

# INDEX

Institute vision and mission	3
Department vision and mission	3
Program Educational Objectives (PEOs)	4
Program Outcomes (POs)	4
Program Specific Outcomes (PSOs)	6
Course Syllabus	7
Instructions to the students	11
Rubrics for Continuous Internal Evaluation (CIE)	12
Make-up laboratory sessions	16
Laboratory experiments Calendar	17
List of experiments to be performed	20
<b>Experiments</b>	
E1.1	Create a JFrame program to display “Good Morning” if current time is between “6 AM to 12 PM” and “Good Afternoon” if the current time is between “12 PM to 6PM”, and “Good Evening” if the current time is between “6PM to 12AM”
E1.2	Create a JFrame program to perform basic arithmetic calculations on given two numbers with the help of button events.
E2.1	Create a JFrame program from which you can open another frames with the help of button events
E2.2	Design different JFrame’s to demonstrate different layouts like Flow layout, Border layout, Grid layout & null layout
E2.3	Create a JFrame program to work with window events
E3.1	Create a JFrame to add a menu bar with which you can select different options from different menus and perform some action on selection of every menu item
E3.2	Create a JFrame program to open the textfile using JFileChooser and display the selected text file content on the JTextArea
E3.3	Design a registration form with the help of a JFrame and save the details in to the text file
E4.1	Create a JFrame program to insert, delete & update the records of a database table
E4.2	Create a JFrame program to select a database table using JComboBox component and display the content of the selected database table in JTablecomponent
E5.1	Write a java program to demonstrate generic class
E5.2	Write a java program to demonstrate methods and constructors in generics
E5.3	Write a java program to demonstrate multiple type parameters in generic classes
E5.4	Write a java program to demonstrate inheritances in generics

E6.1	Write a java program to perform following operations on ArrayList, LinkedList, HashSet and LinkedHashSet i.) Insertion ii.) Deletion iii.) Traversing using traditional-for, for-each, Iterator and ListIterator iv.) Display the elements in reverse order	
E6.2	Write a program that will have a Vector which is capable of storing Employee objects. Use an Iterator and enumeration to list all the elements of the Vector	
E7.1	Write a java program to perform different operations on inbuilt Stack class	
E7.2	Write a java program to perform different operations on inbuilt Queue class	
E7.3	Write a java program to perform insertion, deletion, traversing and searching operations on HashMap and TreeMap	
E8.1	Write a java program to store and retrieve user defined class objects from TreeSet	
E8.2	Write a java program to read a set of values and display the count of occurrences of each number using collection concept	
E9.1	Write a java program to display ArrayList values in sorted order	
E9.2	Write a java program to demonstrate Comparable interface for sorting user defined data type	
E9.3	Write a java program to demonstrate Comparator interface for sorting user defined data type	
E10.1	Write a java program to test simple arithmetic operations of Calculator class using JUnit concept	
E10.2	Write a java program to demonstrate different Assert methods and annotations	
E11.1	Write a java program to demonstrate lambda expression with no parameter	
E11.2	Write a java program to demonstrate lambda expression with single and multiple parameters	
E11.3	Write a java program to iterate the List and Map using lambda expressions	
E11.4	Create two threads using lambda expressions, where one thread displays even numbers for every half second and the other thread displays odd numbers for every second	
E12.1	Write a java program to demonstrate following methods using streams on a List a) filter b) sorted c) distinct d) limit e) count	
E12.2	Write a java program to read a string and collect upper case characters, lower case characters & digits into each individual ArrayList using streams and display them	

## INSTITUTE VISION & MISSION

<b>Vision of Institute:</b>
<ul style="list-style-type: none"><li>• To make our students technologically superior and ethically strong by providing quality education with the help of our dedicated faculty and staff and thus improve the quality of human life.</li></ul>
<b>Mission of the Institute:</b>
<ul style="list-style-type: none"><li>• To provide latest technical knowledge, analytical and practical skills, managerial competence and interactive abilities to students, so that their employability is enhanced.</li><li>• To provide a strong human resource base for catering to the changing needs of the Industry and Commerce.</li><li>• To inculcate a sense of brotherhood and national integrity.</li></ul>

## DEPARTMENT VISION & MISSION

<b>Vision of the department</b>
<ul style="list-style-type: none"><li>• Attaining centre of excellence status in various fields of Computer Science and Engineering (Networks) by offering worth full education, training and research to improve quality of software services for ever growing needs of the industry and society.</li></ul>
<b>Mission of the department</b>
<ul style="list-style-type: none"><li>• Practice qualitative approach and standards to provide students better understanding and profound knowledge in the fundamentals and concepts of computer science with its allied disciplines.</li><li>• Motivate students in continuous learning to enhance their technical, communicational, and managerial skills to make them competent and cope with the latest trends, technologies, and improvements in computer science to have a successful career with professional ethics.</li><li>• Involve students to analyze, design and experiment with contemporary research problems in computer science to impact socio-economic, political and environmental aspects of the globe.</li></ul>



## Program - B.Tech. Computer Science&Engineering(Networks)

### PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

**Within first few years after graduation, the Computer Science&Engineering(NETWORKS)**

PEO1	Graduates with fundamental knowledge should escalate the technical skills within and across disciplines of Computer Science Engineering for productive career by maintaining professional ethics
PEO2	Graduates should develop and exercise their capabilities to demonstrate their creativity in engineering practice and exhibit leadership with responsibility in teamwork
PEO3	Graduates should refine their knowledge and skills to attain professional competence through life-long learning such as higher education, research and professional activities

### PROGRAM OUTCOMES (POs)

Program Outcomes		Engineering graduates will be able to
PO1	<b>Engineering knowledge</b>	Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2	<b>Problem analysis</b>	Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	<b>Design/development of solutions</b>	Design solutions for complex engineering 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	<b>Conduct investigations of complex problems</b>	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	<b>Modern tool usage</b>	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO6	<b>The engineer and society</b>	Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	<b>Environment and sustainability</b>	Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8	<b>Ethics</b>	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice
PO9	<b>Individual and teamwork</b>	Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	<b>Communication</b>	Communicate effectively on complex engineering activities with the engineering 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	<b>Project management and finance</b>	Demonstrate knowledge and understanding of the engineering 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	<b>Life-long learning</b>	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

<b>PROGRAM SPECIFIC OUTCOMES (PSOs)</b>
---

<b>PSO</b>	<i>The Computer Science and Engineering(NETWORKS) graduates will be able to</i>
PSO1	Apply engineering knowledge gained through graduation to provide solutions to the real world problems during professional career
PSO2	Demonstrate management principles by applying those to own work as a member and leader in a team. Work independently and as part of team during project implementation
PSO3	Demonstrate professional competency through life-long learning by developing novel solutions for complex engineering problems by continues learning and stay up with latest technologies

## COURSE SYLLABUS

**U18IN507R22**

**ADVANCED JAVA PROGRAMMING LABORATORY**

**Class:**B.Tech.V-Semester

**Branch:**Computer Science & Engineering(IoT)

**Teaching Scheme :**

L	T	P	C
3	-	-	3

**Examination Scheme :**

Continuous Internal Evaluation	40 marks
End Semester Examination	60 marks

### Course Learning Objectives (LO):

This course will develop students' knowledge in/on...

**LO1:** developing GUI based programs using the concept of swings

**LO2:** the concepts of generics and collections

**LO3:** sorting user-defined data using Comparable and comparator interfaces and performing the unit testing with JUnit

**LO4:** lambda expressions and StreamAPI

### Experiment-I

1. Create a JFrame program to display "Good Morning" if current time is between "6 AM to 12 PM" and "Good Afternoon" if the current time is between "12 PM to 6PM", and "Good Evening" if the current time is between "6PM to 12AM".
2. Create a JFrame program to perform basic arithmetic calculations on given two numbers with the help of button events.

### Experiment-II

1. Create a JFrame program from which you can open another frames with the help of button events.
2. Design different JFrame's to demonstrate different layouts like Flow layout,



Border layout, Grid layout & null layout.

3. Create a JFrame program to work with window events.

#### **Experiment -III**

1. Create a JFrame to add a menu bar with which you can select different options from different menus and perform some action on selection of every menu item.
2. Create a JFrame program to open the text file using JFileChooser and display the selected text file content on the JTextArea.
3. Design a registration form with the help of a JFrame and save the details into the textfile.

#### **Experiment -IV**

1. Create a JFrame program to insert, delete & update the records of a database table.
2. Create a JFrame program to select a database table using JComboBox component and display the content of the selected database table in JTable component.

#### **Experiment -V**

1. Write a java program to demonstrate generic class.
2. Write a java program to demonstrate methods and constructors in generics.
3. Write a java program to demonstrate multiple type parameters in generic classes.
4. Write a java program to demonstrate inheritances in generics.

#### **Experiment -VI**

1. Write a java program to perform following operations on ArrayList, LinkedList, HashSet and LinkedHashSet
  - i. Insertion
  - ii. Deletion
  - iii. Traversing using traditional-for ,for-each Iterator and ListIterator.
  - iv. Display the elements in reverse order.
2. Write a program that will have a Vector which is capable of storing Employee objects. Use an Iterator and enumeration to list all the elements of the Vector.

#### **Experiment -VII**

1. Write a java program to perform different operations on inbuilt Stack class.
2. Write a java program to perform different operations on inbuilt Queue class.
3. Write a java program to perform insertion, deletion, traversing and searching

operations on HashMap and TreeMap.

#### **Experiment -VIII**

1. Write a java program to store and retrieve user defined class objects from TreeSet.
2. Write a java program to read a set of values and display the count of occurrences of each number using collection concept.

#### **Experiment-IX**

1. Write a java program to display ArrayList values in sorted order.
2. Write a java program to demonstrate Comparable interface for sorting user defined datatype.
3. Write a java program to demonstrate Comparator interface for sorting user defined datatype.

#### **Experiment-X**

1. Write a java program to test simple arithmetic operations of Calculator class using JUnit concept
2. Write a java program to demonstrate different Assert methods and annotations.

#### **Experiment-XI**

1. Write a java program to demonstrate lambda expression with no parameter.
2. Write a java program to demonstrate lambda expression with single and multiple parameters.
3. Write a java program to iterate the List and Map using lambda expressions.
4. Create two threads using lambda expressions, where one thread displays even numbers for every half second and the other thread displays odd numbers for every second.

#### **Experiment-XII**

1. Write a java program to demonstrate following methods using streams on a List.  
a) filter      b) sorted      c) distinct      d) limit      e)count
2. Write a java program to read a string and collect upper case characters, lower case characters & digits into each individual ArrayList using streams and display them.

**Laboratory Manual:**

1. Advanced Java Programming laboratory manual, prepared by faculty of Dept. of CSE.

**Reference Books:**

- [1] Herbert Schildt, JAVA The Complete Reference, 10th ed. New York: McGraw-Hill Education India Pvt.Ltd, 2017.
- [2] Sachin Malhotra, Saurabh Choudhary, Programming in JAVA, 2nd ed. New Delhi: Oxford University Press, 2013.
- [3] Uttam K. Roy, Advanced JAVA Programming, New Delhi: Oxford University Press, 2015.
- [4] Pual Deitel, Harvey Deitel, Java How to program, 10th ed. Chennai: Pearson Education, 2016.
- [5] Sujoy Acharya, Mastering Unit Testing Using Mockito and JUnit, Birmingham: Packt Publishing Limited, 2014.

**Course Learning Outcomes (COs):**

On completion of this course, students will be able to

**CO1:** design GUI programs by using the concept of swings.

**CO2:** apply the concept of generics & collections to work on dynamic data.

**CO3:** demonstrate correct usage of Comparable & Comparator interfaces and examine the test cases to perform unit testing using the concept of JUnit.

**CO4:** apply the lambda expressions instead of anonymous class and effectively process collection of objects using Stream API

## INSTRUCTIONS TO THE STUDENTS

1. This laboratory MANUAL & RECORD BOOK is essential for the student and must be brought to every laboratory session.
2. Students should read the manual carefully before coming to the laboratory class and be sure about what is supposed to be done.
3. In every laboratory session, the student -
  - a. should draw the circuit diagram/ experimental circuit and list the components required and draw the observation tables (in the RECORD WORK space provided at the end of each experiment)
  - b. should collect the apparatus
  - c. should get the connections checked by the laboratory instructor before switching ON the supply
  - d. is advised to not exceed the permissible values of current, voltage and/or speed of any machine, apparatus, wire, load, etc.,
  - e. is expected to perform the experiment as per the instructions and do the specimen calculations independently
  - f. should not hesitate to approach the faculty in case of difficulty
  - g. is expected to maintain utmost discipline in the laboratory
  - h. should return the apparatus in good condition after completion of each lab session.  
Any damage to the apparatus causes heavy penalty.
4. After completion of the experiment, the student should complete the Write-up and attend Viva-voce. The student
  - a. should perform calculations on the observations and draw necessary graphs
  - b. should interpret the results and draw reasonable conclusions which are to be mentioned in the 'Interpretation of results and discussions' section in the RECORD WORK.
  - c. should answer the questions at the end of RECORD WORK at the end of the experiment in the space provided.

- d. should appear for viva-voce after completing the write-up of RECORD WORK
- e. should get their record work evaluated by the teacher before leaving the laboratory session.

## RUBRICS FOR CONTINUOUS INTERNAL EVALUATION (CIE)

Continuous Internal Evaluation (CIE) for Practical (Laboratory) Course shall carry 40% weightage. CIE throughout the semester shall consist of the following for each experiment/lab.

CIE- Assessment for experiments done in every lab		Weightage
Experimentation/ Job work/ Coding	Knowledge (K)	10%
Participation as an individual while doing experiment (5%)	Participation (P)	10%
Participation as a member on team while doing experiment (5%)		
Write-up for Record Work	Write-Up (W)	10 %
Viva-voce	Viva-voce (V)	10%

Every laboratory session is evaluated for a total of 40 marks. The details have been listed below.

### A. Before conduction of experiment

#### 1. K - Knowledge

Student should come prepared to the lab session and shall answer the following, prior to the start of the experiment (knowledge):

- Aim – what he/she has to perform?
- Procedure – how to perform?
- Expected result – what he/she will obtain at the end of experiment conduction?

A total of 3-5 questions shall be asked on the above three topics and marks shall be awarded based on his/her performance, as below:

<i>% of questions answered satisfactorily</i>		<i>Marks awarded</i>
80-100	:	10
50-80	:	7-9
30-50	:	3-6
0-30	:	0-2

## B. During Experiment

### 2. P- Participation

Once the student is allowed to perform the experiment, marks will be awarded based on his/her participation **as an individual and as a member on team** while doing the experiment. Effective completion of experiment will be considered as group activity while the answering of questions related to procedure and results will be considered as individual activity. Marks shall be awarded as below:

<i>After the completion of experiment...</i>		<i>Marks awarded</i>
Completed the experiment effectively as team <u>without the assistance of faculty</u> and answered all the questions related to conduction of experiment	:	10
Completed the experiment effectively as team with <u>partial assistance of faculty</u> but able to answer the questions when asked individually	:	7-9
Completed the experiment only with <u>full assistance of faculty</u> and unable to answer the questions even after cooperation of faculty	:	3-6
<u>Unable to complete</u> the experiment even with assistance from the faculty	:	0-2

### C. Tasks/activities to be completed by student after completion of

**experiment:** After completion of experiment, student should complete the Write-up and Viva-voce. **The student has to come to viva-voce after completing the write-up of Record Work**

#### 3. *W-Write-up*

The student should complete the write-up, related to the experiment conducted, in the manual itself, in the designated space for Record Work. The write-up must be on the following:

- Experimental circuits/diagrams
- Observations & Data Collection experimental data
- Calculations
- Results
- Interpretation of Results & Discussion
- Suggestions on ALTERNATIVE APPROACH for experiment
- Suggestions on EXTENSION OF EXPERIMENT

Marks shall be awarded as below...

<i>After the completion of experiment...</i>	<i>Marks awarded</i>
Completed the write-up in the laboratory scheduled time <u>without any mistakes</u> (grammatical & technical) along with <u>good suggestions on alternative approach to experiment and extension of experiment</u>	10
Completed the write-up in the laboratory scheduled time <u>with few mistakes</u> but <u>good suggestions</u> provided	7-9
Completed the write-up in the laboratory scheduled time <u>with few mistakes</u> but <u>minor suggestions</u> provided	3-6
Completed write-up with mistakes and <u>no suggestions</u> provided	0-2

#### 4. V-Viva-voce

After completing the write-up, the student should attend viva-voce to answer the following:

- What are the observations made in the experiment?
- Interpretation of results
- What conclusions have been drawn from conducting the experiment?

Viva-voce should not be limited to only the sample questions listed in VIVA-VOCE questions at the end of experiment, but should go beyond to test the student's involvement in the experiment and also the technical competency.

- i. What did you learn from this experiment based on objectives?
- ii. How will you apply knowledge gained, by performing this experiment, in future?

Student during viva-voce should be asked to comment on suggestions, which he/she recorded in the write-up, on the following:

- iii. **Alternative approach:** How shall the procedure be modified to enhance his/her learning experience?
- iv. **Extension of experiment:** How can the experiment be extended to improve his/her learning?
- v. Any other ideas related to the experiment

Marks will be awarded based on student's performance, as below:

<i>After the completion of experiment...</i>		<b>Marks awarded</b>
Reasonable conclusions drawn with <u>good interpretation of results</u> and answered <u>80-100% of the viva-voce</u> questions perfectly	:	10
<u>Reasonable conclusions</u> drawn but answered <u>50-80% of the viva-voce</u> questions		7-9
<u>Poor conclusions and interpretation</u> of results with only <u>30-50% of viva-voce</u> questions answered	:	3-6
Conclusions <u>without interpretation</u> of results and answered less than <u>30% of viva-voce</u> questions posed	:	0-2



## MAKE-UP LAB SESSIONS

1. Missing lab sessions due to holidays or unforeseen circumstances / disturbances will cause a big loss to student learning
2. To compensate for this loss, lab course faculty has to plan and conduct additional lab sessions, called **Make-up Lab Sessions**, beyond working hours of the institute (or) on Saturdays / Sundays, by giving prior information to students
3. The lab course faculty has to ensure that **Make-up Lab Sessions** are arranged in the following cases
  - (i) to compensate for the lab sessions to be lost due to holidays
  - (ii) to compensate for the lab sessions to be lost due to unforeseen circumstances
4. The dates for Make-up lab sessions for case (i) i.e., **for the sessions which are expected to be lost due to holidays**, are to be announced very much at the beginning of semester itself and printed, in the **Lab Experiments Calendar**
5. The dates for Make-up lab sessions, for case (ii) i.e., **for the sessions which are expected to be lost due to unforeseen circumstances**, are to be announced, conducted and recorded as and when the lab sessions get disturbed

### Important Note:

1. **Doing all stipulated experiments is mandatory for the students to appear for Laboratory End Semester Examination (ESE).**
2. **It is student's responsibility to complete all experiments**
3. If any student is absent for any laboratory session due to valid/genuine reasons, he/she must complete the experiment within a week time by seeking permission from the lab course faculty.
4. Upon completion of the experiments of lab sessions which were missed due to valid/genuine reasons, student will be evaluated for only **50 % of the maximum marks** of the experiment and the corresponding **attendance will not be counted**.
5. Student is allowed to perform only **one experiment during any laboratory session** beyond working hours.

## LABORATORY EXPERIMENTS - CALENDER

Week	Date	Title of the experiment
1.		E1.1) Create a JFrame program to display “Good Morning” if current time is between “6 AM to 12 PM” and “Good Afternoon” if the current time is between “12 PM to 6PM”, and “Good Evening” if the current time is between “6PM to 12AM”
		E1.2) Create a JFrame program to perform basic arithmetic calculations on given two numbers with the help of button events.
2.		E2.1) Create a JFrame program from which you can open another frames with the help of button events.
		E2.2) Design different JFrame’s to demonstrate different layouts like Flow layout, Border layout, Grid layout & null layout.
3.		E2.3) Create a JFrame program to work with window events.
		E3.1) Create a JFrame to add a menu bar with which you can select different options from different menus and perform some action on selection of every menu item.
4.		E3.2) Create a JFrame program to open the textfile using JFileChooser and display the selected text file content on the JTextArea.
		E3.3) Design a registration form with the help of a JFrame and save the details in to the text file.
5.		E4.1) Create a JFrame program to insert, delete & update the records of a database table.
		E4.2) Create a JFrame program to select a database table using JComboBox component and display the content of the selected database table in JTablecomponent.
6.		E5.1) Write a java program to demonstrate generic class.
		E5.2) Write a java program to demonstrate methods and constructors in generics.
7.		E5.3) Write a java program to demonstrate multiple type parameters in generic classes.
		E5.4) Write a java program to demonstrate inheritances in generics.
8.		E6.1) Write a java program to perform following operations on ArrayList, LinkedList, HashSet and LinkedHashSet i.) Insertion ii.) Deletion iii.) Traversing using traditional-for, for-each, Iterator and ListIterator iv.) Display the elements in reverse order.

		E6.2) Write a program that will have a Vector which is capable of storing Employee objects. Use an Iterator and enumeration to list all the elements of the Vector
7.		E7.1) Write a java program to perform different operations on inbuilt Stack class.
		E7.2) Write a java program to perform different operations on inbuilt Queue class.
		E7.3) Write a java program to perform insertion, deletion, traversing and searching operations on HashMap and TreeMap.
8.		E8.1) Write a java program to store and retrieve user defined class objects from TreeSet.
		E8.2) Write a java program to read a set of values and display the count of occurrences of each number using collection concept.
9.		E9.1) Write a java program to display ArrayList values in sorted order.
		E9.2) Write a java program to demonstrate Comparable interface for sorting user defined data type.
		E9.3) Write a java program to demonstrate Comparator interface for sorting user defined data type.
10.		E10.1) Write a java program to test simple arithmetic operations of Calculator class using JUnit concept.
		E10.2) Write a java program to demonstrate different Assert methods and annotations.
11.		E11.1) Write a java program to demonstrate lambda expression with no parameter.
		E11.2) Write a java program to demonstrate lambda expression with single and multiple parameters.
		E11.3) Write a java program to iterate the List and Map using lambda expressions.
		E11.4) Create two threads using lambda expressions, where one thread displays even numbers for every half second and the other thread displays odd numbers for every second.
12.		E12.1) Write a java program to demonstrate following methods using streams on a List a) filter b) sorted c) distinct d) limit e) count.
		E12.2) Write a java program to read a string and collect upper case characters, lower case characters & digits into each individual ArrayList using streams and display them.
13.		END SEMESTER EXAMINATION
14.		END SEMESTER EXAMINATION

## LAB EXPERIMENTS CALENDAR - MAKE-UP SESSIONS

S. No.	Make-up Lab on (Date)	Time	Title of the experiment
Make-up lab sessions - for sessions lost due to holidays			
1.			
2.			
3.			
4.			
Make-up lab sessions - for sessions lost due to unforeseen circumstances			
1.			
2.			
3.			
4.			

## LIST OF EXPERIMENTS

Expt No.	Title of experiment	Date of conduction	Marks awarded (40)	Signature of faculty
E1.	E1.1) Create a JFrame program to display “Good Morning” if current time is between “6 AM to 12 PM” and “Good Afternoon” if the current time is between “12 PM to 6PM”, and “Good Evening” if the current time is between “6PM to 12AM” E1.2) Create a JFrame program to perform basic arithmetic calculations on given two numbers with the help of button events.			
E2.	E2.1) Create a JFrame program from which you can open another frames with the help of button events. E2.2) Design different JFrame’s to demonstrate different layouts like Flow layout, Border layout, Grid layout & null layout. E2.3) Create a JFrame program to work with window events.			
E3.	E3.1) Create a JFrame to add a menu bar with which you can select different options from different menus and perform some action on selection of every menu item. E3.2) Create a JFrame program to open the textfile using JFileChooser and display the selected text file content on the JTextArea. E3.3) Design a registration form with the help of a JFrame and save the details in to the text file.			
E4.	E4.1) Create a JFrame program to insert, delete & update the records of a database table E4.2) Create a JFrame program to select a database table using JComboBox component and display the content of the selected database table in JTablecomponent			
E5.	E5.1) Write a java program to demonstrate generic class. E5.2) Write a java program to demonstrate methods and constructors in generics. E5.3) Write a java program to demonstrate multiple type parameters in generic classes.			

	E5.4) Write a java program to demonstrate inheritances in generics.			
E6.	E6.1) Write a java program to perform following operations on ArrayList, LinkedList, HashSet and LinkedHashSet i.) Insertion ii.) Deletion iii.) Traversing using traditional-for, for-each, Iterator and ListIterator iv.) Display the elements in reverse order. E6.2) Write a program that will have a Vector which is capable of storing Employee objects. Use an Iterator and enumeration to list all the elements of the Vector.			
E7.	E7.1) Write a java program to perform different operations on inbuilt Stack class. E7.2) Write a java program to perform different operations on inbuilt Queue class. E7.3) Write a java program to perform insertion, deletion, traversing and searching operations on HashMap and TreeMap.			
E8.	E8.1) Write a java program to store and retrieve user defined class objects from TreeSet. E8.2) Write a java program to read a set of values and display the count of occurrences of each number using collection concept.			
E9.	E9.1) Write a java program to display ArrayList values in sorted order. E9.2) Write a java program to demonstrate Comparable interface for sorting user defined data type. E9.3) Write a java program to demonstrate Comparator interface for sorting user defined data type.			
E10.	E10.1) Write a java program to test simple arithmetic operations of Calculator class using JUnit concept. E10.2) Write a java program to demonstrate different Assert methods and annotations.			
E11.	E11.1) Write a java program to demonstrate lambda expression with no parameter. E11.2) Write a java program to demonstrate lambda expression with single and multiple parameters. E11.3) Write a java program to iterate the List and Map using lambda expressions.			

	E11.4) Create two threads using lambda expressions, where one thread displays even numbers for every half second and the other thread displays odd numbers for every second.			
E12.	E12.1) Write a java program to demonstrate following methods using streams on a List a) filter b) sorted c) distinct d) limit e) count. E12.2) Write a java program to read a string and collect upper case characters, lower case characters & digits into each individual ArrayList using streams and display them.			

## **WEEK-1**

### **INTRODUCTION**

Graphical User Interface (GUI) and Command Line Interface (CLI) are the two means of interaction between a user and an electronic device.

#### **Command Line Interface**

- A CLI uses a keyboard for text input of commands that are required to navigate the machine.
- CLI requires commands to be used. Users need to know these commands to use a CLI.
- This is generally why CLI is less used than a GUI. However, CLI requires a smaller amount of RAM and CPU processing time.
- CLIs use a command-line interpreter to support scripting or task automation.

#### **Graphical User Interface**

- More than likely, a user's first interaction with a computer is through a GUI.
- It is more common and user friendly. A GUI uses a mouse and a keyboard for navigation throughout the machine. Visual components represent potential user actions.
- Although easier to use, a GUI is considered slower due to more visual output, which requires more memory than a text-based CLI.
- GUIs are excellent for general computer use and for users that want to use a machine without prior knowledge of its interface.
- More users are comfortable with a GUI because it is practical and usable.

#### **Console Based Application**

The console application is a computer program that can be used only with a text-only computer interface, such as a text terminal, the command-line interface of some operating systems (Unix, DOS, etc.) or the text-based interface.

#### **GUI based Application**

A graphical user(GUI) interface is an application that has buttons, windows, and lots of other widgets that the user can use to interact with your application.



## **Graphics programming**

There are current three sets of Java APIs for graphics programming:

- AWT (Abstract Windowing Toolkit)
- Swing
- JavaFX.

### **AWT in Java**

AWT stands for Abstract window toolkit is an Application programming interface (API) for creating Graphical User Interface (GUI) in Java. It allows Java programmers to develop window-based applications.

AWT components use the resources of the operating system, i.e., they are platform-dependent, which means, component's view can be changed according to the view of the operating system.

The classes for AWT are provided by the java.awt package for various AWT components.

Abstract Winowing Toolkit(AWT) is huge! It consists of 12 packages of 370 classes .

Among which only 2 packages - java.awt and java.awt.event are commonly-used.

The java.awt package contains the core AWT graphics classes:

- GUI Component classes, such as Button, TextField, and Label.
- GUI Container classes, such as Frame and Panel.
- Layout managers, such as FlowLayout, BorderLayout and GridLayout.
- Custom graphics classes, such as Graphics, Color and Font.

### **Swing in Java**

Swing is the graphical user interface toolkit that is used for developing windows based java applications or programs. It is the successor of AWT.

Swing is the collection of user interface components for Java programs. It is part of Java Foundation classes that are referred to as JFC.

The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.

The javax.swing package provides classes for java swing API.

## Features of Swing

The features of the Swing are as follows:

1. **Platform Independent:** It is platform-independent; the swing components used to build the program are not platform-specific. It can be used on any platform and anywhere.
2. **Lightweight:** Swing components are lightweight, which helps in creating the UI lighter. The swings component allows it to plug into the operating system user interface framework, including the mappings for screens or devices and other user interactions like keypress and mouse movements.
3. **Pluggable Look and Feel:** This feature enables the user to switch the look and feel of Swing components without restarting an application. The Swing library supports components look and feel that remains the same across all platforms wherever the program runs.
4. **Manageable:** It is easy to manage and configure. Its mechanism and composition pattern also allows changing the settings at run time. The uniform changes can be provided to the user interface without any changes to the application code.
5. **MVC: Model-View-Controller**, the **model** corresponds to the state information associated with the Component. The **view** determines how the component is displayed on the screen, including any aspects of the view that are affected by the current state of the model. The **controller** determines how the component reacts to the user

There are two groups of GUI elements:

- **Container:** Containers, such as Frame and Panel, are used to hold components in a specific layout (such as FlowLayout or GridLayout). A container can also hold sub-containers.
- **Component (Widget, Control):** Components are elementary GUI entities, such as Button, Label, and TextField. They are also called widgets, controls in other graphics systems.

