Course File On
Java Programming

By

Sana Mateen

Assistant Professor

Computer Science and Engineering

Global Institute Of Engineering and Technology

2019-2020

HOD
CSE

Principal
GIET

GLOBAL INSTITUTE OF ENGINEERING AND TECHNOLOGY

COURSE FILE

Subject : **JAVA PROGRAMMING**

Name : **SANA MATEEN**

Designation : **Assistant Professor**

Regulation/Course Code: R 18 / CS405PC

Year / Semester : **II / II**

Department : **CSE**

Academic Year : **2019-20**

Contents

PART-2

S.N.	Topics
1	Attendance Register/Teacher Log Book
2	Time Table
3	Academic Calendar
4	Continuous Evaluation-marks (Test, Assignments etc)
5	Status Request internal Exams and Syllabus coverage
6	Teaching Diary/Daily Delivery Record
7	Continuous Evaluation – MID marks
8	Assignment Evaluation- marks /Grades
9	Special Descriptive Tests Marks
10	Sample students descriptive answer sheets
11	Sample students assignment sheets

2. Vision

The vision of the department is to produce professional computer science engineers who can meet the expectations of the globe and contributes to the advancements of engineering and technology which involves creativity and innovation by providing an excellent learning environment with best quality facilities.

3. Mission

1. To provide the students with a practical and qualitative education in a modern technical environment that will help to improve their abilities and skills in solving programming problems effectively with different ideas and knowledge.

2. To infuse the scientific temper in the students towards the research and development in Computer Science and Engineering trends.

3. To mould the graduates to assume leadership roles by possessing good communication skills, an appreciation for their social and ethical responsibility in a global setting, and the ability to work effectively as team members.

4. Program Educational Objectives, Program Outcomes and Program Specific Outcomes

Program Educational Objectives

PEO 1. To provide graduates with a good foundation in mathematics, sciences and engineering fundamentals required to solve engineering problems that will facilitate them to find employment in MNC's and / or to pursue postgraduate studies with an appreciation for lifelong learning.

PEO 2. To provide graduates with analytical and problem-solving skills to design algorithms, other hardware / software systems, and inculcate professional ethics, interpersonal skills to work in a multi-cultural team.

PEO 3. To facilitate graduates to get familiarized with the art software / hardware tools, imbibing creativity and innovation that would enable them to develop cutting-edge technologies of multi-disciplinary nature for societal development.

Program Outcomes:

PO 1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.

PO 2: Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural science and engineering sciences.

PO 3: Design/development of solutions: 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.

PO 4: Conduct investigations of complex problems: use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO 5: Modern tool usage: create, select and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO 6: The engineer and society: apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO 7: Environment sustainability: understand the impact of the professional engineering solutions in the societal and environmental contexts, and demonstrate the knowledge of, and the need for sustainable development.

PO 8: Ethics: apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO 9: Individual and teamwork: function effectively as an individual and as a member or leader in diverse teams, and in multidisciplinary settings.

PO 10: Communication: 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.

PO 11: Project management and finance: 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.

PO 12: Lifelong learning: recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broader context of technological change.

Program Specific Outcomes:

PSO 1: An Ability to Apply the fundamentals of mathematics, Computer Science and Engineering Knowledge to analyze and develop computer programs in the areas related to Algorithms, System Software, Web Designing, Networking and Data mining for efficient Design of computer based system to deal with Real time Problems.

PSO 2: Ability to implement the Professional Engineering solutions for the betterment of Society, and able to communicate with professional Ethics effectively.

5. Syllabus Copy

CS405PC: JAVA PROGRAMMING

B.TECH II Year II Sem. L T P C
3 1 0 4

Course Objectives:

- To introduce the object oriented programming concepts.
- To understand object oriented programming concepts, and apply them in solving
- problems.
- To introduce the principles of inheritance and polymorphism; and demonstrate how they
- relate to the design of abstract classes
- To introduce the implementation of packages and interfaces
- To introduce the concepts of exception handling and multithreading.
- To introduce the design of Graphical User Interface using applets and swing controls.

Course Outcomes:

- Able to solve real world problems using OOP techniques.
- Able to understand the use of abstract classes.
- Able to solve problems using java collection framework and I/o classes.
- Able to develop multithreaded applications with synchronization.
- Able to develop applets for web applications.
- Able to design GUI based applications

UNIT - I

Object-Oriented Thinking- A way of viewing world – Agents and Communities, messages and methods, Responsibilities, Classes and Instances, Class Hierarchies- Inheritance, Method binding, Overriding and Exceptions, Summary of Object-Oriented concepts. Java buzzwords, An Overview of Java, Data types, Variables and Arrays, operators, expressions, control statements, Introducing classes, Methods and Classes, String handling.

Inheritance– Inheritance concept, Inheritance basics, Member access, Constructors, Creating Multilevel hierarchy, super uses, using final with inheritance, Polymorphism-ad hoc polymorphism, pure polymorphism, method overriding, abstract classes, Object class, forms of inheritancespecialization, specification, construction, extension, limitation, combination, benefits of inheritance,costs of inheritance.

UNIT - II

Packages- Defining a Package, CLASSPATH, Access protection, importing packages.
Interfaces- defining an interface, implementing interfaces, Nested interfaces, applying interfaces, variables in interfaces and extending interfaces.

Stream based I/O (java.io) – The Stream classes-Byte streams and Character streams, Reading console Input and Writing Console Output, File class, Reading and writing Files, Random access file operations, The Console class, Serialization, Enumerations, auto boxing, generics.

UNIT - III

Exception handling - Fundamentals of exception handling, Exception types, Termination or resumptive models, Uncaught exceptions, using try and catch, multiple catch clauses, nested try statements, throw, throws and finally, built- in exceptions, creating own exception sub classes.

Multithreading- Differences between thread-based multitasking and process-based multitasking, Java thread model, creating threads, thread priorities, synchronizing threads, inter thread communication.

UNIT - IV

The Collections Framework (java.util)- Collections overview, Collection Interfaces, The Collection classes- Array List, Linked List, Hash Set, Tree Set, Priority Queue, Array Deque. Accessing a Collection via an Iterator, Using an Iterator, The For-Each alternative, Map

Interfaces and Classes, Comparators, Collection algorithms, Arrays, The Legacy Classes and Interfaces- Dictionary, Hashtable ,Properties, Stack, Vector More Utility classes, String Tokenizer, Bit Set, Date, Calendar, Random, Formatter, Scanner

UNIT - V

GUI Programming with Swing – Introduction, limitations of AWT, MVC architecture, components,containers. Understanding Layout Managers, Flow Layout, Border Layout, Grid Layout, Card Layout,Grid Bag Layout.

Event Handling- The Delegation event model- Events, Event sources, Event Listeners, Event classes, Handling mouse and keyboard events, Adapter classes, Inner classes, Anonymous Inner classes.

A Simple Swing Application, Applets – Applets and HTML, Security Issues, Applets and Applications, passing parameters to applets. Creating a Swing Applet, Painting in Swing, A Paint example, Exploring Swing Controls- JLabel and Image Icon, JText Field, The Swing ButtonsJButton, JToggle Button, JCheck Box, JRadio Button, JTabbed Pane, JScroll Pane, JList, JCombo Box, Swing Menus, Dialogs.

TEXT BOOKS:

1. Java The complete reference, 9th edition, Herbert Schildt, McGraw Hill Education (India) Pvt. Ltd.
2. Understanding Object-Oriented Programming with Java, updated edition, T. Budd, Pearson Education.

REFERENCE BOOKS:

1. An Introduction to programming and OO design using Java, J. Nino and F.A. Hosch, John Wiley & sons
2. Introduction to Java programming, Y. Daniel Liang, Pearson Education.
3. Object Oriented Programming through Java, P. Radha Krishna, University Press.
4. Programming in Java, S. Malhotra, S. Chudhary, 2nd edition, Oxford Univ. Press.
5. Java Programming and Object-oriented Application Development, R. A. Johnson, Cengage Learning.

6. Course Objectives And Course Outcomes

(a) Course Objectives

At the end of the course, the students should be able to:

1. To introduce the object oriented programming concepts.
2. To understand object oriented programming concepts, and apply them in solving
3. problems.
4. To introduce the principles of inheritance and polymorphism; and demonstrate how they
5. relate to the design of abstract classes
6. To introduce the implementation of packages and interfaces
7. To introduce the concepts of exception handling and multithreading.
8. To introduce the design of Graphical User Interface using applets and swing controls.

(b) Course Outcomes

After completing this course the student must demonstrate the knowledge and ability to:

CS405PC1.1: Analyze the importance of Object Oriented Programming for solving real world problems.

CS405PC1.2:Develop applications using Object Oriented Programming Techniques.

CS405PC1.3:Implement multiple inheritance through interface and create user defined packages.

CS405PC1.4: Apply exception handling mechanism and thread model for developing efficient applications.

CS405PC1.5:Write efficient programs by implementing collections framework.

CS405PC1.6: Apply event handling for developing GUI using swings and applets

7. CO and PO Mapping and CO and PSO Mapping

CO and PO Mapping

CO/PO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
C212.1	3	3	3	3	2			1	1		3	2
C212.2	3	3	3	3	3			1	1		3	2
C212.3	3	3	3	3	3			1	1		3	2
C212.4	3	3	3	3	3			1	1		3	2

C212.5	3	3	3	3	3			1	1		3	2
C212.6	3	3	3	3	3			1	1		3	2

Where

1. Low
2. Medium
3. High

CO and PSO Mapping

CO/PO	PSO1	PSO2
C212.1	3	1
C212.2	3	1
C212.3	3	1
C212.4	3	1
C212.5	3	1
C212.6	3	1

Where

1. Low
2. Medium
3. High

8. Brief Notes on the Importance of the Course

This course introduces students to object oriented programming and design. Students will be exposed to the principles of the object oriented programming paradigm specifically including abstraction, encapsulation, inheritance and polymorphism

9. Prerequisites if any

A course on “Programming for Problem Solving using C”.

10. Instructional Learning Outcomes

At the end of the course, the students are able to achieve the following course learning outcomes:

Lecture No.	Topics to be covered	Topics Outcome
L1	A way of viewing world – Agents and Communities, messages and methods, Responsibilities, Classes and Instances	Explain and apply Object Oriemted Concepts
L2-L3	Class Hierarchies- Inheritance, Method binding, Overriding and Exceptions, Summary of Object-Oriented concepts. Java buzzwords	Explain features of Java Language
L4-L5	An Overview of Java, Data types, Example programs	Write standalone java programs
L6-L7	Variables and Arrays, operators, expressions, control statements, Introducing classes	Apply different operatorsand control statements while developing a program
L8-L9	Methods and Classes, Inheritance concept, Inheritance basics, Member access,	Apply Inheritance for code reusability
L10-L11	Constructors, Creating Multilevel hierarchy, Super uses, using final with inheritance	Apply super and final uses for developing an application
L12-L14	Polymorphism-ad hoc polymorphism, pure polymorphism, Method overriding, Abstract classes, Object class	Apply Polymorphism technique for creating multiple variations of application and use abstraction for real world programs
L15	Forms of inheritance, benefits of inheritance, costs of inheritance	Analyze different forms of inheritance and apply the best in real world
L16	Tutorial Classes	Revision of above concepts
L17-L18	**UNIT-2** Defining a Package, CLASSPATH, Access protection, importing packages	Analyze the importance of classpath and access specifier. Create user defined packages
L19	Defining an interface, implementing interfaces	implement interface methods in a class
L20-L21	Nested interfaces, applying interfaces, Variables in interfaces and extending interfaces	Apply interface for multiple inheritance
L22-L24	The Stream classes-Byte streams and Character streams, Reading console Input and Writing Console Output	Analyze the difference between Byte stream and Character stream and use them for different applictaion
L25-L27	File class, Reading and writing Files, Random access file operations	Apply different file methods for reading and writing content to a file
L28-L30	The Console class, Serialization, Enumerations, Auto boxing, generics	Evaluate the importance of Serialization, Autoboxing and Generics for developing efficient programs

L31	Tutorial Classes	Revision of above concepts
L32	**UNIT-3** Fundamentals of exception handling, Exception types	Illustrate what is an exception and its different types
L33-L34	Termination or resumptive models, Uncaught exceptions	Differentiate amongst Termination or resumptive models for exception handling.Determine what is uncaught exception.
L35-L36	Using try and catch, multiple catch clauses, nested try statements	Analyze and Apply exception handling model
MID EXAMINATION-I		
L37-L38	throw, throws and finally ,built- in exceptions, creating own exception sub classes	Analyze the importance of throw,finally and throws clause and apply them in application development
L39-L40	Differences between thread-based multitasking and process-based multitasking, Java thread model	Analyze the pros and cons of thread based and process based multitasking
L41-L42	creating threads, thread priorities	Develop thread based applications
L43-L44	Synchronizing threads, inter thread communication	Develop thread safeapplications
L45	Tutorial Classes	Revision of above concepts
L46-L47	**UNIT-4** Collections overview, Collection Interfaces, The Collection classes- Array List, Linked List	Analyze different collection classes
L48-L49	Hash Set, Tree Set, Priority Queue	Apply collection classes for developing an application.
L50-L51	Array Deque. Accessing a Collection via an Iterator, Using an Iterator, The For-Each alternative	Analyze the difference between Enumeration and Iterator.
L52-L53	Map Interfaces and Classes, Comparators, Collection algorithms, Arrays	Analyze the importance of Map Interface and Classes.And apply different collection algorithms
L54-L55	The Legacy Classes and Interfaces- Dictionary, Hashtable ,Properties	Apply Different legacy classes to develop an application
L56-L58	Stack, Vector, More Utility classes, Random,Formatter,Scanner, Bit Set, String Tokenizer, Date, Calendar	Apply different utility classesto real world application
L59	Tutorial Classes	Revision of above concepts
L60-L61	**UNIT - V** Introduction, limitations of AWT, MVC architecture, components	Understand the advantages of Analyze what is GUI and limitations of AWT.Apply MVC architecture and use different components to develop GUI
L62	containers, Understanding Layout Managers, Flow Layout,	Develop GUI by applying different Layout Managers

L63-L64	Border Layout, Grid Layout, Card Layout, Grid Bag Layout	Develop GUI by applying different Layout Managers
L65-L66	The Delegation event model- Events, Event sources Event Listeners, Event classes	Understand and Apply Event Handling Techniques for GUI
L67-L68	Handling mouse and keyboard events, Adapter classes, Inner classes,	Apply the technique to rethrow the exception
L69-L72	Anonymous Inner classes, A Simple Swing Application, Applets – Applets and HTML, Security Issues, Applets and Applications, passing parameters to applets	Develop applications using Anonymous Inner Classes and analyze the usage of Applet
L73	Creating a Swing Applet, Painting in Swing,, A Paint example, Exploring Swing Controls- JLabel and Image Icon, JText Field, The Swing Buttons	Develop a swing based application
L74	Tutorial Classes	Revision of above concepts
MID EXAMINATION-II		

11. COURSE INFORMATION SHEET (CIS)

(a) Course description

PROGRAMME: **B. Tech.** (Computer Science and Engineering)	DEGREE: **B.TECH**
COURSE: **Java Programming**	YEAR: **II** SEM**: II** CREDITS: **4**
COURSE CODE: **C405PC** REGULATION: **R18**	COURSE TYPE:
COURSE AREA/DOMAIN: **CSE/ PROGRAMMING**	CONTACT HOURS: (T+L)) hours/Week. 6

(b) Syllabus

Unit No.	Details	Hours
I	**Object-Oriented Thinking**- A way of viewing world – Agents and Communities, messages and methods, Responsibilities, Classes and Instances, Class Hierarchies- Inheritance, Method binding, Overriding and Exceptions, Summary of Object-Oriented concepts. Java buzzwords, An Overview of Java, Data types, Variables and Arrays, operators, expressions, control statements, Introducing classes, Methods and Classes, String handling. **Inheritance**– Inheritance concept, Inheritance basics, Member access, Constructors, Creating Multilevel hierarchy, super uses, using final with inheritance,	

	Polymorphism-ad hoc polymorphism, pure polymorphism, method overriding, abstract classes, Object class, forms of inheritancespecialization, specification, construction, extension, limitation, combination, benefits of inheritance,costs of inheritance.	
II	**Packages-** Defining a Package, CLASSPATH, Access protection, importing packages. Interfaces- defining an interface, implementing interfaces, Nested interfaces, applying interfaces, variables in interfaces and extending interfaces. **Stream based I/O (java.io)** – The Stream classes-Byte streams and Character streams, Reading console Input and Writing Console Output, File class, Reading and writing Files, Random access file operations, The Console class, Serialization, Enumerations, auto boxing, generics.	
III	**Exception handling** - Fundamentals of exception handling, Exception types, Termination or resumptive models, Uncaught exceptions, using try and catch, multiple catch clauses, nested try statements, throw, throws and finally, built- in exceptions, creating own exception sub classes. **Multithreading**- Differences between thread-based multitasking and process-based multitasking, Java thread model, creating threads, thread priorities, synchronizing threads, inter thread communication.	
IV	**The Collections Framework (java.util)-** Collections overview, Collection Interfaces, The Collection classes- Array List, Linked List, Hash Set, Tree Set, Priority Queue, Array Deque. Accessing a Collection via an Iterator, Using an Iterator, The For-Each alternative, Map Interfaces and Classes, Comparators, Collection algorithms, Arrays, The Legacy Classes and Interfaces- Dictionary, Hashtable ,Properties, Stack, Vector More Utility classes, String Tokenizer, Bit Set, Date, Calendar, Random, Formatter, Scanner	
V	**GUI Programming with Swing** – Introduction, limitations of AWT, MVC architecture, components,containers. Understanding Layout Managers, Flow Layout, Border Layout, Grid Layout, Card Layout,Grid Bag Layout. **Event Handling**- The Delegation event model- Events, Event sources, Event Listeners, Event classes, Handling mouse and keyboard events, Adapter classes, Inner classes, Anonymous Inner classes. **A Simple Swing Application, Applets** – Applets and HTML, Security Issues, Applets and Applications, passing parameters to applets. Creating a Swing Applet, Painting in Swing, A Paint example, Exploring Swing Controls- JLabel and Image Icon, JText Field, The Swing ButtonsJButton, JToggle Button, JCheck Box, JRadio Button, JTabbed Pane, JScroll Pane, JList, JCombo	
	Contact classes for syllabus coverage	**65**
	Lectures beyond syllabus	**02**
	Tutorial classes	**10**

	Total No. of classes	**69**

(c) Gaps in syllabus

S.No.	Topic	Propose Action	No. of classes
1	No Gaps Observed	Guest Lecture / industry visit	Nil

(d) Topics beyond Syllabus

S.No.	Topic	Propose Action	No. of Classes
1	**JDBC Implementation**	PPT	2

(e) Web Source References

S.No.	Name of book/ website
1	https://www.javatpoint.com/java-tutorial
2	https://www.w3schools.com/java/
3	https://dziretocode.wordpress.com

(f) Delivery / Instructional Methodologies:

☑ CHALK & TALK	☑ STUD. ASSIGNMENT	☑ WEB RESOURCES
☑ LCD/SMART BOARDS	☑ STUD. SEMINARS	☐ ADD-ON COURSES

(g) Assessment Methodologies - Direct

☑ Assignments	☑ Stud. Seminars	☑ Tests/Model Exams	☑ Univ. Examination
☑ Stud. Lab Practices	☑ Stud. Viva	☐ Mini/Major Projects	☐ Certifications
☐ Add-On Courses	☐ Others		

(h) Assessment Methodologies - Indirect

þ Assessment Of Course Outcomes (By Feedback, Once)	☑ Student Feedback On Faculty (Twice)
☐Assessment Of Mini/Major Projects By Ext. Experts	☐ Others

(i) Textbooks and References

Text Books	
1.	Java The complete reference, 9th edition, Herbert Schildt, McGraw Hill Education (India) Pvt. Ltd.
2.	Understanding Object-Oriented Programming with Java, updated edition, T. Budd, Pearson Education.
Suggested / Reference Books	
1.	An Introduction to programming and OO design using Java, J. Nino and F.A. Hosch, John Wiley & sons
2.	Object Oriented Programming through Java, P. Radha Krishna, University Press..
3.	Java Programming and Object-oriented Application Development, R. A. Johnson, Cengage Learning.

