her name and the other for his campus. For subsequent visits, he/she should be greeted with the message similar to the one below “Hi Ajay from Chennai Campus!!”. Write a servlet program to do the needful. b)Build an application using JSF framework to implement a Celsius to Fahrenheit converter. Note: Fahrenheit=(Celsius*9/5)+32 Low Level: Use cookies to track the client information and display as mentioned. Medium Level: Also display the time when he visited the web site at last. High Level: Use HttpSession interface to achieve a similar effect
15.	Using Hibernate framework, simulate the course registration process for Advanced Java Programming. Let the registration number and name of the students who register for the course, be stored in a database. The tool should allow deletion of the registered course for a particular student, if he/she wishes. At any instant, the list of students who have registered for the course should be displayed, if requested for. Low Level: Implement the above said operations. Middle Level: Another table stores the registration number and name of the students who have registered for ‘Data Mining’. Display the registration number and names of those students who have registered for both Java and Data Mining. High Level: Modify your code appropriately to connect to a different database server to retrieve information from the two tables stored in that database.

Total Laboratory Hours	30 hours

Recommended by Board of Studies	DD-MM-YYYY		
Approved by Academic Council	No. xx	Date	DD-MM-YYYY