## Container Class

Any class which has other components in it is called as a container class. For building GUI applications at least one container class is necessary.

A Swing application requires a *top-level container*. There are three top-level containers in Swing:

1. **JFrame:** used for the application's main window (with an icon, a title, minimize/maximize/close buttons, an optional menu-bar, and a content-pane), as illustrated.
2. **JDialog:** used for secondary pop-up window (with a title, a close button, and a content-pane).
3. **JApplet:** used for the applet's display-area (content-pane) inside a browser's window.

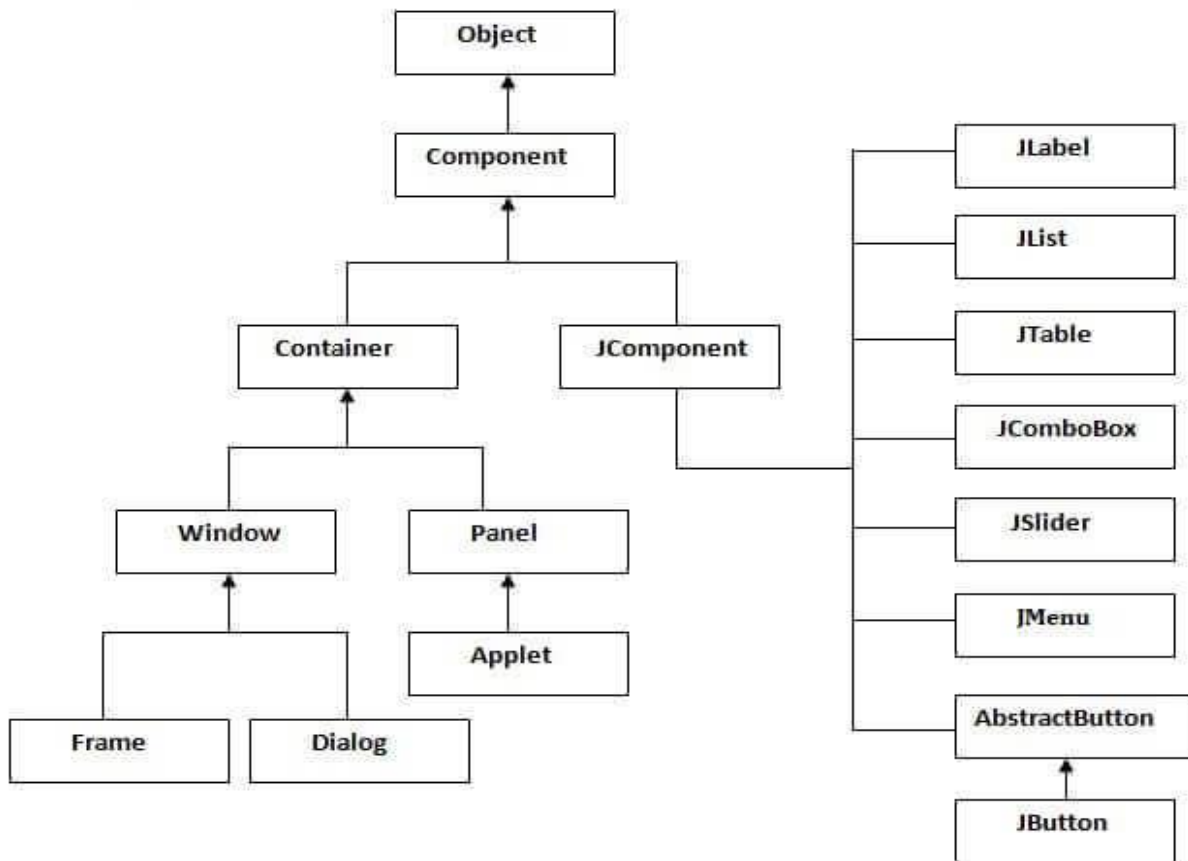
There are *secondary containers* such as JPanel and scrollpane which can be used to group and layout relevant components.

### Component classes

Swing components are the basic building blocks of an application. Swing components are the interactive elements in a Java application.

Swing component classes (in package javax.swing) begin with a prefix "J", e.g., JButton, JTextField, JLabel, JPanel, JFrame, or JApplet.

The hierarchy of java swing API is



## Event Handling in Java

An **event** can be defined as changing the state of an object or behavior by performing actions. Actions can be a button click, cursor movement, keypress through keyboard or page scrolling, etc.

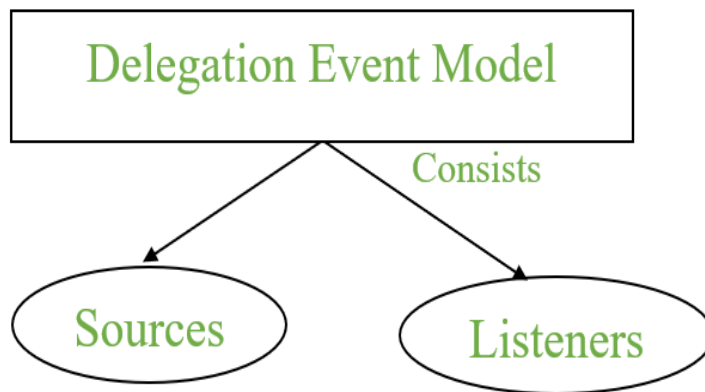
The **java.awt.event** package can be used to provide various event classes.

### Event Handling

It is a mechanism to **control the events** and to **decide what should happen after an event** occur. To handle the events, Java follows the *Delegation Event model*.

### Delegation Event model

- It has Sources and Listeners.



- **Source:** Events are generated from the source. There are various sources like buttons, checkboxes, list, menu-item, choice, scrollbar, text components, windows, etc., to generate events.
- **Listeners:** Listeners are used for handling the events generated from the source. Each of these listeners represents interfaces that are responsible for handling events.

To perform Event Handling, we need to register the source with the listener.

### Registering the Source With Listener

Different Classes provide different registration methods.

#### Syntax:

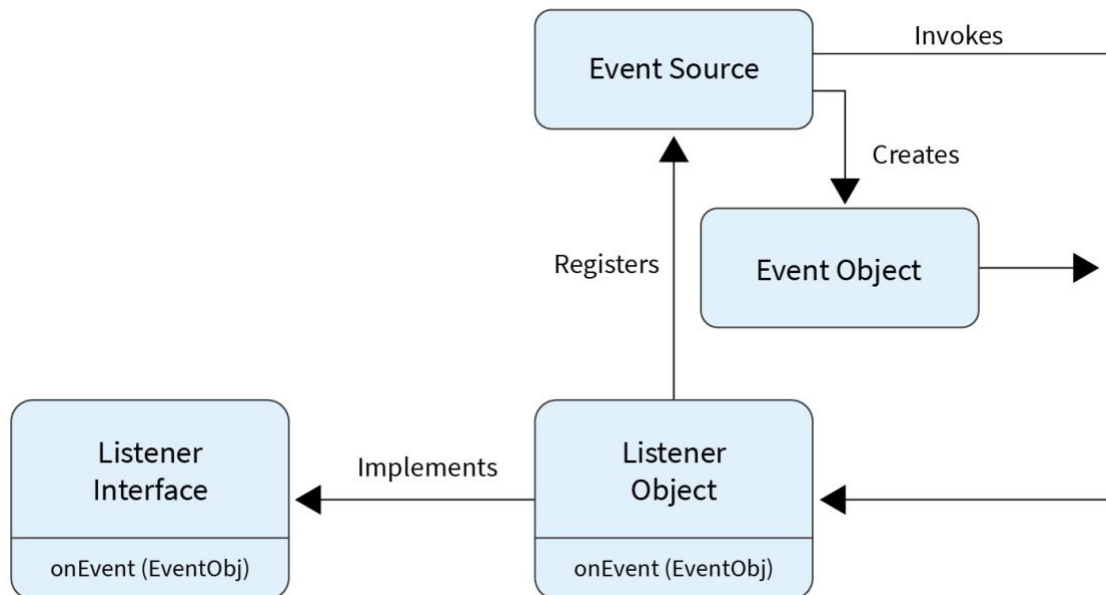
`addTypeListener()`

where Type represents the type of event.

**Example 1:** For **KeyEvent** we use `addKeyListener()` to register.

**Example 2:** For **ActionEvent** we use `addActionListener()` to register.

The image below shows the flow chart of the event delegation model.



A source generates an event and sends it to one or more listeners. The listener sits and waits for an event to occur. When it gets an event, it is processed by the listener and returned. The user interface elements can delegate the processing of an event to a different function.

### **Event Classes in Java**

<b>Event Class</b>	<b>Listener Interface</b>	<b>Description</b>
ActionEvent	ActionListener	An event that indicates that a component-defined action occurred like a button click or selecting an item from the menu-item list.
AdjustmentEvent	AdjustmentListener	The adjustment event is emitted by an Adjustable object like Scrollbar.
ComponentEvent	ComponentListener	An event that indicates that a component moved, the size changed or changed its visibility.
ContainerEvent	ContainerListener	When a component is added to a container (or) removed from it, then this event is generated by a container object.
FocusEvent	FocusListener	These are focus-related events, which include focus, focusin, focusout, and blur.
ItemEvent	ItemListener	An event that indicates whether an item was selected or not.
KeyEvent	KeyListener	An event that occurs due to a sequence of keypresses on the keyboard.
MouseEvent	MouseListener & MouseMotionListener	The events that occur due to the user interaction with the mouse (Pointing Device).

Event Class	Listener Interface	Description
MouseEvent	MouseListener	An event that specifies that the mouse wheel was rotated in a component.
TextEvent	TextListener	An event that occurs when an object's text changes.
WindowEvent	WindowListener	An event which indicates whether a window has changed its status or not.

*As Interfaces contains abstract methods which need to implemented by the registered class to handle events.*

Different interfaces consists of different methods which are specified below.

Listener Interface	Methods
ActionListener	<ul style="list-style-type: none"> <li>actionPerformed()</li> </ul>
AdjustmentListener	<ul style="list-style-type: none"> <li>adjustmentValueChanged()</li> </ul>
ComponentListener	<ul style="list-style-type: none"> <li>componentResized()</li> <li>componentShown()</li> <li>componentMoved()</li> <li>componentHidden()</li> </ul>
ContainerListener	<ul style="list-style-type: none"> <li>componentAdded()</li> <li>componentRemoved()</li> </ul>
FocusListener	<ul style="list-style-type: none"> <li>focusGained()</li> <li>focusLost()</li> </ul>
ItemListener	<ul style="list-style-type: none"> <li>itemStateChanged()</li> </ul>

Listener Interface	Methods
KeyListener	<ul style="list-style-type: none"><li>• keyTyped()</li><li>• keyPressed()</li><li>• keyReleased()</li></ul>
MouseListener	<ul style="list-style-type: none"><li>• mousePressed()</li><li>• mouseClicked()</li><li>• mouseEntered()</li><li>• mouseExited()</li><li>• mouseReleased()</li></ul>
MouseMotionListener	<ul style="list-style-type: none"><li>• mouseMoved()</li><li>• mouseDragged()</li></ul>
MouseWheelListener	<ul style="list-style-type: none"><li>• mouseWheelMoved()</li></ul>
TextListener	<ul style="list-style-type: none"><li>• textChanged()</li></ul>
WindowListener	<ul style="list-style-type: none"><li>• windowActivated()</li><li>• windowDeactivated()</li><li>• windowOpened()</li><li>• windowClosed()</li><li>• windowClosing()</li><li>• windowIconified()</li><li>• windowDeiconified()</li></ul>

### Flow of Event Handling

1. User Interaction with a component is required to generate an event.
2. The object of the respective event class is created automatically after event generation, and it holds all information of the event source.
3. The newly created object is passed to the methods of the registered listener.
4. The method executes and returns the result.



## **JFrame**

JFrame is a Swing's top-level container that renders a window on screen. A frame is a base window on which other components rely, such as menu bar, panels, labels, text fields, buttons, etc. Almost every Swing application starts with JFrame window.

### **Creating A JFrame in Java**

To create a JFrame, we have to create the instance of JFrame class. We have many constructors to create a JFrame.

- **JFrame():** It creates a frame but is invisible
- **JFrame(GraphicsConfiguration gc):** It creates a frame having a blank title and graphics configuration of the screen of device.
- **JFrame(String title):** It creates a JFrame having a title.
- **JFrame(String title, GraphicsConfiguration gc):** It creates a JFrame with specific Graphics configuration as well as a specified title.

create a JFrame just like any other Java objects:

#### **Example:**

```
JFrame frame = new JFrame("Demo program for JFrame");
```

Or create a frame window by creating a class that extends javax.swing.JFrame class:

#### **Example:**

```
public class SwingJFrameDemo extends JFrame {
```

### **Setting layout manager**

The default layout of the frame is BorderLayout, we can set another layout like this:

1. `frame.setLayout(new GridBagLayout());`
2. `frame.setLayout(new GridLayout());`
3. `frame.setLayout(new CardLayout());`
4. `frame.setLayout(new FlowLayout());`

Or using absolute layout:

1. `frame.setLayout(null);`
2. `frame.getContentPane().setLayout(layout);`

### **Specifying window closing behavior for JFrame**

We can specify which action will be executed when the user clicks on the frame's close button:

- Do nothing (default):

```
frame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
```

- Hide the frame:

```
frame.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
```

- In this case, the frame becomes invisible. To show it again, call:

```
frame.setVisible(true);
```

- Dispose the frame:

```
frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
```

In this case, the frame is removed and any resources used by the frame are freed. If the frame is the last displayable window on screen, the JVM may terminate.

- Exit the program (JVM terminates):

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

### **Showing JFrame on screen**

- Make sure we set size for the frame before making it visible:

```
frame.setSize(300, 200);
```

```
frame.setVisible(true);
```

- Center the frame on screen:

```
frame.setLocationRelativeTo(null);
```

- Maximize the frame window:

```
frame.setExtendedState(JFrame.MAXIMIZED_BOTH);
```

- Set window location on screen:

```
frame.setLocation(100, 100);
```

- Set size and location together: We can use the `setBounds(x, y, width, height)` method to set size and location of the frame in one call:

```
frame.setBounds(100, 100, 300, 400);
```

- Pack the frame: If we don't want to specify or (don't know) the exact size of the frame, we can use the method `pack()` to let the frame resizes itself in a manner which ensures all its subcomponents have their preferred sizes:

```
frame.pack();
```

- Set the frame window always on top other windows:

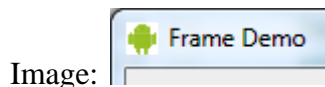
```
frame.setAlwaysOnTop(true);
```

### Customizing JFrame's appearance

- Set icon image for the frame: **The icon image is in the file system:**

```
Image icon = new ImageIcon("path of the image on the system").getImage();
```

```
frame.setIconImage(icon);
```



**NOTE:** if the image could not be found, the frame still has the default icon - which is the coffee cup.

- Disable resizing: By default, the users can resize the frame. Use the following code if we want to prevent the frame from being resized:

```
frame.setResizable(false);
```

- Set background color:

```
frame.getContentPane().setBackground(Color.GREEN);
```

- Undecorate the frame:

```
frame.setUndecorated(true);
```

If the frame is undecorated, its border, title bar and window buttons are all removed, only keep its content pane visible.

### **ContentPane of a Swing Container**

In Java Swing, the top-level containers manage its components via invisible pane. It is called **Content Pane**. For example, let us take the JFrame. Here, the menus, frame titles do not take part on the content pane. But the components like labels, text boxes, buttons etc. get added to it.

The getContentPane() method retrieves the content pane layer so that you can add an object to it. The content pane is an object created by the Java run time environment.

### **Adding child components to JFrame**

- We can use the method **add(Component)** to add a component to the frame's content pane. For example, adding a text field :

```
JTextField textFieldUserName = new JTextField(50);  
  
frame.add(textFieldUserName);
```

The call add(Component) is equivalent to this call:

```
frame.getContentPane().add(Component);
```

- Add a component with a layout constraint:
  - with BorderLayout:

```
add(textFieldUserName, BorderLayout.CENTER);
```

- with GridBagLayout:

```
GridBagConstraints constraint = new GridBagConstraints();  
  
constraint.gridx = 1;  
  
constraint.gridy = 0;  
  
JTextField textFieldUserName = new JTextField(20);
```

```
add(textFieldUserName, constraint);
```

## **JLabel**

JLabel is a class of java Swing . JLabel is used to display a short string or an image icon. JLabel can display text, image or both .

JLabel is only a display of text or image and it cannot get focus .

JLabel is inactive to input events such a mouse focus or keyboard focus.

JLabel is generally used along with those components that do not have their own ability to explain or demonstrate their purpose. The JLabel object created will provide our user, the text instructions or information on our GUI.

### **Constructors of JLabel**

1. **JLabel()** : creates a blank label with no text or image in it.
2. **JLabel(String s)** : creates a new label with the string specified.
3. **JLabel(Icon i)** : creates a new label with a image on it.
4. **JLabel(String s, Icon i, int align)** : creates a new label with a string, an image and a specified horizontal alignment
5. **JLabel(String text, int horizontalAlignment)**:creates a specified text along with horizontal alignment.
6. **JLabel (String text, Icon icon, int horizontalAlignment)**: creates a specified image or icon, text as well as its alignment as ‘horizontal’.

**The icon and text associated with the label can be obtained by the following methods:**

- Icon getIcon( )
- String getText( )

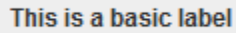
**The icon and text associated with a label can be set by these methods:**

- void setIcon(Icon icon)
- void setText(String str)

## Creating a JLabel object

- Create a basic label with some text:

```
JLabel label = new JLabel("This is a basic label");
```



- Create a label with empty text and set the text later:

```
JLabel label = new JLabel();
```

```
label.setText("This is a basic label");
```

- Create a label with only an icon (the icon file is in the file system and relative to the program):

```
ImageIcon img=new ImageIcon("path of the image");
```

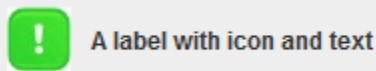
```
JLabel label = new JLabel(img);
```



This is the common way to display an image/icon in Swing.

- Create a label with both text and icon and horizontal alignment is center:

```
JLabel label = new JLabel("A label with icon and text", new ImageIcon  
("images/attention.jpg"), SwingConstants.CENTER);
```



- In this case, we can set the gap between the icon and the text as follows:

```
label.setIconTextGap(10);
```

That will set 10 pixels gap between the icon and the text.

## Common Methods Used in JLabel

1. **getIcon()** : returns the image that the label displays

2. **setIcon(Icon i)** : sets the icon that the label will display to image i
3. **getText()** : returns the text that the label will display
4. **setText(String s)** : sets the text that the label will display to string s

### **Adding the label to a container**

- A JLabel is usually added to a JPanel , a JFrame, a JDialog or a JApplet:

```
frame.add(label);
```

```
dialog.add(label);
```

```
panel.add(label);
```

```
applet.getContentPane().add(label);
```

- Adding a JLabel to a container with a specific layout manager:

```
frame.add(label, BorderLayout.CENTER);
```

```
panel.add(label, gridbagConstraints);
```

### **Customizing JLabel's appearance**

- Change font style, background color and foreground color of the label:

```
label.setFont(new Font("Arial", Font.ITALIC, 16));
```

```
label.setOpaque(true);
```

```
label.setBackground(Color.WHITE);
```

```
label.setForeground(Color.BLUE);
```



When the frame layout is null we have to set the position of the label

```
label.setBounds(200,200,30,10);
```

where 200,200 are x,y coordinates and 30 is the width,10 is the height of the label.

By default, the label's background is transparent, so if you want to set background, you have to set the label's opaque property to true.

- Instead of using the methods above, we can use HTML code to customize the label's appearance. For example:

```
label.setText("<html><font color=red size=4><b>WARNING!</b></html>");
```



### **Labeling a component**

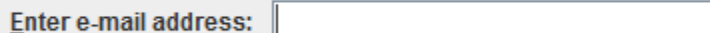
A JLabel is usually used for labeling a component such as a JTextField. If we want to allow the users accessing a text field using shortcut key (mnemonic), use the following code:

```
JTextField textEmail = new JTextField(20);
```

```
JLabel label = new JLabel("Enter e-mail address:");
```

```
label.setLabelFor(textEmail);
```

```
label.setDisplayedMnemonic('E');
```



The 'E' letter in the label is underlined, so the users can type **Alt + E** to get focus on the text field.



## **JButton**

JButton is a fundamental Swing component that renders a button on screen and responds to user's clicking event for performing a specific task.

The button component can contain text, image or both.

### **Three of its constructors are :**

JButton(Icon icon)

JButton(String str)

JButton(String str, Icon icon)

### **Different Methods of JButton Class**

void setText(String str)

String getText( )

void setIcon(Icon icon)

Icon getIcon()

setToolTipText(String str);

void setActionCommand( String actionName)

public void addActionListener(ActionListener listner)

### **Listener(s) for JButton**

ActionListener

**void actionPerformed(ActionEvent e)**

### **Event Name**

ActionEvent

String getActionCommand( )

public Object getSource()

### **Creating a JButton object**

1. Create a default button with a caption:

```
JButton button = new JButton("Edit");
```



2. Create a button with only an icon in the file system:

```
JButton button = new JButton(new ImageIcon("images/start.gif"));
```

Here the icon file start.gif is placed under images directory which is relative to the program.



3. Create a button with only icon inside a jar file or in classpath:

```
String iconPath = "/net/codejava/swing/jbutton/stop.jpg";
```

```
Icon icon = new ImageIcon(getClass().getResource(iconPath));
```

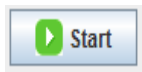
```
JButton button = new JButton(icon);
```

Here the icon file stop.jpg is placed under a specific package in the classpath.



4. Create a button with a caption and an icon:

```
JButton button = new JButton("Start", new ImageIcon("images/start.gif"));
```



### **Adding the button to a container**

1. A JButton is usually added to a JPanel, a JFrame, a JDialog or a JApplet:

```
frame.add(button);
```

```
dialog.add(button);
```

```
panel.add(button);
```

```
applet.getContentPane().add(button);
```

2. Adding a JButton to a container with a specific layout manager:

```
frame.add(button, BorderLayout.CENTER);
```

```
panel.add(button, gridbagConstraints);
```

### **Adding event listener for JButton**

- Adding an event listener using anonymous class (shortcut way):

```
button.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent evt) {  
        // do everything here...  
    }  
});
```

- Adding an event listener using anonymous class and an event handler method (recommended):

```
button.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent evt) {  
        // delegate to event handler method  
        buttonActionPerformed(evt);  
    }  
});
```

- The event handler method is declared as follows:

```
private void buttonActionPerformed(ActionEvent evt) {  
    // event handling code }
```

- Let the container listens to JButton's click event (not recommended):

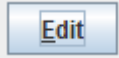
```
public class App extends JFrame implements ActionListener {  
  
    public App() {  
        button.addActionListener(this);  
    }  
    public void actionPerformed(ActionEvent evt) {
```

```
        // event handling code }  
    }
```

### Setting mnemonic for JButton

- Set mnemonic key **Alt + E** for the button whose caption is “Edit”:

```
button.setMnemonic(KeyEvent.VK_E);
```

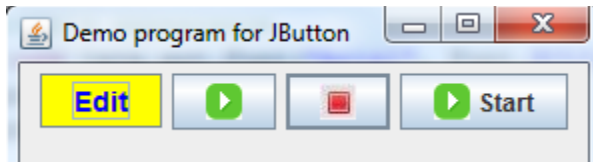
The letter **E** is underlined:  so user can invoke that button's action by pressing **Alt + E** instead of clicking mouse.

### Setting a JButton as the default button

A window can have a default button whose action will be invoked when the user hits **Enter** key. Here is the code to set the button as default button in the frame window:

```
getRootPane().setDefaultButton(button);
```

The default button is bold in the window like the 3<sup>rd</sup> button:



### Customizing JButton's appearance

- Change font style, background color and foreground color of the button:

```
button.setFont(new java.awt.Font("Arial", Font.BOLD, 14));
```

```
button.setBackground(Color.YELLOW);
```

```
button.setForeground(Color.BLUE);
```



- Change font style using HTML code:

```
button.setText("<html><color=blue><b>Edit</b></font></html>");
```

### E1.1 Java program to display “Good Morning, Good Afternoon, Good Evening” based on the time with the help of button event using JFrame.

**OBJECTIVE:**

*This lab will develop students’ knowledge in /on*

- Using JFrames, JButton , event handling using ActionListener interface

*Upon completion of this lab, students will be able to*

- develop programs on JFrames,JButton and handling events using ActionListener interface

Step 1: import the packages awt, awt.event, javax.swing, util. Calendar, util.Formatter

Step 2: Define a class FirstJFrame which implements the interface ActionListener.

Step 3: Define a constructor FirstJFrame()

Step 4: Create a JFrame window by creating an object for it, specify the Layout of JFrame as null, set the size and visibility of the JFrame, set close operation to EXIT\_ON\_CLOSE.

Step 5: create objects for JLabel and JButton labeled as “click” , set the bounds of the components and add the components to JFrame.

Step 6: Add Action listener to the button for handling events.

Step 7: Define actionPerformed() method, create a variable for Calendar.getInstance(),create an object for Formatter class.

Step 8:Using the formatter object get the hour specifier from calendar as follows:

```
String s=fmt.Formatter("%tH",cal);
```

Where fmt is the object for Formatter class.

Step 9: convert the hour format to integer as follows:

```
int hr = Integer.parseInt(s);
```

Step 10: Based on the value of **hr** set the text of label to “Good Morning” or “Good Afternoon” or “Good Evening”.