12. Micro Lesson Plan

Lesson No.	Topics to be covered	Planned Date	Actual Date
UNIT – 1			
L1	A way of viewing world – Agents and Communities, messages and methods, Responsibilities, Classes and Instances	16/12/2019	
L2	Class Hierarchies- Inheritance, Method binding, Overriding and Exceptions	16/12/2019	
L3	Summary of Object-Oriented concepts. Java buzzwords	17/12/2019	
L4	An Overview of Java, Data types	23/12/2019	

L5	Example programs	23/12/2019	
L6	Variables and Arrays, operators	24/12/2019	
L7	expressions, control statements, Introducing classes	27/12/2019	
L8	Methods and Classes	28/12/2019	
L9	Inheritance concept, Inheritance basics, Member access	28/12/2019	
L10	Constructors, Creating Multilevel hierarchy	30/12/2019	
L11	Super uses, using final with inheritance	30/12/2019	
L12	Polymorphism-ad hoc polymorphism, pure polymorphism	31/12/2019	
L13	Method overriding	03/01/2020	
L14	Abstract classes, Object class	04/01/2020	
L15	Forms of inheritance, benefits of inheritance, costs of inheritance	04/01/2020	
L16	Tutorial Class	06/01/2020	
UNIT – 2			
L17	Defining a Package, CLASSPATH	06/01/2020	
L18	Access protection, importing packages	07/01/2020	
L19	Defining an interface, implementing interfaces	11/01/2020	
L20	Nested interfaces, applying interfaces	11/01/2020	
L21	Variables in interfaces and extending interfaces	13/01/2020	
L22	The Stream classes-Byte streams	13/01/2020	
L23	Character streams	14/01/2020	
L24	Reading console Input and Writing Console Output	17/01/2020	
L25	File class	18/01/2020	
L26	Reading and writing Files	18/01/2020	
L27	Random access file operations	20/01/2020	
L28	The Console class	20/01/2020	

L29	Serialization	21/01/2020	
L30	Enumerations, Auto boxing, generics	25/01/2020	
L31	Tutorial Class	25/01/2020	
UNIT – 3			
L32	Fundamentals of exception handling, Exception types	27/01/2020	
L33	Termination or resumptive models	27/01/2020	
L34	Uncaught exceptions	28/01/2020	
L35	Using try and catch, multiple catch clauses	31/01/2020	
L36	nested try statements	01/02/2020	
I MID EXAMINATION			
L37	throw, throws and finally, built- in exceptions	01/02/2020	
L38	creating own exception sub classes	03/02/2020	
L39	Differences between thread-based multitasking and process-based multitasking	03/02/2020	
L40	Java thread model	04/02/2020	
L41	creating threads	08/02/2020	
L42	thread priorities	08/02/2020	
L43	Synchronizing threads	10/02/2020	
L44	inter thread communication	10/02/2020	
L45	Tutorial Class	11/02/2020	
UNIT – 4			
L46	Collections overview, Collection Interfaces	25/02/2020	
L47	The Collection classes- Array List, Linked List	28/02/2020	
L48	Hash Set, Tree Set	29/02/2020	
L49	Priority Queue	29/02/2020	
L50	Array Deque. Accessing a Collection via an Iterator	02/03/2020	
L51	Using an Iterator, The For-Each alternative	03/03/2020	

L52	Map Interfaces and Classes, Comparators	06/03/2020	
L53	Collection algorithms, Arrays	07/03/2020	
L54	The Legacy Classes and Interfaces- Dictionary	07/03/2020	
L55	Hashtable ,Properties	13/03/2020	
L56	Stack, Vector, inter thread communication	14/03/2020	
L57	Random,Formatter,Scanner, Bit Set	14/03/2020	
L58	String Tokenizer ,Date, Calendar	16/03/2020	
L59	Tutorial Class	16/03/2020	
UNIT – 5			
L60	Introduction, limitations of AWT, MVC architecture	17/03/2020	
L61	components	21/03/2020	
L62	containers,Understanding Layout Managers, Flow Layout	21/03/2020	
L63	Border Layout, Grid Layout	23/03/2020	
L64	Card Layout, Grid Bag Layout	23/03/2020	
L65	The Delegation event model- Events, Event sources	24/03/2020	
L66	Event Listeners, Event classes	27/03/2020	
L67	Handling mouse and keyboard events	28/03/2020	
L68	Adapter classes, Inner classes	28/03/2020	
L69	Anonymous Inner classes, A Simple Swing Application	30/03/2020	
L70	Applets – Applets and HTML, Security Issues, Applets and Applications, passing parameters to applets	30/03/2020	
L71	Creating a Swing Applet, Painting in Swing, A Paint example, Exploring Swing Controls- JLabel and Image Icon	31/03/2020	
L72	JText Field, The Swing Buttons- JButton, JToggle Button, JCheck Box	03/04/2020	
L73	Tutorial Class	04/04/2020	

II MID EXAMINATION

13. Teaching Schedule

Subject	Java Programming			
Text Books (to be purchased by the Students)				
Book 1	Java The complete reference, 9th edition, Herbert Schildt, McGraw Hill Education (India) Pvt. Ltd.			
Reference Books				
Book 2	An Introduction to programming and OO design using Java, J. Nino and F.A. Hosch, John Wiley & sons			
Unit	**Topic**	**Chapters No's**		**No. of classes**
		Book 1	**Book 2**	
I	Object-Oriented Thinking- A way of viewing world – Agents and Communities, messages and methods, Responsibilities	-	0	
	Classes and Instances, Class Hierarchies- Inheritance, Method binding, Overriding and Exceptions, Summary of Object-Oriented concepts.	6	2	
	Java buzzwords, An Overview of Java, Data types, Variables and Arrays, operators, expressions, control statements, Introducing classes, Methods and Classes,	1	4	
	String handling.	16	-	
	Inheritance– Inheritance concept, Inheritance basics, Member access, Constructors, Creating Multilevel hierarchy, super uses, using final with inheritance, Polymorphism-ad hoc polymorphism, pure polymorphism, method overriding, abstract classes, Object class, forms of inheritancespecialization, specification, construction, extension, limitation, combination, benefits of inheritance,costs of inheritance.	8	10	
II	Packages- Defining a Package, CLASSPATH, Access protection, importing packages.	9	-	
	Interfaces- defining an interface, implementing interfaces, Nested interfaces, applying interfaces, variables in interfaces and extending interfaces.	9	9	
	Stream based I/O (java.io) – The Stream classes-Byte streams and Character streams, Reading console Input and Writing Console Output, File class, Reading and writing Files, Random access file operations, The Console class, Serialization, Enumerations, auto boxing, generics.	-	7	
III	Exception handling - Fundamentals of exception handling, Exception types, Termination or resumptive models, Uncaught exceptions, using try and catch,	10	15	

	multiple catch clauses, nested try statements, throw, throws and finally, built- in exceptions, creating own exception sub classes.			
	Multithreading- Differences between thread-based multitasking and process-based multitasking, Java thread model, creating threads, thread priorities, synchronizing threads, inter thread communication.	11	-	
IV	The Collections Framework (java.util)- Collections overview, Collection Interfaces, The Collection classes- Array List, Linked List, Hash Set, Tree Set, Priority Queue, Array Deque.	18	12	
	Accessing a Collection via an Iterator, Using an Iterator, The For-Each alternative, Map Interfaces and Classes, Comparators, Collection algorithms, Arrays,	18	22	
	The Legacy Classes and Interfaces- Dictionary, Hashtable ,Properties, Stack, Vector	18	-	
	More Utility classes, String Tokenizer, Bit Set, Date, Calendar, Random, Formatter, Scanner	19	-	
V	GUI Programming with Swing – Introduction, limitations of AWT, MVC architecture, components,containers.	31	-	
	Understanding Layout Managers, Flow Layout, Border Layout, Grid Layout, Card Layout,Grid Bag Layout.	26	-	
	Event Handling- The Delegation event model- Events, Event sources, Event Listeners, Event classes, Handling mouse and keyboard events, Adapter classes, Inner classes, Anonymous Inner classes.	24	-	
	A Simple Swing Application, Applets – Applets and HTML, Security Issues, Applets and Applications, passing parameters to applets. Creating a Swing Applet, Painting in Swing, A Paint example, Exploring Swing Controls- JLabel and Image Icon, JText Field, The Swing ButtonsJButton, JToggle Button, JCheck Box, JRadio Button, JTabbed Pane, JScroll Pane, JList, JCombo Box, Swing Menus, Dialogs.	31	17	
Contact classes for syllabus coverage				**55**
Lectures beyond syllabus				**01**
Tutorial classes				**12**
Classes for gaps & Add-on classes				**02**
Total No. of classes				**70**

14. Mid Descriptive and Quiz Question Papers with Answers

Global Institute of Engineering &Technology **(Approved by AICTE, Affiliated to JNTUH)** **Chilkur (Vil), Moinabad (Mdl), RR District**			**College Code** **U6**	
Name of the Exam:	**I Mid Examinations**		**Marks:**	**10**
Year-Sem & Branch:	**II-II &CSE**	**Duration:**	**60 Min**	
Subject:	**Java Programming**	**Date & Session**	**06/03/2020&FN**	
Answer **ANY TWO** of the following Questions			2**X5=10**	

Q.NO	**QUESTION**	**CO MAPPING**	**BLOOMS TAXONOMY**
1	Differentiate between StringBuffer and StringBuilder class with example program?	CO1	Evaluate
2	Write a Java Program to create an abstract class named Shape that contains two integers and an empty method named printArea().Create three classes named Rectangle, Triangle,Circle to define the printArea() for each respective class.	CO1	Create
3	Write a Java Program to extend an interface in Java?	CO2	Create
4	What is Exception? Explain the Exception Handling Mechanism with example program?	CO3	Create

MID I Descriptive Questions and Answers

1. **Differentiate between StringBuffer and StringBuilder class with example program?**

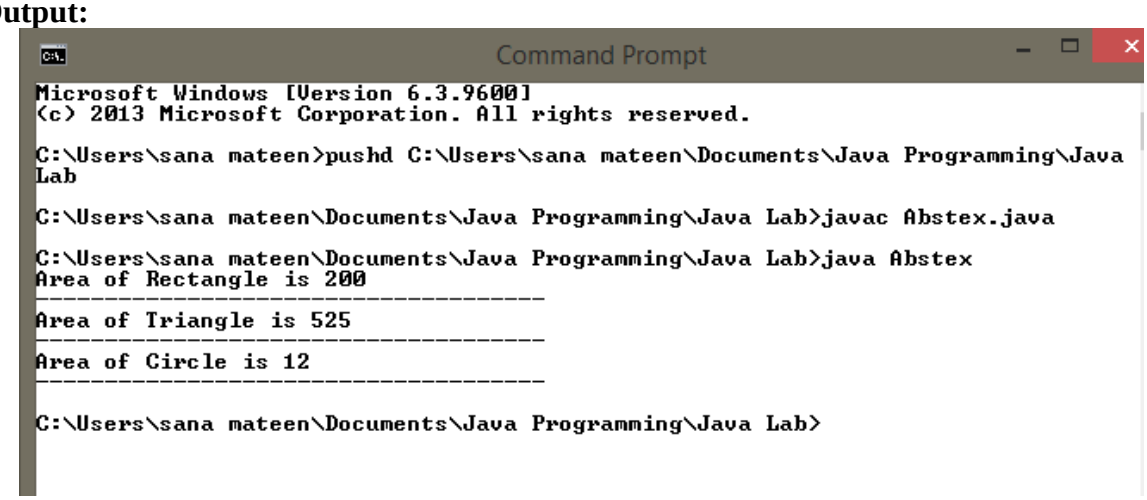
StringBuffer	StringBuilder
StringBuffer is synchronized i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.	StringBuilder is non-synchronized i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.
StringBuffer is less efficient than StringBuilder.	StringBuilder is more efficient than StringBuffer.
public class StringBufferE{ **public static void main(String[] args){** **StringBuffer buffer=new StringBuffer("hello");** **buffer.append("sana mateen");** **System.out.println(buffer);** **}** **}**	**public class StringBuilderr{** **public static void main(String[] args){** **StringBuilder builder=new StringBuilder("hello");** **builder.append("sana mateen");** **System.out.println(builder);** **}** **}**
C:\Users\sana mateen>rename StringBufferEx.java StringBufferE.java C:\Users\sana mateen>javac StringBufferE.java C:\Users\sana mateen>java StringBufferE hellosana mateen C:\Users\sana mateen>	C:\Users\sana mateen>start notepad StringBuilderr.java C:\Users\sana mateen>javac StringBuilderr.java C:\Users\sana mateen>java StringBuilderr hellosana mateen C:\Users\sana mateen>

2. **Write a Java Program to create an abstract class named Shape that contains two integers and an empty method named printArea().Create three classes named Rectangle, Triangle,Circle to define the printArea() for each respective class.**

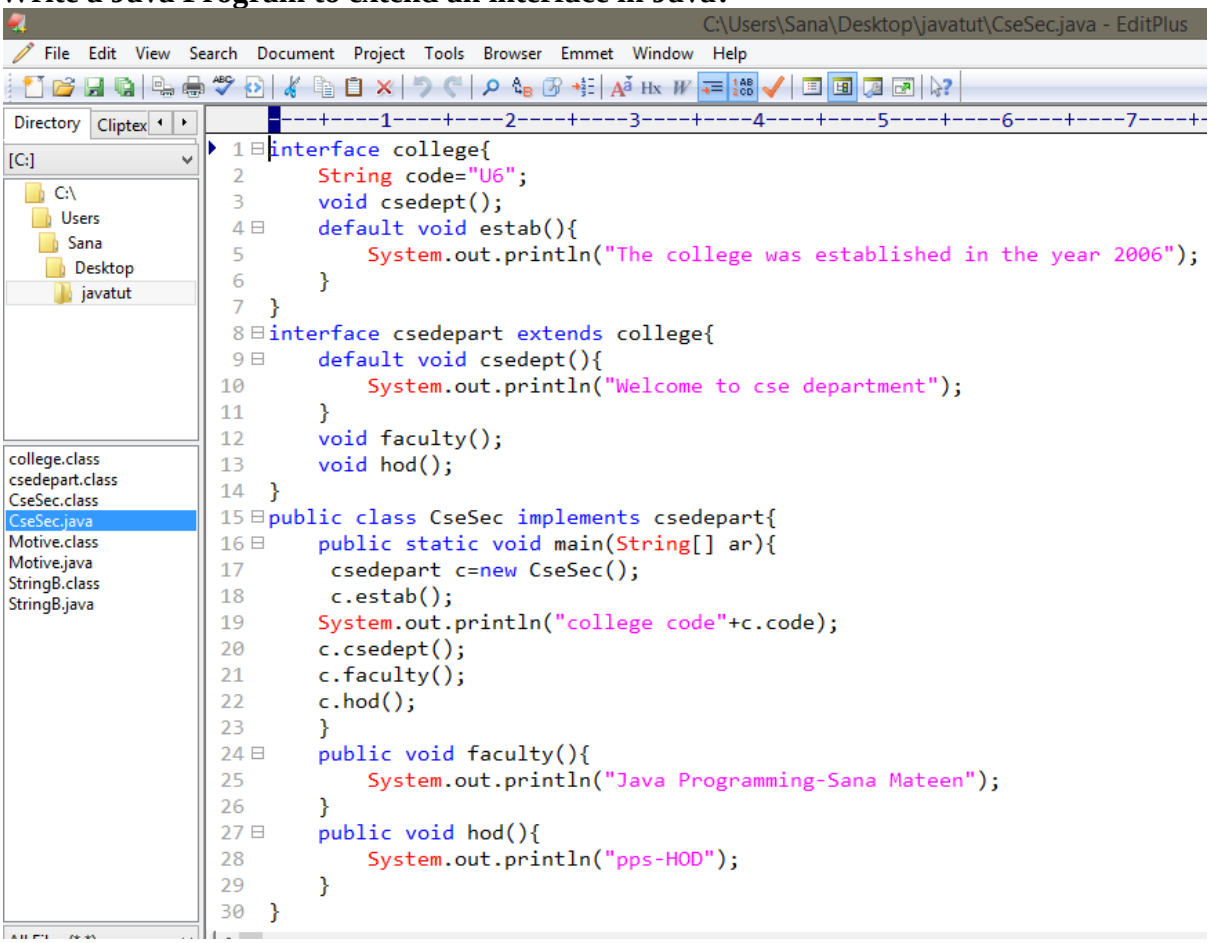
```
abstract class shape
{
        public int x, y;
        public abstract void printArea();
}
class Rectangle extends shape
{
        public void printArea()
        {
                System.out.println("Area of Rectangle is " + x * y);
        }
}
class Triangle extends shape
{
        public void printArea()
        {
                System.out.println("Area of Triangle is " + (x * y) / 2);
        }
}
class Circle extends shape
```

```
{
        public void printArea()
        {
                System.out.println("Area of Circle is " + (22 * x * x) / 7);
        }
}
public class Abstex
{
        /** * @param args the command line arguments */
        public static void main(String[] args)
        {
                Rectangle r = new Rectangle();
                r.x = 10; r.y = 20;
                r.printArea();
                System.out.println("-------------------------------------");
                Triangle t = new Triangle();
                t.x = 30;
                t.y = 35;
                t.printArea();
                System.out.println("-------------------------------------");
                Circle c = new Circle();
                c.x = 2;
                c.printArea();
                System.out.println("-------------------------------------");
        }
}
```

Output:



3. Write a Java Program to extend an interface in Java?



Output

```
C:\Users\sana mateen\Desktop>pushd C:\Users\Sana\Desktop\javatut

C:\Users\Sana\Desktop\javatut>start editplus CseSec.java

C:\Users\Sana\Desktop\javatut>javac CseSec.java

C:\Users\Sana\Desktop\javatut>java CseSec
The college was established in the year 2006
college codeU6
Welcome to cse department
Java Programming-Sana Mateen
pps-HOD
```

4. What is Exception? Explain the Exception Handling Mechanism with example program?

Exception Handling

Bug: An error in a program is called bug.

Debugging: The process of removing errors from program or application is called as Debugging.

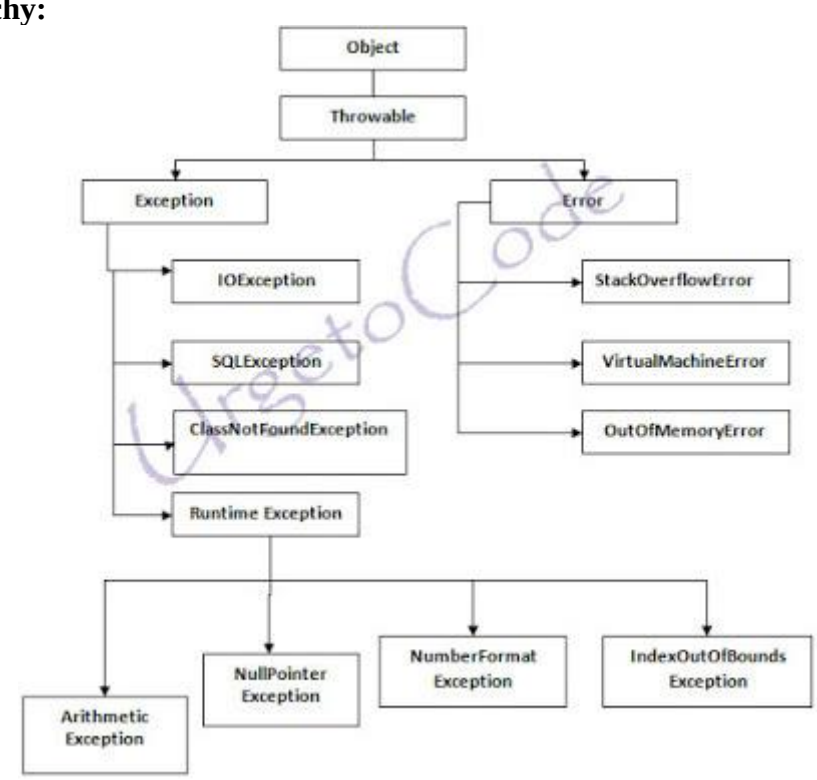
Exception: It is an abnormal event in the program.

Exception Handling: It is the mechanism to handle runtime errors. So that normal flow of the application is maintained.

Exceptions are of two kinds Compile time exceptions and runtime exceptions. The following Figure lists the same.

Errors are irrecoverable. java.lang.Error is a subclass of Throwable.
Example: OutOfMemoryError, StackOverflowError, AssertionError etc.

Class Hierarchy:



1. **Checked Exceptions/Compile time Exceptions**

The classes that extend Throwable except Runtime Exception class and Error are known as Checked Exception.
Checked Exceptions are checked at Compile time i.e. if we get exceptions after compiling the java code.
Example:
IOException, **SQLException**
Program:

```
import java.io.FileReader;
```

```
import java.io.PrintWriter;
class IOExampleException
{
          public static void main(String[] a){
          FileReader f=new FileReader("app.txt");

        for(int i=1;(i=f.read())!=-1;i++)   {
         System.out.print((char)i);

                    }
         f.close();
          }
}
```

Snap:

```
import java.io.FileReader;
import java.io.PrintWriter;
class IOExampleException
{
    public static void main(String[] a){
    FileReader f=new FileReader("app.txt");

         for(int i=1;(i=f.read())!=-1;i++)   {
          System.out.print((char)i);

          }
          f.close();
    }
}
```

Output:

```
F:\jpgm>javac IOExampleException.java
IOExampleException.java:6: error: unreported exception FileNotFoundException; mu
st be caught or declared to be thrown
       FileReader f=new FileReader("app.txt");
                  ^
IOExampleException.java:8: error: unreported exception IOException; must be caught or
declared to be thrown
        for(int i=1;(i=f.read())!=-1;i++)   {
                        ^
IOExampleException.java:12: error: unreported exception IOException; must be caught or
declared to be thrown
         f.close();
               ^
3 errors

F:\jpgm>
```

```
F:\jpgm>javac IOExampleException.java
IOExampleException.java:6: error: unreported exception FileNotFoundException; mu
st be caught or declared to be thrown
        FileReader f=new FileReader("app.txt");
                     ^
IOExampleException.java:8: error: unreported exception IOException; must be caug
ht or declared to be thrown
        for(int i=1;(i=f.read())!=-1;i++)   {
                        ^
IOExampleException.java:12: error: unreported exception IOException; must be cau
ght or declared to be thrown
          f.close();
                  ^
3 errors
```

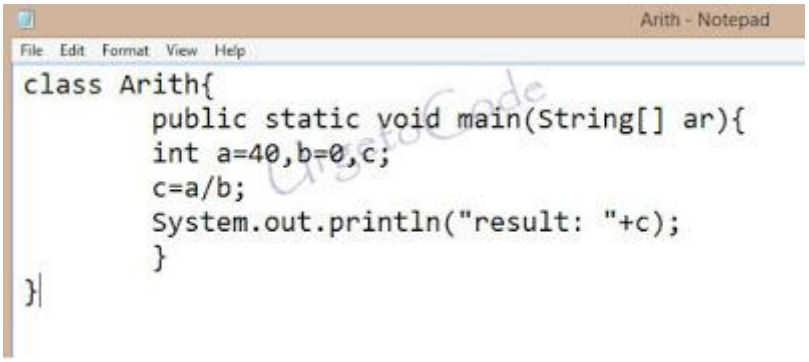
2. **Unchecked Exception/ Runtime Exception**

The classes that extend Runtime Exception class are known as Unchecked Exception.
Example:
ArithmeticException (Divide by zero), **NullPointerException** (trying to convert null to string (String s=null; System.out.println (s.length ()))), **ArrayIndexOutOfBoundsException**

Program:

```
class Arith{
        public static void main(String[] ar){
        int a=40,b=0,c;
        c=a/b;
        System.out.println("result: "+c);
        }
}
```

Snap:



Output:

```
F:\jpgm>javac Arith.java

F:\jpgm>java Arith
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Arith.main(Arith.java:4)

F:\jpgm>
```

```
F:\jpgm>java Arith
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Arith.main(Arith.java:4)

F:\jpgm>
```

3 . Error:

It is irrecoverable.