Step 11: define public static void main() method and create an object for the class FirstJFrame

Step 12: Save the program, compile the program and execute

**E1. 1** Write a java program to display “Good Morning, Good Afternoon, Good Evening” based on the time with the help of button event using JFrame.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Calendar;
import java.util.Formatter;

class FirstJFrame implements ActionListener
{
    JFrame jfrm;
    JLabel jlbl1;
    JButton jbtn;
    FirstJFrame()
    {
        jfrm = new JFrame("First JFrame Program");
        jfrm.setSize(500, 400);
        jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jfrm.setVisible(true);
        jfrm.setLayout(null);
        jlbl1 = new JLabel("HELLO");
        jlbl1.setBounds(100, 80, 200, 50);
        jfrm.add(jlbl1);
        jbtn = new JButton("Click");
        jbtn.setBounds(100, 150, 100, 60);
        jfrm.add(jbtn);
        jbtn.addActionListener(this);
    }
    public void actionPerformed(ActionEvent e)
    {
        Calendar cal = Calendar.getInstance();
        Formatter fmt=new Formatter();
        String s=fmt.Formatter("%tH",cal);
        int hr = Integer.parseInt(s);
        String str = "";
        if(hr< 12)
            str = "Good Morning";
        else if(hr>=12 &&hr< 17)
            str = "Good Afternoon";
        else
            str = "Good Evening";
        jlbl1.setText(str);
    }
}
```

```
public static void main(String[] args) {  
    new FirstJFrame(); }  
}
```

**OUTPUT:**

## **JTextField**

JTextField is a fundamental Swing's component that allows users editing a single line of text.

JTextField inherits the JTextComponent class and uses the interface SwingConstants.

The constructor of the class are :

1. **JTextField()** : constructor that creates a new TextField
2. **JTextField(int columns)** : constructor that creates a new empty TextField with specified number of columns.
3. **JTextField(String text)** : constructor that creates a new empty text field initialized with the given string.
4. **JTextField(String text, int columns)** : constructor that creates a new empty textField with the given string and a specified number of columns .
5. **JTextField(Document doc, String text, int columns)** : constructor that creates a textfield that uses the given text storage model and the given number of columns.

### **Methods of the JTextField are:**

1. **setColumns(int n)** :set the number of columns of the text field.
2. **setFont(Font f)** : set the font of text displayed in text field.
3. **addActionListener(ActionListener l)** : set an ActionListener to the text field.
4. **int getColumns()** :get the number of columns in the textfield.
5. **setEditable(Boolean)** :Sets or indicates whether the user can edit the text in the text field.
6. **setText(String)** : Sets the textof the textfield to the given.
7. **getText()** : *Returns the text of the textfield*

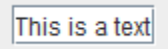
### **Creating a JTextField object**

When creating a text field component, it's common to specify some initial text and/or a number of columns from which the field's width is calculated.

- Create a text field with some initial text:

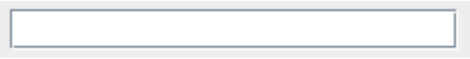


```
JTextField textField = new JTextField("This is a text");
```



- Create a text field with a specified number of columns:

```
JTextField textField = new JTextField(20);
```



- Create a text field with both initial text and number of columns:

```
JTextField textField = new JTextField("This is a text", 20);
```



- Create a default and empty text field then set the text and the number of columns later:

```
JTextField textField = new JTextField();
```

```
textField.setText("This is a text");
```

```
textField.setColumns(20);
```

1. If the initial text is not specified, its default value is null (the text field is empty).
2. If the number of columns is not specified, its default value is 0 (then the text field's width is calculated based on the initial text).

### **Adding the text field to a container**

- A JTextField can be added to any container like JFrame, JPanel, JDialog or JApplet:

```
frame.add(textField);
```

```
panel.add(textField);
```

```
dialog.add(textField);
```

- Add a JTextField to the container with a specific layout manager:

```
frame.add(textField, BorderLayout.CENTER);
```

```
panel.add(textField, gridbagConstraints);
```

### Getting or setting content of the text field

- Getting all content:

```
String content = textField.getText();
```

- Getting a portion of the content:

```
int offset = 5;  
int length = 10;  
try {  
    content = textField.getText(offset, length);  
}  
catch (BadLocationException ex) {  
    // invalid offset/length  
}
```

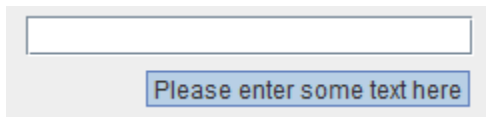
That will return 10 characters from position 5<sup>th</sup> in the text.

- Setting content:

```
textField.setText("another text");
```

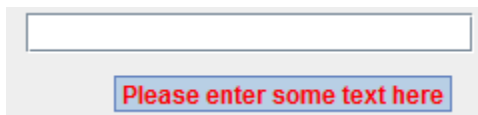
### Setting tooltip text for JTextField

```
textField.setToolTipText("Please enter some text here");
```



We can also set HTML for the tooltip text:

```
textField.setToolTipText("<html><b><font color=red>"  
    + "Please enter some text here" + "</font></b></html>");
```



### Setting input focus for JTextField

Normally, the text field gets focused when the user is clicking on it or pressing the TAB key. To set input focus programmatically, use the following code:

- Setting input focus initially just after the container (such as a JFrame) is displayed:

```
frame.setVisible(true);
```

```
textField.requestFocusInWindow();
```

### **Adding event listeners for JTextField**

- We can capture the event in which the user hits Enter key while typing in the text field.  
For example:

```
textField.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent event) {  
        System.out.println("The entered text is: " + textField.getText());  
    }  
});
```

- Capture key events which happen while the user is typing into the text field:

```
textField.addKeyListener(new KeyListener() {  
    public void keyTyped(KeyEvent event) {  
        System.out.println("key typed");  
    }  
  
    public void keyReleased(KeyEvent event) {  
        System.out.println("key released");  
    }  
  
    public void keyPressed(KeyEvent event) {  
        System.out.println("key pressed");  
    }  
});
```

The order of key events is key pressed, key typed and key released. We can use this technique to validate field's content on-the-fly. In the following example, we check the field's content whenever the user is typing. If the content is empty, disable the action button; otherwise enable the button:

```
textField.addKeyListener(new KeyAdapter() {  
    public void keyReleased(KeyEvent event) {  
  
        String content = textField.getText();  
        if (!content.equals("")) {
```

```
        button.setEnabled(true);
    }
    else {
        button.setEnabled(false);
    }
}
});
```

In this case, we use the KeyAdapter class which implements the KeyListener interface, so we have to override only the method we want.

### Working with text selection in JTextField

We can programmatically select the text field's content.

- Select all text:

```
textField.selectAll();
```



- Select only a portion of text:

```
textField.setSelectionStart(8);
```

```
textField.setSelectionEnd(12);
```



- Set color for the selection and the selected text:

```
textField.setSelectionColor(Color.YELLOW);
```

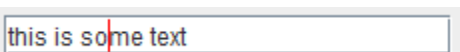
```
textField.setSelectedTextColor(Color.RED);
```



- Set position of the caret and its color:

```
textField.setCaretColor(Color.RED);
```

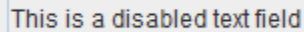
```
textField.setCaretPosition(10);
```



### Customizing JTextField's appearance

- Disable editing the field's content:

```
textField.setEditable(false);
```



- Set horizontal alignment of text:

```
textField.setHorizontalAlignment(JTextField.CENTER);
```



- Valid values for the method setHorizontalAlignment() are:

JTextField.LEFT

JTextField.CENTER

JTextField.RIGHT

JTextField.LEADING

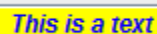
JTextField.TRAILING

- Set font style, background color and foreground color:

```
textField.setFont(new java.awt.Font("Arial", Font.ITALIC | Font.BOLD, 12));
```

```
textField.setForeground(Color.BLUE);
```

```
textField.setBackground(Color.YELLOW);
```



## E1.2 JFrame program to perform basic arithmetic calculations on given two numbers with the help of button events

### OBJECTIVE:

*This lab will develop students' knowledge in /on*

- Using JTextField and its methods

*Upon completion of this lab, students will be able to*

- develop programs on JTextField and the methods to read from and set the text to text field

Step 1: import the packages awt, awt.event, javax.swing

Step 2: Define a class calculator and define public static void main method inside it.

Step 3: Create a JFrame window by creating an object for it, specify the Layout of JFrame as null, set the size and visibility of the JFrame, set close operation to EXIT\_ON\_CLOSE.

Step 4: create three objects for JLabel which displays the labels for the textfields as "Enter Number 1: ", "Enter Number 2: ", "Result: ", three objects for JTextField, four objects for JButton labeled as "add", "subtract", "multiply", "divide". Set the bounds of the components and add the components to JFrame.

Step 5: Add ActionListener to all the buttons for handling events.

Step 6: Define actionPerformed() method for the button "add". Declare two integer variables n1, n2 to get the values from two textfields using text1.getText() and text2.getText() where text1, text2 are objects for JTextField respectively.

```
int n1=Integer.parseInt(text1.getText());
```

```
int n2=Integer.parseInt(text2.getText());
```

Step 7: Perform the addition operation for n1, n2 and set the result to the third textfield.

```
int cal=n1+n2;
```

```
text3.setText(Integer.toString(cal));
```

where text3 is the object for textfield.

Step 8: Similarly define the actionPerformed() methods for the buttons "subtract", "multiply", "Divide" to perform the operations subtraction, multiplication, division.

Step 9: Save the program, compile the program and execute .

**E1. 2** Create a JFrame program to perform basic arithmetic calculations on given two numbers with the help of button events

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
class calculator
{
    public static void main(String[] args)
    {
        JFrame f=new JFrame();
        f.setLayout(null);
        JLabel lab1=new JLabel("Enter Number 1: ");
        JLabel lab2=new JLabel("Enter Number 2: ");
        JLabel lab3=new JLabel("Result: ");

        final JTextField text1=new JTextField(20);
        final JTextField text2=new JTextField(20);
        final JTextField text3=new JTextField(20);

        JButton b1=new JButton("Add");
        JButton b2=new JButton("Subtract");
        JButton b3=new JButton("Multiply");
        JButton b4=new JButton("Division");

        lab1.setBounds(20,20,100,20);
        text1.setBounds(140,20,100,20);
        lab2.setBounds(20,50,100,20);
        text2.setBounds(140,50,100,20);
        lab3.setBounds(20,80,100,20);
        text3.setBounds(140,80,100,20);

        b1.setBounds(80,120,80,20);
        b2.setBounds(180,120,80,20);
        b3.setBounds(280,120,80,20);
        b4.setBounds(380,120,80,20);
        b1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ae){
                int n1=Integer.parseInt(text1.getText());
                int n2=Integer.parseInt(text2.getText());
                int cal=n1+n2;
                text3.setText(Integer.toString(cal));
            }
        });

        b2.addActionListener(new ActionListener() {
```

```
        public void actionPerformed(ActionEvent ae){
            int n1=Integer.parseInt(text1.getText());
            int n2=Integer.parseInt(text2.getText());
            int cal=0;
            if(n1>n2){
                cal=n1-n2;
            }
            else if(n2>n1){
                cal=n2-n1;
            }
            text3.setText(Integer.toString(cal));
        }
    });
    b3.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae){
            int n1=Integer.parseInt(text1.getText());
            int n2=Integer.parseInt(text2.getText());
            int cal=n1*n2;
            text3.setText(Integer.toString(cal));
        }
    });
    b4.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae){
            int n1=Integer.parseInt(text1.getText());
            int n2=Integer.parseInt(text2.getText());
            int cal=0;
            if(n1>n2){
                cal=n1/n2;
            }
            else if(n2>n1){
                cal=n2/n1;
            }
            text3.setText(Integer.toString(cal));
        }
    });
    f.add(lab1);
    f.add(text1);
    f.add(lab2);
    f.add(text2);
    f.add(lab3);
    f.add(text3);
    f.add(b1);
    f.add(b2);
    f.add(b3);
    f.add(b4);
    f.setVisible(true);
```



```
f.setSize(700,250);  
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
}  
}
```

**OUTPUT:**

**Viva Questions**

1. What are the differences between AWT and Swing?
2. What are the containers and components in Java Swing?
3. Explain about event handling in Java.
4. Specify various constants defined by *javax.swing.JFrame* to close the JFrame as per our requirements.
5. How to customize JLabel's appearance?

## **WEEK-2**

### **E2. 1 JFrame program from which you can open another frames with the help of button events**

**OBJECTIVE:**

*This lab will develop students' knowledge in /on*

- opening one frame from another frame

*Upon completion of this lab, students will be able to*

- develop programs for opening one frame from another frame

Step 1: import the packages awt, awt.event, javax.swing

Step 2: Define a class NewFrame

Step 3: Define a constructor NewFrame ()

Step 4: Create a JFrame window by creating an object for it, specify the Layout of JFrame as null, set the size and visibility of the JFrame, set close operation to EXIT\_ON\_CLOSE.

Step 5: create objects for JLabel and JButton labeled as “click me” , set the bounds of the components and add the components to JFrame.

Step 6: Add Action listener to the button for handling events.

Step 7: Define actionPerformed() method, Create a new JFrame window by creating an object for it, specify the Layout of JFrame as FlowLayout(), set the Location Relative To null and visibility of the JFrame, set close operation to EXIT\_ON\_CLOSE.

Step 8: define public static void main() method and create an object for the class NewFrame

Step 9: Save the program, compile the program and execute

**E2. 1** Create a JFrame program from which you can open another frames with the help of button events.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class NewFrame {
    JFrame jfrm;
    JButton jbtn;
    JLabel jlbl;

    NewFrame(){
        jfrm = new JFrame("Jframe 1");
        jfrm.setSize(400,300);
        jfrm.setLayout(new FlowLayout());

        jlbl = new JLabel("Label:");
        jfrm.add(jlbl);

        jbtn = new JButton("Clickme");
        jfrm.add(jbtn);

        jbtn.addActionListener(new ActionListener(){

            public void actionPerformed(ActionEvent e) {

                JFrame nf = new JFrame("New Frame");
                nf.setSize(600,300);
                nf.setLayout(new FlowLayout());
                nf.setLocationRelativeTo(null);
                nf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                nf.setVisible(true);

            }

        });

        jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jfrm.setVisible(true);
    }

    public static void main(String args[]){
        new NewFrame();
    }
}
```

**OUTPUT:**

## **Java Swing Layouts**

In Java swing, Layout manager is used to position all its components, with setting properties, such as the size, the shape, and the arrangement. Different layout managers could have varies in different settings on their components.

The following are some layout managers:

- FlowLayout
- BorderLayout
- CardLayout
- BoxLayout
- GridLayout
- GridBagLayout
- GroupLayout
- SpringLayout

### **FlowLayout**

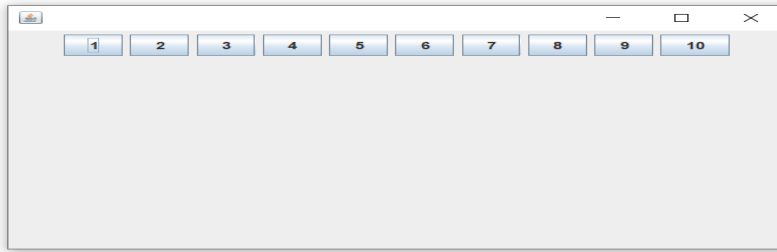
The FlowLayout arranges the components in a directional flow, either from left to right or from right to left. Normally all components are set to one row, according to the order of different components. If all components cannot be fit into one row, it will start a new row and fit the rest in.

To construct a FlowLayout, three options could be chosen:

- FlowLayout(): construct a new FlowLayout object with center alignment and horizontal and vertical gap to be default size of 5 pixels.
- FlowLayout(int align): construct similar object with different settings on alignment
- FlowLayout(int align, int hgap, int vgap): construct similar object with different settings on alignment and gaps between components.

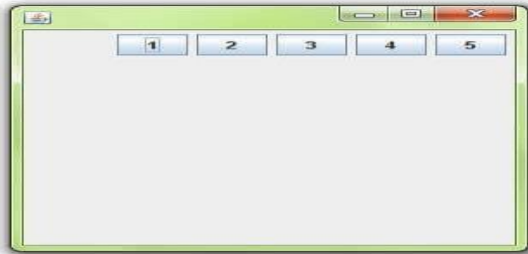
### **Example:**

```
frameObj.setLayout(new FlowLayout());  
frameObj.add(b1); frameObj.add(b2); frameObj.add(b3); frameObj.add(b4); frameObj.add(b5);  
frameObj.add(b6); frameObj.add(b7); frameObj.add(b8);frameObj.add(b9); frameObj.add(b10);
```



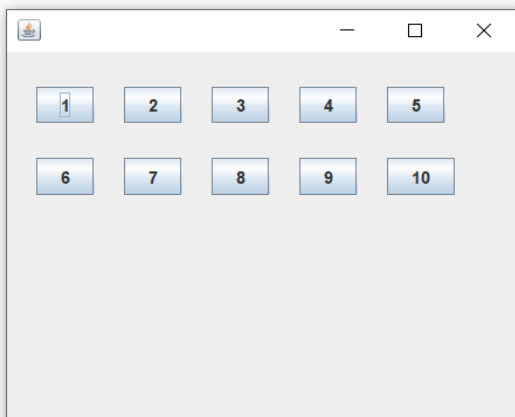
**Example of FlowLayout class: Using FlowLayout(align) constructor**

```
f.setLayout(new FlowLayout(FlowLayout.RIGHT));  
f.add(b1); f.add(b2); f.add(b3); f.add(b4); f.add(b5);
```



**Example of FlowLayout class: Using FlowLayout(align, int hgap, int vgap) constructor**

```
frameObj.setLayout(new FlowLayout(FlowLayout.LEFT, 20, 25));  
frameObj.add(b1); frameObj.add(b2); frameObj.add(b3); frameObj.add(b4); frameObj.add(b5);  
frameObj.add(b6); frameObj.add(b7); frameObj.add(b8); frameObj.add(b9); frameObj.add(b10);
```



## **BorderLayout**

The BorderLayout is used to arrange the components in five regions: north, south, east, west, and center. Each region (area) may contain one component only. It is the default layout of a frame or window. The BorderLayout provides five constants for each region:

1. public static final int NORTH
2. public static final int SOUTH
3. public static final int EAST
4. public static final int WEST
5. public static final int CENTER

### **Constructors of BorderLayout class:**

- BorderLayout(): creates a border layout but with no gaps between the components.
- BorderLayout(int hgap, int vgap): creates a border layout with the given horizontal and vertical gaps between the components.

### **Example of BorderLayout class: Using BorderLayout() constructor**

```
JButton b1 = new JButton("NORTH");; // the button will be labeled as NORTH

JButton b2 = new JButton("SOUTH");; // the button will be labeled as SOUTH

JButton b3 = new JButton("EAST");; // the button will be labeled as EAST

JButton b4 = new JButton("WEST");; // the button will be labeled as WEST

JButton b5 = new JButton("CENTER");; // the button will be labeled as CENTER

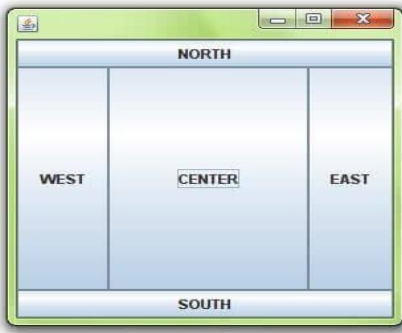

f.add(b1, BorderLayout.NORTH); // b1 will be placed in the North Direction

f.add(b2, BorderLayout.SOUTH); // b2 will be placed in the South Direction

f.add(b3, BorderLayout.EAST); // b2 will be placed in the East Direction

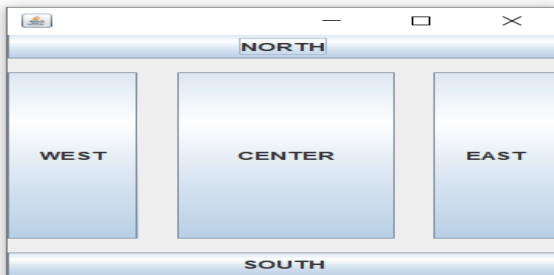
f.add(b4, BorderLayout.WEST); // b2 will be placed in the West Direction

f.add(b5, BorderLayout.CENTER);
```



### Example of BorderLayout class: Using BorderLayout(int hgap, int vgap) constructor

```
jframe.setLayout(new BorderLayout(20, 15));  
  
jframe.add(btn1, BorderLayout.NORTH);  
  
jframe.add(btn2, BorderLayout.SOUTH);  
  
jframe.add(btn3, BorderLayout.EAST);  
  
jframe.add(btn4, BorderLayout.WEST);  
  
jframe.add(btn5, BorderLayout.CENTER);
```



### GridLayout

The Java GridLayout class is used to arrange the components in a rectangular grid. One component is displayed in each rectangle.

#### Constructors of GridLayout class

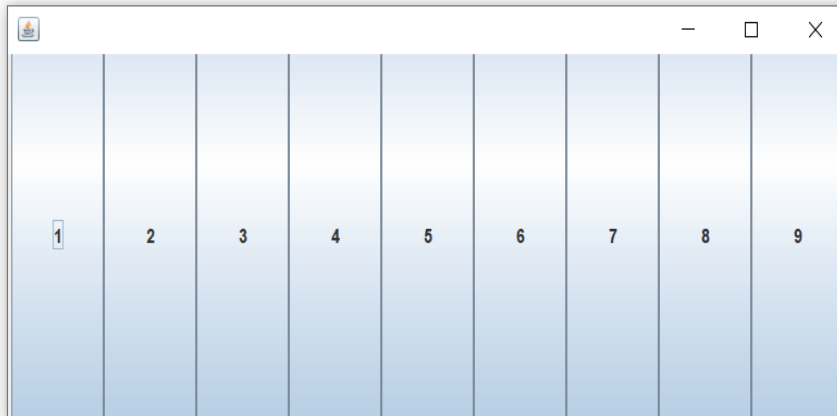
1. **GridLayout():** creates a grid layout with one column per component in a row.
2. **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.



3. **GridLayout(int rows, int columns, int hgap, int vgap):** creates a grid layout with the given rows and columns along with given horizontal and vertical gaps.

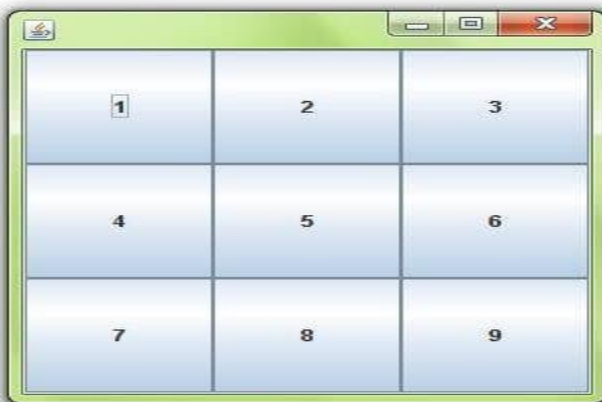
**Example of GridLayout class: Using GridLayout() Constructor**

```
frameObj.setLayout(new GridLayout());  
frameObj.add(btn1); frameObj.add(btn2); frameObj.add(btn3);  
frameObj.add(btn4); frameObj.add(btn5); frameObj.add(btn6);  
frameObj.add(btn7); frameObj.add(btn8); frameObj.add(btn9);
```



**Example of GridLayout class: Using GridLayout(int rows, int columns) Constructor**

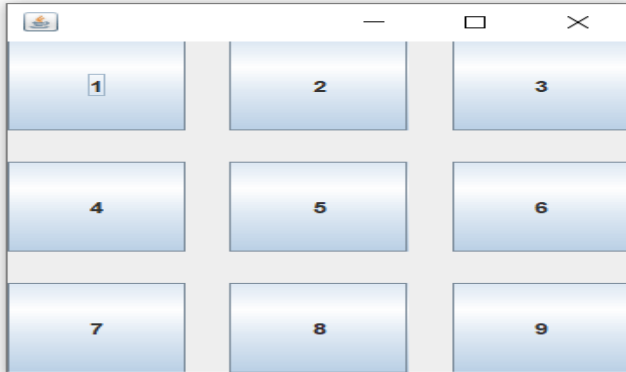
```
f.setLayout(new GridLayout(3,3));  
f.add(b1); f.add(b2); f.add(b3); f.add(b4); f.add(b5); f.add(b6); f.add(b7); f.add(b8); f.add(b9);
```



**Example of GridLayout class: Using GridLayout(int rows, int columns, int hgap, int vgap)  
Constructor**

```
f.setLayout(new GridLayout(3, 3, 20, 25));
```

```
f.add(b1); f.add(b2); f.add(b3); f.add(b4); f.add(b5); f.add(b6); f.add(b7); f.add(b8); f.add(b9);
```



## **E2. 2 Design different JFrame's to demonstrate different layouts like Flow layout, Border layout, Grid layout & null layout.**

### **OBJECTIVE:**

*This lab will develop students' knowledge in /on*

- Layout manager of container

*Upon completion of this lab, students will be able to*

- develop programs on JFrame's different layouts

### **Border Layout**

Step 1: import the packages awt, javax.swing

Step 2: Define a class Border

Step 3: Define a constructor Border ()

Step 4: Create a JFrame window by creating an object for it, specify the Layout of JFrame as BorderLayout, set the size and visibility of the JFrame, set close operation to EXIT\_ON\_CLOSE.

Step 5: Create five objects for JButton labeled as "north", "south", "East", "West", "Center".

Step 6: Add the button to BorderLayout as:

```
add(textFieldUserName, BorderLayout.CENTER);
```

Step 7: Similarly add the remaining buttons to NORTH, SOUTH, EAST, WEST for BorderLayout.

Step 8: define public static void main() method and create an object for the class Border

Step 9: Save the program, compile and execute the program

### **Flow Layout**

Step 1: import the packages awt, javax.swing

Step 2: Define a class Flow

Step 3: Define a constructor Flow ()

Step 4: Create a JFrame window by creating an object for it, specify the Layout of JFrame as

Flow Layout, set the size and visibility of the JFrame, set close operation to EXIT\_ON\_CLOSE.

Step 5: Create five objects for JButton labeled as “1”, ”2”, ”3”, ”4”, ”5”.

Step 6: Add the buttons to FlowLayout as:

```
frame.add(button);
```

Step 7: define public static void main() method and create an object for the class Flow

Step 8: Save the program, compile and execute the program

### **Grid Layout**

Step 1: import the packages awt, javax.swing

Step 2: Define a class Grid

Step 3: Define a constructor Grid ()

Step 4: Create a JFrame window by creating an object for it, specify the Layout of JFrame as GridLayout, set the size and visibility of the JFrame, set close operation to EXIT\_ON\_CLOSE.

Step 5: Create five objects for JButton labeled as “1”, ”2”, ”3”, ”4”, ”5”.

Step 6: Add the buttons to GridLayout as:

```
frame.add(button);
```

Step 7: define public static void main() method and create an object for the class Grid

Step 8: Save the program, compile and execute the program

**E2. 2**Design different JFrame's to demonstrate different layouts like Flow layout, Border layout, Grid layout & null layout

### **BORDER LAYOUT**

```
import javax.swing.*;
import java.awt.*;
public class Border {
    Border(){
        JFrame jfrm = new JFrame("BorderLayout");
        jfrm.setSize(400,300);
        BorderLayout layout = new BorderLayout();
        layout.setHgap(10);
        layout.setVgap(10);
        jfrm.setLayout(layout);

        JButton btn1 = new JButton("NORTH");
        JButton btn2 = new JButton("EAST");
        JButton btn3 = new JButton("SOUTH");
        JButton btn4 = new JButton("WEST");
        JButton btn5 = new JButton("CENTER");

        jfrm.add(btn1, BorderLayout.NORTH);
        jfrm.add(btn2, BorderLayout.EAST);
        jfrm.add(btn3, BorderLayout.SOUTH);
        jfrm.add(btn4, BorderLayout.WEST);
        jfrm.add(btn5, BorderLayout.CENTER);

        jfrm.setVisible(true);
        jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String args[]){
        new Border();
    }
}
```

### **OUTPUT:**

## **FLOW LAYOUT**

```
import javax.swing.*;
import java.awt.*;
public class Flow {
    Flow(){
        JFrame jfrm = new JFrame("FLOwLayout");
        jfrm.setSize(400,300);
        jfrm.setLayout(new FlowLayout());

        JButton btn1 = new JButton("1");
        JButton btn2 = new JButton("2");
        JButton btn3 = new JButton("3");
        JButton btn4 = new JButton("4");
        JButton btn5 = new JButton("5");
        jfrm.add(btn1);
        jfrm.add(btn2);
        jfrm.add(btn3);
        jfrm.add(btn4);
        jfrm.add(btn5);

        jfrm.setVisible(true);
        jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String args[]){
        new Flow();
    }
}
```

## **OUTPUT:**

## **GRID LAYOUT**

```
import javax.swing.*;
import java.awt.*;

public class Grid {
    Grid(){
        JFrame jfrm = new JFrame("GridLayout");
        jfrm.setSize(400,300);
        jfrm.setLayout(new GridLayout(3,3,10,10));

        JButton btn1 = new JButton("1");
        JButton btn2 = new JButton("2");
        JButton btn3 = new JButton("3");
        JButton btn4 = new JButton("4");
        JButton btn5 = new JButton("5");
        jfrm.add(btn1);
        jfrm.add(btn2);
        jfrm.add(btn3);
        jfrm.add(btn4);
        jfrm.add(btn5);

        jfrm.setVisible(true);
        jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String args[]){
        new Grid();
    }
}
```

## **OUTPUT:**

## **Window Event**

The Java WindowListener is notified whenever you change the state of window. It is notified against WindowEvent. The WindowListener interface is found in java.awt.event package. It has three methods.

### **WindowListener interface declaration**

**public interface** WindowListener **extends** EventListener

### **Methods of WindowListener interface**

The signature of 7 methods found in WindowListener interface with their usage are given below:

<b>S.no.</b>	<b>Method signature</b>	<b>Description</b>
1.	public abstract void windowActivated (WindowEvent e);	It is called when the Window is set to be an active Window.
2.	public abstract void windowClosed (WindowEvent e);	It is called when a window has been closed as the result of calling dispose on the window.
3.	public abstract void windowClosing (WindowEvent e);	It is called when the user attempts to close the window from the system menu of the window.
4.	public abstract void windowDeactivated (WindowEvent e);	It is called when a Window is not an active Window anymore.
5.	public abstract void windowDeiconified (WindowEvent e);	It is called when a window is changed from a minimized to a normal state.
6.	public abstract void windowIconified (WindowEvent e);	It is called when a window is changed from a normal to a minimized state.
7.	public abstract void windowOpened (WindowEvent e);	It is called when window is made visible for the first time.



### E2.3 JFrame program to work with window events

**OBJECTIVE:**

*This lab will develop students' knowledge in /on*

- window events

*Upon completion of this lab, students will be able to*

- develop programs on using window events with frames

Step 1: import the packages awt, awt.event, javax.swing

Step 2: Define a class WindowEvents

Step 3: Define a constructor WindowEvents()

Step 4: Create a JFrame window by creating an object for it, specify the Layout of JFrame as FlowLayout, set the size and visibility of the JFrame, set close operation to EXIT\_ON\_CLOSE.

Step 5: Add WindowListener for handling window events.

Step 6: Define the methods

windowClosing(), windowClosed(), windowIconified(), windowDeiconified(), windowActivated(), windowDeactivated() ,and print a statement for each occurrence of window event method using frame.setTitle("statement");

Step 7: define public static void main() method and create an object for the class WindowEvents

Step 8: Save the program, compile and execute the program.

**E2. 3** Create a JFrame program to work with window events

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.JFrame;
import javax.swing.JLabel;
public class WindowEvents {
    JFrame jfrm;
    WindowEvents(){
        jfrm = new JFrame("Mouse Events");
        jfrm.setSize(400,300);
        jfrm.setLayout(new FlowLayout());
        jfrm.addWindowListener(new WindowListener() {

            public void windowOpened(java.awt.event.WindowEvent e) {
                jfrm.setTitle("windowOpened");
            }

            public void windowClosing(java.awt.event.WindowEvent e) {
                jfrm.setTitle("windowClosing");
            }

            public void windowClosed(java.awt.event.WindowEvent e) {
                jfrm.setTitle("windowClosed");
            }

            public void windowIconified(java.awt.event.WindowEvent e) {
                jfrm.setTitle("windowIconified");
            }

            public void windowDeiconified(java.awt.event.WindowEvent e) {
                jfrm.setTitle("windowDeiconified");
            }

            public void windowActivated(java.awt.event.WindowEvent e) {
                jfrm.setTitle("windowActivated");
            }

            public void windowDeactivated(java.awt.event.WindowEvent e) {
                jfrm.setTitle("windowDeactivated");
            }
        });
        jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jfrm.setVisible(true);
    }

    public static void main(String args[]){
```

```
new WindowEvents();  
}
```

**OUTPUT:**

**Viva Questions**

1. Mention different Methods of JButton Class.
2. How to set or get content from JTextField ?
3. What is Swing Layout manager? What is the default Layout of the Frame or Window?
4. Mention various constructors for GridLayout.
5. Mention the abstract Methods of WindowListener interface.

## **WEEK-3**

### **JMenuBar, JMenu and JMenuItem**

JMenuBar, JMenu and JMenuItem are a part of Java Swing package.

JMenuBar is an implementation of menu bar . The JMenuBar contains one or more JMenu objects, when the JMenu objects are selected they display a popup showing one or more JMenuItem .

JMenu basically represents a menu . It contains several JMenuItem Object . It may also contain JMenu Objects (or submenu)

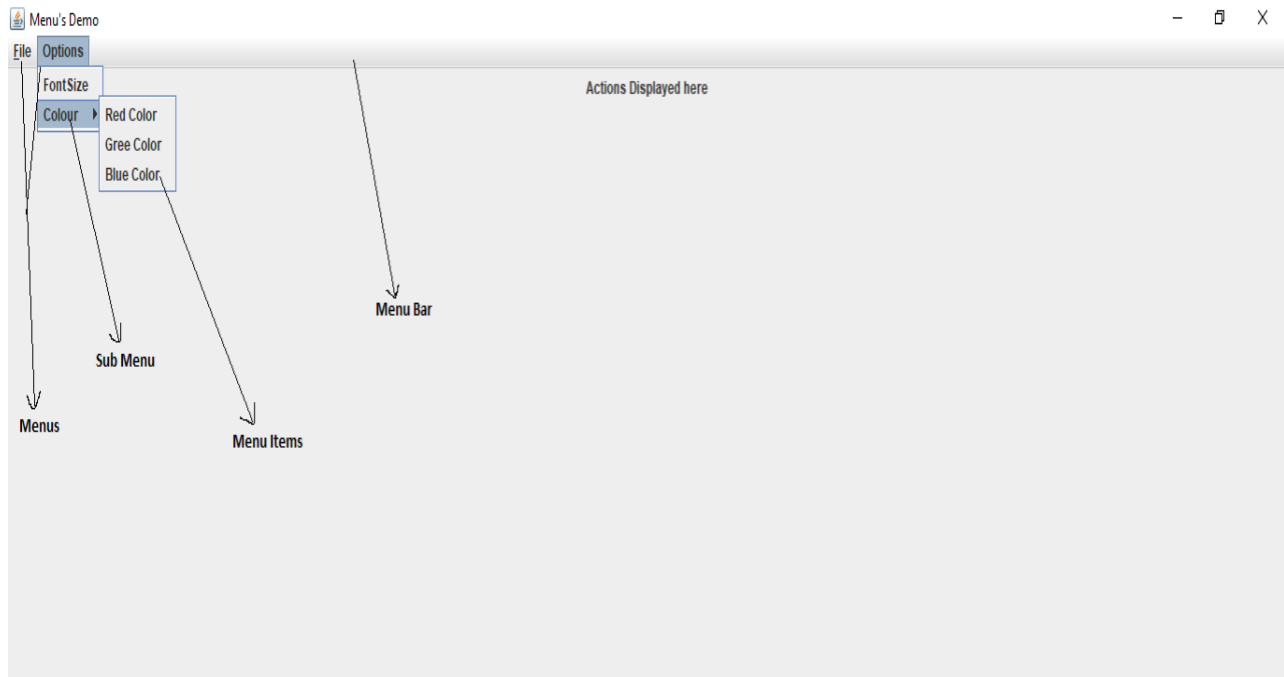
#### **Constructors :**

1. **JMenuBar()** : Creates a new MenuBar.
2. **JMenu()** : Creates a new Menu with no text.
3. **JMenu(String name)** : Creates a new Menu with a specified name.
4. **JMenu(String name, boolean b)** : Creates a new Menu with a specified name and boolean value specifies it as a tear-off menu or not. A tear-off menu can be opened and dragged away from its parent menu bar or menu.
5. **JMenu(Action a)** : Creates an instance of JMenu whose properties are taken from the specified Action.
6. **JMenuItem()** : Creates a JMenuItem instance without icon or text.
7. **JMenuItem(Icon icon)** : Creates a JMenuItem instance with a given icon.
8. **JMenuItem(String text)** : Creates a JMenuItem instance with a given text.
9. **JMenuItem(String text, Icon icon)** : Creates a JMenuItem instance with a given text and icon
10. **JMenuItem(String text, int mnemonic)** : Creates a JMenuItem instance with the given text and keyboard mnemonic.
11. **JMenuItem(Action a)** : Creates a JMenuItem instance whose properties are taken from the a given Action.

#### **Commonly used methods:**

1. **add(JMenu c)** : Adds menu to the menu bar. Adds JMenu object to the Menu bar.

2. **add(Component c)** : Add component to the end of JMenu
3. **add(Component c, int index)** : Add component to the specified index of JMenu
4. **add(JMenuItem menuItem)** : Adds menu item to the end of the menu.
5. **add(String s)** : Creates a menu item with specified string and appends it to the end of menu.
6. **getItem(int index)** : Returns the specified menuitem at the given index



**E3.1 JFrame to add a menu bar with which you can select different options from different menus and perform some action on selection of every menu item.**

**OBJECTIVE:**

*This lab will develop students' knowledge in /on*

- JMenuBar, JMenu, JMenuItem

*Upon completion of this lab, students will be able to*

- develop programs on creating a JMenuBar, adding JMenu's and JMenuItem's to it

Step 1: import the packages awt, awt.event, javax.swing

Step 2: Define a class JMenuExample

Step 3: Define a constructor JMenuExample()

Step 4: Create a JFrame window by creating an object for it, specify the Layout of JFrame as null, set the size and visibility of the JFrame, set close operation to EXIT\_ON\_CLOSE.

Step 5: create three objects for JMenu ,an object for JMenuBar,one object for JLabel labeled as "Actions displayed here", object for Font class, set the bounds of the JLabel and add the JLabel component to JFrame.

Step 6: Create a new JMenu as "File", add items "Open","Save", "Save as ", "Exit" for the "File" menu using JMenuItem as:

```
JMenuItem jmiOpen = new JMenuItem("Open");
```

Step 7: create shortcut keys the menu items "File" and "save" using setMnemonic() metod.

Step 8: Add the items "Open",Save", "Save as, "Exit" to JMenu "File".

Step 9: Create a new JMenu "Options" , new JMenu "Colors".

Step 10: Create JMenuItem s "Font size","Blue", "Green", "Red

Step 11: Add the items "Font size", "Colors" to JMenu "Options".

Step 12: Add the items "Green", "Blue", "Red" to JMenu "Colors".

Step 13: Create a new JMenuBar and add the items "File", "options" to JMenuBar.

Step 14: Add ActionListener interfaces for the items “Open”, “Save”, “Save as”, create actionPerformed() methods for each of the items which displays the statement in JLabel the type of action performed by the user.

Step 15: Add ActionListener interface for the item “Exit”, create actionPerformed() method with the statement System.exit(0) to exit the current frame.

Step 16: Add ActionListener interface for the item “Font”, create actionPerformed() method with the statement to change the font of the text in JLabel.

Step 17: Add ActionListener interfaces for the items “Green”, “Blue”, “Red”. Create actionPerformed() methods for each of the items which changes the foreground colour of the text in JLabel to the user selected colour.

Step 18: define public static void main() method and create an object for the class JMenuExample

Step 19: Save the program, compile and execute the program.

**E3. 1** Create a JFrame to add a menu bar with which you can select different options from different menus and perform some action on selection of every menu item

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class JMenuExample {

    JMenu jmFile;
    JMenu jmOptions, jmColor;
    JMenuBar jmb;
    JFrame jfrm;
    JLabel jlbl;
    Font f;

    JMenuExample(){

        jfrm = new JFrame("Menu's Demo");
        jfrm.setLayout(new FlowLayout());
        jfrm.setSize(500, 400);
        jfrm.setLocationRelativeTo(null);

        jlbl = new JLabel("Actions Displayed here");
        jfrm.add(jlbl);

        jmFile = new JMenu("File");
        JMenuItem jmiOpen = new JMenuItem("Open");
        JMenuItem jmiSave = new JMenuItem("Save");
        JMenuItem jmiSaveAs = new JMenuItem("Save As");
        JMenuItem jmiExit = new JMenuItem("Exit");

        jmFile.setMnemonic(KeyEvent.VK_F);
        jmiSave.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S,
        InputEvent.CTRL_DOWN_MASK));

        jmFile.add(jmiOpen);
        jmFile.add(jmiSave);

        jmiSave.setMnemonic(KeyEvent.VK_S);

        jmFile.add(jmiSaveAs);
        jmFile.addSeparator();
        jmFile.add(jmiExit);

        jmb = new JMenuBar();
```



```
jmb.add(jmFile);

jfrm.setJMenuBar(jmb);
jmOptions = new JMenu("Options");
jmColor = new JMenu("Colour");

JMenuItem jmiFontSize = new JMenuItem("FontSize");
JMenuItem jmiRedColor = new JMenuItem("Red Color");
JMenuItem jmiGreenColor = new JMenuItem("Green Color");
JMenuItem jmiBlueColor = new JMenuItem("Blue Color");

jmOptions.add(jmiFontSize);

jmColor.add(jmiRedColor);
jmColor.add(jmiGreenColor);
jmColor.add(jmiBlueColor);

jmb.add(jmOptions);
jmOptions.add(jmColor);

jmiOpen.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
jlbl.setText(e.getActionCommand() + " Selected");
}
});

jmiSave.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
jlbl.setText(e.getActionCommand() + " Selected");
}
});

jmiSaveAs.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
jlbl.setText(e.getActionCommand() + " Selected");
}
});

jmiExit.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
System.exit(0);
}
});

jmiFontSize.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
```

```
jlbl.setFont(new Font("ALGERIAN", Font.BOLD, 30));
}
});

jmiRedColor.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
jlbl.setForeground(Color.red);
}
});

jmiGreenColor.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
jlbl.setForeground(Color.green);
}
});

jmiBlueColor.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
jlbl.setForeground(Color.blue);
}
});
jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
jfrm.setVisible(true);
}

public static void main(String[] args) {
new JMenuExample();
}
}
```

**OUTPUT:**

## **JFileChooser**

JFileChooser is a part of java Swing package.

JFileChooser is a easy and an effective way to prompt the user to choose a file or a directory.

The object of JFileChooser class represents a dialog window from which the user can select file. It inherits JComponent class.

### **Commonly used Constructors:**

1. **JFileChooser():**Constructs a JFileChooser pointing to the user's default directory.
2. **JFileChooser(File currentDirectory) :** Constructs a JFileChooser using the given File as the path.
3. **JFileChooser(String currentDirectoryPath) :** Constructs a JFileChooser using the given path.

### **open file dialog using JFileChooser**

Create a new instance ofJFileChooser class:

```
JFileChooser jfchooser = new JFileChooser();  
jfchooser.setMultiSelectionEnabled(Boolean.FALSE);
```

### **Check if the user selects a file or not:**

```
if(jfchooser.showOpenDialog(null)== JFileChooser.APPROVE_OPTION){  
    // user selects file  
    FName = jfchooser.getSelectedFile().getAbsolutePath(); // get the path of selected file  
    jta.setText(FName);} // show the path of the selected file in the given text area
```

## **JTextArea**

JTextArea is a part of java Swing package . It represents a multi line area that displays text. It is used to edit the text .

JTextArea inherits JComponent class. The text in JTextArea can be set to different available fonts and can be appended to new text . A text area can be customized to the need of user .

### **Constructors of JTextArea**

- **JTextArea():** This is used to construct a new blank text-based area.
- **JTextArea(int row, int column):** This JTextArea is similar to that of the unparameterized JTextArea, with the difference is that it uses the rows and column parameters. It is used to construct a new text field-based area and a fixed number of rows and columns.
- **JTextArea(String s):** It is used to construct a new text-based area along with a given initial text.
- **JTextArea(String s, int row, int column):** This one is much more similar to those like the string values or the ones containing row and column parameterized values, so, therefore, this constructs a given initial text and a fixed number of rows and column values.

### **Methods of JTextArea**

- **Append(String s):** As the name suggests, this method is used for appending one given string with the text of the text area.
- **setFont(Font f):** This method is used to fix the font size and font type of text area to the given font.
- **getLineCount():** This function is used to get the number of lines in the text area text field.
- **setColumns(int c):** This is used to set the column number of the text area along with the given integer.
- **setRows(int r):** This function is used to set the rows of the text area along with the given integer.
- **getColumns():** This function is used to fetch the number of columns along with the text area field.
- **getRows():** This function is used to get the number of rows of a particular text area.

## **JScrollPane**

JScrollPane is used to give a scrollable view to your component.

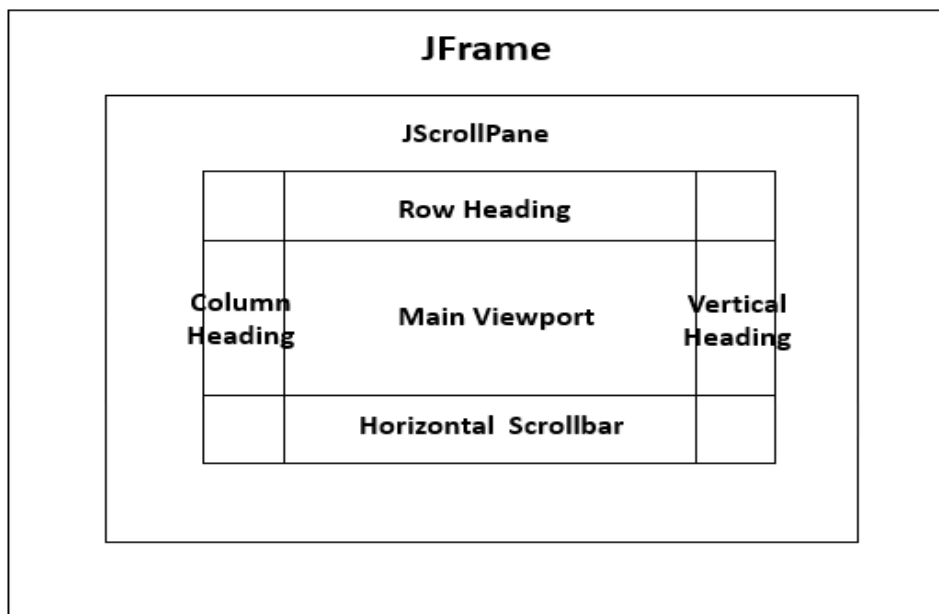
When the screen size is small or limited, we can use a scroll pane to showcase a large component or a component whose size changes dynamically.

The component should be lightweight like image, table, text, textarea, etc.

JScrollPane component should be inside the container like JFrame or JPanel

When we have limited screen size, then we need to use scroll pane for the following two conditions:

1. To display a large component.
2. To display a dynamically size changeable component.



JScrollPane class is a combination of viewports and scrollbars. It will connect our viewport with the scrollbar.

We can control our scrollbars appearances by using scrollbar display policy properties: `verticalScrollbarPolicy` and `horizontalScrollbarPolicy`.

Both these properties can have values `AS_NEEDED`, `ALWAYS`, or `NEVER`. It also has two additional viewports:

1. `rowHeading` – Used to scroll horizontally

2. columnHeading – Used to scroll vertically

**Constructors:**

1. **JScrollPane()** : Creates an empty scroll pane (no viewport). It can have both vertical and horizontal scrollbars when needed.
2. **JScrollPane(Component c)** : Creates a scroll pane with the specified component. When the component content is larger than the view, then horizontal and vertical scrollbar appears.
3. **JScrollPane(int vsPolicy, int hsPolicy)** : Creates a scroll pane with the specified scroll policies.
4. **JScrollPane(Component c, int vsPolicy, int hsPolicy)** : Creates a scroll pane with the specified component. The component position is controlled with a pair of scrollbars.

**Methods in JScrollPane Class**

Method	Description
setColumnHeaderView(Component)	It sets the column header for the scroll pane.
setRowHeaderView(Component)	It sets the row header for the scroll pane.
setCorner(String, Component)	It sets or gets the specified corner. The int parameter specifies which corner and must be one of the following constants defined in JScrollPaneConstants: UPPER_LEFT_CORNER, UPPER_RIGHT_CORNER, LOWER_LEFT_CORNER, LOWER_RIGHT_CORNER, LOWER_LEADING_CORNER, LOWER_TRAILING_CORNER, UPPER_LEADING_CORNER, UPPER_TRAILING_CORNER.
getCorner(String)	
setViewportView(Component)	Set the scroll pane's client.

### E3.2 JFrame program to open the text file using JFileChooser and display the selected text file content on the JTextArea

**OBJECTIVE:**

*This lab will develop students' knowledge in /on*

- JFileChooser, JTextArea, JScrollPane classes

*Upon completion of this lab, students will be able to*

- develop programs using JFileChooser, JTextArea, JScrollPane classes

Step 1: Import the packages javax.swing, awt, awt.event, util.Scanner, io.File, io.FileNotFoundException, util.logging.Level, util.logging.Logger

Step 2: Define the class FileReadJTextArea

Step 3: Define the constructor FileReadJTextArea()

Step 4: Create a JFrame window by creating an object for it, specify the Layout of JFrame as FlowLayout, set the size and visibility of the JFrame, set close operation to EXIT\_ON\_CLOSE.

Step 5: Create two objects for JButton labeled as "File chooser", "Read File" and add them to the frame.

Step 6: Create an object for JTextArea, set row size & column size of the JTextArea and add it to the frame

Step 7: Create a new JScrollPane and add it to the JTextArea.

Step 8: Add ActionListener interface for the button "File Chooser", define the actionPerformed() method.

Step 9: Create a new JFileChooser, set its method setMultiSelectionEnabled to FALSE.

Step 10: Approve the user selected option, get the Absolute Path of the Selected File and add the path to Text Area

Step 11: Add ActionListener interface for the button "Read File", define the actionPerformed() method to read the contents of the selected file from Text Area.

**E3. 2** Create a JFrame program to open the text file using JFileChooser and display the selected text file content on the JTextArea

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.logging.Level;
import java.util.logging.Logger;

public class FileReadJTextArea {
    JFrame jfrm;
    JButton jbtn1, jbtn2;
    JTextArea jta;
    JScrollPane jsp;
    JFileChooser jfchooser;
    String FName;

    FileReadJTextArea(){
        FName = "";
        jfrm = new JFrame("Reading File and Display in Text Area");
        jfrm.setSize(400, 300);
        jfrm.setLayout(new FlowLayout());
        jbtn1 = new JButton("File Chooser");
        jfrm.add(jbtn1);

        jbtn2 = new JButton("Read File");
        jfrm.add(jbtn2);

        jta = new JTextArea();
        jta.setRows(10);
        jta.setColumns(20);

        jsp = new JScrollPane(jta);
        jfrm.add(jsp);

        jbtn1.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {

                jfchooser = new JFileChooser();
                jfchooser.setMultiSelectionEnabled(Boolean.FALSE);

                if(jfchooser.showOpenDialog(null)== JFileChooser.APPROVE_OPTION){
```



```
FName = jfchooser.getSelectedFile().getAbsolutePath();
jta.setText(FName);
}
}
});

jbtn2.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent e) {

        String res = "";
        try (Scanner sc = new Scanner(new File(FName))) {
            while(sc.hasNext())
            {
                res += sc.nextLine() + "\n";
            }
            jta.setText(res);
        }
        catch (FileNotFoundException ex) {
            Logger.getLogger(FileReadJTextArea.class.getName()).log(Level.SEVERE,
            null, ex);
        }
    }
});
jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
jfrm.setVisible(true);
}

public static void main(String[] args) {
    new FileReadJTextArea();
}
}
```

**OUTPUT:**

## **JTable**

JTable is used to edit or display 2-D data which consists of rows and columns. It is almost similar to a spreadsheet that contains data in a tabular form.

### **Constructors of JTable**

- **JTable():** A new table will be created with empty cells.
- **JTable(int r, int c):** A table will be created with the size as r\*c.
- **JTable(Object[ ][ ] d, Object [ ]col):** A table will be created with the specified data where []col describes the names of column.

### **Methods:**

- **getSelectedRows ():** The index of selected rows will be returned.
- **getSelectedRow ():** The index of the selected row which is selected first will be returned. If no row is selected, -1 will be returned.
- **getSelectedColumn ():** The index of the selected column which is selected first will be returned. If no column is selected, -1 will be returned.
- **getSelectedColumnCount ():** A count of selected columns will be returned.
- **getSelectedColumns ():** The index of the selected columns will be returned.
- **getSelectedRowCount ():** Count of selected rows will be returned.
- **getValueAt(int row, int column):** Returns the cell value at row and column
- **isRowSelected(int row):** Returns true if the specified index is in the valid range of rows, and the row at that index is selected.
- **isEditing():** Returns true if a cell is being edited.

### E3.3 Design a registration form with the help of a JFrame and save the details in to the text file

**OBJECTIVE:**

*This lab will develop students' knowledge in /on*

- JTable

*Upon completion of this lab, students will be able to*

- develop programs on JTable creation

Step 1: import the packages `awt.event`, `io.*`, `util.Scanner`, `util.StringTokenizer`, `util.logging.Level`, `util.logging.Logger`, `javax.swing.*` .

Step 2: Define the class `RegForm`

Step 3: Define the constructor `RegForm ()`

Step 4: Create a JFrame window by creating an object for it, specify the Layout of JFrame as `FlowLayout`, set the size and visibility of the JFrame, set close operation to `EXIT_ON_CLOSE`.

Step 5: Create three JLabel objects labeled as "Name", "Roll no", "Mobl no", three JTextField objects, create two JButton objects as "Save", "Display", set the bounds of the components and add them to the frame.

Step 6: Add ActionListener for the button "Save", define `actionPerformed()` method which creates a new file called "Test1.txt", merge the data from three JTextFields, and write the data to the file "Test1.txt".

Step 7: Add ActionListener for the button "Display", define `actionPerformed()` method in which read the contents of the file "Test1.txt", divide the data of the file into tokens using `StringTokenizer`, arrange the tokens in the 2-D array called "data".

Step 8: create a `JTable(data, column)`, add to the frame.

**E3. 3**Design a registration form with the help of a JFrame and save the details in to the text file

```
import java.awt.event.*;
import java.io.*;
import java.util.Scanner;
import java.util.StringTokenizer;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.*;

public class RegForm {

    JFrame jfrm;
    JLabel jln,jlr,jlm;
    JTextField jtn,jtr,jtm;
    JScrollPane jsp;
    JTable jtbl;
    JButton jbsave,jbdisp;

    RegForm()
    {
        jfrm=new JFrame ("File Chooser");
        jfrm.setSize(500,500);
        jfrm.setLayout(null);

        jln=new JLabel("Name:");
        jln.setBounds(100,10,50,30);
        jfrm.add(jln);

        jtn=new JTextField("");
        jtn.setBounds(160,10,100,30);
        jfrm.add(jtn);

        jlr=new JLabel("Roll No:");
        jlr.setBounds(100,60,50,30);
        jfrm.add(jlr);

        jtr=new JTextField("");
        jtr.setBounds(160,60,100,30);
        jfrm.add(jtr);

        jlm=new JLabel("Mbl No:");
        jlm.setBounds(100,110,50,30);
        jfrm.add(jlm);
        jtm=new JTextField("");
        jtm.setBounds(160,110,100,30);
```

```
jfrm.add(jtm);

jbsave=new JButton("Save");
jfrm.add(jbsave);
jbsave.setBounds(100,150,100,30);

jbsave.addActionListener(new ActionListener()
{

    public void actionPerformed(ActionEvent e) {
        String t="";
        try(FileWriter fw=new FileWriter(new File("Test1.txt"),true)){
            t=jtn.getText()+":"+jtr.getText()+":"+jtm.getText()+"\n";

            if(t.length()>3)
            {
                fw.write(t); //write the contents
                JOptionPane.showMessageDialog(jfrm, "ONE ROW Successfully Entered");
            }
            else{
                JOptionPane.showMessageDialog(jfrm, "NO DATA FOUND");
            }

        } catch (IOException ex) {
            System.out.println(ex.toString());
        }

        jtn.setText("");
        jtr.setText("");
        jtm.setText("");
    }
});

jbdisp=new JButton("Display");
jfrm.add(jbdisp);
jbdisp.setBounds(180,150,100,30);

jbdisp.addActionListener(new ActionListener()
{

    public void actionPerformed(ActionEvent e) {
        int cnt=0;
        try(Scanner sc=new Scanner(new File("Test1.txt")))
        {
            for(;sc.hasNext();sc.nextLine(),cnt++);
        }
    }
});
```

```
} catch (FileNotFoundException ex) {
System.out.println(ex.toString());
}
if(cnt!=0)
{
String Data[][]=new String[cnt][3];
String colHeader[]={ "Name", "Roll", "Mbl" };
System.out.println("Cnt="+cnt);
try(Scanner sc=new Scanner(new File("Test1.txt")))
{
String text="";
int i=0;
while(sc.hasNext())
{
text=sc.nextLine();
int k=0;
StringTokenizer st=new StringTokenizer(text,":");
while(st.hasMoreTokens())
{
Data[i][k++]=st.nextToken();
}
++i;
}
} catch (FileNotFoundException ex) {
System.out.println(ex.toString());
}

jtbl=new JTable(Data,colHeader);

jsp=new JScrollPane(jtbl);
jsp.setBounds(60,200,300,150);
jfrm.add(jsp);

jtn.setText("");
jtr.setText("");
jtm.setText("");
}
}
});
jfrm.setVisible(true);
jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
public static void main(String[] args) {
new RegForm();
}
}
```

**OUTPUT:**

**Viva Questions**

1. What is JMenu? What are the benefits of utilizing Jmenubars in an application ?
2. How to use JFileChooser to select multiple files?
3. What are the different constructors for JTextArea?
4. What is the use of JScrollpane?
5. Mention 6 methods that are implemented by JTable?

## **WEEK-4**

### **Establishing JDBC Connection in Java**

JDBC is an acronym for Java Database Connectivity. It's an advancement for ODBC ( Open Database Connectivity ).

JDBC is a standard API specification developed in order to move data from frontend to the backend. This API consists of classes and interfaces written in Java.

It basically acts as an interface (not the one we use in Java) or channel between your Java program and databases i.e it establishes a link between the two so that a programmer could send data from Java code and store it in the database for future use.

#### **Steps For Connectivity Between Java Program and Database**

1. Import the Packages
2. Load the drivers using the *forName() method*
3. Register the drivers *using DriverManager*
4. Establish a connection *using the Connection class object*
5. Create a statement
6. Execute the query
7. Close the connections

#### **Step 1: Import the Packages**

```
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.sql.Statement;
```

#### **Step 2: Loading the drivers**

In order to begin with, you first need to load the driver or register it before using it in the program. Registration is to be done once in your program. You can register a driver in one of two ways mentioned below as follows:

##### **2-A Class.forName()**

Here we load the driver's class file into memory at the runtime. No need of using new or create objects. The following example uses Class.forName() to load the Oracle driver as shown below as follows:

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

##### **2-B DriverManager.registerDriver()**



DriverManager is a Java inbuilt class with a static member register. Here we call the constructor of the driver class at compile time. The following example uses DriverManager.registerDriver() to register the Oracle driver as shown below:

**DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver())**

**Step 3:** Establish a connection *using the Connection class object*

After loading the driver, establish connections as shown below as follows:

**Connection con = DriverManager.getConnection(url,user,password)**

- **user:** Username from which your SQL command prompt can be accessed.
- **password:** password from which the SQL command prompt can be accessed.
- **con:** It is a reference to the Connection interface.
- **Url:** Uniform Resource Locator which is created as shown below:

**String url = “ jdbc:oracle:thin:@localhost:1521:xe”**

Where oracle is the database used, thin is the driver used, @localhost is the IP Address where a database is stored, 1521 is the port number and xe is the service provider.

**Step 4:** Create a statement

Once a connection is established you can interact with the database. The JDBCStatement, CallableStatement, and PreparedStatement interfaces define the methods that enable you to send SQL commands and receive data from your database.

Use of JDBC Statement is as follows:

**Statement st = con.createStatement();**

*Here, con is a reference to Connection interface used in previous step .*

**Step 5:** Execute the query

The query here is an SQL Query. Now we know we can have multiple types of queries. Some of them are as follows:

- The query for updating/inserting a table in a database.
- The query for retrieving data.

The executeQuery() method of the **Statement interface** is used to execute queries of retrieving values from the database. This method returns the object of ResultSet that can be used to get all the records of a table.

The executeUpdate(sql query) method of the Statement interface is used to execute queries of updating/inserting.