Example:

OutOfMemoryError:

When an application attempts to add more data into heap space area, but there is not enough space.

VirtualMachineError:

It is thrown to indicate that JVM is broken or has run out of resources.

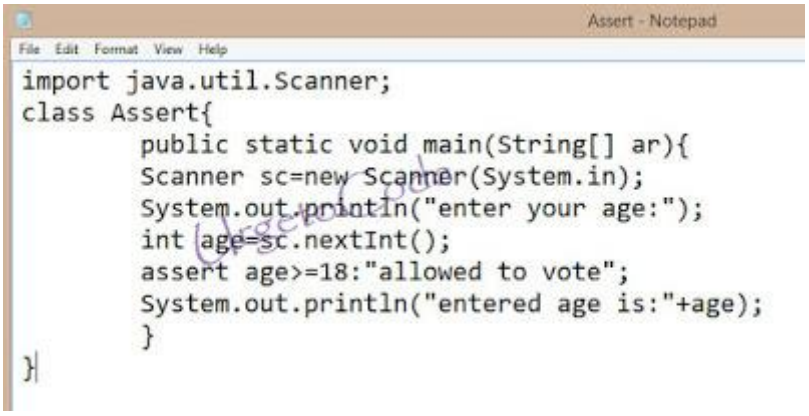
AssertionError:

The assertion is effective ways of detecting and correcting the error.To enable Assertion use –ea option

Program:

```
import java.util.Scanner;
class Assert{
        public static void main(String[] ar){
        Scanner sc=new Scanner(System.in);
        System.out.println("enter your age:");
        int age=sc.nextInt();
        assert age>=18:"allowed to vote";
        System.out.println("entered age is:"+age);
        }
}
```

Snap:



Output:

```
F:\jpgm>java -ea Assert
enter your age:
23
Exception in thread "main" java.lang.AssertionError: not allowed to vote
        at Assert.main(Assert.java:7)
```

```
F:\jpgm>java -ea Assert
enter your age:
12
entered age is:12
F:\jpgm>
```

MID I OBJECTIVE QUESTION PAPER

1.Java is not purely object oriented programming languages as it supports_____[ ]
a. Primitive data type b.pointers c.weakly typed d.strongly typed

2.What is the output of the following code? [ ]

```
import java.util.Scanner;
class Motive{
        public static void main(String[] ar){
                System.out.println("enter your stream");
                Scanner s=new Scanner(System.in);
                String n=s.next();
                if(n.equalsIgnoreCase("cse")){
                        System.out.println("I can develop anything");
                }
                else{
                        System.out.println("I will do it");
                }
        }
}
```

a. enter your stream
CSE
I can develop anything

b. enter your stream
CSIT
I will do it

c. enter your stream
IT
I will do it

d. enter your stream
MSC
I will do it

3. The fields in an interface are implicitly specified as, [ ]
a. static only b. protected c. private d.both static and final e. none of the above.
4. What will be the output of the following Java program? [ ]

```
class c
{
      public void main( String[] args )
      {
         System.out.println( "Hello" + args[0] );
      }
}
```

a. Hello c b. Hello c. Hello world d.Runtime Error

5. What is the process by which we can control what parts of a program can access the members of a class? [ ]

a.Polymorphism b. Abstraction c. Encapsulation d. Recursion

6. Which of these method of class String is used to compare two String objects for their equality? [ ]

a. equals() b. Equals() c. isequal() d. Isequal()

7. Super keyword in java is used to [ ]

a.Refer immediate parent class instance variables. b.Invoke immediate parent class methods.
c.Invoke immediate parent class constructor. d.All

8. If a class inheriting an abstract class does not define all of its function then it will be known as? [ ]

a. Abstract b. A simple class c. Static class d. None of the mentioned

9. Which of these packages contains Scanner class? [ ]

a. java.lang b. java.util c. java.io d. java.system

10. Which of these keywords is not a part of exception handling? [ ]

a. try b. finally c. thrown d. catch

Fill in the Blanks:

11._____ is used to achieve multiple inheritance.

12._____ is a collection of classes and interfaces.

13._____ function gives the string representation for objects.

14.final keyword is used to create_____ and prevents_____

15._____ specifies multiple constructors with different signature.

16.Objects and instance variables live in _____memory

17.method calls and local variables lives in _____memory

18.An exception can be handled using _____ and _____

19._____ clause is used to throw user defined exceptions.

20.An interface which is declared inside another interface or class is _____

Key

S.No	MCQ	Fill in Blanks
1	a	interface
2	a	package
3	c	toString()
4	d	constants,inheritance
5	c	constructor overloading
6	a	heap
7	d	stack
8	a	try and catch
9	b	throw
10	c	nested interface

15. Previous Question Papers

Code No: 133BM **R16**

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD
B.Tech II Year I Semester Examinations, November/December - 2018
OBJECT ORIENTED PROGRAMMING THROUGH JAVA
(Common to CSE, IT)

Time: 3 Hours **Max. Marks: 75**

Note: This question paper contains two parts A and B.
Part A is compulsory which carries 25 marks. Answer all questions in Part A.
Part B consists of 5 Units. Answer any one full question from each unit. Each question carries 10 marks and may have a, b, c as sub questions.

PART- A

(25 Marks)

1a)	What is inheritance? Give example.	[2]
b)	Define the basic characteristics of object oriented programming.	[3]
c)	What is Console class? What is its use in java?	[2]
d)	What is the use of auto boxing in java? Explain.	[3]
e)	What is thread based preemptive multitasking?	[2]
f)	How do we set priorities for threads?	[3]
g)	What is a Java Priority queue?	[2]
h)	What is a Collection Class? Give an example.	[3]
i)	What is Swing in Java? How it differs from Applet.	[2]
j)	How do applets differ from application program?	[3]

PART-B

(50 Marks)

2.a)	Describe the structure of a typical Java program with an example.	
b)	Write the significance of Java Virtual Machine.	[5+5]
	OR	
3.a)	How do we implement polymorphism in JAVA? Explain briefly.	
b)	What is an array? How do you declare the array in java? Give examples.	[5+5]
4.a)	How to design and implement an interface in Java? Give an example.	
b)	Give an example where interface can be used to support multiple inheritance.	[5+5]
	OR	
5.a)	What are the methods available in the Character Streams? Discuss.	
b)	Distinguish between Byte Stream Classes and Character Stream Classes.	[5+5]
6.a)	What is an Exception? How is an Exception handled in JAVA?	
b)	Write a java program that illustrates the application of multiple catch statements.	[5+5]
	OR	
7.a)	Differentiate between multiprocessing and multithreading. What is to be done to implement these in a program?	
b)	Write a program that creates two threads. Fist thread prints the numbers from 1 to 100 and the other thread prints the numbers from 100 to 1.	[5+5]

8. a) What are similarities and difference between ArrayList and Vector? Explain.
b) What is different between Iterator and ListIterator? Explain different ways to iterate over a list. [5+5]

OR

9.a) What are the best practices related to Java Collections Framework? Discuss.
b) What is Comparable and Comparator interface? Differentiate between them. [5+5]

10.a) What is the difference between init( ) and start ( ) methods in an Applet? When will each be executed?
b) Write the applets to draw the Cube and Circle shapes. [5+5]

OR

11.a) Explain various layout managers in JAVA.
b) Write a program to create a frame window that responds to mouse clicks. [5+5]

---oo0oo---

26

Code No: 133BM **R16**

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD
B.Tech II Year I Semester Examinations, April/May - 2018
OBJECT ORIENTED PROGRAMMING THROUGH JAVA
(Common to CSE, IT)

Time: 3 Hours **Max. Marks: 75**

Note: This question paper contains two parts A and B.
Part A is compulsory which carries 25 marks. Answer all questions in Part A.
Part B consists of 5 Units. Answer any one full question from each unit. Each question carries 10 marks and may have a, b, c as sub questions.

PART- A

(25 Marks)

1.a)	What is abstract class? Give example.	[2]
b)	Explain the use of 'for' statement in Java with an example.	[3]
c)	Define a Package? What is its use in java? Explain.	[2]
d)	List out the benefits of Stream oriented I/O.	[3]
e)	How do we start and stop a thread?	[2]
f)	Write the complete life cycle of a thread.	[3]
g)	What is the benefit of Generics in Collections Framework?	[2]
h)	Differentiate between Enumeration and Iterator interface.	[3]
i)	What are the limitations of AWT?	[2]
j)	Why do applet classes need to be declared as public?	[3]

PART-B

(50 Marks)

2.a)	What is inheritance and how does it help to create new classes quickly.	
b)	Describe different levels of access protection available in Java.	[5+5]
	OR	
3.a)	List the primitive data types available in Java and explain.	
b)	What is polymorphism? Explain different types of polymorphisms with examples.	[5+5]
4.a)	What is an interface? What are the similarities between interfaces and classes?	
b)	How can you extend one interface by the other interface? Discuss.	[5+5]
	OR	
5.a)	Discuss about CLASSPATH environment variables.	
b)	Discuss the different levels of access protection available in Java.	[5+5]
6.a)	What are advantages of using Exception handling mechanism in a program?	
b)	Write a java program that demonstrates how certain exception types are not allowed to be thrown.	[5+5]
	OR	
7.a)	What are the different ways that are possible to create multiple threaded programs in java? Discuss the differences between them.	
b)	Write a program to create four threads using Runnable interface.	[5+5]

8.a) What is Java Collections Framework? List out some benefits of Collections framework and explain.

b) What is the importance of hashCode() and equals() methods? [5+5]

OR

9.a) What are the common algorithms implemented in Collections Framework? Discuss.

b) What is difference between ArrayList and LinkedList in collection framework? Explain. [5+5]

10.a) What is an applet? Explain the life cycle of Applet with a neat sketch.

b) Write the applets to draw the Cube and Cylinder shapes. [5+5]

OR

11.a) What is an Layout manager? Explain different types of Layout managers.

b) Write a program to create a frame window that responds to key strokes. [5+5]

---oo0oo---

16. Assignment Topics

MID I

1. **Explain interface in detail along with example?**
2. **Explain Abstraction with an example program?**
3. **Explain StringBuffer and String Builder along with example program?**
4. **Explain features of java?**
5. **Explain the characteristics of OOP?**
6. **Explain superkeyword and its usage?**

MID II

1. **Explain the difference between process based and thread based multitasking?**
2. **Explain the concept of Collections Framework?**
3. **Write a Java Program to implement a Traffic Light?**
4. **Write a Java Program to implement Producer Consumer Problem?**

17. Unit wise Question Bank

Short Answer Questions:

Q.No	Questions
	UNIT 1
1	What are the features of Java Language?
2	What is Data Abstraction?
3	List String manipulation functions of Java String Class?
4	What is inheritance?Give example?
5	Write about Object Oriented Programming Characteristics?
6	What is an abstract class?Give example?
7	Explain the use of for statement in java with an example?
8	Differentiate between objects and classes?Write an example program?
9	Write a JavaProgram to differentiate between print() and println() ?
10	Differentiate between break and continue statement with the help of program?
11	Write a Java Program to implement typecasting?
12	Write a Java Program to implement method overloading?
13	What is super keyword?Mention its usage along with example program?
14	Write about Object class?
15	Write about final keyword and its use along with example program?
16	Write about different forms of inheritance?
17	What are the benefits of inheritance?
18	Explain the use of final keyword
	UNIT 2
1	What is Console class?What is its use in java?
2	What is the use of autoboxing in java?Explain.
3	What is static inner class?
4	Explain implicit and explicit import statements?
5	Define Package?What is its use in java?Explain
6	List out the benefits of Stream Oriented I/O
7	Differentiate between interface and abstract class
8	Explain Access protection with the help of a program?
9	List the Byte Stream classes?
10	Define Serialization and Deserialization?
11	Write about CLASSPATH environment variable?
12	What is the purpose of nested interface?
13	Write a Java Program to read an entire line through Keyboard?
	UNIT 3
1	What is thread based preemptive multitasking?
2	How do we set priorities for threads?
3	Explain the difference between process and threads

4	Explain how a multiple catch statement works?
5	How do we start and stop thread?
6	Write the complete life cycle of a thread?
7	List any six built-in exceptions in Java?
8	What is Termination or Resumptive Model?
9	Differentiate between compiletime error and runtime error?
10	Differentiate between throw and throws clause?
11	Explain the types of Exception?
12	Which class is the super class for all types of errors and exceptions in Java?
13	What are the two ways of creating thread?
14	What is Thread State?
15	List any four Thread constructors?
16	What is synchronization?
UNIT 4	
1	What is Java Priority queue?
2	What is Collection Class?Give an example?
3	Explain the use of String tokenizer with an example?
4	Explain any three methods defined by Iterator?
5	What is the benefit of Generics in Collections Framework?
6	Differentiate between Enumeration and iterator interface?
7	Differentiate between Arrays and Vector?
8	Differentiate between ArrayList and LinkedList?
9	Write about HashSet and TreeSet class?
10	Write a Java Program for traversing through ArrayList?
11	Define Map?Mention the different type of Map classes?
12	Write a Java Program to implement Date Class?
13	What is ArrayDeque?Specify its features?
14	List the HashTable constructors?
15	Differentiate between HashMap andTreeMap?
16	What are benefits of Collection framework?
17	What is collection?
18	What is the difference between Collection and Collections?
UNIT 5	
1	What is Swing in Java?How it differs from Applet
2	How do applets differ from application program?
3	Explain the use of layout managers?
4	Explain the life cycle of an applet?
5	What are the limitations of AWT?
6	Why do Applet classes need to be declared as public?
7	What are the containers available in swing?
8	What are the advantages of event delegation Model?
9	What are inner classes?Specify any two properties of inner class?

10	Mention the situation in which anonymous inner class is used?
11	What is adapter class?
12	What is the relationship between an event listener interface and an event adapter class?
13	What is AWT?
14	What is container and component?
15	What are the limitations of AWT?
16	What is the function of SetLayout and FlowLayout?
17	What is an Applet?
18	Explain the lifecycle of Applet?
19	What is JPanel Object?
20	Differentiate between GridBagLayout and GridLayout?

Long Answer Questions:

Q. No	Questions
	UNIT 1
1	Explain the object oriented programming characteristics?
2	What is the usage of enumerated data type?Give example?
3	Explain Final Keyword and it's usage along with example program?
4	With the help of program explain the usage of super keyword?
5	Explain about polymorphism I detail?
6	What is an array?How do you declare an array in java?Give examples?
7	Explain the structure of java program with an example?
8	Explain the importance of Java Virtual Machine?
9	Briefly explain the different data types used in java along with an example?
10	Explain the difference between method overloading and method overriding?
11	Write a java program to find the sum of the given number?
12	Write a java program to implement String Handling Functions?
13	Write a java program to implement Constructor Overloading?
14	Explain Polymorphism in detail along with example program?
15	Explain abstract classes along with example program?
16	Explain different forms of inheritance?
17	Write a Java program to implement Multi level inheritance?
18	Write a Java program to implement Single Inheritance?
19	Write a Java program to implement Hierarchical Inheritance?
	UNIT 2
1	What is nested class?Differentiate between static nested classes and non static nested classes?
2	Define package?How to access,import a package?Explain with example program?
3	Explain Generics with an example program?
4	How to design and implement interfacein java?Give an example?
5	Write a java program to implement multiple inheritance?

6	What are the methods available in the Character Streams?Discuss
7	Distinguish between Byte Stream Classes and Character Stream Classes?
8	Explain different types of access specifiers along with example program?
9	With an example program explain how to create user defined package and import in another class?
10	Write a java program to extend one interface from another interface?
11	Explain in detail about Console Class?
12	Explain the process of reading console input and writing console output with an example program?
13	Write a java program to implement serialization and deserialization?
14	Explain autoboxing in detail with an example program?
15	Write a java program to read from and write content to a file?
16	Explain File class in detail along with an example program?
17	Write a java program to implement DataInputStream and DataOutputStream?
UNIT 3	
1	Explain user-defined exception with an example program?
2	What is meant by re-throwing exception?Explain with an example program?
3	Does java support thread priorities?Justify your answer?
4	Describe producer-consumer problem using inter-thread communication?
5	What is an Exception?Explain Exception Handling mechanism with an example program?
6	Write a java program that illustrates the application of multiple catch statements?
7	Write a program that creates two threads First thread prints the numbers from 1 to 100 and other thread prints the numbers from 100 to 1?
8	Explain thread creation along with an example program?
9	Explain thread synchronization with the help of example program?
10	Write a java program to generate multiplication of two numbers using thread?
11	Explain thread class in detail along with an example?
12	Write a java program to implement nested try catch blocks?
13	Differentiate between throw and throws clause along with an example?
UNIT 4	
1	Write a java program to implement Stack class?
2	Explain Random collection class?
3	Differentiate between ArrayList and Vector?Explain
4	What is the difference between Iterator and ListIterator?Explain different ways to iterate over a list?
5	What are the best practices related to Java Collection Framework?Discuss
6	What is Comparable and Comparator interface?Differentiate between them?
7	Explain about StringTokenizer in detail?
8	Explain Collection algorithms in detail along with an example program?
9	Write a java program to implement Stack class methods?
10	Explain Vector class in detail?

11	Explain Dictionary Class along with an example program?
12	Explain TreeMap in detail along with an example?
13	Explain TreeSet class in detail along with an example?
14	Explain HashSet class in detail along wih an example?
15	Explain implementation of map using HashTable?
16	Explain calendar class in detail?
UNIT 5	
1	What is the significance of layout managers?Discuss briefly various layout managers?
2	Explain about JButton class along with an example?
3	Explain delegation event model?
4	Write an applet to draw a smiley picture and accept user name as a parameter and display welcome message?
5	What is the difference between init() and start() methods in an Applet?Explain through an example program?
6	Write the applets to draw Cube and Circle shapes?
7	Explain various layout managers?
8	Write a program to create a frame windows that responds to mouse clicks?
9	Explain Frame class along with an example program?
10	Differentiate between AWT and Swings?
11	Write a java program to create a registration form using AWT?
12	Explain MVC Architecture in detail?
13	Briefly Explain hierarchy of swing components?
14	What is an event? Mention the different event classes and event listener interfaces and write a java program to implement ActionListener interface?
15	Explain AdjustmentEvent class and its usage along with an example program?
16	Write a java program to implement Mouse Events?
17	Write a java program to implement Adapter Class?
18	Write a java program to implement Anonymous Inner class?
19	Explain the life cycle methods of Applet with an example program?
20	Write a java program to display an image using ImageIcon class?
21	Explain JScrollPane along with an example program?
22	Explain JTabbedPane along with an example program?
23	Discuss in detail about MenuBars,Menus and Dialog boxes?
24	Explain about JAPPLET in detail?
25	Explain the steps involved increating JCheckBox,JLabel,JRadioButton,JButton?

UNIT-1

Short Answer Question for 2 Marks:

1.What are the features of Java Language?

1. Simple
2. Object-Oriented
3. Portable
4. Platform independent
5. Secured
6. Robust
7. Architecture neutral
8. Interpreted
9. High Performance
10. Multithreaded
11. Distributed
12. Dynamic

2.What is Data Abstraction?

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only important things to the user and hides the internal details for example sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery.

Abstraction lets you focus on what the object does instead of how it does it.

There are two ways to achieve abstraction in java

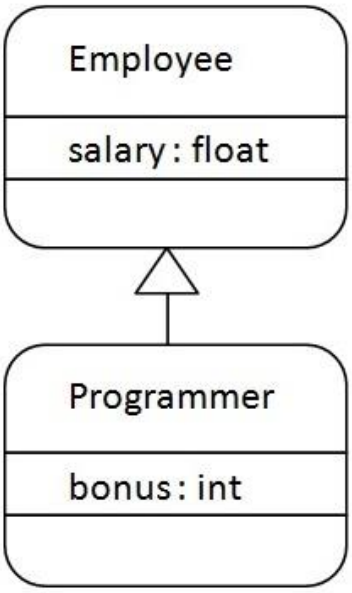
Abstract class (0 to 100%)

Interface (100%)

3.What is inheritance?Give example?

- **Inheritance in java** is a mechanism in which one object acquires all the properties and behaviors of parent object.

The idea behind inheritance in java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of parent class, and you can add new methods and fields also.



4.What is an abstract class?Give example?

A class which contains the abstract keyword in its declaration is known as abstract class.Abstract classes may or may not contain abstract methods, i.e., methods without body (public void get();)

But, if a class has at least one abstract method, then the class must be declared abstract.If a class is declared abstract, it cannot be instantiated. To use an abstract class, you have to inherit it from another class, provide implementations to the abstract methods in it.

5.Write about different forms of inheritance?

- Inheritance is used in a variety of way and for a variety of different purposes .
    - Inheritance for Specialization
    - Inheritance for Specification
    - Inheritance for Construction
    - Inheritance for Extension
    - Inheritance for Limitation
    - Inheritance for Combination

6.What are the benefits of inheritance?

- Software Reusability (among projects)
- Increased Reliability (resulting from reuse and sharing of well-tested code)
- Code Sharing (within a project)
- Consistency of Interface (among related objects)
- Software Components

7.Write about Object class?

- The **Object class** is the parent class of all the classes in java by default. In other words, it is the topmost class of java.
- The Object class is beneficial if you want to refer any object whose type you don't know. Notice that parent class reference variable can refer the child class object, know as upcasting.
- Let's take an example, there is getObject() method that returns an object but it can be of any type like Employee,Student etc, we can use Object class reference to refer that object. For example:
- Object obj=getObject();

Short Answer Questions for 3 Marks:

8..List String manipulation functions of Java String Class?

concat() method

String str = "Java";

String str2 = s.concat(str);

equals() method compares two strings for equality. Its general syntax is,

boolean equals (Object str)

By compareTo() method

compareTo() method compares values and returns an int which tells if the string compared is

less than, equal to or greater than the other string. It compares the String based on natural

Ordering i.e alphabetically.

Its general syntax is,

int compareTo(String str)

charAt()

charAt() function returns the character located at the specified index.

equalsIgnoreCase() determines the equality of two Strings, ignoring thier case (upper or lower case doesn't matters with this fuction).

replace() method replaces occurances of character with a specified new character.