#### **Step 6: Closing the connections**

So finally we have sent the data to the specified location and now we are on the verge of completing our task. By closing the connection, objects of Statement and ResultSet will be closed automatically. The close() method of the Connection interface is used to close the connection. It is shown below as follows:

```
con.close();
```

**E4. 1** Create a JFrame program to insert, delete & update the records of a database table

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.*.*;

class DBConnection
{
    Connection con;
    Statement stmt;
    ResultSet rs;

    DBConnection()
    {
        try {
            DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
            con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE", "system",
            "123456");
        }
        catch (SQLException ex) {
            Logger.getLogger(DBConnection.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    Connection getConnection()
    {
        return con;
    }

    Object getResults(String sql, boolean flag)
    {
        int n = 0;
        try {
            if(flag == true)
            {
                stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
                rs = stmt.executeQuery(sql);
                return rs;
            }
        }
    }
}
```

```
else
{
stmt = con.createStatement();
n = stmt.executeUpdate(sql);
return n;
}
}
catch (SQLException ex) {
Logger.getLogger(DBConnection.class.getName()).log(Level.SEVERE, null, ex);
}
return n;
}
}
public class JDBCUpdate {

JFramejfrm;
JTextField jtf1, jtf2;
JButtonjbtnFirst, jbtnLast, jbtnNext, jbtnPrev, jbtnQuery, jbtnDelete, jbtnUpdate,
jbtnInsert;
JLabel jlbl1, jlbl2;
boolean flag;
ResultSets;
String c1 = "";
String c2 = "";

JDBCUpdate()
{
flag = false;
jfrm = new JFrame("JFrame for DB Updates");
jfrm.setSize(350, 200);
jfrm.setLayout(null);

jlbl1 = new JLabel("C1: ");
jlbl1.setBounds(60, 10, 40, 30);
jfrm.add(jlbl1);

jtf1 = new JTextField();
jtf1.setBounds(90, 10, 50, 30);
jfrm.add(jtf1);

jlbl1 = new JLabel("C2: ");
jlbl1.setBounds(170, 10, 40, 30);
jfrm.add(jlbl1);

jtf2 = new JTextField();
jtf2.setBounds(200, 10, 50, 30);
```

```
jfrm.add(jtf2);

jbtnQuery = new JButton("Query");
jbtnQuery.setBounds(20, 50, 80, 30);
jfrm.add(jbtnQuery);

jbtnQuery.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e) {
        try {
            flag = true;
            rs = (ResultSet)new DBConnection().getResults("select * from test1", flag);
            if(rs != null)
            {
                if(rs.next())
                {
                    jtf1.setText(rs.getString("C1"));
                    jtf2.setText(rs.getString("C2"));
                    c1 = jtf1.getText();
                    c2 = jtf2.getText();
                }
            }
        } catch (SQLException ex) {
            Logger.getLogger(JDBCUpdate.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
});

jbtnDelete = new JButton("Delete");
jbtnDelete.setBounds(90, 50, 80, 30);
jfrm.add(jbtnDelete);

jbtnDelete.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e) {

        if(!jtf1.getText().isEmpty() && !jtf2.getText().isEmpty())
        {
            flag = false;
            int n = (int)new DBConnection().getResults("delete from test1 where C1 = " +
            jtf1.getText()+" and C2 = " + jtf2.getText()+"", flag);
            JOptionPane.showMessageDialog(jfrm, "Deleted " + n + " Records in DB", "Alert",
            JOptionPane.INFORMATION_MESSAGE);
            if(n != 0)
            {
                try {
```

```
flag = true;
rs = (ResultSet)new DBConnection().getResults("select * from test1", flag);
if(rs != null)
{
    if(rs.first())
    {
        jtf1.setText(rs.getString("C1"));
        jtf2.setText(rs.getString("C2"));
        c1 = jtf1.getText();
        c2 = jtf2.getText();
    }
}
else
{
    jtf1.setText("");
    jtf2.setText("");
    c1 = jtf1.getText();
    c2 = jtf2.getText();
}
} catch (SQLException ex) {
    Logger.getLogger(JDBCUpdate.class.getName()).log(Level.SEVERE, null,
ex);
}
}
}
}
});
```

```
jbtnUpdate = new JButton("Update");
jbtnUpdate.setBounds(160, 50, 80, 30);
jfrm.add(jbtnUpdate);
```

```
jbtnUpdate.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e) {
```

```
JOptionPane.showMessageDialog(jfrm, c1 + ", " + c2, "Alert",
JOptionPane.INFORMATION_MESSAGE);
if(!jtf1.getText().isEmpty() && !jtf2.getText().isEmpty())
{
    flag = false;
```

```
int n = (int)new DBConnection().getResults("update test1 set C1 = '" +
jtf1.getText()+"', C2 = '" + jtf2.getText()+"' where C1 = '" + c1 + "' and C2 = '" + c2 + "'", flag);
JOptionPane.showMessageDialog(jfrm, "Updated " + n + " Records in DB",
"Alert", JOptionPane.INFORMATION_MESSAGE)
```

```
if(n != 0)
{
    try {
        flag = true;
        rs = (ResultSet)new DBConnection().getResults("select * from test1", flag);
        if(rs != null)
        {
            if(rs.first())
            {
                jtf1.setText(rs.getString("C1"));
                jtf2.setText(rs.getString("C2"));
            }
        }
        else
        {
            jtf1.setText("");
            jtf2.setText("");
        }
    } catch (SQLException ex) {
        Logger.getLogger(JDBCUpdate.class.getName()).log(Level.SEVERE, null,
ex);
    }
}
});

jbtnInsert = new JButton("Insert");
jbtnInsert.setBounds(230, 50, 80, 30);
jfrm.add(jbtnInsert);

jbtnInsert.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e) {

if(!jtf1.getText().isEmpty() && !jtf2.getText().isEmpty())
{
    flag = false;
    int n = (int)new DBConnection().getResults("insert into test1 values(" +
jtf1.getText()+", " + jtf2.getText()+")", flag);
    JOptionPane.showMessageDialog(jfrm, "Inserted " + n + " Records in DB", "Alert",
JOptionPane.INFORMATION_MESSAGE);

if(n != 0)
{
    try {
```

```
flag = true;
rs = (ResultSet)new DBConnection().getResults("select * from test1", flag);
if(rs != null)
{
    if(rs.first())
    {
        jtf1.setText(rs.getString("C1"));
        jtf2.setText(rs.getString("C2"));
        c1 = jtf1.getText();
        c2 = jtf2.getText();
    }
}
else
{
    jtf1.setText("");
    jtf2.setText("");
    c1 = jtf1.getText();
    c2 = jtf2.getText();
}
} catch (SQLException ex) {
    Logger.getLogger(JDBCUpdate.class.getName()).log(Level.SEVERE, null,
ex);
}
}
}
}

});
jbtnFirst = new JButton("First");
jbtnFirst.setBounds(20, 100, 80, 30);
jfrm.add(jbtnFirst);

jbtnFirst.addActionListener(new ActionListener()
{

    public void actionPerformed(ActionEvent e) {

        if(rs == null)
        {
            flag = true;
            rs = (ResultSet)new DBConnection().getResults("select * from test1", flag);
        }

        if(rs != null)
        {
            try {
```



```
if(!rs.isFirst())
{
rs.first();
jtf1.setText(rs.getString("C1"));
jtf2.setText(rs.getString("C2"));
c1 = jtf1.getText();
c2 = jtf2.getText();
}
} catch (SQLException ex) {
Logger.getLogger(JDBCUpdate.class.getName()).log(Level.SEVERE, null, ex);
}
}
}
});

jbtnNext = new JButton("Next");
jbtnNext.setBounds(90, 100, 80, 30);
jfrm.add(jbtnNext);

jbtnNext.addActionListener(new ActionListener()
{
@Override
public void actionPerformed(ActionEvent e) {
//throw new UnsupportedOperationException("Not supported yet."); //To change
body of generated methods, choose Tools | Templates.
if(rs == null)
{
flag = true;
rs = (ResultSet)new DBConnection().getResults("select * from test1", flag);
}
if(rs != null)
{
try {
if(!rs.isLast())
{
rs.next();
jtf1.setText(rs.getString("C1"));
jtf2.setText(rs.getString("C2"));
c1 = jtf1.getText();
c2 = jtf2.getText();
}
} catch (SQLException ex) {
Logger.getLogger(JDBCUpdate.class.getName()).log(Level.SEVERE, null, ex);
}
}
} });
```

```
jbtnPrev = new JButton("Prev");
jbtnPrev.setBounds(160, 100, 80, 30);
jfrm.add(jbtnPrev);

jbtnPrev.addActionListener(new ActionListener()
{

    public void actionPerformed(ActionEvent e) {

        if(rs == null)
        {
            flag = true;
            rs = (ResultSet)new DBConnection().getResults("select * from test1", flag);
        }
        if(rs != null)
        {
            try {
                if(!rs.isFirst())
                rs.previous();
                jtf1.setText(rs.getString("C1"));
                jtf2.setText(rs.getString("C2"));
                c1 = jtf1.getText();
                c2 = jtf2.getText();
            } catch (SQLException ex) {
                Logger.getLogger(JDBCUpdate.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }
});

jbtnLast = new JButton("Last");
jbtnLast.setBounds(230, 100, 80, 30);
jfrm.add(jbtnLast);
jbtnLast.addActionListener(new ActionListener()
{

    public void actionPerformed(ActionEvent e) {

        if(rs == null)
        {
            flag = true;
            rs = (ResultSet)new DBConnection().getResults("select * from test1", flag);
        }
        if(rs != null)
        {
            try {
                rs.last();
```

```
jtf1.setText(rs.getString("C1"));
jtf2.setText(rs.getString("C2"));
c1 = jtf1.getText();
c2 = jtf2.getText();
} catch (SQLException ex) {
    Logger.getLogger(JDBCUpdate.class.getName()).log(Level.SEVERE, null, ex);
}
}
}
});

jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
jfrm.setVisible(true);

jfrm.setResizable(false);
}

public static void main(String[] args) {

    new JDBCUpdate();
}
}
```

**OUTPUT:**

**E4. 2** Create a JFrame program to select a database table using JComboBox component and display the content of the selected database table in JTablecomponent

```
package tablesdisplay;
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.logging.Level;
import java.util.logging.Logger;
class DBConnection
{
    Connection con;
    Statement stmt;
    ResultSets;

    DBConnection()
    {
        try {
            DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
            con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE", "system",
            "Soora123");
        } catch (SQLException ex) {
            Logger.getLogger(DBConnection.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    Connection getConnection()
    {
        return con;
    }

    ResultSetgetResults(String sql)
    {
        try {
            stmt = con.createStatement();
            rs = stmt.executeQuery(sql);
        } catch (SQLException ex) {
            Logger.getLogger(DBConnection.class.getName()).log(Level.SEVERE, null, ex);
        }
        return rs;
    }
}
```

```
}

public class TablesDisplay {

    JFrame jfrm;
    JTable jtbl;
    JComboBox<String> jcmbTable;
    ResultSets;
    int nRows, nColumns;
    String selectedTable;
    //JLabel jlbl;
    JScrollPane jsp;

    TablesDisplay()
    {
        jfrm = new JFrame("Display selected table data from ComboBox");
        jfrm.setSize(500, 400);
        jfrm.setLayout(new FlowLayout());

        jcmbTable = new <String>JComboBox();
        jcmbTable.addItem("");
        jcmbTable.addItem("EMP");
        jcmbTable.addItem("DEPT");

        jfrm.add(jcmbTable);
        jtbl = null;
        jsp = null;

        jcmbTable.addActionListener(new ActionListener()
        {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    if(jcmbTable.getSelectedItem().toString().length() > 0)
                    {
                        selectedTable = jcmbTable.getSelectedItem().toString();

                        rs = new DBConnection().getResults("select count(*) as \"ROWS\" from "
                            + selectedTable);
                        rs.next();
                        nRows = Integer.parseInt(rs.getString("ROWS"));

                        rs = new DBConnection().getResults("select count(*) as \"COLUMNS\" from "
                            + "user_tab_columns where table_name = '" + selectedTable + "'");
```

```
rs.next();
nColumns = Integer.parseInt(rs.getString("COLUMNS"));
String colHeadings[] = new String[nColumns];
String colDataTypes[] = new String[nColumns];

rs = new DBConnection().getResults("select column_name, data_type from "
+ "user_tab_columns where table_name = '" + selectedTable + "'");
for(int i=0; rs.next(); i++)
{
colHeadings[i] = rs.getString("column_name");
colDataTypes[i] = rs.getString("data_type");
}
String data[][] = new String[nRows][nColumns];

rs = new DBConnection().getResults("select * from " + selectedTable);

for(int r=0; rs.next(); r++)
{
for(int i=0; i<nColumns; i++)
{
switch(colDataTypes[i])
{
case "NUMBER":
data[r][i] = rs.getInt(colHeadings[i])+"";
break;
case "DATE":
data[r][i] = rs.getDate(colHeadings[i])+"";
break;
case "VARCHAR2":
data[r][i] = rs.getString(colHeadings[i]) + "";
}
}
}

if(jsp != null)
{
jsp.remove(jtbl);
jsp.revalidate();
jsp.repaint();
jfrm.remove(jsp);
jfrm.revalidate();
jfrm.repaint();
}
jtbl = new JTable(data, colHeadings);
jsp = new JScrollPane(jtbl);
jfrm.add(jsp);
```

```
jfrm.revalidate();
jfrm.repaint();
jfrm.repaint();
}
} catch (SQLException ex) {
    Logger.getLogger(TablesDisplay.class.getName()).log(Level.SEVERE, null, ex);
}
}
});

jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
jfrm.setVisible(true);
}
public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable()
    {
        public void run() {
            new TablesDisplay();
        }
    });
}
}
```

**OUTPUT:**

## **WEEK-5**

### **Java Generics**

The Java Generics allows us to create a single class, interface, and method that can be used with different types of data (objects).

Java Generics was introduced to deal with type-safe objects.

Generic means parameterized types. Using generics, the idea is to allow any data type to be it Integer, String, or any user-defined Datatype and it is possible to create classes that work with different data types.

#### **Advantages of Java Generics**

##### **1. Code Reusability**

With the help of generics in Java, we can write code that will work with different types of data

##### **2. Compile-time Type Checking**

The **type parameter** of generics provides information about the type of data used in the generics code

**3. Type casting is not required:** There is no need to typecast the object.

#### **Types of Java Generics**

- **Generic method:** Generic Java method takes a parameter and returns some value after performing a task.

It is exactly like a normal function, however, a generic method has **type** parameters which are cited by actual type. This allows the generic method to be used in a more general way.

```
public <T> void genericMethod(T data) { ... }
```

#### **Example:**

```
// create a generics method
```

```
public <T> void genericsMethod(T data) {  
    System.out.println("Generics Method:");  
    System.out.println("Data Passed: " + data);  
}
```



- **Generic classes:** A Generic class simply means that the **items or functions** in that class can be generalized with the parameter(example **T**) to specify that we can add any type as a parameter in place of **T** like Integer, Character, String, Double or any other user-defined type.

A generic class is implemented exactly like a non-generic class. The only difference is that it contains a **type** parameter section. There can be more than one type of parameter, separated by a comma. The classes, which accept one or more parameters, are known as parametrized classes or parameterized types.

```
class GenericsClass<T,V,...> { ... }
```

**Example:**

```
// create a generics class
class GenericsClass<T> {

    // variable of T type
    private T data;

    public GenericsClass(T data) {
        this.data = data;
    }

    // method that return T type variable
    public T getData() {
        return this.data;
    }
}

// initialize generic class with Integer data

GenericsClass<Integer> intObj = new GenericsClass<>(5);
//GenericsClass<String> obj2=new Genericsclass<>("Hello");
System.out.println("Generic Class returns: " + intObj.getData());

// initialize generic class with String data

GenericsClass<String> stringObj = new GenericsClass<>("Java Programming");
System.out.println("Generic Class returns: " + stringObj.getData());
```

## **Bounded Types with Generics**

There may be times when you want to restrict the types that can be used as type arguments in a parameterized type. For example, a method that operates on numbers might only want to accept instances of Numbers or their subclasses. This is what bounded type parameters are for.

### **How to Declare a Bounded Type Parameter in Java?**

1. List the type parameter's name,
2. Along with the extends keyword
3. And by its upper bound.

#### **Syntax**

`<T extends superClassName>`

#### **Example:**

`<T extends Number>`

### E5.1 Java program to demonstrate generic class

**OBJECTIVE:**

*This lab will develop students' knowledge in /on*

- Java Genrics

*Upon completion of this lab, students will be able to*

- develop programs on Java Generic classes

Step 1: Create a Generic class as follows:

```
class GenericsClass<T> {
```

Step 2: Declare a data member of type T, define a method with return type T to return the data member.

Step 3: Create another class, define main() method.

Step 4: Initialize generic class with Integer data, call the method of the generic class.

Initialize generic class with String data, call the method of the generic class.

Step 5: Save the program, compile and execute the program.

**E5. 1** Write a java program to demonstrate generic class

```
class genClass {
    public static void main(String[] args) {

        // initialize generic class
        // with Integer data
        GenericsClass<Integer> intObj = new GenericsClass<>(5);
        System.out.println("Generic Class returns: " + intObj.getData());

        // initialize generic class
        // with String data
        GenericsClass<String> stringObj = new GenericsClass<>("Java Programming");
        System.out.println("Generic Class returns: " + stringObj.getData());
    }
}

// create a generics class
class GenericsClass<T> {

    // variable of T type
    private T data;

    public GenericsClass(T data) {
        this.data = data;
    }

    // method that return T type variable
    public T getData() {
        return this.data;
    }
}
```

**OUTPUT:**

## E5.2 Java program to demonstrate methods and constructors in generics

### **OBJECTIVE:**

*This lab will develop students' knowledge in /on*

- Java Genrics

*Upon completion of this lab, students will be able to*

- develop programs on Java Generic methods, Generic constructors

Step 1: Create a Generic class , declare a data member of type T.

Step 2: Define a constructor of generic class with one parameter of type T.

Step 3: Define a method to display the value of data member ,define a method to display the data type of the generic class.

Step 4: Create another class, define main() method.

Step 5: Initialize generic class with double data, call the methods of generic class.

Step 6: Save the program, compile and execute the program.

**E5. 2** Write a java program to demonstrate methods and constructors in generics

```
class Gen<T>{
    T ob;
    Gen(T ob1){
        ob = ob1;
    }
    void disp(){
        System.out.println("ob = "+ob);
    }
    void showType()
    {
        System.out.println("Type of T: " + ob.getClass().getName());
    }
}
public class genCons {
    public static void main(String args[]){
        Gen<Double> a = new Gen<Double>(10.3);
        a.disp();
        a.showType();
    }
}
```

**OUTPUT:**

### E5.3 Java program to demonstrate multiple type parameters in generic classes.

**OBJECTIVE:**

*This lab will develop students' knowledge in /on*

- Java Genrics

*Upon completion of this lab, students will be able to*

- develop programs on multiple type parameters to generic classes

Step 1: Create a generic class with two type parameters as follows

```
class Gen < T, V > {
```

Step 2: Define two data members one with type T and another with type V.

Step 3: Define a constructor with two parameters of type T and V.

Step 4: Define a method with return type T and return the data member of type T.

Step 5: Define a method with return type V and return the data member of type V.

Step 6: Define two methods to display the data types of T and V.

Step 7: Create another class, define main() method.

Step 8: Initialize generic class with two data types for T and V as integer, double. Call all the methods of generic class.

Step 9: Save the program, compile and execute the program.

**E5. 3** Write a java program to demonstrate multiple type parameters in generic classes

```
class Gen < T, V > {
    T ob1;
    V ob2;
    Gen(T o1, V o2) {
        ob1 = o1;
        ob2 = o2;
    }
    T getOb1() {
        return ob1;
    }
    V getOb2() {
        return ob2;
    }
    void showOb1Type() {
        System.out.println("Type of T: " + ob1.getClass().getName());
    }
    void showOb2Type() {
        System.out.println("Type of V: " + ob2.getClass().getName());
    }
}

public class GenWith2Params {
    public static void main(String[] args) {
        Gen < Integer, Double > o1 = new Gen < Integer, Double > (233, 10.5);
        System.out.println(o1.getOb1());
        System.out.println(o1.getOb2());
        o1.showOb1Type();
        o1.showOb2Type();
    }
}
```

**OUTPUT:**



#### E5.4 Java program to demonstrate inheritances in generics

**OBJECTIVE:**

*This lab will develop students' knowledge in /on*

- Java Genrics

*Upon completion of this lab, students will be able to*

- develop programs on inheritance of generic classes

Step 1: Create a generic class as:

```
class A<T>{
```

Step 2: Declare a data member of type T, define a constructor with one parameter of type T, define a method to display the data member.

Step 3: Create a generic class class B<T,V> which inherits class A in which type T belongs to class A and type V belongs to class B

Step 4: Declare a data member of type V, define a constructor with one parameter of type T from super class and another parameter of type V, define a method to display the data member.

Step 5: Create another class, define main() method .

Step 6: Create an object for class B and initialize the object with double, integer values

**E5. 4** Write a java program to demonstrate inheritances in generics

```
class A<T>{
    T a;
    A(T a1){
        a = a1;
    }
    void showT(){
        System.out.println("a = "+a);
    }
}
class B<T,V> extends A<T>{
    V b;
    B(T a1, V b1){
        super(a1);
        b = b1;
    }
    void showV(){
        super.showT();
        System.out.println("b = "+b);
    }
}
class GenericsInheritance {
    public static void main(String args[]){
        B<Integer,Double> o1 = new B<Integer,Double>(10,25.36);
        o1.showV();
    }
}
```

**OUTPUT:**

### **Viva Questions**

1. What is Generics in Java? What are the advantages of using Generics?
2. Write about declaration of Generic methods and Generic classes.
3. Can Java generics be applied to primitive types? Justify.
4. How do you create sub-classes of generic classes?
5. What are bounded types with generics?

## **WEEK-6**

### **Collections in Java**

The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects.

Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

Java Collection means a single unit of objects.

Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

#### **What is Collection in Java**

A Collection represents a single unit of objects, i.e., a group.

#### **What is a framework in Java**

- It provides readymade architecture.
- It represents a set of classes and interfaces.
- It is optional.

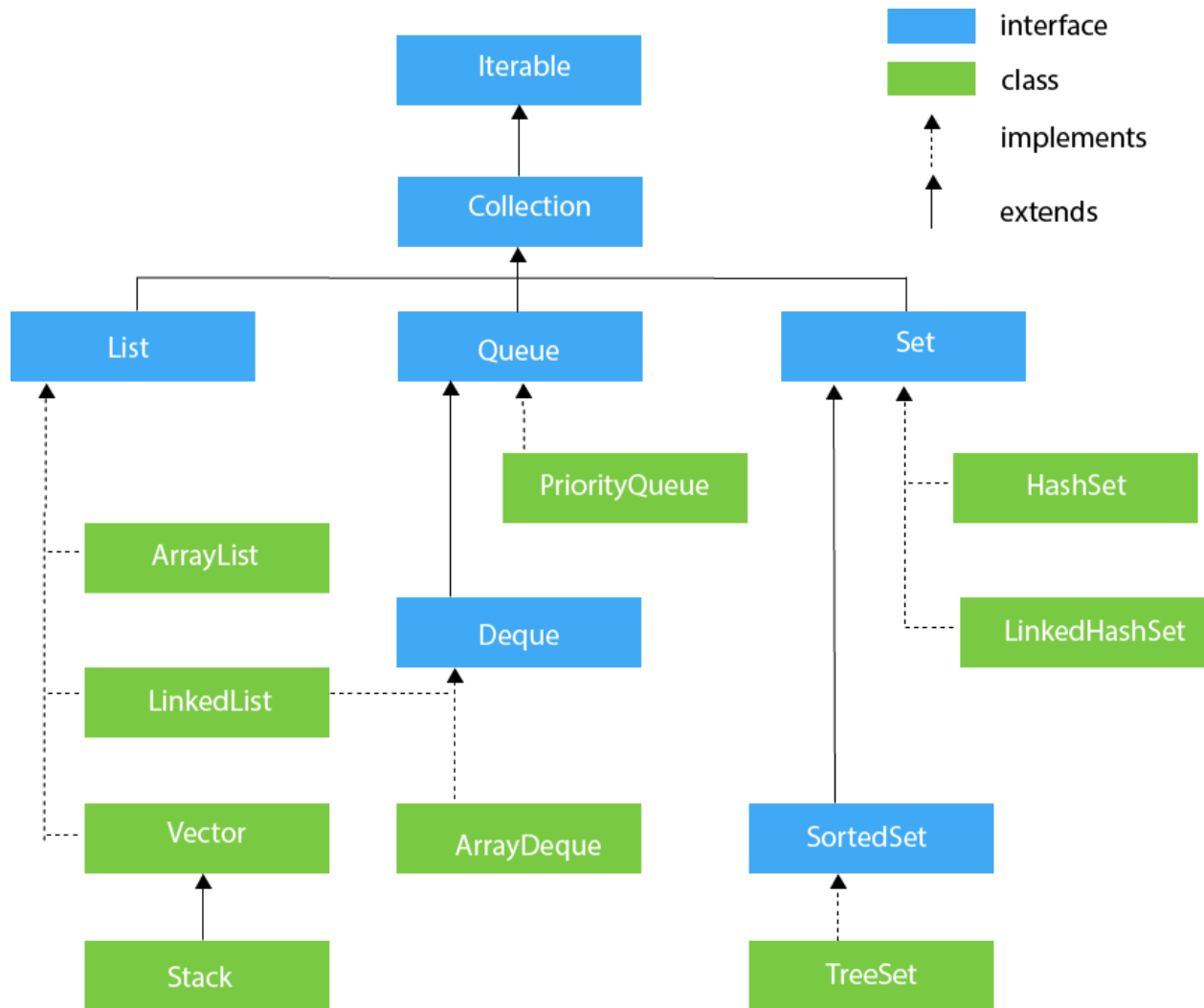
#### **What is Collection framework**

The Collection framework represents a unified architecture for storing and manipulating a group of objects. It has:

1. Interfaces and its implementations, i.e., classes
2. Algorithm

## Hierarchy of Collection Framework

The **java.util** package contains all the classes and interfaces for the Collection framework.



### Methods of Collection interface

There are many methods declared in the Collection interface. They are as follows:

Method	Description
public boolean add(E e)	It is used to insert an element in this collection.

<code>public boolean addAll(Collection&lt;? extends E&gt; c)</code>	It is used to insert the specified collection elements in the invoking collection.
<code>public boolean remove(Object element)</code>	It is used to delete an element from the collection.
<code>public boolean removeAll(Collection&lt;?&gt; c)</code>	It is used to delete all the elements of the specified collection from the invoking collection.
<code>default boolean removeIf(Predicate&lt;? super E&gt; filter)</code>	It is used to delete all the elements of the collection that satisfy the specified predicate.
<code>public boolean retainAll(Collection&lt;?&gt; c)</code>	It is used to delete all the elements of invoking collection except the specified collection.
<code>public int size()</code>	It returns the total number of elements in the collection.
<code>public void clear()</code>	It removes the total number of elements from the collection.
<code>public boolean contains(Object element)</code>	It is used to search an element.
<code>public boolean containsAll(Collection&lt;?&gt; c)</code>	It is used to search the specified collection in the collection.
<code>public Iterator iterator()</code>	It returns an iterator.
<code>public Object[] toArray()</code>	It converts collection into array.
<code>public &lt;T&gt; T[] toArray(T[] a)</code>	It converts collection into array. Here, the runtime type of the returned array is that of the specified array.

public boolean isEmpty()	It checks if collection is empty.
default Splitter<E> splitter()	It generates a Splitter over the specified elements in the collection.
public boolean equals(Object element)	It matches two collections.

## **ArrayList**

Java **ArrayList** class uses a *dynamic array* for storing the elements. It is like an array, but there is *no size limit*. We can add or remove elements anytime. It is found in the *java.util* package.

The important points about the Java ArrayList class are:

- Java ArrayList class can contain duplicate elements.
- Java ArrayList class maintains insertion order.
- Java ArrayList class is non synchronized.
- Java ArrayList allows random access because the array works on an index basis.
- In ArrayList, manipulation is a little bit slower than the LinkedList in Java because a lot of shifting needs to occur if any element is removed from the array list.
- We can not create an array list of the primitive types, such as int, float, char, etc. It is required to use the required wrapper class in such cases. For example:

```
ArrayList<int> al = ArrayList<int>(); // does not work
ArrayList<Integer> al = new ArrayList<Integer>(); // works fine
```

- Java ArrayList gets initialized by the size. The size is dynamic in the array list, which varies according to the elements getting added or removed from the list.

### **ArrayList class declaration**

**public class** ArrayList<E> **extends** AbstractList<E> **implements** List<E>, RandomAccess, Cloneable, Serializable

### **Constructors of ArrayList**

`ArrayList()` : It is used to build an empty array list.

`ArrayList(Collection<? extends E> c)` : It is used to build an array list that is initialized with the elements of the collection `c`.

`ArrayList(int capacity)` : It is used to build an array list that has the specified initial capacity.

### Methods of ArrayList

`ArrayList` can implement all the methods of `List` Interface along with the following methods

Method	Description
<code>void add(int index, E element)</code>	It is used to insert the specified element at the specified position in a list.
<code>E get(int index)</code>	It is used to fetch the element from the particular position of the list.
<code>int indexOf(Object o)</code>	It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
<code>protected void removeRange(int fromIndex, int toIndex)</code>	It is used to remove all the elements lies within the given range.
<code>E set(int index, E element)</code>	It is used to replace the specified element in the list, present at the specified position.
<code>void sort(Comparator&lt;? super E&gt; c)</code>	It is used to sort the elements of the list on the basis of the specified comparator.
<code>List&lt;E&gt; subList(int fromIndex, int toIndex)</code>	It is used to fetch all the elements that lies within the given range.



<code>void trimToSize()</code>	It is used to trim the capacity of this ArrayList instance to be the list's current size.
--------------------------------	---

## **Java LinkedList class**

Java LinkedList class uses a doubly linked list to store the elements. It provides a linked-list data structure. It inherits the AbstractList class and implements List and Deque interfaces.

The important points about Java LinkedList are:

- Java LinkedList class can contain duplicate elements.
- Java LinkedList class maintains insertion order.
- Java LinkedList class is non synchronized.
- In Java LinkedList class, manipulation is fast because no shifting needs to occur.
- Java LinkedList class can be used as a list, stack or queue

### **LinkedList class declaration**

**public class** LinkedList<E> **extends** AbstractSequentialList<E> **implements** List<E>, Deque<E>, Cloneable, Serializable

### **Constructors of Java LinkedList**

Constructor	Description
<code>LinkedList()</code>	It is used to construct an empty list.
<code>LinkedList(Collection&lt;? extends E&gt; c)</code>	It is used to construct a list containing the elements of the specified collection, in the order, they are returned by the collection's iterator.

### **Methods of Java LinkedList**

Method	Description
<code>void add(int index, E element)</code>	It is used to insert the specified element at the specified position index in a list.
<code>boolean addAll(int index, Collection&lt;? extends E&gt; c)</code>	It is used to append all the elements in the specified collection, starting at the specified position of the list.
<code>void addFirst(E e)</code>	It is used to insert the given element at the beginning of a list.
<code>void addLast(E e)</code>	It is used to append the given element to the end of a list.
<code>Iterator&lt;E&gt; descendingIterator()</code>	It is used to return an iterator over the elements in a deque in reverse sequential order.
<code>E element()</code>	It is used to retrieve the first element of a list.
<code>E get(int index)</code>	It is used to return the element at the specified position in a list.
<code>E getFirst()</code>	It is used to return the first element in a list.
<code>E getLast()</code>	It is used to return the last element in a list.
<code>int indexOf(Object o)</code>	It is used to return the index in a list of the first occurrence of the specified element, or -1 if the list does not contain any element.
<code>int lastIndexOf(Object o)</code>	It is used to return the index in a list of the last occurrence of the specified element, or -1 if the list does not contain any element.
<code>boolean offer(E e)</code>	It adds the specified element as the last element of a list.

<code>boolean offerFirst(E e)</code>	It inserts the specified element at the front of a list.
<code>boolean offerLast(E e)</code>	It inserts the specified element at the end of a list.
<code>E peek()</code>	It retrieves the first element of a list
<code>E peekFirst()</code>	It retrieves the first element of a list or returns null if a list is empty.
<code>E peekLast()</code>	It retrieves the last element of a list or returns null if a list is empty.
<code>E poll()</code>	It retrieves and removes the first element of a list.
<code>E pollFirst()</code>	It retrieves and removes the first element of a list, or returns null if a list is empty.
<code>E pollLast()</code>	It retrieves and removes the last element of a list, or returns null if a list is empty.
<code>E pop()</code>	It pops an element from the stack represented by a list.
<code>void push(E e)</code>	It pushes an element onto the stack represented by a list.
<code>E remove()</code>	It is used to retrieve and removes the first element of a list.
<code>E remove(int index)</code>	It is used to remove the element at the specified position in a list.
<code>boolean remove(Object o)</code>	It is used to remove the first occurrence of the specified element in a list.

E removeFirst()	It removes and returns the first element from a list.
boolean removeFirstOccurrence(Object o)	It is used to remove the first occurrence of the specified element in a list (when traversing the list from head to tail).
E removeLast()	It removes and returns the last element from a list.
boolean removeLastOccurrence(Object o)	It removes the last occurrence of the specified element in a list (when traversing the list from head to tail).
E set(int index, E element)	It replaces the element at the specified position in a list with the specified element.

## **Java HashSet**

Java HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.

The important points about Java HashSet class are:

- HashSet stores the elements by using a mechanism called **hashing**.
- HashSet contains unique elements only.
- HashSet allows null value.
- HashSet class is non synchronized.
- HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashcode.
- HashSet is the best approach for search operations.
- The initial default capacity of HashSet is 16, and the load factor is 0.75.

### **HashSet class declaration**

**public class** HashSet<E> **extends** AbstractSet<E> **implements** Set<E>, Cloneable, Serializable

### Constructors of Java HashSet class

Constructor	Description
HashSet()	It is used to construct a default HashSet.
HashSet(int capacity)	It is used to initialize the capacity of the hash set to the given integer value capacity. The capacity grows automatically as elements are added to the HashSet.
HashSet(int capacity, float loadFactor)	It is used to initialize the capacity of the hash set to the given integer value capacity and the specified load factor.
HashSet(Collection<? extends E> c)	It is used to initialize the hash set by using the elements of the collection c.

### Methods of Java HashSet class

HashSet implements all the methods of Collection interface.

## Java LinkedHashSet Class

Java LinkedHashSet class is a Hashtable and Linked list implementation of the Set interface. It inherits the HashSet class and implements the Set interface.

The important points about the Java LinkedHashSet class are:

- Java LinkedHashSet class contains unique elements only like HashSet.
- Java LinkedHashSet class provides all optional set operations and permits null elements.
- Java LinkedHashSet class is non-synchronized.
- Java LinkedHashSet class maintains insertion order.

Note: Keeping the insertion order in the LinkedHashSet has some additional costs, both in terms of extra memory and extra CPU cycles. Therefore, if it is not required to maintain the insertion order, go for the lighter-weight HashMap or the HashSet instead.

### LinkedHashSet Class Declaration

```
public class LinkedHashSet<E> extends HashSet<E> implements Set<E>, Cloneable, Serializable
```

### Constructors of Java LinkedHashSet Class

Constructor	Description
HashSet()	It is used to construct a default HashSet.
HashSet(Collection c)	It is used to initialize the hash set by using the elements of the collection c.
LinkedHashSet(int capacity)	It is used to initialize the capacity of the linked hash set to the given integer value capacity.
LinkedHashSet(int capacity, float fillRatio)	It is used to initialize both the capacity and the fill ratio (also called load capacity) of the hash set from its argument

### **E6.1 Java program to perform following operations on ArrayList, LinkedList, HashSet and LinkedHashSet**

- i. Insertion**
- ii. Deletion**
- iii. Traversing using traditional-for, for-each, Iterator and ListIterator**
- iv. Display the elements in reverse order**

#### **OBJECTIVE:**

*This lab will develop students' knowledge in /on*

- ArrayList, LinkedList, HashSet, LinkedHashset

*Upon completion of this lab, students will be able to*

- develop programs using ArrayList, LinkedList, HashSet, LinkedHashset

#### **ArrayList**

Step 1: import the packages util.ArrayList, util.Iterator, util.Random

Step 2: Create a class MyArrayListDemo and define main() method.

Step 3: Create a new ArrayList generic object with Integer type.

Step 4: Add elements into the ArrayList using Random class inside for loop.

Step 5: Get the size of the array using size() method, display the elements of the ArrayList using get() with for loop

Step 6: Remove a random element from the array, display the updated elements of the ArrayList using get() with for loop

Step 7: Add the element to the ArrayList at the particular index using set().

Step 8: Display the elements of the ArrayList using Iterator.

Step 9: Save the program , compile and execute the program.

#### **LinkedList**

Step 1: import the packages util.LinkedList, util.Random, util.Iterator;

Step 2: Create a class MyLinkedListDemo and define main() method.

Step 3: Create a new LinkedList generic object with Integer type.

Step 4: Add the random elements to the list using Random class with add()

Step 5: perform various operations on LinkedList using the methods getFirst(),removeFirst(), addLast(100), get(3), peek(), element(), poll(), remove(), push(150), pop()

Step 6: Display the elements of LinkedList using Iterator

## **HashSet**

Step 1: import the package java.util.\*

Step 2: Create a class MyHashSetDemo and define main( ) method.

Step 3: Create a new HashSet generic object with Integer type.

Step 4: Add the random elements to HashSet using Random class with for loop

Step 5: Display the size and elements of the HashSet using Iterator.

Step 6: Create an another new HashSet generic object2 of type Integer .

Step 7 : Add all the elements of HashSet1 to HashSet 2.

Step 8: Display the elements of HashSet 2 using or each loop.

## **LinkedHashSet**

Step 1: import the package java.util.\*

Step 2: Create a class LinkedHashSet2 and define main( ) method.

Step 3: Create a new LinkedHashSet generic object of type String.

Step 4: Add a few Strings to LinkedHashSet using add() method.

Step 5 : display the elements of LinkedHashSet using Iterator.

Step 6: Display the elements of LinkedHashSet in reverse order using for loop.



**E6. 1** Write a java program to perform following operations on ArrayList, LinkedList, HashSet and LinkedHashSet i. Insertion ii. Deletion iii. Traversing using traditional-for, for-each, Iterator and ListIterator iv. Display the elements in reverse order

### ArrayList

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Random;
public class MyArrayListDemo {
    public static void main(String args[]) {
        ArrayList<Integer> o1 = new ArrayList<Integer>();
        System.out.println("ArrayList Empty: "+o1.isEmpty());

        for(int i=1; i<= new Random().nextInt(20)+1; i++){
            o1.add(i);
        }
        System.out.println("ArrayListEmpty(After adding elements): "+o1.isEmpty());

        System.out.println("ArrayList Size: "+o1.size());

        System.out.println("Arraylist elements:");
        for(int i=0; i<o1.size();i++){
            System.out.print(o1.get(i)+" ");
        }
        System.out.println();

        Integer v2 = o1.remove(new Random().nextInt(o1.size()));
        System.out.println("The removed element is "+v2);

        System.out.println("Arraylistelements(After removing one element):");
        for(int i=0; i<o1.size();i++){
            System.out.print(o1.get(i)+" ");
        }
        System.out.println("");
        Integer v1 = 5;
        o1.set(0, v1);
        Iterator it = o1.iterator();
        System.out.println("Arraylist elements after updating first element:(Using
        Iterator):");
        while(it.hasNext()){
            System.out.print(it.next()+" ");
        }
    }
}
```

**OUTPUT:**

## LinkedList

```
import java.util.LinkedList;
import java.util.Random;
import java.util.Iterator;
public class MyLinkedListDemo {
    public static void main(String args[]) {
        LinkedList < Integer > ll1 = new LinkedList < Integer > ();
        for (int i = 1; i <= 10; i++)
            ll1.add(new Random().nextInt(100) + 1);
        System.out.println("LinkedList print using reference: " + ll1);
        System.out.println("getFirst: " + ll1.getFirst());
        System.out.println("LinkedList print using reference: " + ll1);
        System.out.println("removeFirst: " + ll1.removeFirst());
        System.out.println("LinkedList print using reference: " + ll1);
        ll1.addLast(100);
        System.out.println("LinkedList print using reference: (After addLast) " + ll1);
        System.out.println("get(3): " + ll1.get(3));
        System.out.println("peek: " + ll1.peek());
        System.out.println("LinkedList print using reference: " + ll1);
        System.out.println("element(): " + ll1.element());
        System.out.println("LinkedList print using reference: " + ll1);
        System.out.println("poll(): " + ll1.poll());
        System.out.println("LinkedList print using reference: " + ll1);
        System.out.println("remove(): " + ll1.remove());
        System.out.println("LinkedList print using reference: " + ll1);
        ll1.push(150);
        System.out.println("LinkedList print using reference: " + ll1);
        System.out.println("pop(): " + ll1.pop());
        System.out.println("LinkedList print using reference: " + ll1);
        System.out.println("LL display using Iterator");
        Iterator it1 = ll1.descendingIterator();
        for (; it1.hasNext(); )
            System.out.print(it1.next() + " ");
        System.out.println();
    }
}
```

## **OUTPUT:**

## **HashSet**

```
import java.util.*;
public class MyHashSetDemo {
    public static void main(String args[]){
        HashSet <Integer> hs1 = new HashSet <Integer>();
        System.out.println("HashSet1 empty(initially):"+hs1.isEmpty());
        Random r = new Random();
        for(int i=0; i<r.nextInt(10)+1;i++){
            hs1.add(r.nextInt(100)+1);
        }
        System.out.println("HashSet1: "+hs1);
        System.out.println("HashSet1 Size: "+hs1.size());
        System.out.println("HashSet1 using Iterator: ");
        Iterator it1 = hs1.iterator();
        for(;it1.hasNext();){
            System.out.print(it1.next()+" ");
        }
        System.out.println("\nHashset1 contains element 0:"+hs1.contains(0));
        HashSet <Integer> hs2 = new HashSet <Integer>();
        hs2.addAll(hs1);
        System.out.println("New HashSet2 after adding elements Hashset1: "+hs2);
        System.out.println("HashSet2 after invoking clear() method:");
        hs2.clear();
        for(Integer v1:hs2){
            System.out.println(v1+ " ");
        }
    }
}
```

## **OUTPUT:**

## **LinkedHashSet**

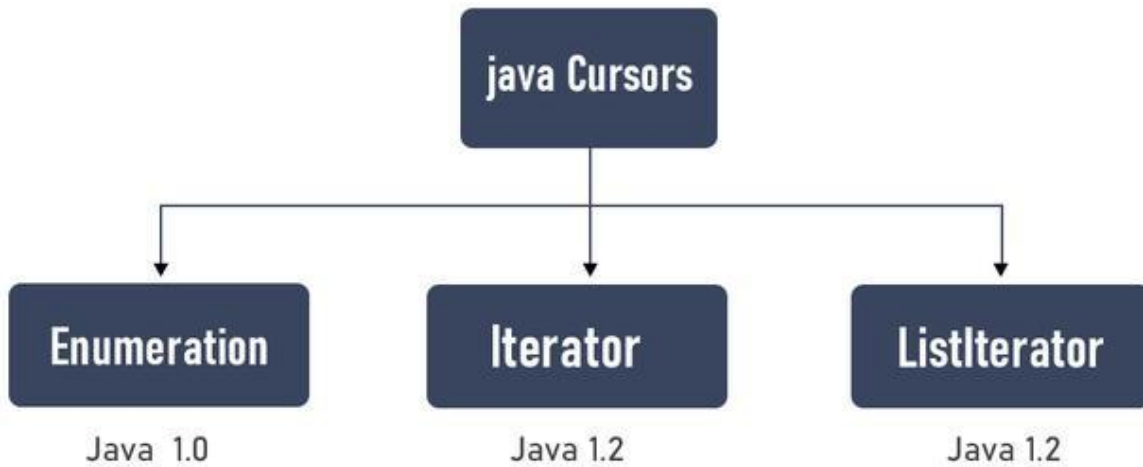
```
import java.util.*;
public class LinkedHashSet2{
public static void main(String args[]){
LinkedHashSet<String> al=new LinkedHashSet<String>();
al.add("Ravi");
al.add("Vijay");
al.add("Ravi");
al.add("Ajay");
Iterator<String>itr=al.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
}
String z[] = new String[al.size()];
al.toArray(z);
System.out.println("LinkedHashSet elements in reverse order:");
for (int i=z.length-1;i>=0;i--)
System.out.print(z[i]+" ");
}
}
```

## **OUTPUT:**

A Java Cursor is an Iterator, which is used to iterate or traverse or retrieve a Collection or Stream object's elements one by one. There are **three cursors in Java**.

1. Iterator
2. Enumeration
3. ListIterator

***Note:** SplitIterator can also be considered as a cursor as it is a type of Iterator only.*



## Java Cursors

### 1. Iterator

- Iterators in Java are used in the Collection framework to retrieve elements one by one.
  - It is a **universal** iterator as we can apply it to any Collection object.
  - By using Iterator, we can perform both read and remove operations. It is an improved version of Enumeration with the additional functionality of removing an element.
  - Iterator must be used whenever we want to enumerate elements in all Collection framework implemented interfaces like Set, List, Queue, Deque, and all implemented classes of Map interface.
  - Iterator is the **only** cursor available for the entire collection framework.
- Iterator object can be created by calling *iterator()* method present in Collection interface.

**Syntax:**

```
Iterator itr = c.iterator();
```

**Methods of Iterator Interface in Java**

Iterator interface defines **three** methods as listed below:

**1. hasNext():** Returns true if the iteration has more elements.

```
public boolean hasNext();
```

**2. next():** Returns the next element in the iteration. It throws **NoSuchElementException** if no more element is present.

```
public Object next();
```

**3. remove():** Removes the next element in the iteration. This method can be called only once per call to next().

```
public void remove();
```

*remove() method can throw two exceptions namely as follows:*

- **UnsupportedOperationException** : If the remove operation is not supported by this iterator
- **IllegalStateException** : If the next method has not yet been called, or the remove method has already been called after the last call to the next method.

**Advantages of Java Iterator**

- We can use it for any Collection class.
- It supports both READ and REMOVE operations.
- It is a Universal Cursor for Collection API.
- Method names are simple and easy to use them.

## **2. Enumeration**

It is an interface used to get elements of legacy collections(Vector, Hashtable).

Enumeration is the first iterator present from JDK 1.0, rests are included in JDK 1.2 with more functionality.

Enumerations are also used to specify the input streams to a *SequenceInputStream*. We can create an Enumeration object by calling *elements()* method of vector class on any vector object

```
// Here "v" is an Vector class object. e is of  
// type Enumeration interface and refers to "v"
```

```
Enumeration e = v.elements();
```

There are **two** methods in the Enumeration interface namely :

**1. public boolean hasMoreElements():** This method tests if this enumeration contains more elements or not.

**2. public Object nextElement():** This method returns the next element of this enumeration. It throws NoSuchElementException if no more element is present

There are certain limitations of enumeration which are as follows:

- Enumeration is for **legacy** classes(Vector, Hashtable) only. Hence it is not a universal iterator.
- Remove operations can't be performed using Enumeration.
- Only forward direction iterating is possible.

### Similarities between Java Enumeration and Iterator

- Both are Java Cursors.
- Both are used to iterate a Collection of object elements one by one.
- Both support READ or Retrieval operation.
- Both are Uni-directional Java Cursors which means support only Forward Direction Iteration.

### Differences between Java Enumeration and Iterator

Enumeration	Iterator
Introduced in Java 1.0	Introduced in Java 1.2
It is used to iterate only Legacy Collection classes.	We can use it for any Collection class.



Enumeration	Iterator
It supports only READ operation.	It supports both READ and DELETE operations.
It's not Universal Cursor.	It is a Universal Cursor.
Lengthy Method names.	Simple and easy-to-use method names.

### **3. ListIterator**

- It is only applicable for List collection implemented classes like ArrayList, LinkedList, etc.
- It provides bi-directional iteration. ListIterator must be used when we want to enumerate elements of List.
- This cursor has more functionality(methods) than iterator. ListIterator object can be created by calling *listIterator()* method present in the List interface.

ListIterator ltr = l.**listIterator()**;

Here “l” is any List object, ltr is of type. ListIterator interface and refers to “l”. ListIterator interface extends the Iterator interface. So all three methods of Iterator interface are available for ListIterator. In addition, there are **six** more methods.

#### **1. Forward direction**

**1.1 hasNext():** Returns true if the iteration has more elements  
public boolean hasNext();

**1.2 next():** Same as next() method of Iterator. Returns the next element in the iteration.  
public Object next();

**1.3 nextIndex():** Returns the next element index or list size if the list iterator is at the end of the list.

```
public int nextIndex();
```

## **2. Backward direction**

**2.1 hasPrevious():** Returns true if the iteration has more elements while traversing backward.

```
public boolean hasPrevious();
```

**2.2 previous():** Returns the previous element in the iteration and can throw **NoSuchElementException** if no more element present.

```
public Object previous();
```

**2.3 previousIndex():** Returns the previous element index or -1 if the list iterator is at the beginning of the list,

```
public int previousIndex();
```

## **3. Other Methods**

**3.1 remove():** Same as remove() method of Iterator. Removes the next element in the iteration.

```
public void remove();
```

**3.2 set(Object obj):** Replaces the last element returned by next() or previous() with the specified element.

```
public void set(Object obj);
```

**3.3 add(Object obj):** Inserts the specified element into the list at the position before the element that would be returned by next()

```
public void add(Object obj);
```

## Vector Class in Java

The Vector class implements a growable array of objects.

It is found in java.util package and implement the List interface, so we can use all the methods of the List interface.

Vector implements a dynamic array which means it can grow or shrink as required. Like an array, it contains components that can be accessed using an integer index.

They are very similar to ArrayList, but Vector is synchronized and has some legacy methods that the collection framework does not contain.

It also maintains an insertion order like an ArrayList. Still, it is rarely used in a non-thread environment as it is **synchronized**, and due to this, it gives a poor performance in adding, searching, deleting, and updating its elements.

The Iterators returned by the Vector class are fail-fast. In the case of concurrent modification, it fails and throws the **ConcurrentModificationException**.

### Constructors

**1. Vector():** Creates a default vector of the initial capacity is 10.

```
Vector<E> v = new Vector<E>();
```

**2. Vector(int size):** Creates a vector whose initial capacity is specified by size.

```
Vector<E> v = new Vector<E>(int size);
```

**3. Vector(int size, int incr):** Creates a vector whose initial capacity is specified by size and increment is specified by incr. It specifies the number of elements to allocate each time a vector is resized upward.

```
Vector<E> v = new Vector<E>(int size, int incr);
```

**4. Vector(Collection c):** Creates a vector that contains the elements of collection c.

**E6.2 A program that will have a Vector which is capable of storing Employee objects. Use an Iterator and enumeration to list all the elements of the Vector**

**OBJECTIVE:**

*This lab will develop students' knowledge in /on*

- Vector class, Iterator, Enumeration

*Upon completion of this lab, students will be able to*

- develop programs using Vector class, Iterator, Enumeration

Step 1: import the packages util.Vector, util.Random, util.Iterator, util.Enumeration

Step 2: Create the class Employee with data members as employee name and id.

Step 3: Define the constructor Employee with parameters and pass the parameters to the data members.

Step 4: Define the method to display the details such as name and id of the employee.

Step 5: Create the class MyVectorDemo and define main() method.

Step 6: Create a new Vector object for class employee as

```
Vector <Employee> o1 = new Vector <Employee>();
```

Step 7: Read the values for employee class using Random class with for loop

Step 8: Display the details of employee using Iterator and Enumeration.

Step 9 : save the program, compile and execute the program.

**E6. 2** Write a program that will have a Vector which is capable of storing Employee objects. Use an Iterator and enumeration to list all the elements of the Vector

```
import java.util.Vector;
import java.util.Random;
import java.util.Iterator;
import java.util.Enumeration;

class Employee {
    private String empName, empNo;
    Employee(String ename, String empno) {
        empName = ename;
        empNo = empno;
    }
    void dispEmpData() {
        System.out.println("Name: " + empName + "\tNo: " + empNo);
    }
}

class MyVectorDemo
{
    public static void main(String args[])
    {
        Vector <Employee> o1 = new Vector <Employee>();
        Random r = new Random();

        for(int i=0; i < r.nextInt(20)+1; i++){
            int n = r.nextInt(100)+1;
            o1.add(new Employee("Name"+n, "Empno"+n));
        }

        System.out.println("Using Iterator:");
        Iterator <Employee> it1 = o1.iterator();
        while(it1.hasNext()){
            it1.next().dispEmpData();
        }

        System.out.println("\nUsing Enumeration:");
        Enumeration <Employee> enum1 = o1.elements();
        while(enum1.hasMoreElements()){
            enum1.nextElement().dispEmpData();
        }
    }
}
```

**OUTPUT:**

**Viva Questions**

1. What is Collection framework? Mention any 5 methods of collection Interface.
2. What is the difference between LinkedList and LinkedHashSet?
3. What is the difference between Iterator and Enumeration ?
4. What is the difference between ArrayList and Vector?
5. Mention all the methods of ListIterator.

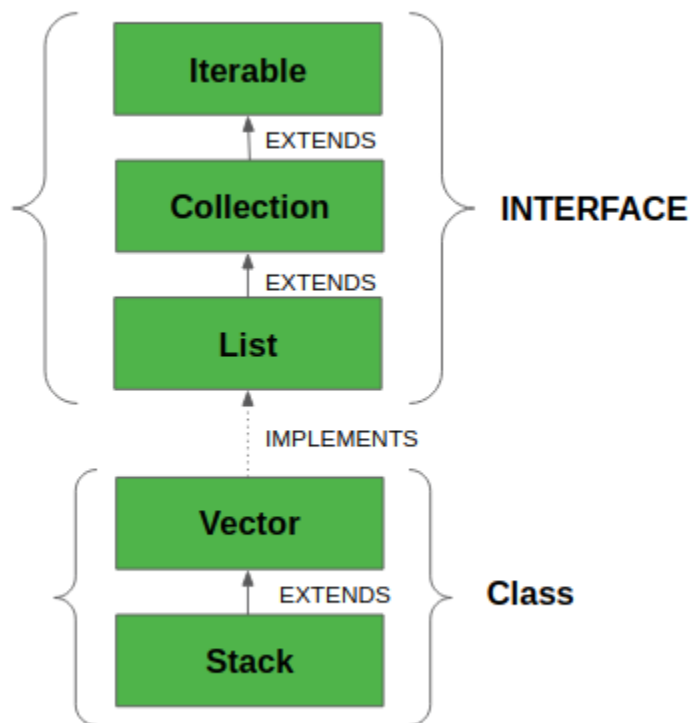
## WEEK-7

### Java Stack Class

The **stack** is a linear data structure that is used to store the collection of objects. It is based on **Last-In-First-Out** (LIFO)

The stack data structure has the two most important operations that are **push** and **pop**. The push operation inserts an element into the stack and pop operation removes an element from the top of the stack.

In Java, **Stack** is a class that falls under the Collection framework that extends the **Vector** class. It also implements interfaces **List**, **Collection**, **Iterable**, **Cloneable**, **Serializable**. It represents the LIFO stack of objects



#### Stack Class Constructor

The Stack class contains only the **default constructor** that creates an empty stack.

```
public Stack()
```

#### Creating a Stack

If we want to create a stack, first, import the java.util package and create an object of the Stack class.

```
Stack stk = new Stack();
```

Or

```
Stack<type> stk = new Stack<>();
```

Where type denotes the type of stack like Integer, String, etc.

### Methods of the Stack Class

We can perform push, pop, peek and search operation on the stack.

The Java Stack class provides mainly five methods to perform these operations. Along with this, it also provides all the methods of the Java Vector class.

Method	Modifier and Type	Method Description
empty()	Boolean	The method checks the stack is empty or not.
push(E item)	E	The method pushes (insert) an element onto the top of the stack.
pop()	E	The method removes an element from the top of the stack and returns the same element as the value of that function.
peek()	E	The method looks at the top element of the stack without removing it.
search(Object o)	Int	The method searches the specified object and returns the position of the object.



### E7.1 Java program to perform different operations on inbuilt Stack class

**OBJECTIVE:**

*This lab will develop students' knowledge in /on*

- inbuilt stack class in java

*Upon completion of this lab, students will be able to*

- develop programs using Stack class

Step 1: import the packages util.Stack, util.Random, util.Iterator

Step 2: Create a class MyStackDemo and define the main() method

Step 3: Create a new Stack generic object of type Integer as:

```
Stack <Integer> s1 = new Stack <Integer>();
```

Step 4: Display the status of the stack such as size, capacity, is Empty

Step 5: Push the elements into the stack using Random class with for loop

Step 6: Display the size, elements of the stack using Iterator

Step 7: Pop the elements of the stack and display the status of the stack.

Step 8: Save the program, compile and execute the program.

**E7. 1** Write a java program to perform different operations on inbuilt Stack class

```
import java.util.Stack;
import java.util.Random;
import java.util.Iterator;

class MyStackDemo
{
    public static void main(String args[])
    {
        Stack <Integer> s1 = new Stack <Integer>();
        System.out.println("capacity: " + s1.capacity() + " size: " + s1.size() + " Empty: "
+ s1.isEmpty());

        int size = new Random().nextInt(10)+1;

        for(int i=0; i<size; i++)
        {
            s1.push(new Random().nextInt(100));
        }

        Iterator <Integer> it1 = s1.iterator();

        System.out.println("Stack Size: " + s1.size());
        while(it1.hasNext())
        {
            System.out.print(it1.next() + " ");
        }
        System.out.println();
        System.out.println("Popped elements are: ");
        while(!s1.empty())
        {
            System.out.print(s1.pop() + " ");
        }
    }
}
```

**OUTPUT:**

### **Java Queue Interface**

The interface Queue is available in the java.util package and does extend the Collection interface.

It is used to keep the elements that are processed in the First In First Out (FIFO) manner. It is an ordered list of objects, where insertion of elements occurs at the end of the list, and removal of elements occur at the beginning of the list.

Being an interface, the queue requires, for the declaration, a concrete class, and the most common classes are the LinkedList and PriorityQueue in Java.

#### **Methods of Java Queue Interface**

Method	Description
boolean add(object)	It is used to insert the specified element into this queue and return true upon success.
boolean offer(object)	It is used to insert the specified element into this queue.
Object remove()	It is used to retrieves and removes the head of this queue.
Object poll()	It is used to retrieves and removes the head of this queue, or returns null if this queue is empty.
Object element()	It is used to retrieves, but does not remove, the head of this queue.
Object peek()	It is used to retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.

## **PriorityQueue Class**

PriorityQueue is also class that is defined in the collection framework that gives us a way for processing the objects on the basis of priority.

**public class** PriorityQueue<E> **extends** AbstractQueue<E> **implements** Serializable

### **Constructors**

**PriorityQueue()** : Creates a PriorityQueue with the default initial capacity that orders its elements according to their **natural ordering**

**PriorityQueue(Collection<? extends E> c)** : Creates a PriorityQueue containing the elements in the specified collection.

**PriorityQueue (int initialCapacity)** : Creates a PriorityQueue with the specified initial capacity that orders its elements according to their **natural ordering**.

**PriorityQueue(int initialCapacity, Comparator<? super E> comparator)** : Creates a PriorityQueue with the specified initial capacity that orders its elements according to the specified comparator.

**PriorityQueue(PriorityQueue<? extends E> c)** :Creates a PriorityQueue containing the elements in the specified priority queue.

**PriorityQueue(SortedSet<? extends E> c)** : Creates a PriorityQueue containing the elements in the specified sorted set.