String str = "Change me"; System.out.println(str.replace('m','M')); Output : Change Me

toLowerCase() method returns string with all uppercase characters converted to lowercase.

String str = "ABCDEF"; System.out.println(str.toLowerCase()); Output : abcdef

toUpperCase()

This method returns string with all lowercase character changed to uppercase.

String str = "abcdef";

System.out.println(str.toUpperCase()); Output : ABCDEF
trim()

This method returns a string from which any leading and trailing whitespaces has been removed.

String str = " hello "; System.out.println(str.trim()); Output : hello

9..Write about Object Oriented Programming Characteristics?

Object-Oriented Programming (OOP) is a programming language model organized around objects rather than “actions”.

The following are the characteristics of Object Oriented Programming:

Abstraction:

Abstraction is the concept of exposing only the required essential characteristics and behavior with respect to a context.

Encapsulation:

Encapsulation in java is a process of wrapping code and data together into a single unit, for example capsule i.e. mixed of several medicines.

Polymorphism:

The ability of different objects to respond, each in its own way, to identical messages is called polymorphism.When one task is performed by different ways i.e. known as polymorphism.

Inheritance:

Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object.

10.Explain the use of for statement in java with an example?

for each variation in java is used to iterate the elements of the array.

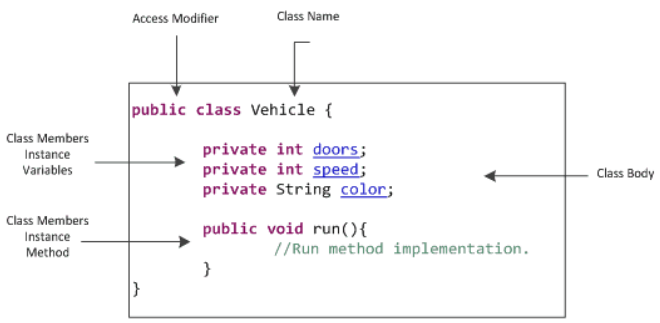
EnhancedForLoop - Notepad

File Edit Format View Help

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

     int[] numbers = {3, 4, 5, -5, 0, 12};
     int sum = 0;

      for (int number: numbers) {
         sum += number;
      }

      System.out.println("Sum = " + sum);
   }
}
```

```
E:\oops>start notepad EnhancedForLoop.java

E:\oops>javac EnhancedForLoop.java

E:\oops>java -cp . EnhancedForLoop
Sum = 19
```

11.Differentiate between objects and classes?Write an example program?

S.NO	Class	Object
1	A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.	An object stores its state in fields called variables and its behavior through methods.An Object is an instance of class.
2		**Vehicle v1=new Vehicle();**

3	Student - Notepad
	File Edit Format View Help
	class Student{
	int id;//field or data member or instance variable
	String name;
	String yclass="2nd IT B";
	public static void main(String args[]){
	Student s1=new Student();//creating an object of Student
	System.out.println(s1.id);//accessing member through reference
	System.out.println(s1.name);
	System.out.println(s1.yclass);
	}
	}

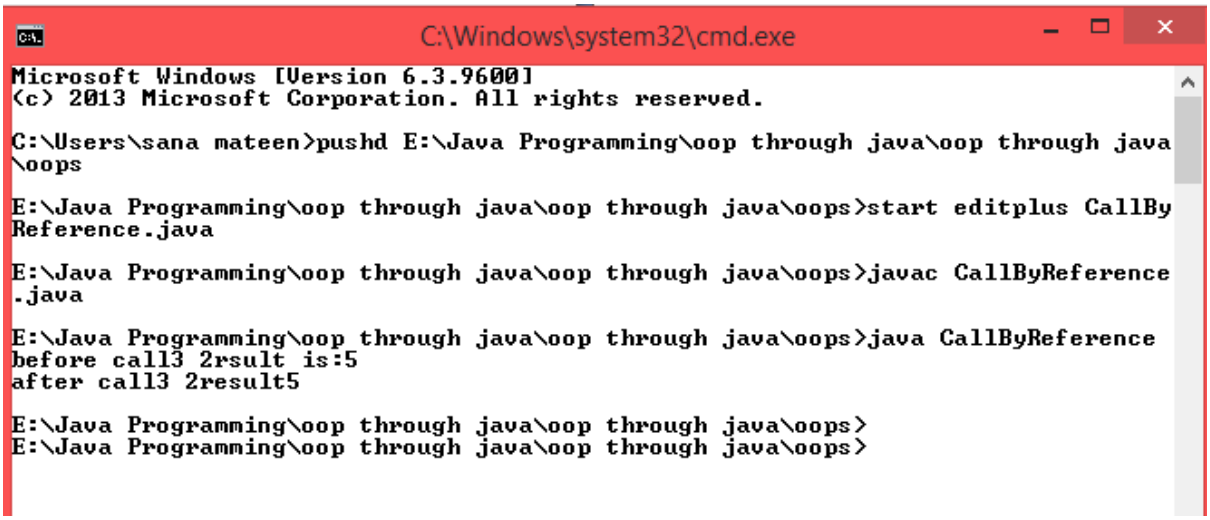
4	C:\Users\sana mateen\Desktop\oops>javac Student.java C:\Users\sana mateen\Desktop\oops>java Student 0 null 2nd IT B

12.Write a Java Program to differentiate between print() and println() ?

E:\Java Programming\oop through java\oop through java\oops\CallByReference.j

Search Document Project Tools Browser Emmet Window Help

```
1  class Add{
2  int a,b,c;
3  Add(int i,int j){
4  a=i;
5  b=j;
6  }
7  int  addition(Add o){
8  c=o.a+o.b;
9  System.out.println("rsult is:"+c);
10 return c;
11 }
12 }
13 class CallByReference{
14 public static void main(String[] ar){
15 Add am=new Add(3,2);
16 System.out.print("before call"+am.a+" "+am.b);
17 int c=am.addition(am);
18 System.out.println("after call"+am.a+" "+am.b+"result"+c);
19 }
20 }
```



13.Differentiate between break and continue statement with the help of program?

Break	**Continue**
The break statement terminates the loop immediately, and the control of the program moves to the next statement following the loop.	**The continue statement skips the current iteration of a loop (for, while, and do...while loop).**
public class MyClass { **public static void main(String[] args) {** **for (int i = 0; i < 10; i++) {** **if (i == 4) {** **break;** **}** **System.out.println(i);** **}** **}** **}**	**public class MyClass {** **public static void main(String[] args) {** **for (int i = 0; i < 10; i++) {** **if (i == 4) {** **continue;** **}** **System.out.println(i);** **}** **}** **}**

14.Write a Java Program to implement typecasting?

The process of converting the value of one data type (int, float, double, etc.) to another data type is known as typecasting.

Program:

```
class TypeCast {
  public static void main(String[] args) {
    // create int type variable
    int num = 10;
    System.out.println("The integer value: " + num);


    // convert into double type
    double data = num;
    System.out.println("The double value: " + data);
  }
}
```

Output:

The integer value: 10

The double value: 10.0

15.Write a Java Program to implement method overloading?

Method Overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different.

```
public class Area
{
  public void calculate()                        // I
  {
    System.out.println("Nothing to calculate");
  }
  public void calculate(int x)                   // II
  {
    System.out.println("Circle Area: " + Math.PI * Math.pow(x, 2));
  }
  public void calculate(double x)                // III
  {
    System.out.println("Circle Perimeter: " + 2 * Math.PI * x);
  }
  public void calculate(int x, int y)            // IV
  {
    System.out.println("Rectangle Area: " +x*y);
  }
  public static void main(String args[])
  {
    Area a1 = new Area();
    a1.calculate();                 // I
    a1.calculate(10, 20);           // IV
    a1.calculate(10.5);             // III
    a1.calculate(10);               // II
  }
```

```
c:\Users\sana mateen\Desktop\oops>start notepad Area.java

c:\Users\sana mateen\Desktop\oops>javac Area.java

c:\Users\sana mateen\Desktop\oops>java Area
Nothing to calculate
Rectangle Area: 200
Circle Perimeter: 65.97344572538566
Circle Area: 314.1592653589793
```

16.What is super keyword?Mention its usage along with example program?

The super keyword in Java is a reference variable which is used to refer immediate parent class object.

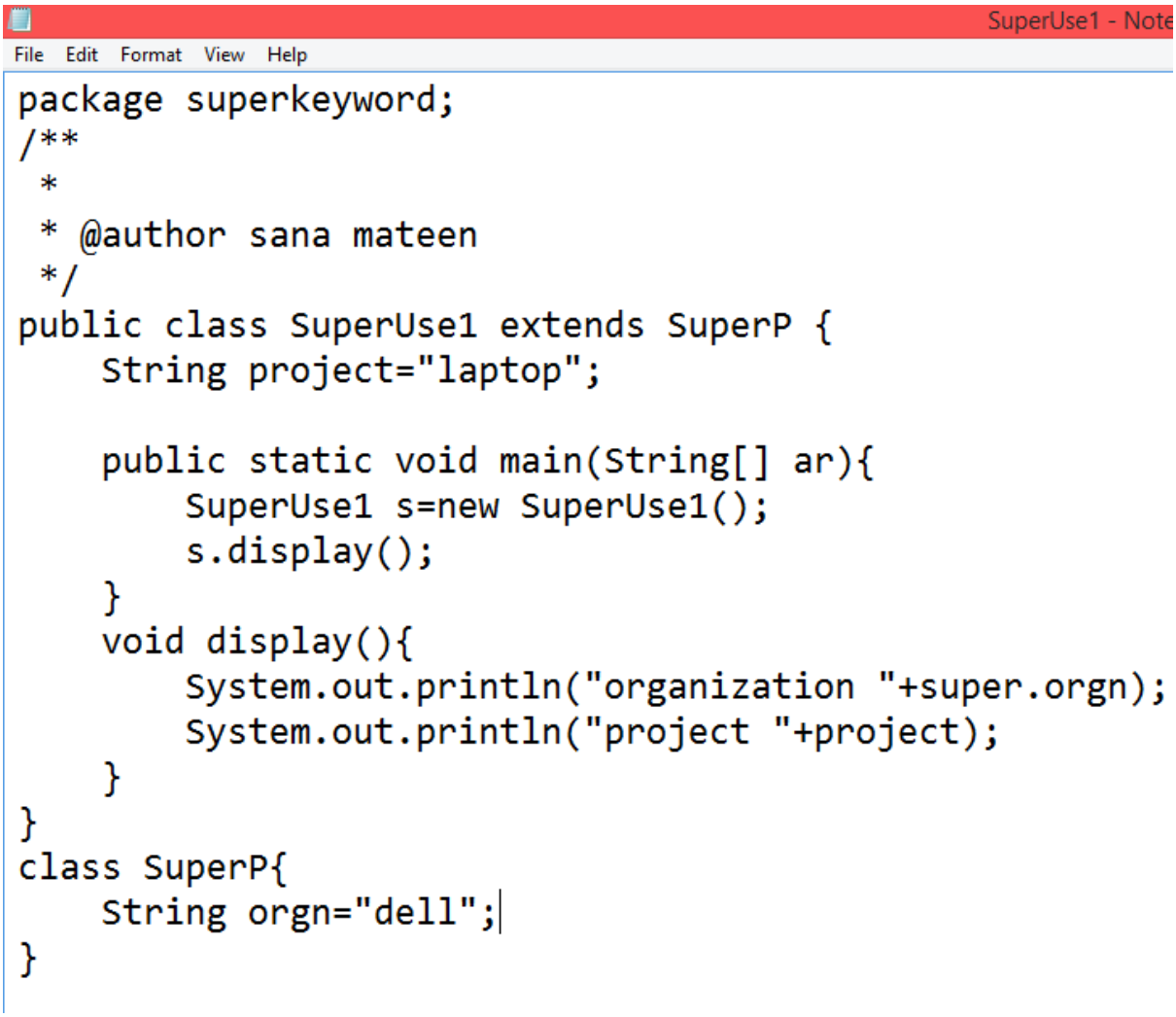
Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword

super can be used to refer immediate parent class instance variable.

super can be used to invoke immediate parent class method.

super() can be used to invoke immediate parent class constructor.



17.Write about final keyword and its use along with example program?

The final keyword in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

- variable
- method
- class

Program:

```
class Bike9{
 final int speedlimit=90;//final variable
 void run(){
  speedlimit=400;
 }
 public static void main(String args[]){
 Bike9 obj=new  Bike9();
 obj.run();
 }
}
```

Output:

Compile Time Error

Long Answer Questions for 5 Marks:

1 Explain the object oriented programming characteristics?

Object-oriented programming: As the name suggests, Object-Oriented Programming or OOPs refers to languages that uses objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

The following are the characteristics of Object Oriented Programming:

Abstraction:

Abstraction is the concept of exposing only the required essential characteristics and behavior with respect to a context.

Hiding of data is known as **data abstraction**. In object oriented programming language this is implemented automatically while writing the code in the form of class and object.

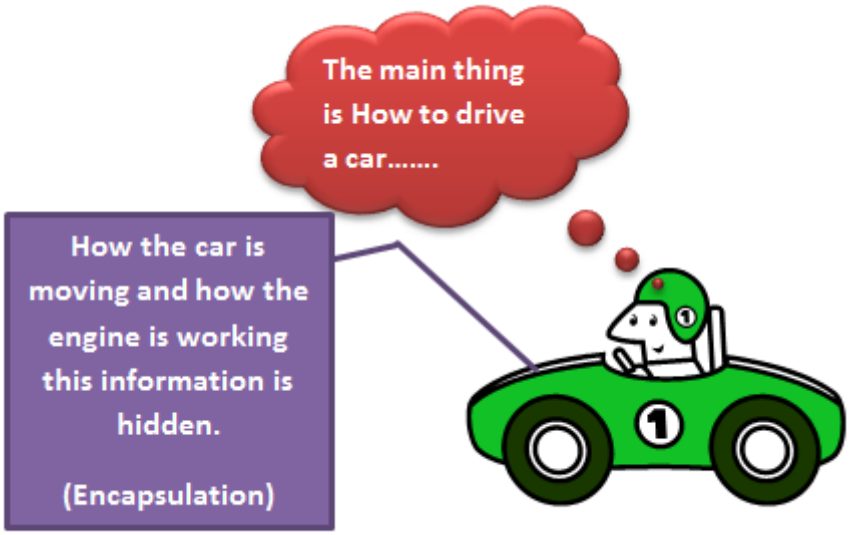
Real Life Example of Abstraction in Java

Abstraction shows only important things to the user and hides the internal details, for example, when we ride a bike, we only know about how to ride bikes but can not know about how it work? And also we do not know the internal functionality of a bike.

Encapsulation:

Encapsulation in java is a process of wrapping code and data together into a single unit, for example capsule i.e. mixed of several medicines.

It is a technique of making the fields in a class private and providing access to fields via public methods

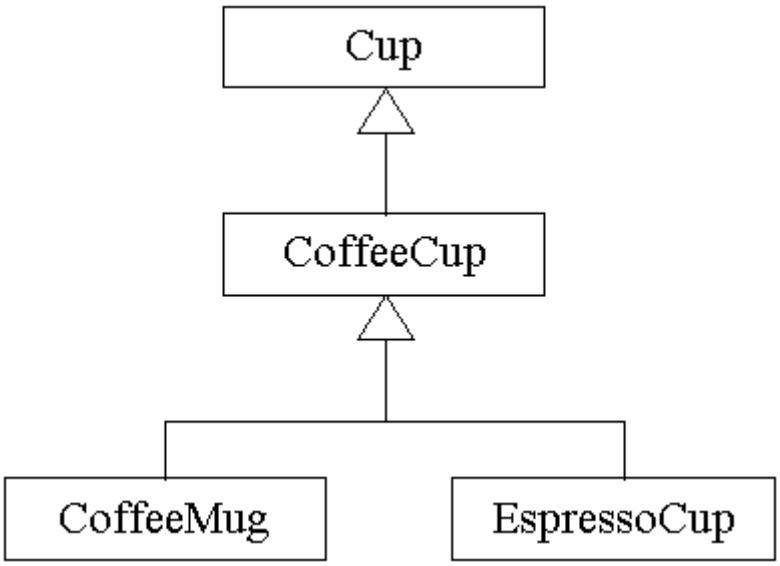


Polymorphism:

It is the concept of **object oriented** programming.The ability of different objects to respond, each in its own way, to identical messages is called **polymorphism.**

When **one task is performed by different ways** i.e. known as polymorphism.

poly mean many morph mean form.



Inheritance:

Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object.

The idea behind inheritance in java is that you can create new classes that are built upon existing classes.

When you inherit from an existing class, you can reuse methods and fields of parent class, and you can add new methods and fields also.

Inheritance represents the **IS-A relationship**, also known as *parent-child* relationship.

2 What is the usage of enumerated data type?Give example?

The Enum in Java is a data type which contains a fixed set of constants.

It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, and SATURDAY) , directions (NORTH, SOUTH, EAST, and WEST), season (SPRING, SUMMER, WINTER, and AUTUMN or FALL), colors (RED, YELLOW, BLUE, GREEN, WHITE, and BLACK) etc. According to the Java naming conventions, we should have all constants in capital letters. So, we have enum constants in capital letters.

Java Enums can be thought of as classes which have a fixed set of constants (a variable that does not change). The Java enum constants are static and final implicitly.

Uses:

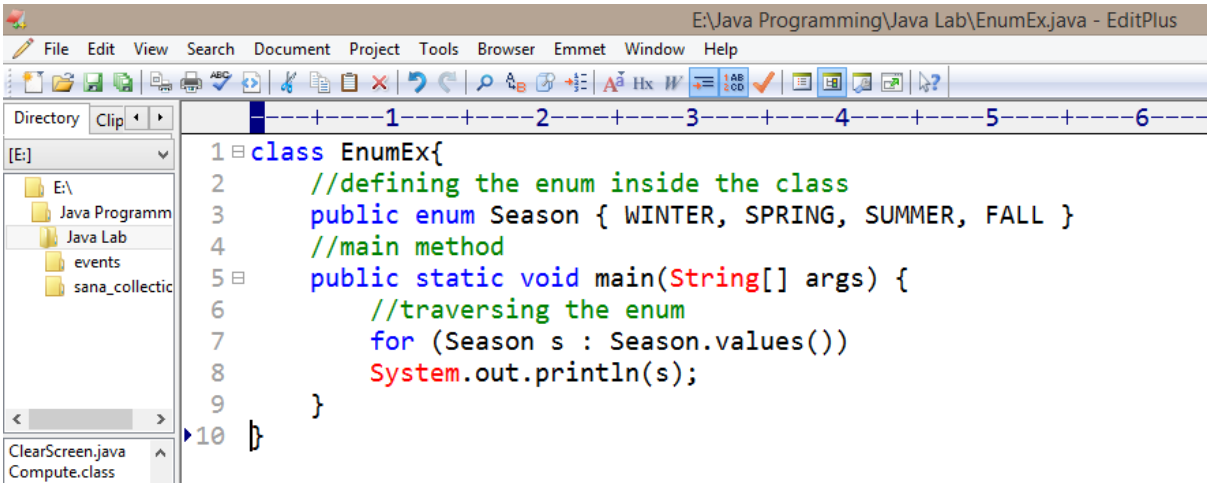
Enums are used to create our own data type like classes. The enum data type (also known as Enumerated Data Type) is used to define an enum in Java. Unlike C/C++, enum in Java is more powerful. Here, we can define an enum either inside the class or outside the class.

Java Enum internally inherits the Enum class, so it cannot inherit any other class, but it can implement many interfaces. We can have fields, constructors, methods, and main methods in Java enum.

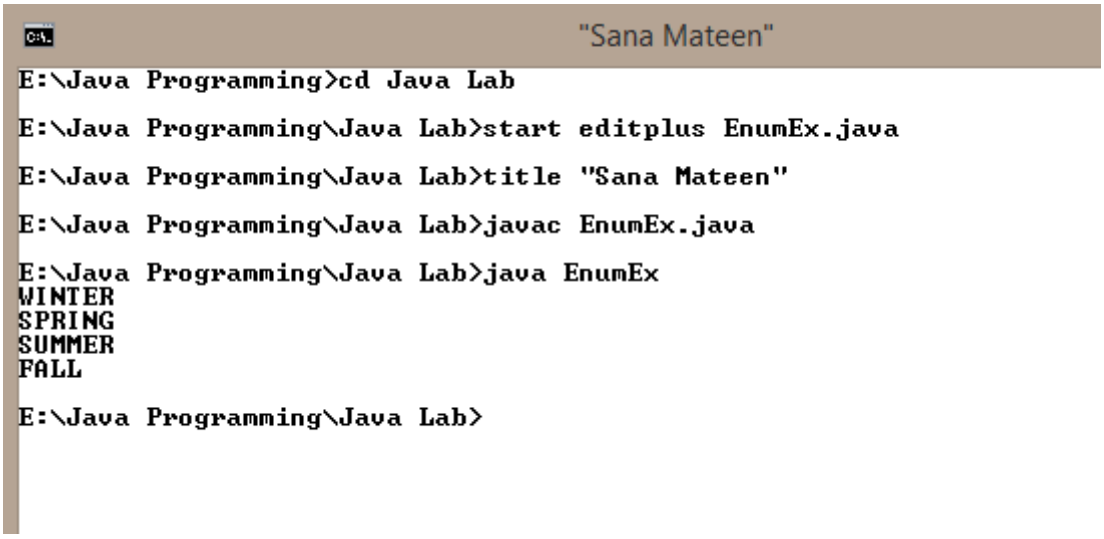
Points to for Java Enum

- Enum improves type safety
- Enum can be easily used in switch
- Enum can be traversed
- Enum can have fields, constructors and methods
- Enum may implement many interfaces but cannot extend any class because it internally extends Enum class.

Example Program:



Output:



3 Explain Final Keyword and it's usage along with example program?

- The keyword final has three uses
- It can be used to create the equivalent of a named constant

Program:

```
class FinExam{
final int marks=40;
void pass()
{
marks=35;
}
public static void main(String[] args)
{
FinExam fo=new FinExam();
fo.pass();
}
}
```

Output:

```
C:\Users\sana mateen>cd desktop

C:\Users\sana mateen\Desktop>cd oops

C:\Users\sana mateen\Desktop\oops>start notepad FinExam.java

C:\Users\sana mateen\Desktop\oops>javac FinExam.java
FinExam.java:5: error: cannot assign a value to final variable marks
marks=35;
^
1 error

C:\Users\sana mateen\Desktop\oops>
```

- Using final to prevent overriding

Program:

```
class SuperE{
        int x;
        SuperE(int x)
        {
                this.x=x;
        }
        final void display()
        {
                System.out.println("Super var x= "+x);
        }
}
class SubE extends SuperE{
        int y;
        SubE(int x,int y)
        {
                super(x);
                this.y=y;
        }
        void display()
        {
                System.out.println("Super var x= "+x);
                System.out.println("Sub class var y= "+y);

        }
}
class FinExam2
{
        public static void main(String[] args)
        {
                SubE s=new SubE(5,10);
                s.display();
        }
}
```

Output:

```
C:\Users\sana mateen\Desktop\oops>javac FinExam2.java
FinExam2.java:19: error: display() in SubE cannot override display() in SuperE
        void display()
             ^
  overridden method is final
1 error
```

- Using final to prevent inheritance

Program:

```
final class SuperE{
        int x;
        SuperE(int x)
        {
                this.x=x;
        }
         void display()
        {
                System.out.println("Super var x= "+x);
        }
}
class SubE extends SuperE{
        int y;
        SubE(int x,int y)
        {
                super(x);
                this.y=y;
        }
        void display()
        {
                System.out.println("Super var x= "+x);
                System.out.println("Sub class var y= "+y);

        }
}
class FinExam2
{
        public static void main(String[] args)
        {
                SubE s=new SubE(5,10);
                s.display();
        }
}
```

Output:

```
C:\Users\sana mateen\Desktop\oops>javac FinExam2.java
FinExam2.java:12: error: cannot inherit from final SuperE
class SubE extends SuperE{
                   ^
1 error
```

4 With the help of program explain the usage of super keyword?

- The **super** keyword in java is a reference variable which is used to refer immediate parent class object.
- Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.
- Usage of java super Keyword
- super can be used to refer immediate parent class instance variable.
- super can be used to invoke immediate parent class method.
- super() can be used to invoke immediate parent class constructor.

Program1:

suse - Notepad

File Edit Format View Help

```
class Job{
String orgnztn="dell";
}
class Project extends Job{
String pname="smartphone";
void display(){
System.out.println(pname);
System.out.println(super.orgnztn);
}
}
class suse{
public static void main(String[] args)
{
Project p=new Project();
p.display();
}
}
```

Output:

```
E:\oops>javac suse.java

E:\oops>java -cp . suse
smartphone
dell
```

Program 2:

Testp - Notepad

File Edit Format View Help

```
/* Base class Person */
class Person
{
    void message()
    {
        System.out.println("This is person class");
    }
}

/* Subclass Student */
class Student extends Person
{
    void message()
    {
        System.out.println("This is student class");
    }

    // Note that display() is only in Student class
    void display()
    {
        // will invoke or call current class message() method
        message();

        // will invoke or call parent class message() method
        super.message();
    }
}

/* Driver program to test */
class Testp
{
    public static void main(String args[])
    {
        Student s = new Student();

        // calling display() of Student
        s.display();
    }
}
```

Output:

```
E:\oops>java -cp . Testp
This is student class
This is person class
```

Program 3:

```
class Person
{
    Person()
    {
        System.out.println("Person class Constructor");
    }
}

/* subclass Student extending the Person class */
class Student extends Person
{
    Student()
    {
        // invoke or call parent class constructor
        super();

        System.out.println("Student class Constructor");
    }
}

/* Driver program to test*/
class Testpe
{
    public static void main(String[] args)
    {
        Student s = new Student();
    }
}
```

Output:

```
E:\oops>java -cp . Testpe
Person class Constructor
Student class Constructor
```

5 Explain about polymorphism in detail?

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

There are two types of polymorphism in java: compile time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

Compile time polymorphism

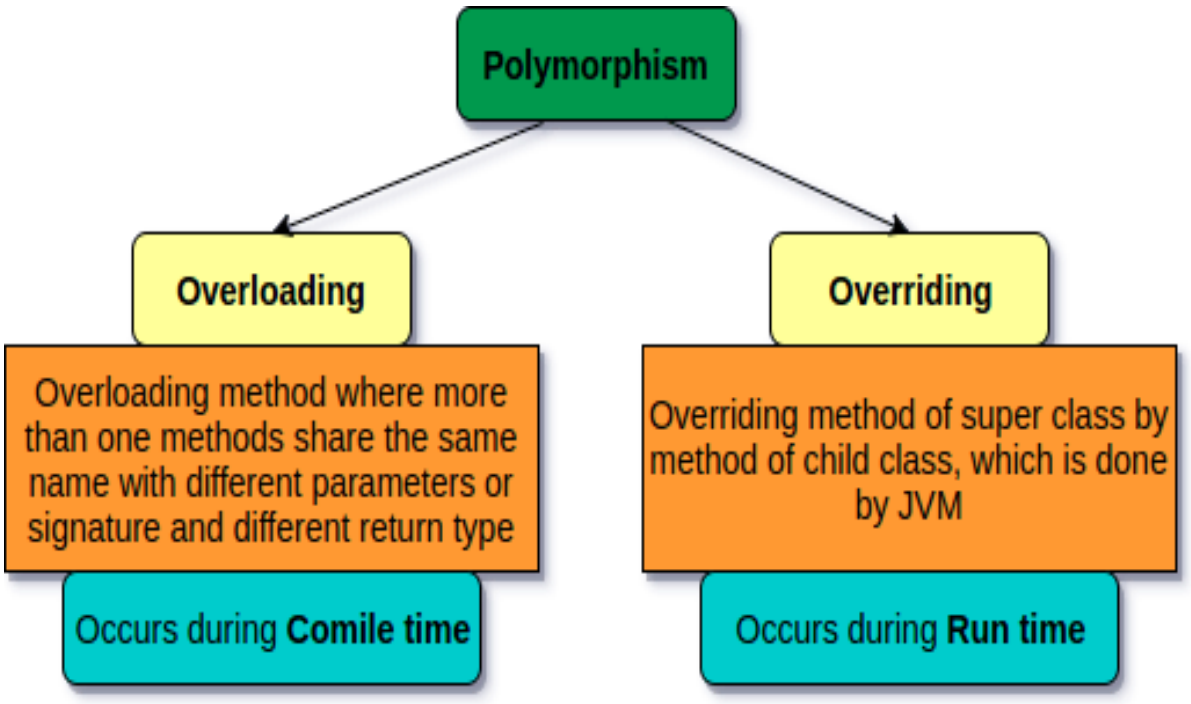
If a method call is resolved at compile time that it is called static binding or earlier binding or static polymorphism or compile time polymorphism.

Method overloading and method overriding by using static method, private method and final method are example for static polymorphism.

Runtime polymorphism

If a method call is resolved at the time of execution it is called dynamic binding or late binding or run time polymorphism.

Example- Method overloading and method overriding by using instance method are example for dynamic binding.



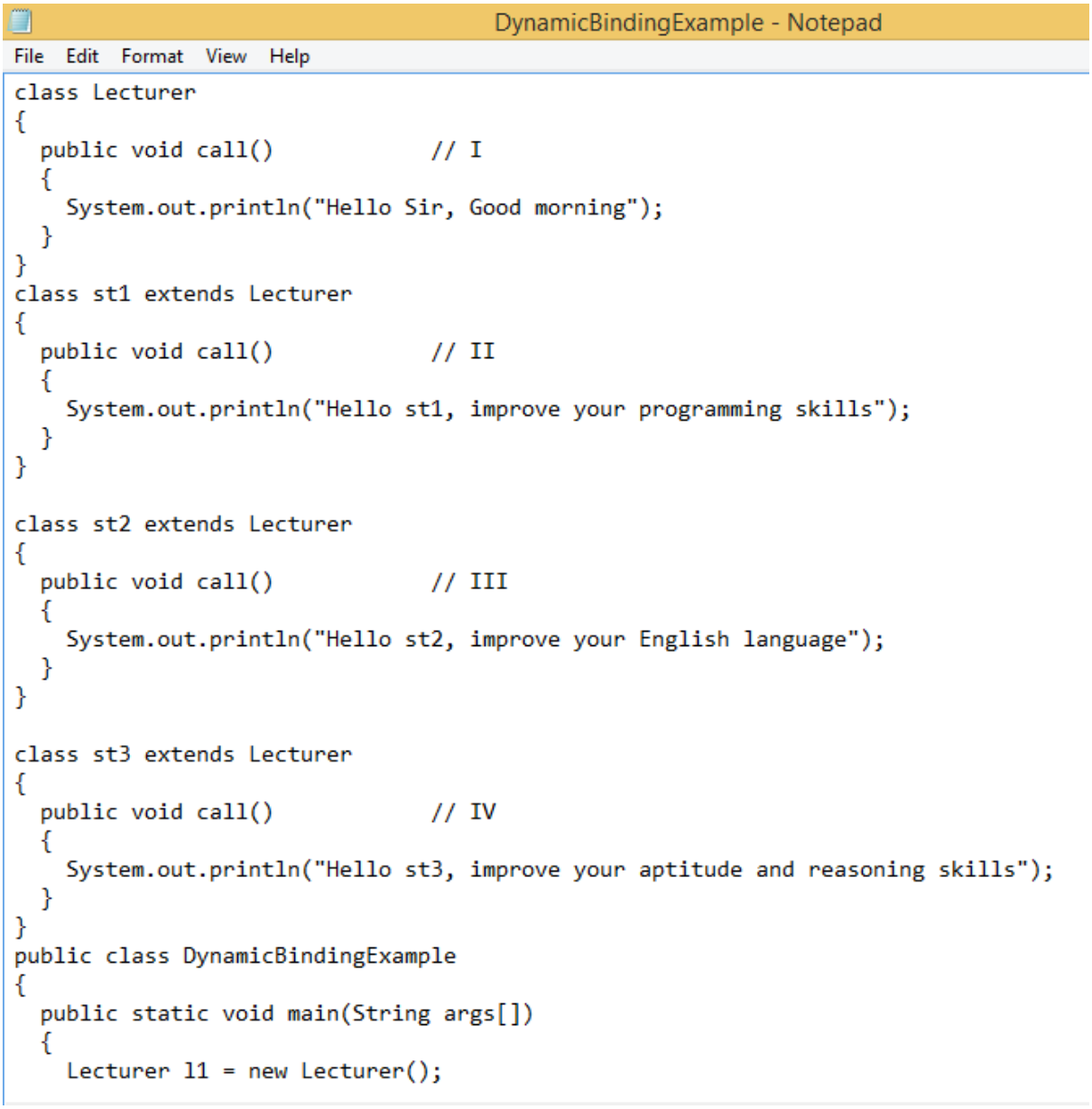
Compile Time Polymorphism:

```
public class Area
{
  public void calculate()                          // I
  {
    System.out.println("Nothing to calculate");
  }
  public void calculate(int x)                     // II
  {
    System.out.println("Circle Area: " + Math.PI * Math.pow(x, 2));
  }
  public void calculate(double x)                  // III
  {
    System.out.println("Circle Perimeter: " + 2 * Math.PI * x);
  }
  public void calculate(int x, int y)              // IV
  {
    System.out.println("Rectangle Area: " +x*y);
  }
  public static void main(String args[])
  {
    Area a1 = new Area();
    a1.calculate();                 // I
    a1.calculate(10, 20);           // IV
    a1.calculate(10.5);             // III
    a1.calculate(10);               // II
  }
```

Output:

```
c:\Users\sana mateen\Desktop\oops>start notepad Area.java

c:\Users\sana mateen\Desktop\oops>javac Area.java

c:\Users\sana mateen\Desktop\oops>java Area
Nothing to calculate
Rectangle Area: 200
Circle Perimeter: 65.97344572538566
Circle Area: 314.1592653589793
```

Runtime Polymorphism:

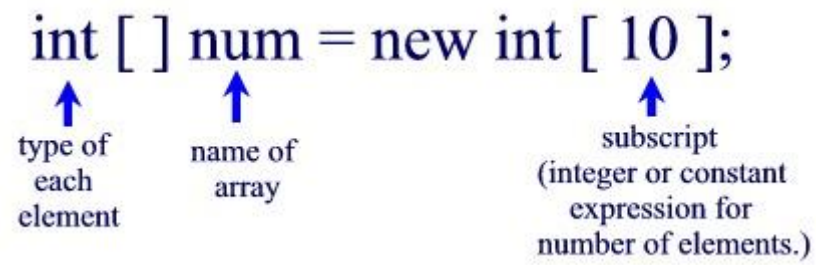

DynamicBindingExample - Notepad
File Edit Format View Help
class Lecturer
{
public void call() // I
{
System.out.println("Hello Sir, Good morning");
}
}
class st1 extends Lecturer
{
public void call() // II
{
System.out.println("Hello st1, improve your programming skills");
}
}
class st2 extends Lecturer
{
public void call() // III
{
System.out.println("Hello st2, improve your English language");
}
}
class st3 extends Lecturer
{
public void call() // IV
{
System.out.println("Hello st3, improve your aptitude and reasoning skills");
}
}
public class DynamicBindingExample
{
public static void main(String args[])
{
Lecturer l1 = new Lecturer();
l1.call(); // I
st1 r1 = new st1();
l1 = r1;
l1.call(); // II
st2 p1 = new st2();
l1 = p1;
l1.call(); // III
st3 j1 = new st3();
l1 = j1;
l1.call(); // IV
}
}

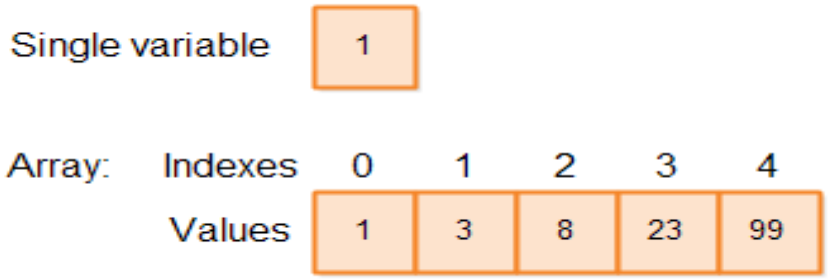
Output:

```
c:\Users\sana mateen\Desktop\oops>start notepad DynamicBindingExample.java

c:\Users\sana mateen\Desktop\oops>javac DynamicBindingExample.java

c:\Users\sana mateen\Desktop\oops>java DynamicBindingExample
Hello Sir, Good morning
Hello st1, improve your programming skills
Hello st2, improve your English language
Hello st3, improve your aptitude and reasoning skills
```

6 What is an array?How do you declare an array in java?Give examples?

- An Array is a group of similar type of variables referred to by common name.





- **Declaring an Array Variable in Java**
- A Java array variable is declared just like you would declare a variable of the desired type, except you add [] after the type.
- int[] intArray;

```
int[] intArray;
int   intArray[];

String[] stringArray;
String   stringArray[];
```

- When you declare a Java array variable you only declare the variable (reference) to the array itself. The declaration does not actually create an array. You create an array like this:
- int[] intArray;
- **intArray = new int[10];**

Example Program 1:

```
class IterateArray{
    public static void main(String[] ar){
        String[] nam=new String[10];
        for(int i=0;i<nam.length;i++)
        {
            nam[i]="string"+i;
        }
        for(int i=0;i<nam.length;i++)
        {
            System.out.println(nam[i]);
        }
    }
}
```

Output:

```
C:\Users\sana mateen\Desktop\oops>javac IterateArray.java

C:\Users\sana mateen\Desktop\oops>java IterateArray
string0
string1
string2
string3
string4
string5
string6
string7
string8
string9
```

Example Program 2:

```
class ForEachArrayIteration{
    public static void main(String[] gs)
    {
        int n=Integer.parseInt(gs[0]);
        int[] num=new int[n];
        for(int i=0; i < num.length; i++)
        {
        num[i] = i;
        }
        for(int i:num)
        {

            System.out.println(i);
        }
    }
}
```

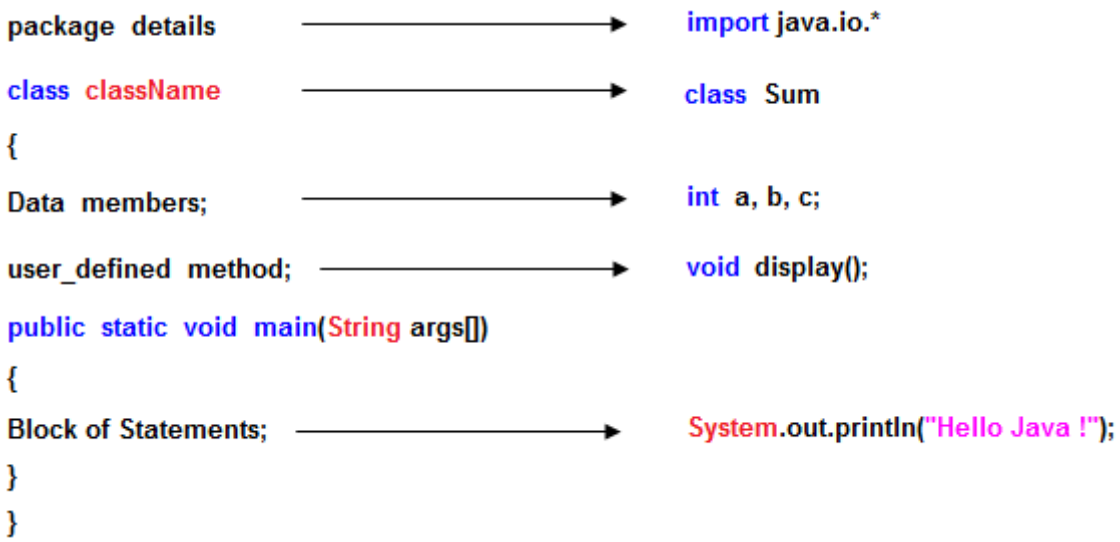
Output:

```
C:\Users\sana mateen\Desktop\oops>javac ForEachArrayIteration.java

C:\Users\sana mateen\Desktop\oops>java ForEachArrayIteration 4
0
1
2
3
```

7 Explain the structure of java program with an example?

A typical structure of a Java program contains the following elements

- Package declaration
- Import statements
- Comments
- Class definition
- Class variables, Local variables
- Methods/Behaviors



A package is a collection of classes, interfaces and sub-packages. A sub package contains collection of classes, interfaces and sub-sub packages etc. java.lang.*; package is imported by default and this package is known as default package.

Class is keyword used for developing user defined data type and every java program must start with a concept of class.

"ClassName" represent a java valid variable name treated as a name of the class each and every class name in java is treated as user-defined data type.

Data member represents either instance or static they will be selected based on the name of the class.

User-defined methods represents either instance or static they are meant for performing the operations either once or each and every time.

Each and every java program starts execution from the main() method. And hence main() method is known as program driver.

Since main() method of java is not returning any value and hence its return type must be void.

Since main() method of java executes only once throughout the java program execution and hence its nature must be static.

Since main() method must be accessed by every java programmer and hence whose access specifier must be public.

Each and every main() method of java must take array of objects of String.

Block of statements represents set of executable statements which are in term calling user-defined methods are containing business-logic.

The file naming conversion in the java programming is that which-ever class is containing main() method, that class name must be given as a file name with an extension .java.

8 Explain the importance of Java Virtual Machine?

The Java Virtual Machine is a program whose purpose is to execute other programs.

What the JVM is used for

The JVM has two primary functions: to allow Java programs to run on any device or operating system (known as the "Write once, run anywhere" principle), and to manage and optimize program memory. When Java was released in 1995, all computer programs were written to a specific operating system, and program memory was managed by the software developer. So the JVM was a revelation.

1. Ensures Security

When Java was initially developed, security was the priority. That is the reason why Java programs run separated in an enclosed area of the Java Virtual Machine which acts as a protecting shield.

Java Virtual Machine allows developers to write Java programs that are highly secure with help of its built-in security features.

Java Virtual Machine keeps the Operating System secure by preventing any malicious software from attacking it. It ensures its safety by not allowing the Java applications to interact with the Operating System resources.

2. Cross Platform

A program that is Cross platform has the capability to run successfully on different types of hardware. Java is also a cross platform language i.e. it is possible to run a single piece of code written on a particular hardware on any other hardware that has JVM installed on it.

Therefore, JVM makes Java a cross platform language. A browser is not always needed for a Java program to execute, there are many java apps available too for the same. They run on the desktop the same way as the regular programs do.

3. Just-in-time compiler

The Java Virtual Machine comes with a Just-in-time compiler which converts the Java code into a low level machine language that can run as fast as the regular applications.

This compiled code goes into the cached memory of the browser, this means that one can use the code again without downloading it again and again compiling it.

9 Briefly explain the different data types used in java along with an example?

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

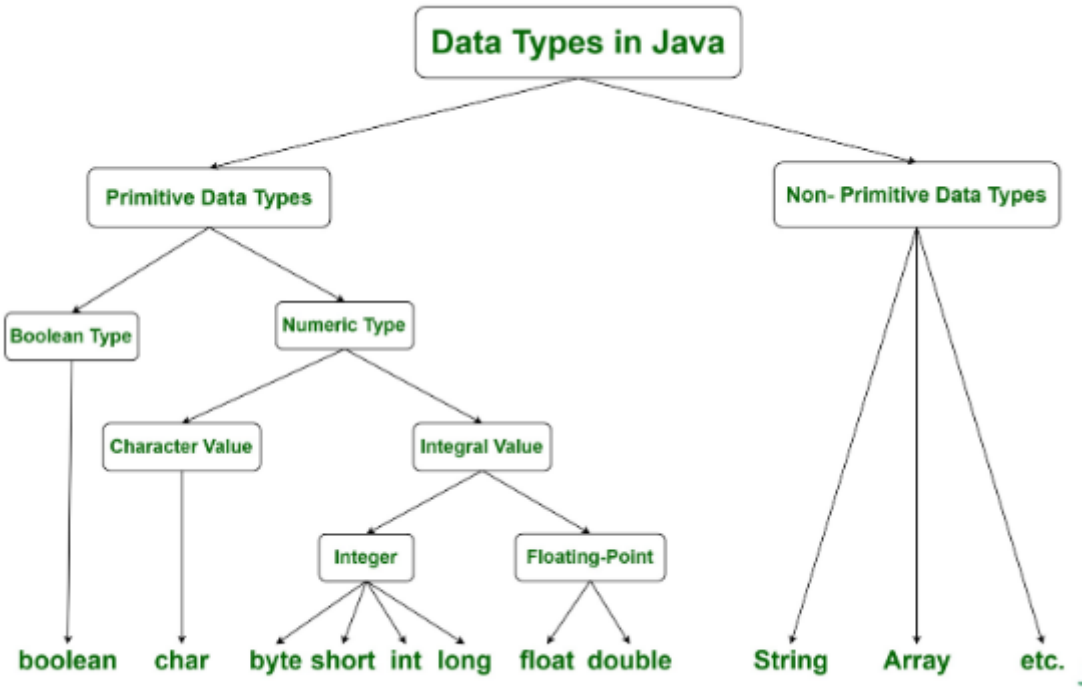
Primitive data types: The primitive data types include boolean, char, byte, short, int, long, float and double.

Non-primitive data types: The non-primitive data types include Classes, Interfaces, and Arrays.

Java Primitive Data Types

In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.

Java is a statically-typed programming language. It means, all variables must be declared before its use. That is why we need to declare variable's type and name.



Example Program:

```
class add
{
    public static void main(String[] args)
    {
        int a=3,b=4,c;
        c=a+b;
        System.out.println("result is "+c);
    }
}
```

Output:

```
"Sana Mateen"
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\sana mateen>pushdE:\Java Programming\oop through java\oop through jav
oops
The filename, directory name, or volume label syntax is incorrect.

C:\Users\sana mateen>pushd E:\Java Programming\oop through java\oop through ja
\oops

E:\Java Programming\oop through java\oop through java\oops>title "Sana Mateen"

E:\Java Programming\oop through java\oop through java\oops>javac add.java

E:\Java Programming\oop through java\oop through java\oops>java add
result is 7

E:\Java Programming\oop through java\oop through java\oops>
```

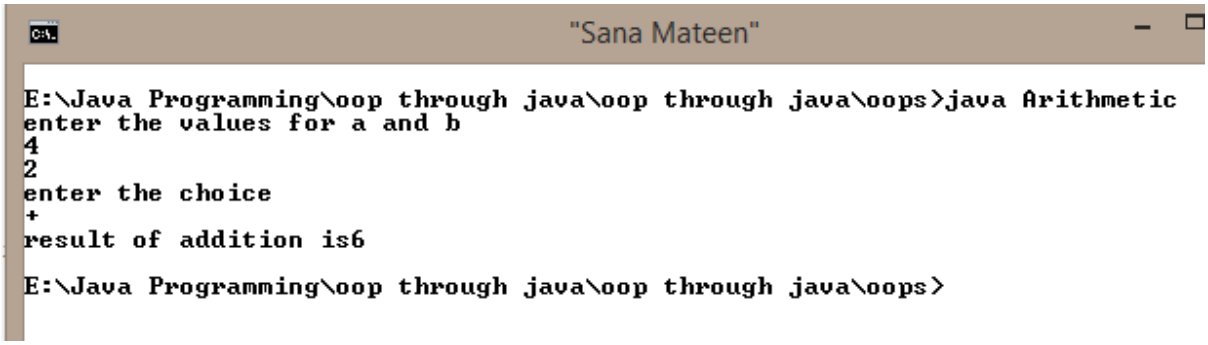
Java Non Primitive Data Types:

The non-primitive data types include Classes, Interfaces, and Arrays.

Example Program:

```
import java.io.*;
class Arithmetic
{
    public static void main(String[] args) throws IOException{
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("enter the values for a and b");
        int a=Integer.parseInt(br.readLine());
        int b=Integer.parseInt(br.readLine());
        System.out.println("enter the choice");

        char ch=(char)br.read();
        int c;
        switch(ch){
            case '+':
                c=a+b;
                System.out.println("result of addition is"+c);
                break;
            case '-':
                c=a-b;
                System.out.println("result of subtraction is"+c);
                break;
            case '*':
                c=a*b;
                System.out.println("result of multiplication is"+c);
                break;
            case '/':
                c=a/b;
                System.out.println("result of division is"+c);
                break;
            case '%':
                c=a%b;
                System.out.println("result of modulus operation is"+c);
                break;
            default:
                System.out.println("try again");
                break;      }
    }
}
```

Output:



10 Explain the difference between method overloading and method overriding?

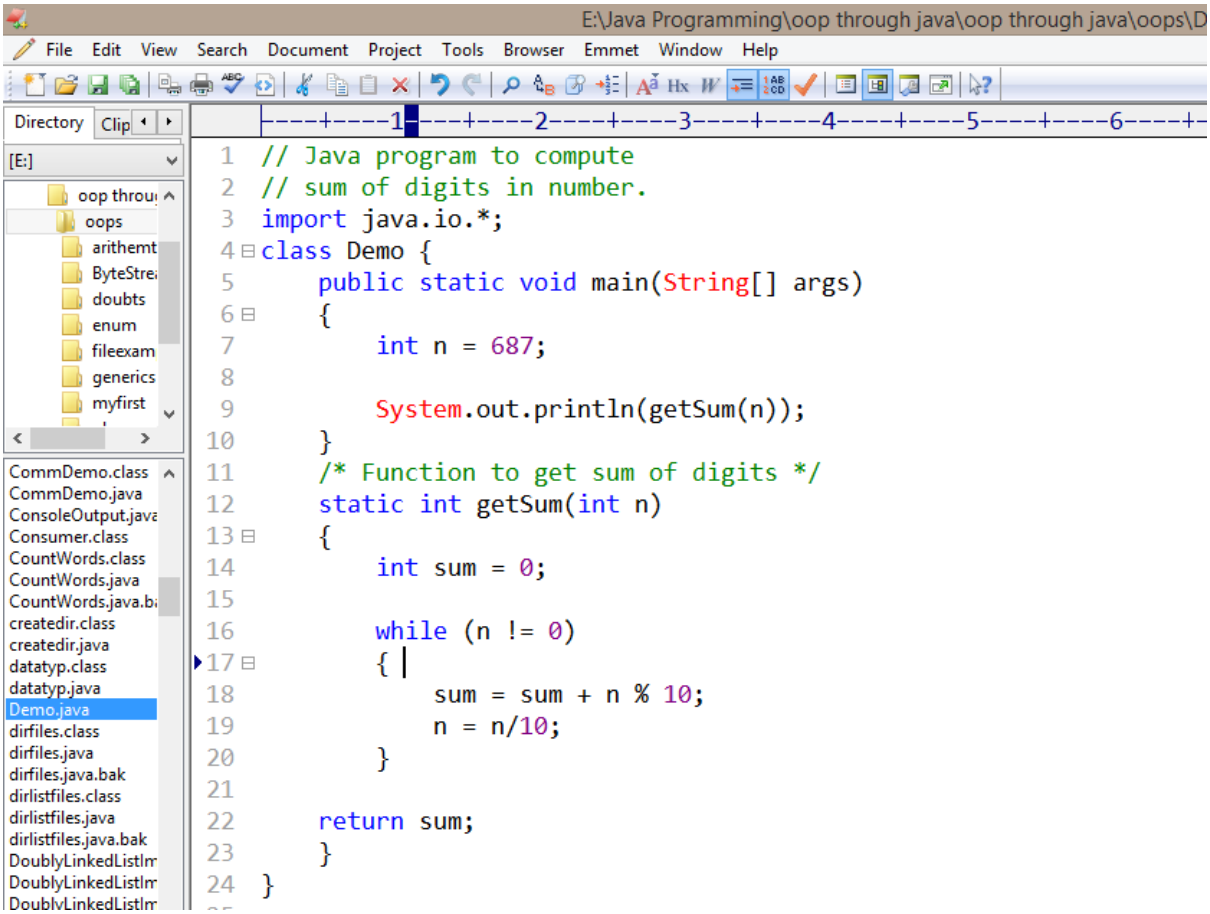
S.NO	METHOD OVERLOADING	METHOD OVERRIDING
1.	Method overloading is used to increase the readability of the program.	Method overriding is used to provide the specific implementation of the method that is already provided by its super class.
2.	Method overloading is performed within class.	Method overriding occurs in two classes that have IS-A (inheritance) relationship.
3.	In case of method overloading, parameter must be different.	In case of method overriding, parameter must be same.
4.	Method overloading is the example of compile time polymorphism.	Method overriding is the example of run time polymorphism.

5.	Overloading
	class Dog{
	public void bark(){
	System.out.println("woof ");
	}
	//overloading method
	public void bark(int num){
	for(int i=0; i<num; i++)
	System.out.println("woof ");
	}
	}
	Same Method Name, Different Parameter
		Overriding
		class Dog{
		public void bark(){
		System.out.println("woof ");
		}
		}
		class Hound extends Dog{
		public void sniff(){
		System.out.println("sniff ");
		}
		public void bark(){
		System.out.println("bowl");
		}
		}
		Same Method Name, Same parameter

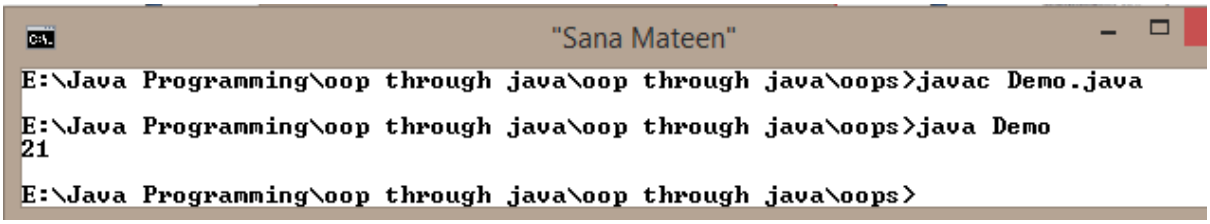
Long Answer Questions for 10 Marks:

11 Write a java program to find the sum of the given number?

Program:

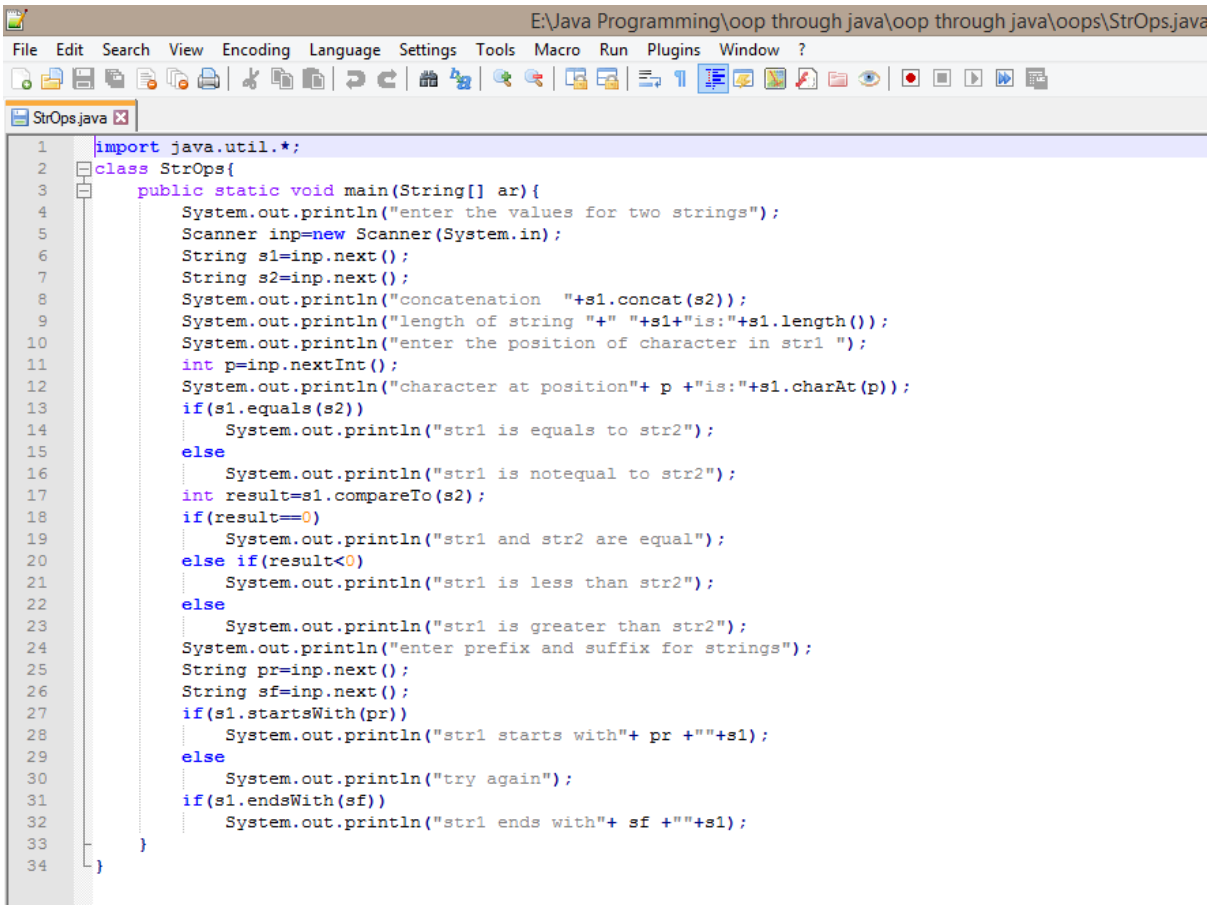


Output:



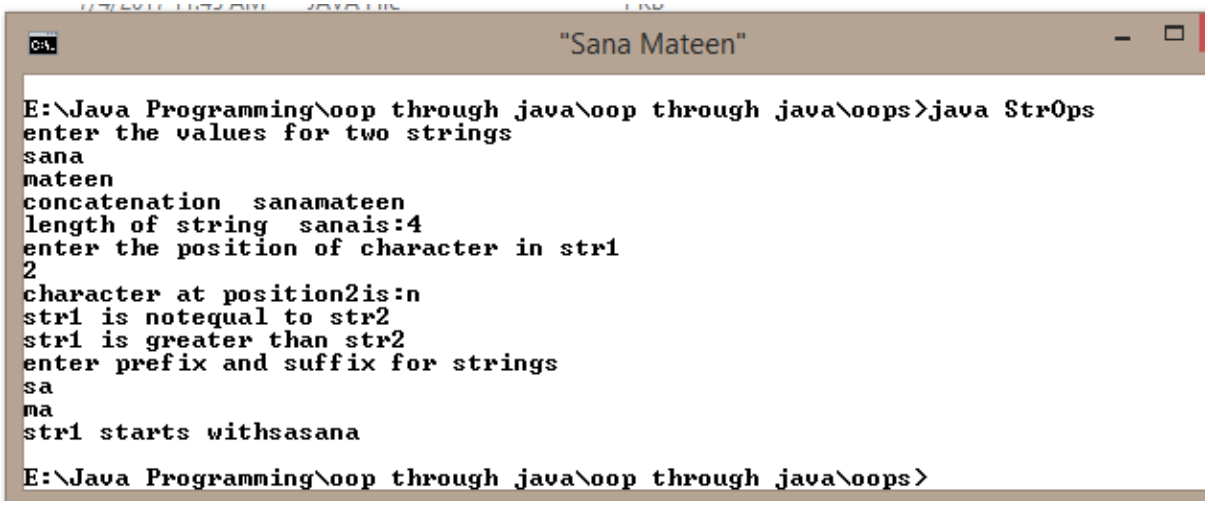
12 Write a java program to implement String Handling Functions?

Program:




```
import java.util.*;
class StrOps{
    public static void main(String[] ar){
        System.out.println("enter the values for two strings");
        Scanner inp=new Scanner(System.in);
        String s1=inp.next();
        String s2=inp.next();
        System.out.println("concatenation  "+s1.concat(s2));
        System.out.println("length of string "+" "+s1+"is:"+s1.length());
        System.out.println("enter the position of character in str1 ");
        int p=inp.nextInt();
        System.out.println("character at position"+ p +"is:"+s1.charAt(p));
        if(s1.equals(s2))
            System.out.println("str1 is equals to str2");
        else
            System.out.println("str1 is notequal to str2");
        int result=s1.compareTo(s2);
        if(result==0)
            System.out.println("str1 and str2 are equal");
        else if(result<0)
            System.out.println("str1 is less than str2");
        else
            System.out.println("str1 is greater than str2");
        System.out.println("enter prefix and suffix for strings");
        String pr=inp.next();
        String sf=inp.next();
        if(s1.startsWith(pr))
            System.out.println("str1 starts with"+ pr +""+s1);
        else
            System.out.println("try again");
        if(s1.endsWith(sf))
            System.out.println("str1 ends with"+ sf +""+s1);
    }
}
```

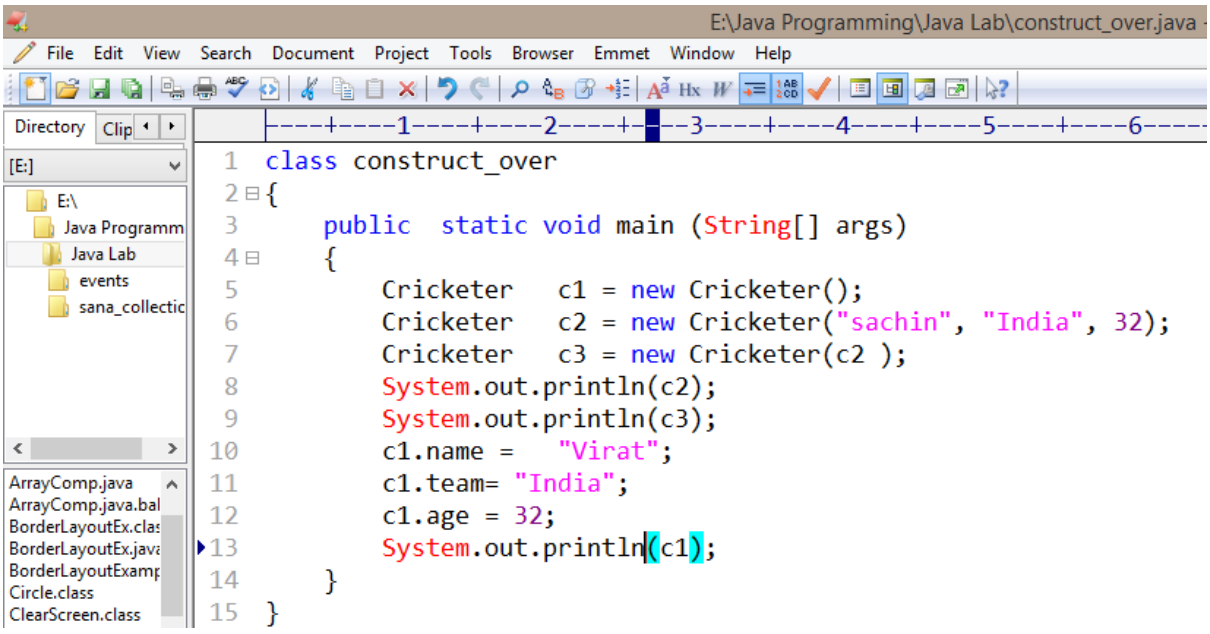
Output:

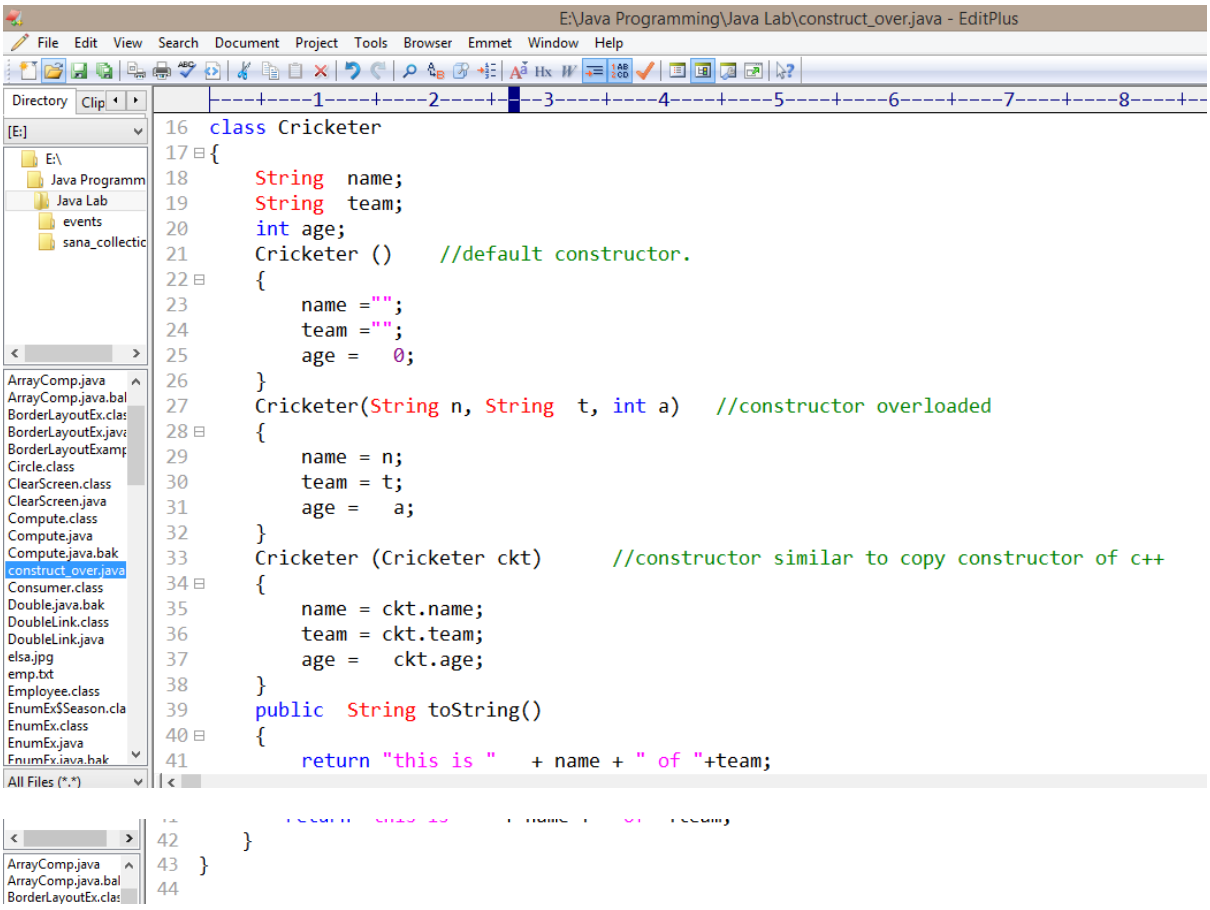