## E7.2 Java program to perform different operations on inbuilt Queue class

### OBJECTIVE:

*This lab will develop students' knowledge in /on*

- inbuilt Queue in java

*Upon completion of this lab, students will be able to*

- develop programs using inbuilt Queue

Step 1: import the packages util.PriorityQueue, util.Random, util.Iterator

Step 2: Create a class MyPriorityQueueDemo and define main() method

Step 3: Create a new PriorityQueue generic object of type Integer as:

```
PriorityQueue< Integer >pq = new PriorityQueue< Integer > ();
```

Step 4: Add the elements into the queue using Random class with for loop.

Step 5: Display the elements of the queue using Iterator.

Step 6: Remove the elements of the queue, display the size of queue, peek element of the queue.

Step 7: Save the program, compile and execute the program.

**E7. 2** Write a java program to perform different operations on inbuilt Queue class

```
import java.util.PriorityQueue;
import java.util.Random;
import java.util.Iterator;
public class MyPriorityQueueDemo {
    public static void main(String args[]) {
        PriorityQueue< Integer > pq = new PriorityQueue< Integer > ();
        int n = new Random().nextInt(10) + 1;
        for (int i = 0; i < n; i++) {
            int n1 = new Random().nextInt(100);
            System.out.print(n1 + " ");
            pq.add(n1);
        }
        System.out.println("\nPrioirity queue: "+pq);
        Iterator <Integer> it1 = pq.iterator();
        System.out.println("\nValues of priority queue using Iterator");
        while (it1.hasNext()) {
            System.out.print(it1.next() + " ");
        }
        System.out.println("\nBefore offer size: " + pq.size());
        Integer v1 = 115;
        System.out.println("peek: " + pq.offer(v1));
        it1 = pq.iterator();
        System.out.println("Elements of priority queue using Iterator after offer");
        while (it1.hasNext()) {
            System.out.print(it1.next() + " ");
        }
        System.out.println("\nAfter offer size: " + pq.size());
        while (pq.size() > 0) {
            Integer i1 = pq.peek();
            System.out.print(i1 + " ");
            pq.remove(i1);
        }
        System.out.println("\nSize of PQ: " + pq.size());
    }
}
```

**OUTPUT:**

## **Java HashMap**

Java **HashMap** class implements the Map interface which allows us to store key and value pair, where keys should be unique.

If you try to insert the duplicate key, it will replace the element of the corresponding key.

It is easy to perform operations using the key index like updation, deletion, etc. HashMap class is found in the java.util package.

It allows us to store the null elements as well, but there should be only one null key.

It is denoted as HashMap<K,V>, where K stands for key and V for value. It inherits the AbstractMap class and implements the Map interface.

There should be only one null key object and there can be any number of null values.

Java HashMap maintains no order.

### **HashMap class declaration**

**public class** HashMap<K,V> **extends** AbstractMap<K,V> **implements** Map<K,V>, Cloneable, Serializable

HashMap class Parameters

- **K**: It is the type of keys maintained by this map.
- **V**: It is the type of mapped values.

### **Constructors in HashMap:**

1. **HashMap()** : It is used to construct a default HashMap.
2. **HashMap(int initialCapacity)** : It is used to initialize the capacity of the hash map to the given integer value, capacity.
3. **HashMap(int initialCapacity, float loadFactor)** : It is used to initialize both the capacity and load factor of the hash map by using its arguments.
4. **HashMap(Map map)** : It is used to initialize the hash map by using the elements of the given Map object m.

### Methods of Java HashMap class

Method	Description
V put(Object key, Object value)	It is used to insert an entry in the map.
void putAll(Map map)	It is used to insert the specified map in the map.
V putIfAbsent(K key, V value)	It inserts the specified value with the specified key in the map only if it is not already specified.
V remove(Object key)	It is used to delete an entry for the specified key.
boolean containsValue(Object value)	This method returns true if some value equal to the value exists within the map, else return false.
boolean containsKey(Object key)	This method returns true if some key equal to the key exists within the map, else return false.
boolean equals(Object o)	It is used to compare the specified Object with the Map.
V get(Object key)	This method returns the object that contains the value associated with the key.
V replace(K key, V value)	It replaces the specified value for a specified key.
int size()	This method returns the number of entries in the map.



## Java TreeMap class

Java TreeMap class is a red-black tree based implementation. It provides an efficient means of storing key-value pairs in sorted order.

Java TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.

Java TreeMap contains only unique elements.

Java TreeMap cannot have a null key but can have multiple null values.

Java TreeMap is non synchronized.

Java TreeMap maintains ascending order.

### TreeMap class declaration

```
public class TreeMap<K,V> extends AbstractMap<K,V> implements NavigableMap<K,V>, Cloneable, Serializable
```

- **K**: It is the type of keys maintained by this map.
- **V**: It is the type of mapped values.

### Constructors of Java TreeMap class

Constructor	Description
TreeMap()	It is used to construct an empty tree map that will be sorted using the natural order of its key.
TreeMap(Comparator<? super K> comparator)	It is used to construct an empty tree-based map that will be sorted using the comparator comp.
TreeMap(Map<? extends K,? extends V> m)	It is used to initialize a treemap with the entries from <b>m</b> , which will be sorted using the natural order of the keys.
TreeMap(SortedMap<K,? extends V> m)	It is used to initialize a treemap with the entries from the SortedMap <b>sm</b> , which will be sorted in the same order as <b>sm</b> .

## Methods of Java TreeMap class

Method	Description
Map.Entry<K,V> ceilingEntry(K key)	It returns the key-value pair having the least key, greater than or equal to the specified key, or null if there is no such key.
K ceilingKey(K key)	It returns the least key, greater than the specified key or null if there is no such key.
Map.Entry firstEntry()	It returns the key-value pair having the least key.
K higherKey(K key)	It is used to return true if this map contains a mapping for the specified key.
Set keySet()	It returns the collection of keys exist in the map.
V put(K key, V value)	It inserts the specified value with the specified key in the map.
V replace(K key, V value)	It replaces the specified value for a specified key.
boolean containsKey(Object key)	It returns true if the map contains a mapping for the specified key.
boolean containsValue(Object value)	It returns true if the map maps one or more keys to the specified value.
V remove(Object key)	It removes the key-value pair of the specified key from the map.
int size()	It returns the number of key-value pairs exists in the hashtable.

### E7.3 Java program to perform insertion, deletion, traversing and searching operations on HashMap and TreeMap

#### OBJECTIVE:

*This lab will develop students' knowledge in /on*

- HashMap, TreeMap

*Upon completion of this lab, students will be able to*

- develop programs using HashMap, TreeMap

#### HashMap

Step 1: import the packages util.HashMap, util.Random, java.util.Set, util.Iterator

Step 2: Create a class MyHashMapDemo and define main() method

Step 3: Create new HashMap generic object as:

```
HashMap < Integer, Integer > hm1 = new HashMap < Integer, Integer > ();
```

Step 4: Add the keys and values to the hash map using **hm1.put(k,v)** where k is the key and v is the value generated by using Random class with for loop.

Step 5: Using the method keySet() retrieve the collection of keys in the map into Set object.

Step 6 : Display the keys and their corresponding values of the hash map using Iterator

Step 7: Delete a Key and its Value from the hash map using the method remove()

Step 8 : Display the keys and their corresponding values of the updated hash map using Iterator

Step 9: Save the program, compile and execute the program

#### TreeMap

Step 1: import the packages util.TreeMap, util.Random, java.util.Set, util.Iterator

Step 2: Create a class MyTreeMapDemo and define main() method

Step 3: Create new TreeMap generic object as:

```
TreeMap<Integer,Integer> tm1 = new TreeMap<Integer,Integer> ();
```

Step 4: Add the keys and values to the hash map using **tm1.put(k,v)** where k is the key and v is the value generated by using Random class with for loop.

Step 5: Using the method keySet() retrieve the collection of keys in the map into Set object.

Step 6 : Display the keys and their corresponding values of the hash map using Iterator

Step 7: Replace the value of a key using the method replace()

Step 8: Delete a Key and its Value from the hash map using the method remove()

Step 9 : Display the keys and their corresponding values of the updated hash map using Iterator

Step 10: Save the program, compile and execute the program

**E7. 3** Write a java program to perform insertion, deletion, traversing and searching operations on HashMap and TreeMap

### HashMap

```
import java.util.HashMap;
import java.util.Random;
import java.util.Set;
import java.util.Iterator;
class MyHashMapDemo {
    public static void main(String args[]) {

        HashMap < Integer, Integer > hm1 = new HashMap < Integer, Integer > ();
        Random randKeys = new Random();
        Random randValues = new Random();
        int N = new Random().nextInt(20) + 1;

        for (int i = 1; i <= N; i++) {
            int k = randKeys.nextInt(20);
            int v = randValues.nextInt(100);
            hm1.put(k, v);
        }

        Set < Integer > s1 = hm1.keySet();
        Iterator < Integer > it1 = s1.iterator();

        System.out.println("size of HashMap: " + hm1.size());
        System.out.println("Elements of HashMap are: ");
        while (it1.hasNext()) {
            int key = it1.next();
            System.out.println(key + "->" + hm1.get(key));
        }
        it1 = s1.iterator();
        int key = 1;
        while (it1.hasNext()) {
            key = it1.next();
        }
        System.out.println("Key: " + key + " removed value is: " + hm1.remove(key));
        it1 = s1.iterator();
        System.out.println("Elements of HashMap after remove are: ");
        while (it1.hasNext()) {
            int key = it1.next();
            System.out.println(key + "->" + hm1.get(key));
        }

        System.out.println("Size of HashMap is: " + hm1.size());
```

```
if (hm1.containsKey(key))
    System.out.println(key + " is present in HashMap");
else
    System.out.println(key + " is not present in HashMap");
}
```

**OUTPUT:**

## TreeMap

```
import java.util.Iterator;
import java.util.Random;
import java.util.Set;
import java.util.TreeMap;
public class MyTreeMapDemo {
    public static void main(String args[]){
TreeMap<Integer,Integer> tm1 = new TreeMap<Integer,Integer> ();
    Random r = new Random();
    System.out.println("Treemap Empty(initially): "+tm1.isEmpty());
    for(int i = 0; i<10; i++){
        int k = r.nextInt(20)+1;
        int v = r.nextInt(100)+1;
        tm1.put(k,v);
    }
    System.out.println("Treemap Size after putting (keys,values): "+tm1.size());
    System.out.println("Values of TreeMap are: ");
    Set < Integer > ks1 = tm1.keySet();
    Iterator < Integer > it1 = ks1.iterator();
    int k=1;
    while (it1.hasNext()) {
        k = it1.next();
        System.out.println(k + "->" + tm1.get(k));
    }
    tm1.replace(k, 0);
    System.out.println("Values of TreeMapare(after replacing last key's value: ");
    Iterator < Integer > it2 = ks1.iterator();
    for(;it2.hasNext();) {
        k = it2.next();
        System.out.println(k + "->" + tm1.get(k));
    }
    tm1.remove(k);
    System.out.println("After removinf last key, The TreeMap is:"+tm1);
    System.out.println("After invoking clear() method:");
    tm1.clear();
    System.out.println("Treemap Size: "+tm1.size());
    }
}
```

**OUTPUT:**

**Viva Questions**

1. Mention the methods provided by Stack class
2. What is a priority queue in Java?
3. Differentiate between List and Set.
4. What is Map interface in Java?
5. What is the difference between HashMap and TreeMap?



## **WEEK-8**

### **Java TreeSet class**

Java TreeSet class implements the Set interface that uses a tree for storage.

It inherits AbstractSet class and implements the NavigableSet interface.

The objects of the TreeSet class are stored in ascending order.

The important points about the Java TreeSet class are:

- Java TreeSet class contains unique elements only like HashSet.
- Java TreeSet class access and retrieval times are quite fast.
- Java TreeSet class doesn't allow null element.
- Java TreeSet class is non synchronized.
- Java TreeSet class maintains ascending order.

#### **Internal Working of The TreeSet Class**

TreeSet is being implemented using a binary search tree, which is self-balancing just like a Red-Black Tree. Therefore, operations such as a search, remove, and add consume  $O(\log(N))$  time.

#### **TreeSet Class Declaration**

**public class** TreeSet<E> **extends** AbstractSet<E> **implements** NavigableSet<E>, Cloneable, Serializable

#### **Constructors of Java TreeSet Class**

Constructor	Description
TreeSet()	It is used to construct an empty tree set that will be sorted in ascending order according to the natural order of the tree set.
TreeSet(Collection<? extends E> c)	It is used to build a new tree set that contains the elements of the collection c.
TreeSet(Comparator<? super E> comparator)	It is used to construct an empty tree set that will be sorted according to given comparator.

TreeSet(SortedSet<E> s)	It is used to build a TreeSet that contains the elements of the given SortedSet.
-------------------------	--

### Methods of Java TreeSet Class

Method	Description
boolean add(E e)	It is used to add the specified element to this set if it is not already present.
boolean addAll(Collection<? extends E> c)	It is used to add all of the elements in the specified collection to this set.
E ceiling(E e)	It returns the equal or closest greatest element of the specified element from the set, or null there is no such element.
E floor(E e)	It returns the equal or closest least element of the specified element from the set, or null there is no such element.
E higher(E e)	It returns the closest greatest element of the specified element from the set, or null there is no such element.
E lower(E e)	It returns the closest least element of the specified element from the set, or null there is no such element.
E pollFirst()	It is used to retrieve and remove the lowest(first) element.
E pollLast()	It is used to retrieve and remove the highest(last) element.

<code>boolean contains(Object o)</code>	It returns true if this set contains the specified element.
<code>boolean isEmpty()</code>	It returns true if this set contains no elements.
<code>boolean remove(Object o)</code>	It is used to remove the specified element from this set if it is present.
<code>void clear()</code>	It is used to remove all of the elements from this set.
<code>E first()</code>	It returns the first (lowest) element currently in this sorted set.
<code>E last()</code>	It returns the last (highest) element currently in this sorted set.
<code>int size()</code>	It returns the number of elements in this set.

### E8.1 Java program to store and retrieve user defined class objects from TreeSet

**OBJECTIVE:**

*This lab will develop students' knowledge in /on*

- TreeSet

*Upon completion of this lab, students will be able to*

- develop programs on TreeSet

Step 1: import the packages util.TreeSet, util.Random, util.Iterator

Step 2: Create a class Student which implements Comparable with data members name and rollno

Step 3: Define a parameterized constructor and pass the parameters to data members.

Step 4: Define a method to display the student data

Step 5: Create a class MyTreeSetDemo and define the main() method

Step 6: Define a new TreeSet as:

```
TreeSet< Student > ts1 = new TreeSet< Student > ();
```

Step 7: Add the elements with add() method using Random class with for loop

Step 8: Display the elements of TreeSet using Iterator.

**E8. 1** Write a java program to store and retrieve user defined class objects from TreeSet

```
import java.util.TreeSet;
import java.util.Random;
import java.util.Iterator;

class Student {
    private String sName;
    private String sRollNo;
    Student() {}
    Student(String Name, String RollNo) {
        sName = Name;
        sRollNo = RollNo;
    }
    void dispStudentData() {
        System.out.println("Name: " + sName + " RollNo: " + sRollNo);
    }
}

public class MyTreeSetDemo {
    public static void main(String args[]) {
        TreeSet< Student > ts1 = new TreeSet< Student > ();
        int N = new Random().nextInt(10) + 1;
        for (int i = 1; i <= N; i++) {
            Random rand = new Random();
            int n = rand.nextInt(100);
            ts1.add(new Student("Name" + n, "RollNo" + n));
        }
        Iterator < Student > it1 = ts1.iterator();
        System.out.println("Data present in TreeSet are: ");
        while (it1.hasNext()) {
            it1.next().dispStudentData();
        }
    }
}
```

**OUTPUT:**

## E8.2 Java program to read a set of values and display the count of occurrences of each number using collection concept

### OBJECTIVE:

*This lab will develop students' knowledge in /on*

- Operations on Collections

*Upon completion of this lab, students will be able to*

- develop programs to perform different operations on the given collections.

Step 1: import the packages util.TreeMap, util.Random, util.Iterator, util.Set

Step 2: Create a class MyTreeMapCountDemo and define the main() method.

Step 3: Declare an array of random size and add elements into the array.

Step 4: Create a new tree map as:

```
TreeMap< Integer, Integer > tm1 = new TreeMap< Integer, Integer > ();
```

Step 5: Read the values of array to tree map as keys and their corresponding count of occurrences as values

Step 6: While reading the values from array if the tree contains the key already increment the count and replace the value of that key and if the tree does not contain that key put it to the tree and its corresponding value is 1.

Step 7: Display the values of the array using for loop

Step 8: Display the keys & values of tree map using Iterator.

**E8. 2** Write a java program to read a set of values and display the count of occurrences of each number using collection concept

```
import java.util.TreeMap;
import java.util.Random;
import java.util.Iterator;
import java.util.Set;

public class MyTreeMapCountDemo {
    public static void main(String args[]) {

        int N = new Random().nextInt(100) + 1;
        int a[] = new int[N];
        Random rand = new Random();
        for (int i = 0; i < N; i++) {
            a[i] = rand.nextInt(20);
        }

        TreeMap< Integer, Integer > tm1 = new TreeMap< Integer, Integer > ();
        for (int i = 0; i < N; i++) {
            int k = a[i];
            if (tm1.containsKey(k)) {
                int v = tm1.get(k);
                ++v;
                tm1.replace(k, v);
            }
            else
                tm1.put(k, 1);
        }
        System.out.println("Array values are: ");
        for (int i = 0; i < N; i++)
            System.out.print(a[i] + " ");
        System.out.println();

        System.out.println("Values of TreeSet are: ");
        Set < Integer > ks1 = tm1.keySet();
        Iterator < Integer > it1 = ks1.iterator();
        while (it1.hasNext()) {
            int k = it1.next();
            System.out.println(k + "->" + tm1.get(k));
        }
    }
}
```

**OUTPUT:**

**Viva Questions**

1. Mention the constructors of TreeSet class.
2. Differentiate between HashSet and TreeSet.
3. What are the various ways to iterate over a list?
4. How to iterate map?



## **WEEK-9**

### **E9.1 Java program to display ArrayList values in sorted order**

**OBJECTIVE:**

*This lab will develop students' knowledge in /on*

- Sorting the ArrayList

*Upon completion of this lab, students will be able to*

- develop programs for Sorting the values of ArrayList

Step 1: import the packages util.ArrayList, util.Iterator, util.Collections

Step 2: Create a class and define main() method

Step 3: Create a new ArrayList and values to the list.

Step 4: Display the ArrayList using Iterator

Step 5: Sort the elements of using Collections.sort() method

Step 6: Display the sorted array list using Iterator.

**E9. 1** Write a java program to display ArrayList values in sorted order

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Collections;
public class MyArrayListIntegerSort {
    public static void main(String args[]) {
        ArrayList< Integer > o1 = new ArrayList< Integer > ();

        o1.add(10);
        o1.add(20);
        o1.add(1);
        o1.add(15);

        Iterator it1 = o1.iterator();
        System.out.println("ArrayList values before sort: ");
        while (it1.hasNext())
            System.out.print(it1.next() + " ");

        Collections.sort(o1);
        System.out.println();

        System.out.println("ArrayList values after sort: ");
        it1 = o1.iterator();
        while (it1.hasNext())
            System.out.print(it1.next() + " ");
        }
    }
```

**OUTPUT:**

## Java Comparable interface

The Comparable interface is used to compare an object of the same class with an instance of that class, it provides ordering of data for objects of the user-defined class.

The class has to implement the `java.lang.Comparable` interface to compare its instance and contains only one method named `compareTo(Object)`.

It provides a single sorting sequence only, i.e., you can sort the elements on the basis of single data member only.

### **compareTo(Object obj) method**

**public int compareTo(Object obj):** It is used to compare the current object with the specified object. It returns

- positive integer, if the current object is greater than the specified object.
- negative integer, if the current object is less than the specified object.
- zero, if the current object is equal to the specified object.

## Java Comparator interface

Java Comparator interface is used to order the objects of a user-defined class.

This interface is found in `java.util` package and contains 2 methods `compare(Object obj1, Object obj2)` and `equals(Object element)`.

It provides multiple sorting sequences, i.e., you can sort the elements on the basis of any data member, for example, rollno, name, age or anything else.

### Methods of Java Comparator Interface

Method	Description
<code>public int compare(Object obj1, Object obj2)</code>	It compares the first object with the second object.
<code>public boolean equals(Object obj)</code>	It is used to compare the current object with the specified object.

## E9.2 Java program to demonstrate Comparable interface for sorting user defined data type

### OBJECTIVE:

*This lab will develop students' knowledge in /on*

- Comparable interface for sorting

*Upon completion of this lab, students will be able to*

- develop programs using Comparable interface for sorting

Step 1: import the packages util.ArrayList, util.Iterator, util.Collections

Step 2: Create a class student implements Comparable <student>

Step 3: Declare the data members name and rollno , define a parameterised constructor and pass the parameters to the data members.

Step 4: Define the method to display the details of the student

Step 5: Define the abstract method public int compareTo(Object o) which compares the names of the student class.

Step 6: Create a class and define main() method

Step 7: Create an ArrayList as:

```
ArrayList< Student > o1 = new ArrayList< Student > ();
```

Step 8: Display the ArrayList using Iterator.

Step 9: Sort the ArrayList using Collections.sort()

Step 10: Display the sorted ArrayList using Iterator.

**E9. 2** Write a java program to demonstrate Comparable interface for sorting user defined data type

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Collections;

class Student implements Comparable < Student > {
    private String sName;
    private String sRollNo;

    Student(String Name, String RollNo) {
        sName = Name;
        sRollNo = RollNo;
    }
    void dispData() {
        System.out.println("Name: " + sName + " RollNo: " + sRollNo);
    }
    public int compareTo(Student s) {
        return sRollNo.compareTo(s.sRollNo);
    }
}

public class MyArrayListUserDefinedSort {
    public static void main(String args[]) {
        ArrayList< Student > o1 = new ArrayList< Student > ();
        o1.add(new Student("Name" + "hhh", "RollNo" + 20));
        o1.add(new Student("Name" + "mmm", "RollNo" + 1));
        o1.add(new Student("Name" + "aaa", "RollNo" + 15));
        o1.add(new Student("Name" + "rrr", "RollNo" + 10));

        Iterator < Student > it1 = o1.iterator();
        System.out.println("ArrayList values before sort: ");
        while (it1.hasNext()){
            it1.next().dispData();
        }
        Collections.sort(o1);

        System.out.println();
        System.out.println("ArrayList values after sort: ");
        it1 = o1.iterator();
        while (it1.hasNext())
            it1.next().dispData();
    }
}
```

**OUTPUT:**

### E9.3 Java program to demonstrate Comparator interface for sorting user defined data type

**OBJECTIVE:**

*This lab will develop students' knowledge in /on*

- Comparator interface for sorting

*Upon completion of this lab, students will be able to*

- develop programs using Comparator interface for sorting

Step 1: import the packages util.ArrayList, util.Iterator, util.Collections, util.Comparator

Step 2: Create a class student implements Comparator <student>

Step 3: Declare the data members name and rollno , define a parameterised constructor and pass the parameters to the data members.

Step 4: Define the method to display the details of the student

Step 5: Define the abstract method public int compare(Object o) which compares the names of the student class.

Step 6: Create a class and define main() method

Step 7: Create an ArrayList as:

```
ArrayList< Student > o1 = new ArrayList< Student > ();
```

Step 8: Display the ArrayList using Iterator.

Step 9: Sort the ArrayList using Collections.sort()

Step 10: Display the sorted ArrayList using Iterator.

**E9. 3** Write a java program to demonstrate Comparator interface for sorting user defined data type

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Collections;
import java.util.Comparator;

class Student implements Comparator < Student > {
    private String sName;
    private String sRollNo;
    Student() {}
    Student(String Name, String RollNo) {
        sName = Name;
        sRollNo = RollNo;
    }
    void dispData() {
        System.out.println("Name: " + sName + " RollNo: " + sRollNo);
    }
    public int compare(Student s1, Student s2) {
        return s1.sName.compareTo(s2.sName);
    }
}

public class MyArrayListUserDefinedComparatorSort {
    public static void main(String args[]) {
        ArrayList< Student > o1 = new ArrayList<> ();
        o1.add(new Student("Name" + 10, "RollNo" + 10));
        o1.add(new Student("Name" + 20, "RollNo" + 20));
        o1.add(new Student("Name" + 1, "RollNo" + 1));
        o1.add(new Student("Name" + 15, "RollNo" + 15));
        Iterator < Student > it1 = o1.iterator();
        System.out.println("ArrayList values before sort: ");
        while (it1.hasNext()){
            it1.next().dispData();
        }
        Collections.sort(o1, new Student());
        System.out.println();
        System.out.println("ArrayList values after sort: ");
        it1 = o1.iterator();
        while (it1.hasNext())
            it1.next().dispData();
    }
}
```



**OUTPUT:**

**Viva Questions**

1. What Is Java Collections Sort?
2. Write the different methods to implement collections sort.
3. Write about Comparator interface and its methods.
4. Differentiate between Comparable and Comparator.

## **WEEK-10**

### **JUnit Testing Framework for Java**

It is an *open-source testing framework* for java programmers. The java programmer can create test cases and test his/her own code.

It is one of the unit testing framework. Current version is junit 4.

Java Developers use this framework to write and execute automated tests.

#### **What is Unit Testing?**

Unit testing, as the name suggests, refers to the testing of small segments of code.

Here, a unit indicates the smallest bit of code that can be fetched out of the system. This small bit can be a line of the code, a method, or a class.

The smaller the chunk of code, the better it is, as smaller chunks will tend to run faster. And this provides a better insight into the code and its performance.

To perform unit testing, we need to create test cases. The unit test case is a code which ensures that the program logic works as expected.

The org.junit package contains many interfaces and classes for junit testing such as Assert, Test, Before, After etc.

#### **Types of unit testing**

There are two ways to perform unit testing: 1) manual testing 2) automated testing.

##### **1) Manual Testing**

If you execute the test cases manually without any tool support, it is known as manual testing. It is time consuming and less reliable.

##### **2) Automated Testing**

If you execute the test cases by tool support, it is known as automated testing. It is fast and more reliable.

#### **Features of JUnit**

- Open Source Network:

JUnit is an open-source network that enables developers to write codes fast and with better quality.

- Provides Annotations:

It provides several annotations to identify test methods.

- Provides Assertions:

There are assertions to test expected results.

- Provides Test Runners:

JUnit has test runners to run tests.

- Improves Code Quality:

JUnit is the most popular testing framework for efficient testing. It allows faster code writing, which results in an increase in the code's quality.

- Automated Test Running:

The test results do not require manual checking. All the tests run automatically on JUnit, the results obtained again automatically checked, and it provides feedback.

- Easily interpretable results:

The test results are represented interactively by showing test progress in a bar, thus making them easily interpretable.

### **Annotations for Junit testing**

The Junit 4.x framework is annotation based, the annotations that can be used while writing the test cases are

**@Test** : annotation specifies that method is the test method.

**@Test(timeout=1000)** : annotation specifies that method will be failed if it takes longer than 1000 milliseconds (1 second).

**@BeforeClass** : annotation specifies that method will be invoked only once, before starting all the tests.

**@Before** : annotation specifies that method will be invoked before each test.

**@After** : annotation specifies that method will be invoked after each test.

**@AfterClass** : annotation specifies that method will be invoked only once, after finishing all the tests.

Assert is a method useful in determining Pass or Fail status of a test case, The assert methods are provided by the class `org.junit.Assert` which extends `java.lang.Object` class.

There are various types of assertions like Boolean, Null, Identical etc.

Junit provides a class named `Assert`, which provides a bunch of assertion methods useful in writing test cases and to detect test failure.

## **JUnit Assert methods**

### **Boolean**

If you want to test the boolean conditions (true or false), you can use following assert methods

1. **`assertTrue(condition)`**
2. **`assertFalse(condition)`**

Here the condition is a boolean value.

### **Null object**

If you want to check the initial value of an object/variable, you have the following methods:

1. **`assertNull(object)`**
2. **`assertNotNull(object)`**

Here object is Java object **e.g.** `assertNull(actual);`

### **Identical**

If you want to check whether the objects are identical (i.e. comparing two references to the same java object), or different.

1. **`assertSame(expected, actual)`**, It will return true if **`expected == actual`**
2. **`assertNotSame(expected, actual)`**

### **Assert Equals**

If you want to test equality of two objects, you have the following methods

- **`assertEquals(expected, actual)`**

It will return true if: **`expected.equals( actual )`** returns true.

### **Assert Array Equals**

If you want to test equality of arrays, you have the following methods as given below:

- **assertArrayEquals(expected, actual)**

Above method must be used if arrays have the same length, for each valid value for **i**, you can check it as given below:

- **assertEquals(expected[i],actual[i])**
- **assertArrayEquals(expected[i],actual[i])**

### **Fail Message**

If you want to throw any assertion error, you have **fail()** that always results in a fail verdict.

- **Fail(message);**

You can have assertion method with an additional **String** parameter as the first parameter. This string will be appended in the failure message if the assertion fails. E.g. **fail( message )** can be written as

- **assertEquals( message, expected, actual)**

## Setup JUnit Environment

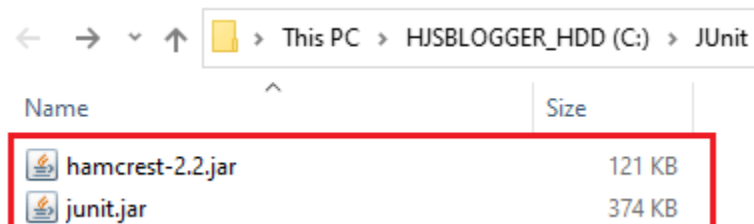
Here we install the JUnit 4. Follow the below-mentioned steps to download and install JUnit, the most important step of JUnit environment setup.

**Step 1** – Visit the [JUnit official site](https://github.com/junit-team/junit4/wiki/Download-and-Install) and click on ‘Download and Install’

The screenshot shows the GitHub Wiki page for JUnit 4. The page title is 'Download and Install' by Marc Philipp. It lists two options for downloading and installing JUnit: 'Plain-old JAR' and 'Maven'. Under 'Plain-old JAR', it instructs to download 'junit.jar' and 'hamcrest-core.jar'. The 'junit.jar' file is highlighted with a red box. On the right side, there is a sidebar with a 'Pages' section containing a list of links: Home, 'Enclosed' test runner example, Aggregating tests in suites, Assertions, Assertions CN, and Assumptions with assume.

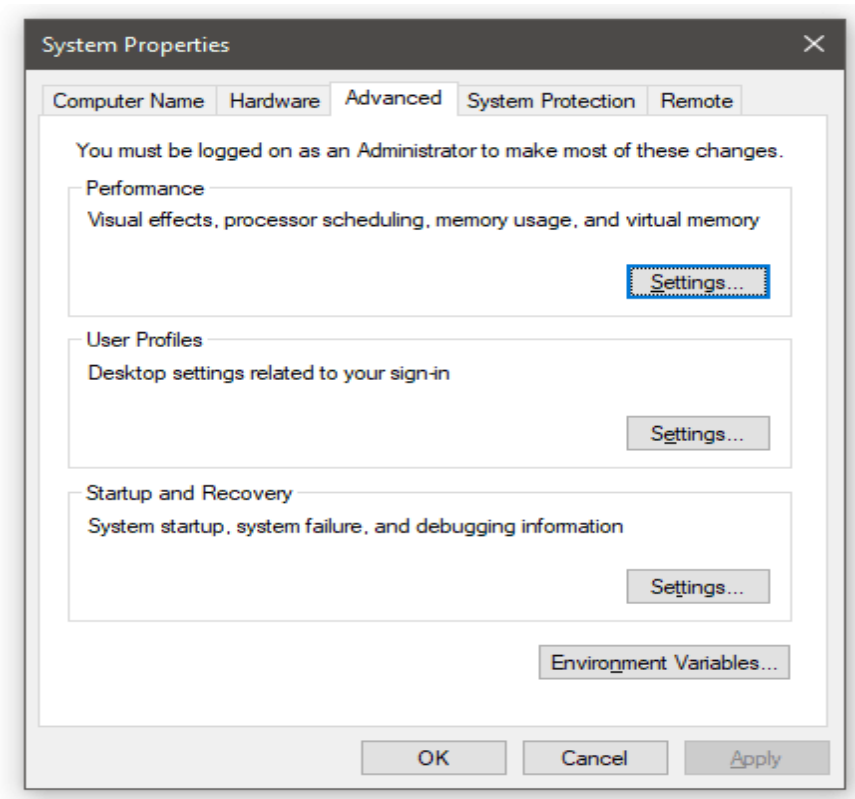
Download junit.jar and hamcrest-core.jar

Place the JUnit and Hamcrest jar files in the **C:\JUnit** folder.



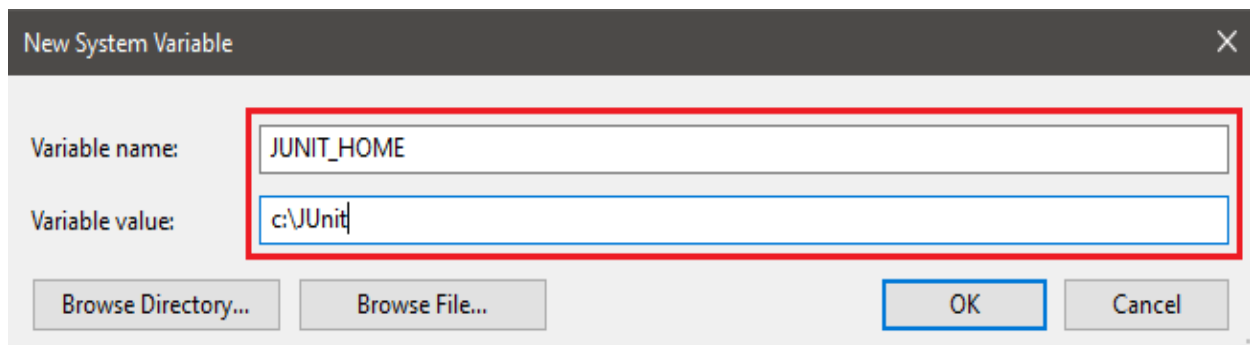
## Setup Environment Variables for JUnit

**Step 1** – Create a new System Variable (under the ‘Environment variables’ section).

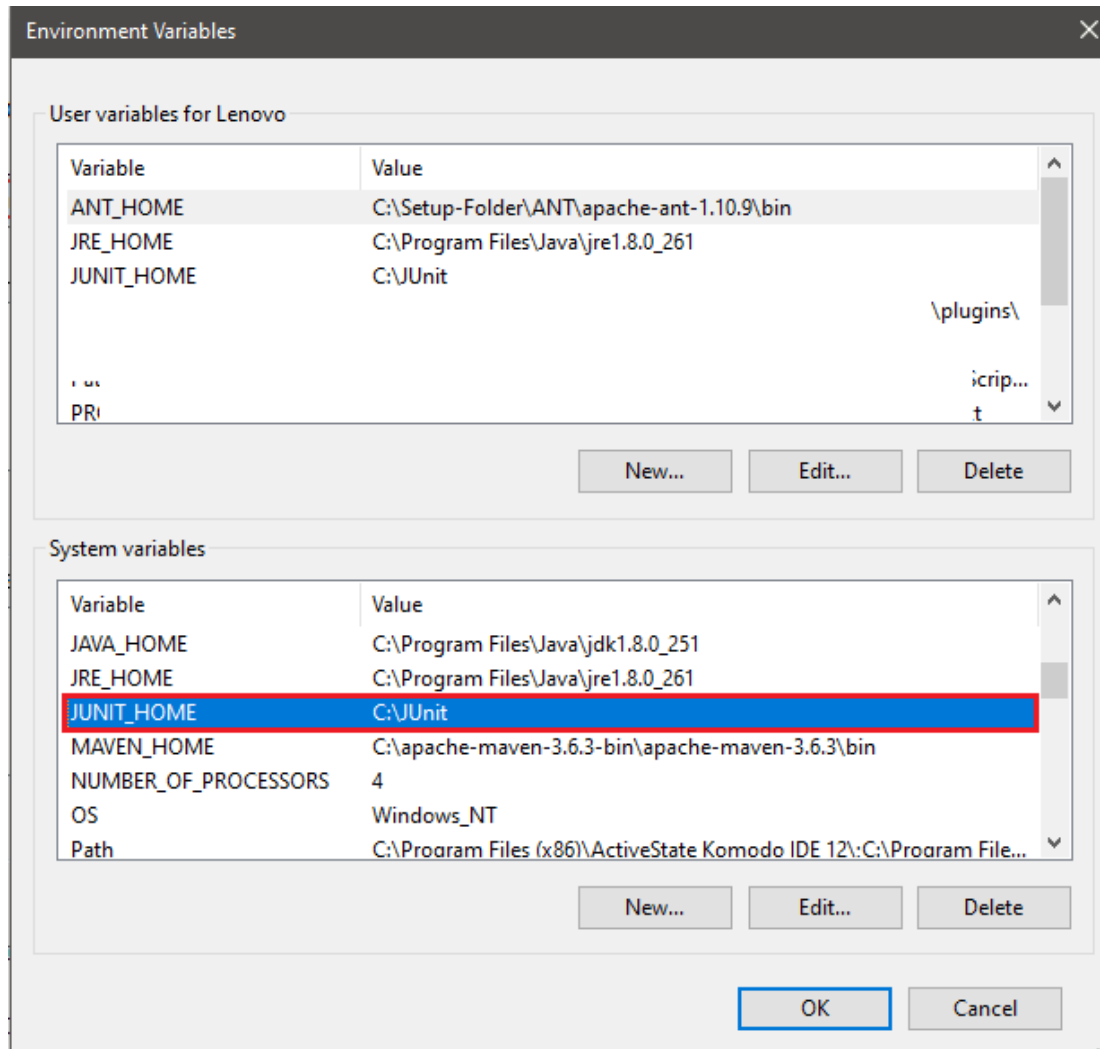


Under the new System Variable window, enter the following details:

- Variable Name – JUNIT\_HOME
- Variable Value – Path to JUnit.jar (i.e., C:\JUnit in our case)



Click on 'OK' to set the newly created environment variable. As seen below, JUNIT\_HOME is visible in the System variable section.



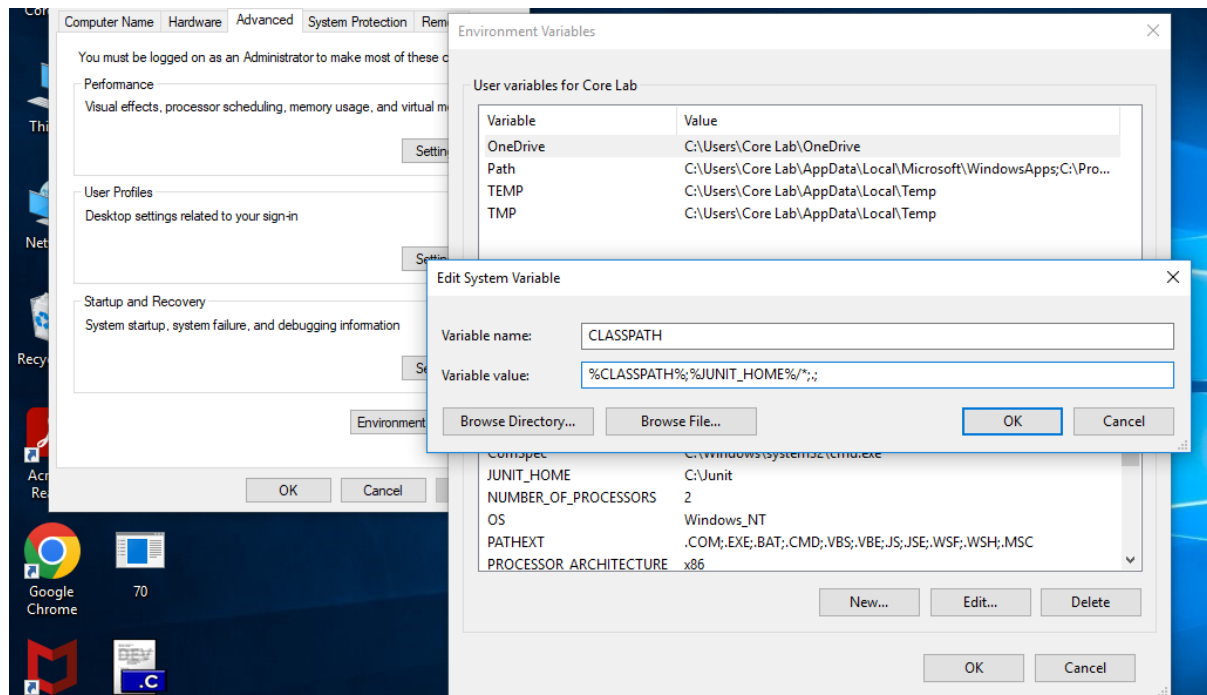
### Setup CLASSPATH Variable for JUnit

Step 1 – In this step, we modify the CLASSPATH environment variable (if it exists) by appending the JUnit jar's location. If the CLASSPATH environment variable is not available in the system, create a new System Variable with the following details:

- Variable Name – CLASSPATH
- Variable Value – %CLASSPATH%;%JUNIT\_HOME%\\*;.;

Class path entries can contain the base name wildcard character (\*), which is considered equivalent to specifying a list of all of the files in the directory with the extension .jar or .JAR.





Click on 'OK' to create (or modify) the environment variable CLASSPATH. As seen below, the system variable CLASSPATH is successfully created (or modified) in the system.

**E10. 1 Java program to test simple arithmetic operations of Calculator class using JUnit concept.**

**E10. 2 Java program to demonstrate different Assert methods and annotations**

**OBJECTIVE:**

*This lab will develop students' knowledge in /on*

- Unit Testing with JUnit Framework

*Upon completion of this lab, students will be able to*

- perform unit testing using JUnit 4

**Environment:**

The below environment is used to run the tests in this example.

- JUnit jar
- Hamcrest core jar
- Java Development Kit (JDK)
- Windows 10 OS

**Step-1:** write a program to perform simple calculations as follows:

```
public class Calculator {
public Calculator() {
}
// Sum method.
Public int add(int a, int b) {
return a + b;
}
//Subtract method.
Public int subtract(int a, int b) {
return a - b;
}
// Multiply method.
public long multiply(int a, int b) {
return a * b;
}
}
```

**Step-2:** Save it as Calculator.java

**Step-3:** compile the program as **javac Calculator.java**

Copy the Calculator.class file to the folder C:\JUnit

**Step-4:** Create another file and write the code to test the Calculator program using org.junit, annotations and assert statements.

```
import org.junit.Test;
import org.junit.Assert;
import org.junit.Before;
import org.junit.After;

public class CalculatorTest {

    private Calculator objCalcUnderTest;

    @Before
    public void setUp() {
        objCalcUnderTest = new Calculator();
    }

    @Test
    public void testAdd() {
        int a = 15;
        int b = 20;
        int expectedResult = 35;
        long result = objCalcUnderTest.add(a, b);
        Assert.assertEquals(expectedResult, result);
    }

    @Test
    public void testSubtract() {
        int a = 25;
        int b = 20;
        int expectedResult = 5;
        long result = objCalcUnderTest.subtract(a, b);
        Assert.assertEquals(expectedResult, result);
    }

    @Test
    public void testMultiply() {
        int a = 10;
        int b = 25;
        long expectedResult = 250;
        long result = objCalcUnderTest.multiply(a, b);
        Assert.assertEquals(expectedResult, result);
    }
}
```

```
@After
public void tearDown() {
    objCalcUnderTest = null;
}
}
```

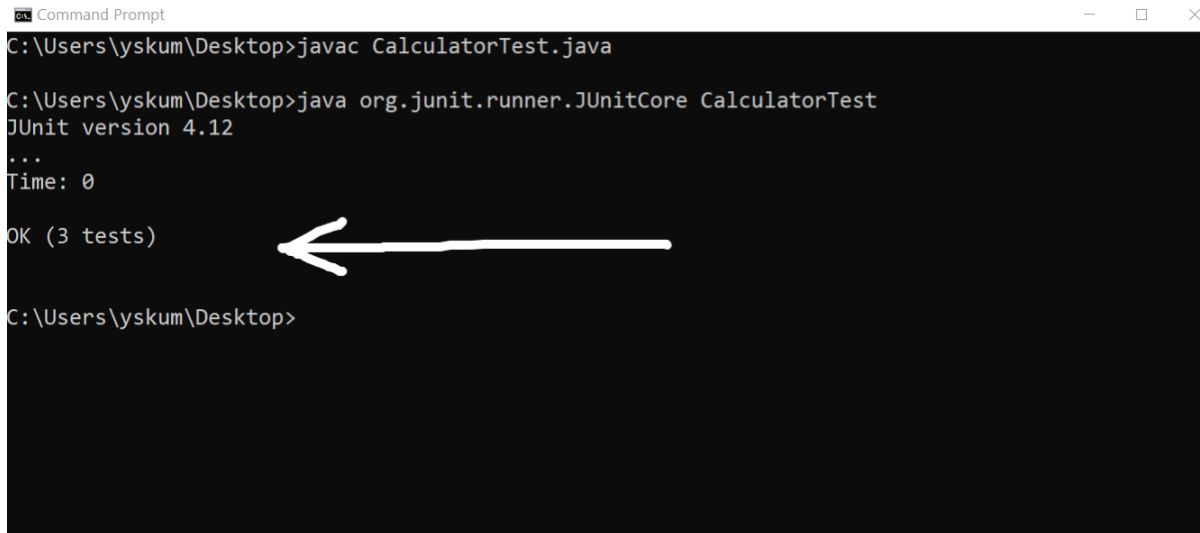
**Step-5:** Save it as CalculatorTest.java

Compile the program as **javac CalculatorTest.java**

**Step-6:** run the tests using the following command.

**java org.junit.runner.JUnitCore CalculatorTest**

**Output:**



```
Command Prompt
C:\Users\yskum\Desktop>javac CalculatorTest.java

C:\Users\yskum\Desktop>java org.junit.runner.JUnitCore CalculatorTest
JUnit version 4.12
...
Time: 0
OK (3 tests)
```

The dots indicate the number of tests in your junit suite. OK result is outputted when all the tests pass.

### **Viva Questions**

1. What is JUnit? What are the features of JUnit?
2. What are the important JUnit annotations?
3. What are the advantages of automated testing?
4. What are the JUnit Assert Methods?
5. How do you test a "private" method?



## **WEEK-11**

### **Java Lambda Expressions**

Lambda expression is a new and important feature of Java which was included in Java SE 8.

An interface with single abstract method is called functional interface. Lambda expressions basically express instances of functional interfaces.

Lambda expressions implement the only abstract function and therefore implement functional interfaces. It is very useful in collection library. It helps to iterate, filter and extract data from collection.

Java lambda expression is treated as a function, so compiler does not create .class file.

Lambda expressions provide below functionalities.

- Enable to treat functionality as a method argument, or code as data.
- A function that can be created without belonging to any class.
- A lambda expression can be passed around as if it was an object and executed on demand.

#### **Java Lambda Expression Syntax**

(argument-list) -> {body}

Java lambda expression is consisted of three components.

**1) Argument-list:** It can be empty or non-empty as well.

**2) Arrow-token:** It is used to link arguments-list and body of expression.

**3) Body:** It contains expressions and statements for lambda expression.

#### **No Parameter Syntax**

```
() -> {  
  
    //Body of no parameter lambda  
  
}
```

#### **One Parameter Syntax**

```
(p1) -> {
```

```
        //Body of single parameter lambda  
    }
```

### **Two Parameter Syntax**

```
(p1,p2) -> {  
    //Body of multiple parameter lambda  
}
```

### **A Java program to demonstrate simple lambda expressions**

```
import java.util.ArrayList;  
class Test  
{  
    public static void main(String args[])  
    {  
        // Creating an ArrayList with elements  
        // {1, 2, 3, 4}  
        ArrayList<Integer> arrL = new ArrayList<Integer>();  
        arrL.add(1);  
        arrL.add(2);  
        arrL.add(3);  
        arrL.add(4);  
  
        // Using lambda expression to print all elements  
        // of arrL  
        arrL.forEach(n -> System.out.println(n));  
  
        // Using lambda expression to print even elements  
        // of arrL  
        arrL.forEach(n -> { if (n%2 == 0) System.out.println(n); });  
    }  
}
```

**E11. 1 Java program to demonstrate lambda expression with no parameter.**

**E11. 2 Java program to demonstrate lambda expression with single and multiple parameters.**

**OBJECTIVE:**

*This lab will develop students' knowledge in /on*

- Lambda expressions

*Upon completion of this lab, students will be able to*

- develop programs using Lambda expressions

```
interface If1
```

```
{    double fun1();  
  
}
```

```
interface If2
```

```
{    long fact(int n);  
  
}
```

```
interface If3
```

```
{    long mult(int x, int y);  
  
}
```

```
class LambdaExpresssionParametersDemo
```

```
{  
    public static void main(String args[])  
    {  
        If1 o1 = () -> 10.2;  
  
        System.out.println("Lambda expression without parameters: " + o1.fun1());  
  
        If2 o2 = (int n) ->  
        {  
            if(n < 0)  
                return -1;  
            else if (n == 0)  
            {  
                return 1;  
            }  
            else  
            {  

```



```
        long prod = 1;
        for(int i=1; i<=n; i++)
            prod *= i;
        return prod;
    }
};

System.out.println("Lambda expression with one parameter:");
System.out.println("Factorial of 4: " + o2.fact(4));
System.out.println("Factorial of 5: " + o2.fact(5));

If3 o3 = (int x, int y) ->
{
    return x*y;
};

System.out.println("lambda expression with more than one parameter: " );
System.out.println("product of 4 and 5 is: " + o3.mult(4,5));
    }
}
```

**OUTPUT:**

### E11. 3 Java program to iterate the List and Map using lambda expressions.

**OBJECTIVE:**

*This lab will develop students' knowledge in /on*

- Lambda expressions

*Upon completion of this lab, students will be able to*

- develop programs using Lambda expressions on list and map

```
import java.util.ArrayList;
import java.util.HashMap;

class LambdaExpressionListHashandMapDisplay
{
    public static void main(String args[])
    {
        ArrayList<Integer> o1 = new ArrayList<Integer>();

        o1.add(10);
        o1.add(20);
        o1.add(30);
        System.out.println("Display of List Elements using Lambda Expression: ");
        o1.forEach((Integer v) -> System.out.print(v + " "));

        HashMap <String, Integer> o2 = new HashMap <String, Integer> ();
        System.out.println();

        o2.put("CSE1", 90);
        o2.put("CSE2", 80);
        o2.put("CSE3", 70);
        System.out.println("Display of Hash Elements using Lambda Expression: ");
        o2.forEach((String k, Integer v) -> System.out.println(k + "->" + v));
    }
}
```

**OUTPUT:**

**E11. 4 Create two threads using lambda expressions, where one thread displays even numbers for every half second and the other thread displays odd numbers for every second.**

**OBJECTIVE:**

*This lab will develop students' knowledge in /on*

- Lambda expressions

*Upon completion of this lab, students will be able to*

- develop programs using Lambda expressions with threads

```
class LambdaExpresssionThreads
{
    public static void main(String args[])
    {
        Runnable o1 = () ->
        {
            for(int i=1; ; i++)
            {
                if(i%2 == 0)
                    System.out.println("i1 = " + i);
                try
                {
                    Thread.currentThread().sleep(500);
                }
                catch(Exception e)
                {
                }
            }
        };

        Runnable o2 = () ->
        {
            for(int i=1; ; i++)
            {
                if(i%2 != 0)
                    System.out.println("i2 = " + i);
                try
                {
                    Thread.currentThread().sleep(1000);
                }
                catch(Exception e)
                {
                }
            }
        }
    }
}
```

```
        };  
        new Thread(o1).start();  
        new Thread(o2).start();  
    }  
}
```

**OUTPUT:**

**Viva Questions**

1. What Is a Functional Interface?
2. What Is a Lambda Expression and What Is It Used For?
3. Explain Lambda Expression Syntax
4. What is Runnable Interface?

## **WEEK-12**

### **Stream In Java**

Java provides a new additional package in Java 8 called java.util.stream. This package consists of classes, interfaces and enum which allow functional-style operations on the elements.

The Stream API is used to process collections of objects. A stream is a sequence of objects that supports various methods which can be pipelined to produce the desired result.

The features of Java stream are –

- A stream is not a data structure instead it takes input from the Collections, Arrays or I/O channels.
- Stream does not store elements. It simply conveys elements from a source such as a data structure, an array, or an I/O channel, through a pipeline of computational operations.
- Streams don't change the original data structure, they only provide the result as per the pipelined methods.
- Each intermediate operation is lazily executed and returns a stream as a result, hence various intermediate operations can be pipelined. Terminal operations mark the end of the stream and return the result.
- The elements of a stream are only visited once during the life of a stream. Like an Iterator, a new stream must be generated to revisit the same elements of the source.

#### **Different Operations On Streams-**

##### **Intermediate Operations:**

1. **map:** The map method is used to returns a stream consisting of the results of applying the given function to the elements of this stream.

##### **Example:**

```
List number = Arrays.asList(2,3,4,5);  
ArrayList square = number.stream().filter(x->((x%2)!=0))  
                             .map(x->x*x)  
                             .collect(Collectors.toList());
```

2. **filter:** The filter method is used to select elements as per the Predicate passed as argument.

**Example:**

```
List names = Arrays.asList("Reflection","Collection","Stream");  
List result = names.stream().filter(s->s.startsWith("S")).collect(Collectors.toList());
```

3. **sorted:** The sorted method is used to sort the stream.

**Example:**

```
List names = Arrays.asList("Reflection","Collection","Stream");  
List result = names.stream().sorted().collect(Collectors.toList());
```

4. **distinct:** the distinct() method is used for finding the distinct elements by field from a Stream. To find the items that are distinct by multiple fields.

**Example:**

```
Collection<String> list = Arrays.asList("A", "B", "C", "D", "A", "B", "C");  
  
List<String> distinctElements = list.stream().distinct().collect(Collectors.toList());  
  
System.out.println(distinctElements);
```

5. **limit:** The limit(n) method is an intermediate operation that returns a stream not longer than the requested size, the *n* parameter can't be negative.

**Example:**

```
Stream.of(1, 2, 3, 4, 5, 6, 7, 8, 9, 10).filter(i -> i % 2 == 0).limit(2)  
  
    .forEach(i -> System.out.print(i + " "));
```

**Terminal Operations:**

1. **collect:** The collect method is used to return the result of the intermediate operations performed on the stream.

**Example:**

```
List number = Arrays.asList(2,3,4,5,3);  
Set square = number.stream().map(x->x*x).collect(Collectors.toSet());
```

2. **forEach:** The forEach method is used to iterate through every element of the stream.

**Example:**

```
List number = Arrays.asList(2,3,4,5);  
number.stream().map(x->x*x).forEach(y->System.out.println(y));
```

3. **count:** long count() returns the count of elements in the stream.

**Example:**

```
List<Integer> list = Arrays.asList(0, 2, 4, 6, 8, 10, 12);  
long total = list.stream().count();  
System.out.println(total);
```

4. **reduce:** The reduce method is used to reduce the elements of a stream to a single value. The reduce method takes a BinaryOperator as a parameter.

**E12. 1 Java program to demonstrate following methods using streams on a List a) filter  
b) sorted c) distinct d) limit e) count**

**OBJECTIVE:**

*This lab will develop students' knowledge in /on*

- Stream API

*Upon completion of this lab, students will be able to*

- develop programs on stream API using different operations.

```
import java.util.Arrays;
import java.util.IntSummaryStatistics;
import java.util.List;
import java.util.stream.Collectors;
public class JavaStreamsExample
{

    public static void main(String args[]) {

// Count the empty strings

        List<String> strList = Arrays.asList("abc", "", "bcd", "", "defg", "jk");

        long count = strList.stream() .filter(x -> x.isEmpty()) .count(); //2

        System.out.printf("List %s has %d empty strings %n", strList, count);

// Count String with length more than 3

        long num = strList.stream() .filter(x -> x.length()> 3) .count(); //1

        System.out.printf("List %s has %d strings of length more than 3 %n", strList,
num);

// Remove all empty Strings from List

        List<String> filtered = strList.stream() .filter(x -> !x.isEmpty())

                                .collect(Collectors.toList());

        System.out.printf("Original List : %s, List without Empty Strings : %s %n",
strList, filtered);

// Convert String to Uppercase and join them using coma

        List<String> G7 = Arrays.asList("USA","Japan","France","Germany","Italy","U
K","Canada");
```



```
String G7Countries = G7.stream() .map(x -> x.toUpperCase())
                                .collect(Collectors.joining(", "));

System.out.println(G7Countries);

// Create List of square of all distinct numbers

List<Integer> numbers = Arrays.asList(9, 10, 3, 4, 7, 3, 4);
List<Integer> distinct = numbers.stream() .map( i -> i*i).distinct()
                                .collect(Collectors.toList());

System.out.printf("Original List : %s, Square Without duplicates : %s %n",
numbers, distinct);

//Get count, min, max, sum, and average for numbers

List<Integer> primes = Arrays.asList(2, 3, 5, 7, 11, 13, 17, 19, 23, 29);
IntSummaryStatistics stats = primes.stream() .mapToInt((x) -> x)
                                .summaryStatistics();

System.out.println("Highest prime number in List : " + stats.getMax());
System.out.println("Lowest prime number in List : " + stats.getMin());
System.out.println("Sum of all prime numbers : " + stats.getSum());
System.out.println("Average of all prime numbers : " + stats.getAverage());

    }
}
```

**OUTPUT:**

**E12. 2 Java program to read a string and collect upper case characters, lower case characters & digits into each individual ArrayList using streams and display them.**

**OBJECTIVE:**

*This lab will develop students' knowledge in /on*

- Stream API

*Upon completion of this lab, students will be able to*

- develop programs on stream API using different operations.

```
import java.util.*;
import java.util.stream.Collectors;

public class prg1 {
    public static List<Character> convertStringToCharList(String str)
    {
        List<Character> chars = str .chars().mapToObj(e -> (char)e).collect(Collectors.toList());
        return chars;
    }

    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        String str=sc.nextLine();
        List<Character> chars= convertStringToCharList(str);
        ArrayList<Character> upperChars= chars.stream().filter(x -> Character.isUpperCase(x))
                                                .collect(Collectors.toCollection(ArrayList::new));
        ArrayList<Character> lowerChars= chars.stream().filter(x -> Character.isLowerCase(x))
                                                .collect(Collectors.toCollection(ArrayList::new));
        ArrayList<Character> digitss= chars.stream().filter(x -> Character.isDigit(x))
                                                .collect(Collectors.toCollection(ArrayList::new));

        System.out.println("Upper Case Characters in the given string are: "+upperChars);
        System.out.println("Lower Case Characters in the given string are: "+lowerChars);
        System.out.println("Digits in the given string are: "+digitss);
    }
}
```

**OUTPUT:**

**Viva Questions**

1. What is a Stream API? Why do we require the Stream API?
2. What is the difference between Collection and Stream?
3. What are the most commonly used Intermediate operations?
4. What are the most common type of Terminal operations?