```
E:\Java Programming\oop through java\oop through java\oops>java StrOps
enter the values for two strings
sana
mateen
concatenation  sanamateen
length of string  sanais:4
enter the position of character in str1
2
character at position2is:n
str1 is notequal to str2
str1 is greater than str2
enter prefix and suffix for strings
sa
ma
str1 starts withsasana

E:\Java Programming\oop through java\oop through java\oops>
```

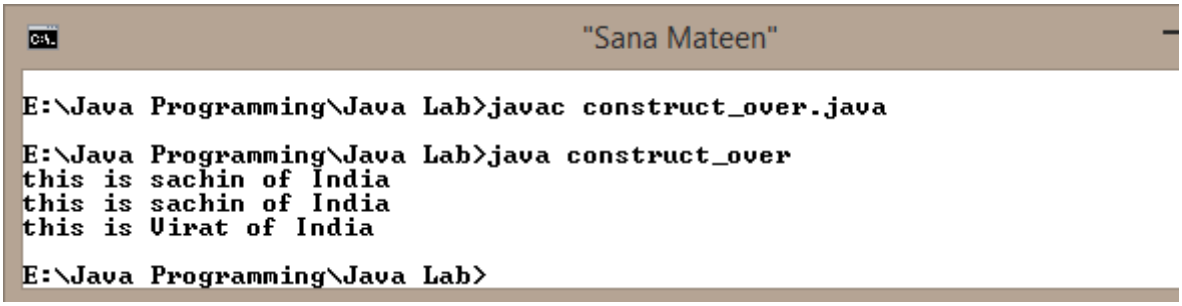
13 Write a java program to implement Constructor Overloading?

Program:





Output:



14 Explain Polymorphism in detail along with example program?

- **Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.**
- The word "poly" means many and "morphs" means forms. So polymorphism means many forms.
- There are two types of polymorphism in java: compile time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.
- **Compile time polymorphism**
- If a method call is resolved at compile time that it is called static binding or earlier binding or static polymorphism or compile time polymorphism.
- **Method overloading and method overriding by using static method, private method and final method are example for static polymorphism.**
- **Runtime polymorphism**
- If a method call is resolved at the time of execution it is called dynamic binding or late binding or run time polymorphism.
- Example- Method overloading and method overriding by using instance method are example for dynamic binding.

Program:

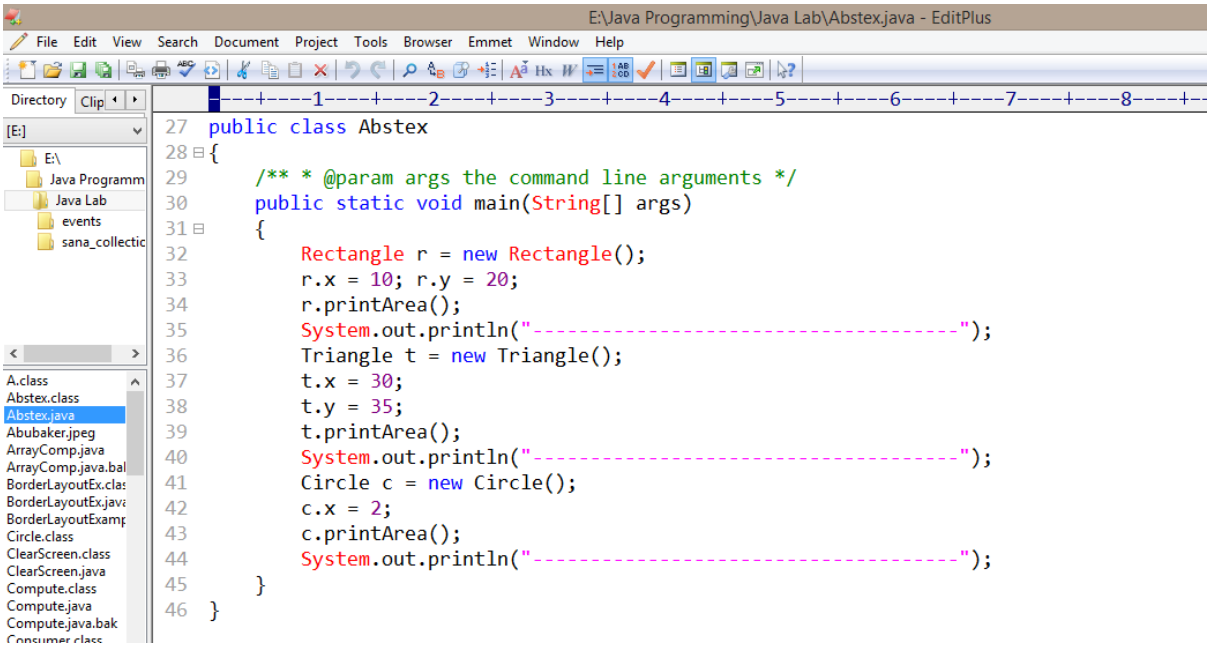
```
public class Area
{
  public void calculate()                          // I
  {
    System.out.println("Nothing to calculate");
  }
  public void calculate(int x)                     // II
  {
    System.out.println("Circle Area: " + Math.PI * Math.pow(x, 2));
  }
  public void calculate(double x)                  // III
  {
    System.out.println("Circle Perimeter: " + 2 * Math.PI * x);
  }
  public void calculate(int x, int y)              // IV
  {
    System.out.println("Rectangle Area: " +x*y);
  }
  public static void main(String args[])
  {
    Area a1 = new Area();
    a1.calculate();                  // I
    a1.calculate(10, 20);            // IV
    a1.calculate(10.5);              // III
    a1.calculate(10);                // II
  }
```

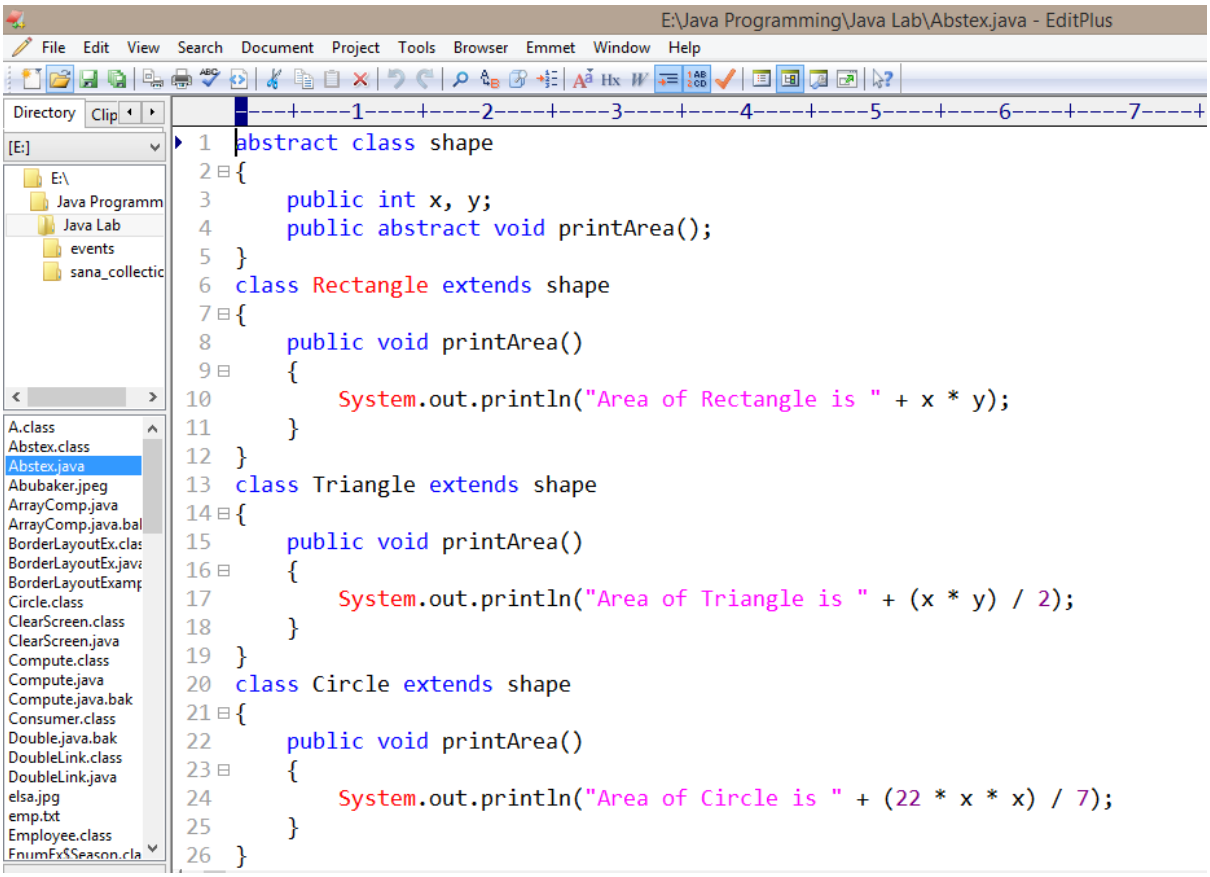
Output:

```
c:\Users\sana mateen\Desktop\oops>start notepad Area.java

c:\Users\sana mateen\Desktop\oops>javac Area.java

c:\Users\sana mateen\Desktop\oops>java Area
Nothing to calculate
Rectangle Area: 200
Circle Perimeter: 65.97344572538566
Circle Area: 314.1592653589793
```

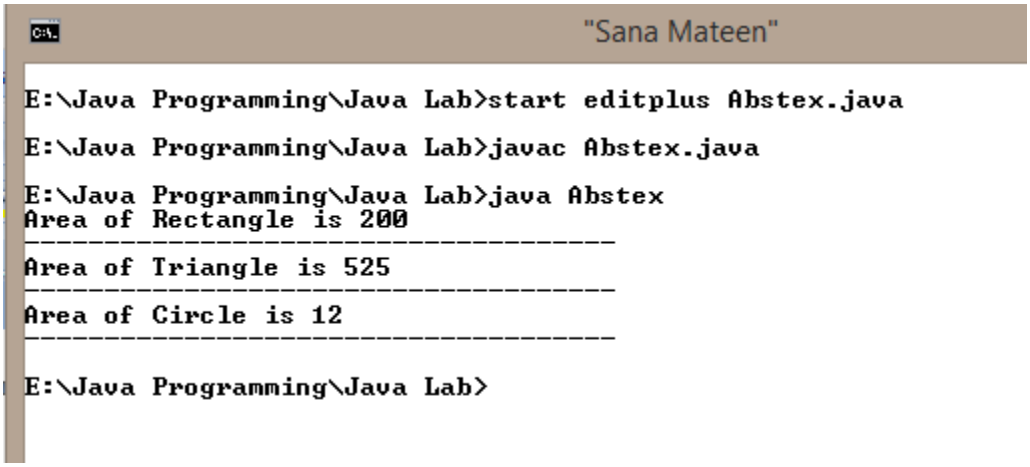
15 Explain abstract classes along with example program?

Program:





Output:



16 Explain different forms of inheritance?

All objects eventually inherit from Object, which provides useful methods such as equals and toString.

In general we want to satisfy substitutability: if B is a subclass of A, anywhere we expect an instance of A we can use an instance of B.

Inheritance gets used for a number of purposes in typical object-oriented programming:

specialization -- the subclass is a special case of the parent class (e.g. Frame and CannonWorld)

- By far the most common form of inheritance is for specialization.
  - Child class is a specialized form of parent class
  - Principle of substitutability holds
- A good example is the Java hierarchy of Graphical components in the AWT:
  - Component
    - Label
    - Button
    - TextComponent
      - TextArea
      - TextField
    - CheckBox
    - ScrollBar

specification -- the superclass just specifies which methods should be available but doesn't give code. This is supported in java by interfaces and abstract methods.

- The next most common form of inheritance involves specification. The parent class specifies some behavior, but does not implement the behavior
  - Child class implements the behavior
  - Similar to Java interface or abstract class
  - When parent class does not implement actual behavior but merely defines the behavior that will be implemented in child classes
- Example, Java 1.1 Event Listeners:

ActionListener, MouseListener, and so on specify behavior, but must be subclassed.

construction -- the superclass is just used to provide behavior, but instances of the subclass don't really act like the superclass. Violates substitutability. Exmample: defining Stack as a subclass of Vector. This is not clean -- better to define Stack as having a field that holds a vector.

- The parent class is used only for its behavior, the child class has no *is-a* relationship to the parent.
  - Child modify the arguments or names of methods
- An example might be subclassing the idea of a *Set* from an existing *List* class.
  - Child class is not a more specialized form of parent class; no substitutability

extension -- subclass adds new methods, and perhaps redefines inherited ones as well.

- The child class generalizes or extends the parent class by providing more functionality
  - In some sense, opposite of subclassing for specialization
- The child doesn't change anything inherited from the parent, it simply adds new features
  - Often used when we cannot modify existing base parent class
- Example, ColoredWindow inheriting from Window
  - Add additional data fields
  - Override window display methods

limitation -- the subclass restricts the inherited behavior. Violates substitutability. Example: defining Queue as a subclass of Dequeue.

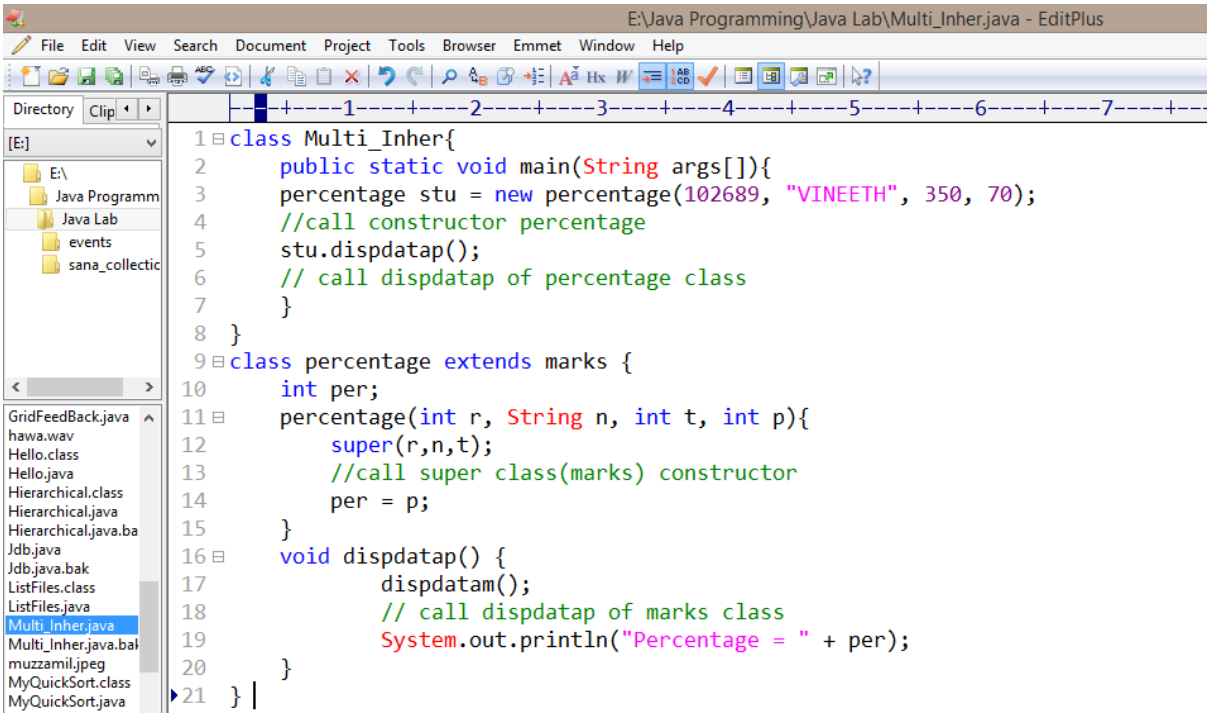
- The child class limits some of the behavior of the parent class.
- Example, you have an existing List data type, and you want a Stack
- Inherit from List, but override the methods that allow access to elements other than top so as to produce errors.

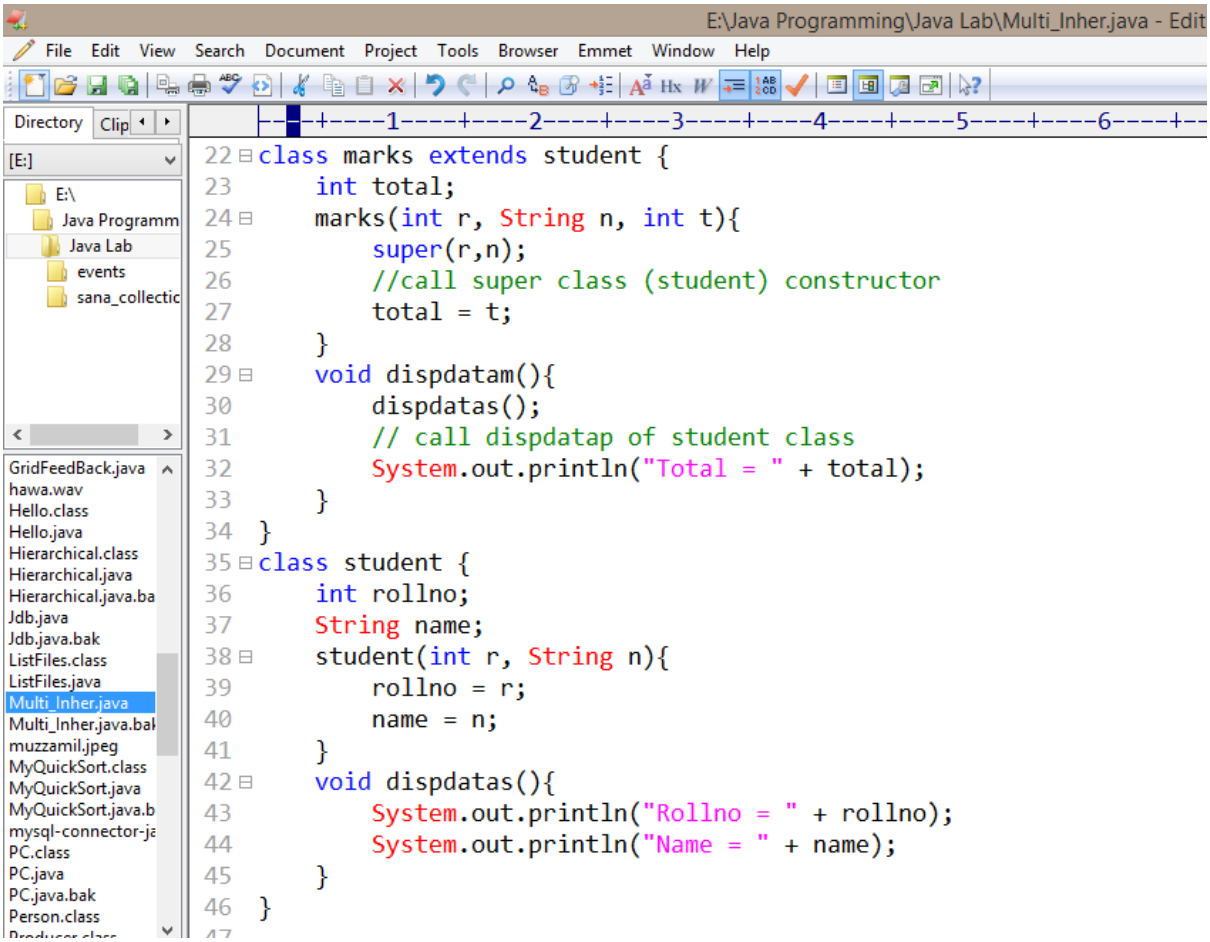
combination -- multiple inheritance. Provided in part by implementing multiple interfaces.

- Two or more classes that seem to be related, but its not clear who should be the parent and who should be the child.
- Example: Mouse and TouchPad and JoyStick
- Better solution, abstract out common parts to new parent class, and use subclassing for specialization.

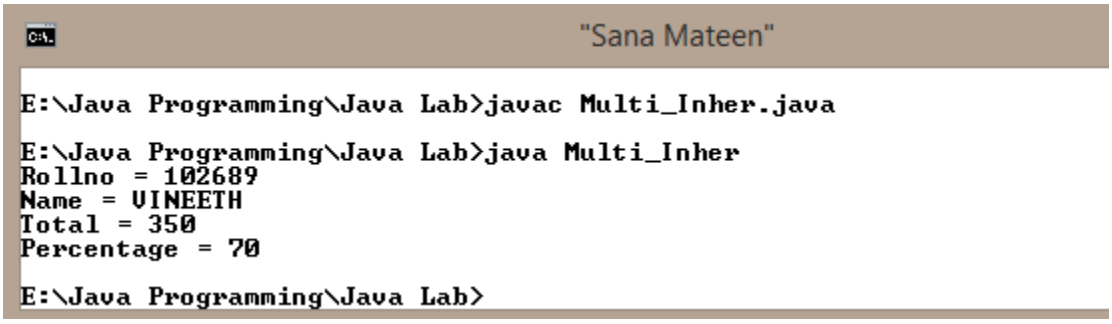
17 Write a Java program to implement Multi level inheritance?

Program



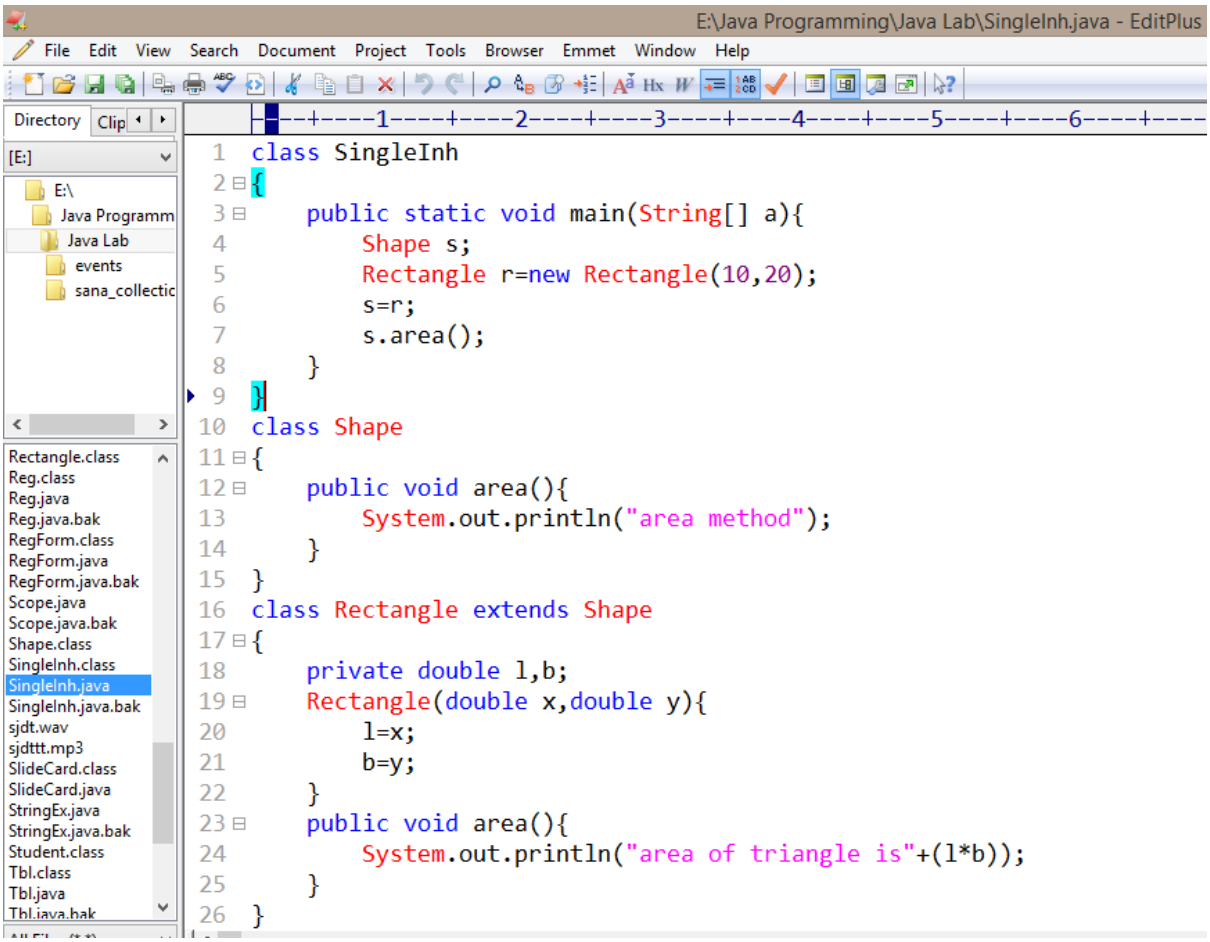

E:\Java Programming\Java Lab\Multi_Inher.java - Edit
File Edit View Search Document Project Tools Browser Emmet Window Help
Directory Clip
[E:]
E:\
Java Programm
Java Lab
events
sana_collectic
GridFeedBack.java
hawa.wav
Hello.class
Hello.java
Hierarchical.class
Hierarchical.java
Hierarchical.java.ba
Jdb.java
Jdb.java.bak
ListFiles.class
ListFiles.java
Multi_Inher.java
Multi_Inher.java.bak
muzzamil.jpeg
MyQuickSort.class
MyQuickSort.java
MyQuickSort.java.b
mysql-connector-ja
PC.class
PC.java
PC.java.bak
Person.class
22 class marks extends student {
23 int total;
24 marks(int r, String n, int t){
25 super(r,n);
26 //call super class (student) constructor
27 total = t;
28 }
29 void dispdatam(){
30 dispdatas();
31 // call dispdatap of student class
32 System.out.println("Total = " + total);
33 }
34 }
35 class student {
36 int rollno;
37 String name;
38 student(int r, String n){
39 rollno = r;
40 name = n;
41 }
42 void dispdatas(){
43 System.out.println("Rollno = " + rollno);
44 System.out.println("Name = " + name);
45 }
46 }

Output:

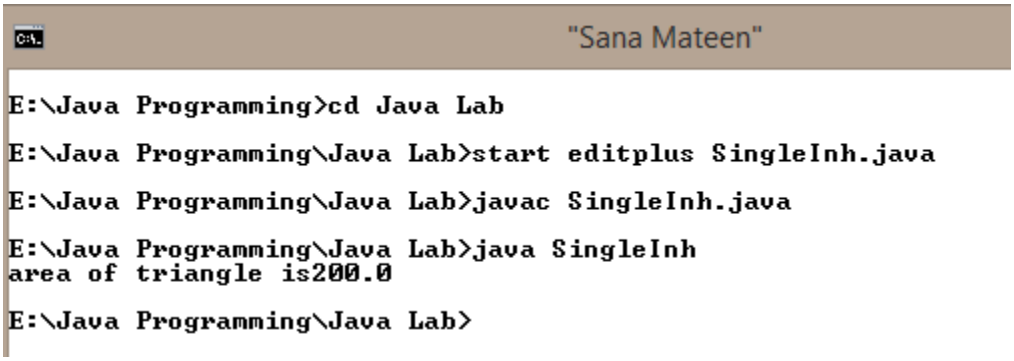

"Sana Mateen"
E:\Java Programming\Java Lab>javac Multi_Inher.java
E:\Java Programming\Java Lab>java Multi_Inher
Rollno = 102689
Name = VINEETH
Total = 350
Percentage = 70
E:\Java Programming\Java Lab>

18 Write a Java program to implement Single Inheritance?

Program:

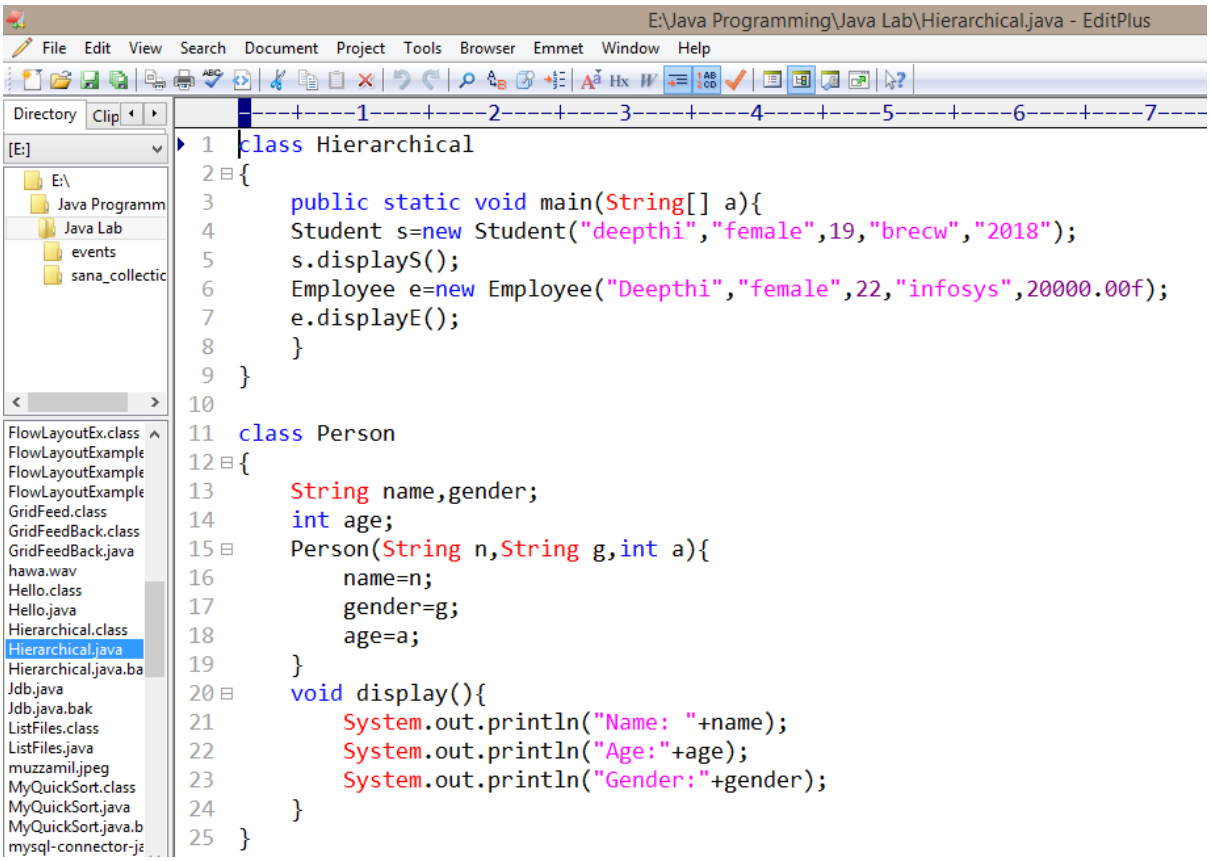


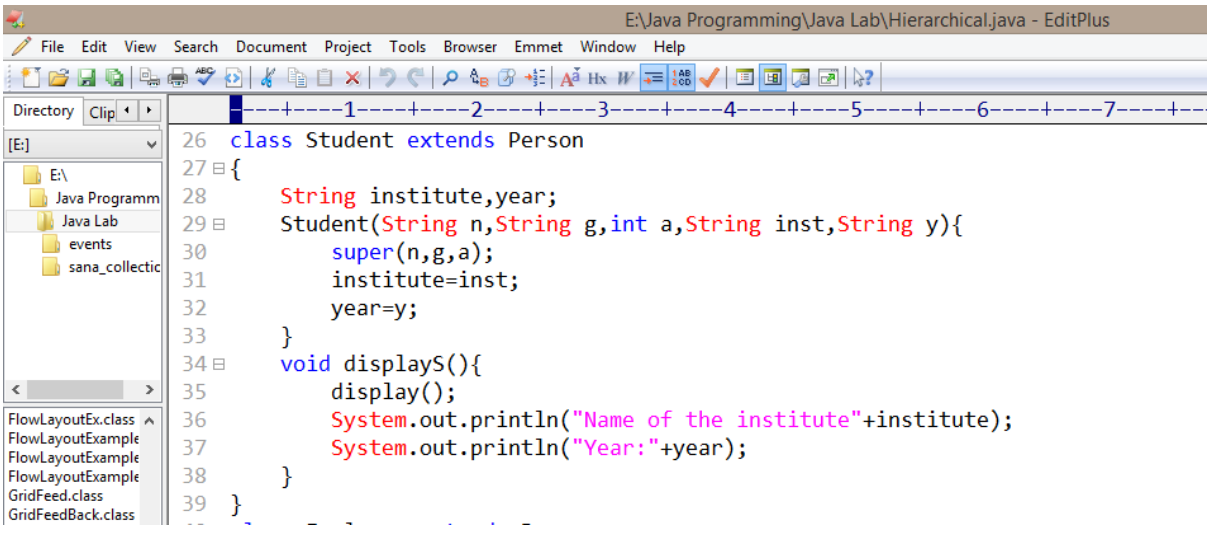
Output:

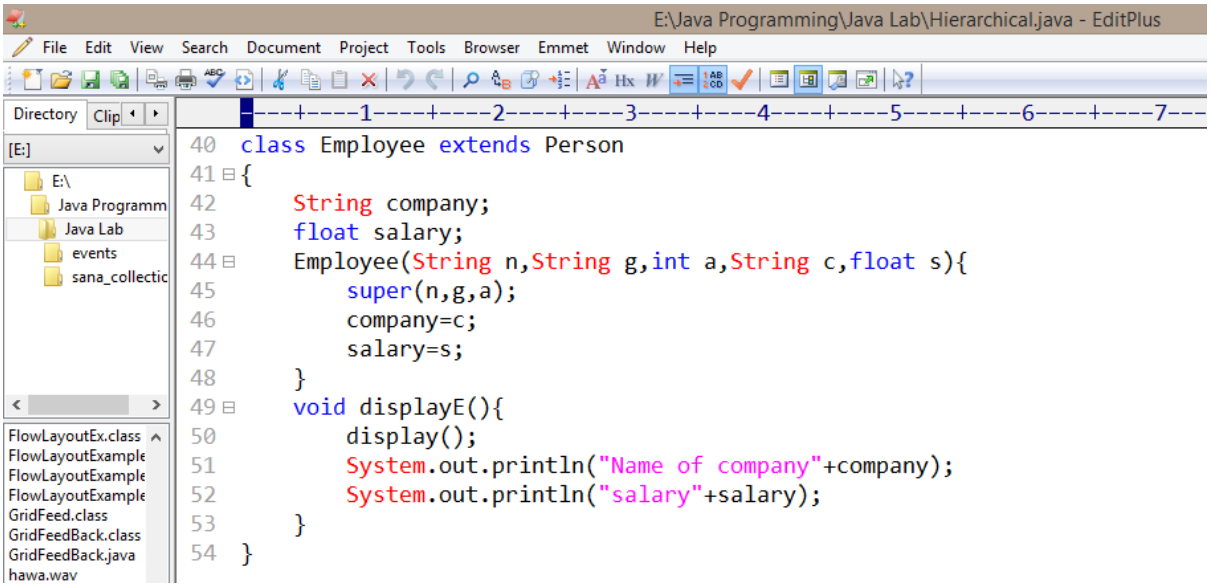


19 Write a Java program to implement Hierarchical Inheritance?

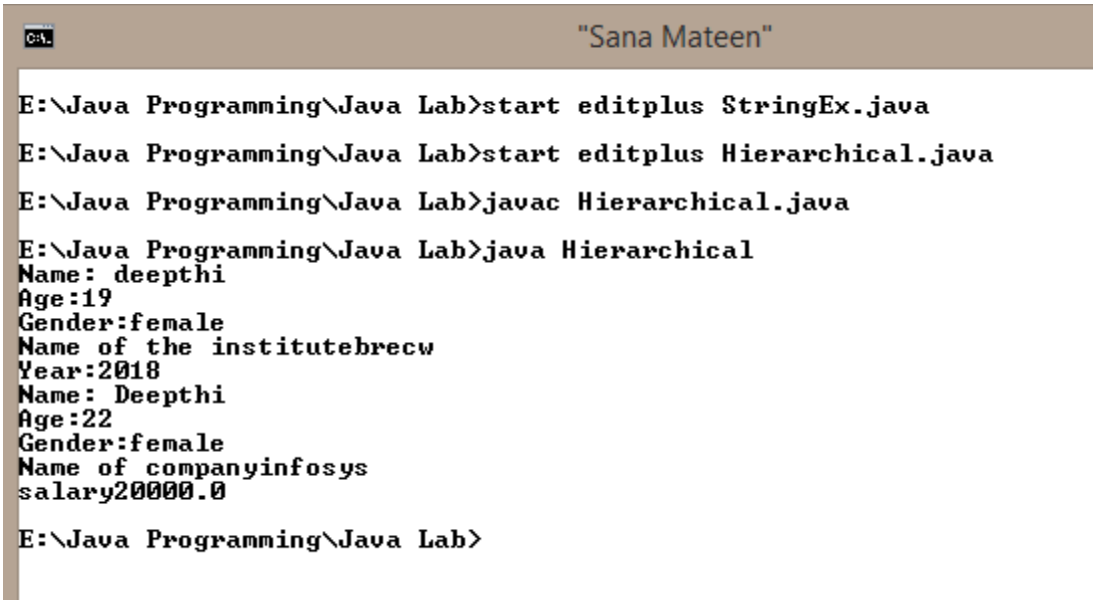
Program:






E:\Java Programming\Java Lab\Hierarchical.java - EditPlus
File Edit View Search Document Project Tools Browser Emmet Window Help
Directory Clip
[E:]
E:\
Java Programm
Java Lab
events
sana_collectic
FlowLayoutEx.class
FlowLayoutExample
FlowLayoutExample
FlowLayoutExample
GridFeed.class
GridFeedBack.class
GridFeedBack.java
hawa.wav
class Employee extends Person
{
String company;
float salary;
Employee(String n,String g,int a,String c,float s){
super(n,g,a);
company=c;
salary=s;
}
void displayE(){
display();
System.out.println("Name of company"+company);
System.out.println("salary"+salary);
}
}

Output:


"Sana Mateen"
E:\Java Programming\Java Lab>start editplus StringEx.java
E:\Java Programming\Java Lab>start editplus Hierarchical.java
E:\Java Programming\Java Lab>javac Hierarchical.java
E:\Java Programming\Java Lab>java Hierarchical
Name: deepthi
Age:19
Gender:female
Name of the institutebrecw
Year:2018
Name: Deepthi
Age:22
Gender:female
Name of companyinfosys
salary20000.0
E:\Java Programming\Java Lab>

UNIT-2

Short Answer Questions for 2 Marks:

1. What is Console class?What is its use in java?

It is used to read from and write to the console, if one exists. It implements the Flushable interface.

Console is primarily a convenience class because most of its functionality is available through System.in and System.out.

Console supplies no constructors.

Instead, a Console object is obtained by calling System.console( ), which is shown here:

static Console console( )

If a console is available, then a reference to it is returned. Otherwise, null is returned.

2. What is the use of autoboxing in java?Explain.

Autoboxing: The automatic conversion of primitive data types into its equivalent Wrapper type is known as boxing.

For example, converting int to Integer class. The Java compiler applies autoboxing when a primitive value is:

Passed as a parameter to a method that expects an object of the corresponding wrapper class.

Assigned to a variable of the corresponding wrapper class.

3. What is static inner class?

A static class i.e. created inside a class is called static nested class in java. It cannot access non-static data members and methods. It can be accessed by outer class name.

It can access static data members of outer class including private.

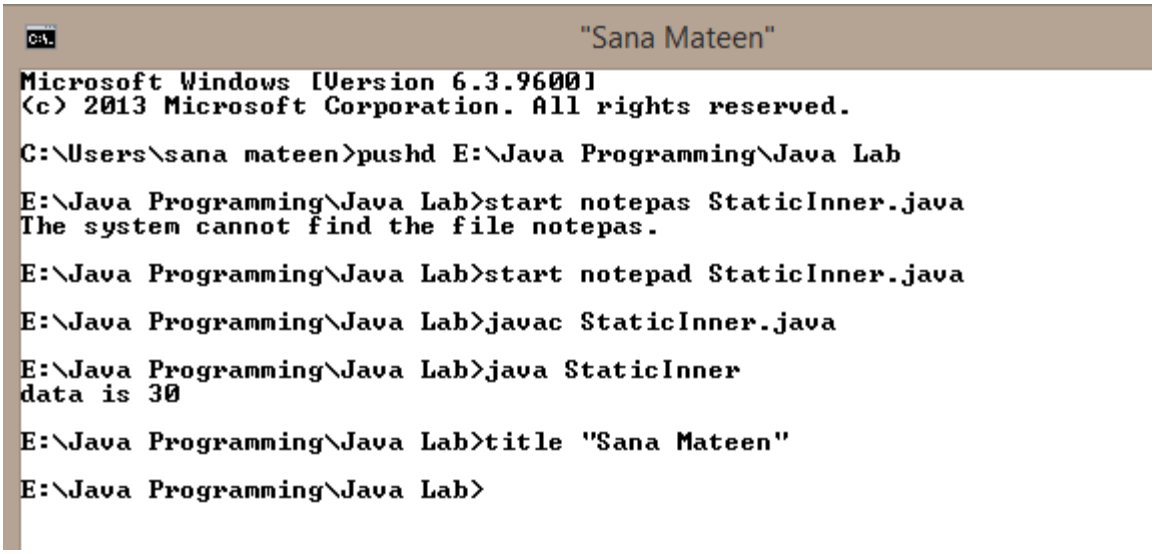
Static nested class cannot access non-static (instance) data member or method.

StaticInner - Notepad

File Edit Format View Help

```
class StaticInner{
        static int data=30;
        static class Inner{
                void msg(){
                        System.out.println("data is "+data);
                }
        }
        public static void main(String args[]){
                StaticInner.Inner obj=new StaticInner.Inner();
                obj.msg();
        }
}
```

Output:



4. Explain implicit and explicit import statements?

Imports can be categorized as *explicit* (for example import java.util.List;) or *implicit* (also known as 'on-demand', for example import java.util.*;):

- Implicit imports give access to all visible types in the type (or package) that precedes the ".*"; types imported in this way never shadow other types.
- Explicit imports give access to just the named type; they can shadow other types that would normally be visible through an implicit import, or through the normal package visibility rules.

5. Define Package?What is its use in java?Explain

A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Classes and packages are both means of encapsulating and containing the name space and scope of variables and methods.

Packages act as containers for classes and other subordinate packages. Classes act as containers for data and code.

6. List out the benefits of Stream Oriented I/O

The streaming interface to I/O in java provides a clean abstraction for a complex and often cumbersome task. The composition of the filtered stream classes allows you to dynamically build the custom streaming interface to suit your data transfer requirements. Java programs written to adhere to the abstract, high-level Java InputStream Class, Java OutputStream Class, Reader Class In Java, and Java Writer Class classes will function properly in the future even when new and improved concrete stream classes are invented. This model works very well when we switch from a file system-based set of streams to the network and socket streams. Finally Serialization I/O classes provides a portable solution to this sometimes tricky task.

Short Answer Questions for 3 Marks:

7. Differentiate between interface and abstract class

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance.**	Interface **supports multiple inheritance.**
3) Abstract class **can have final, non-final, static and non-static variables.**	Interface has **only static and final variables.**
4) Abstract class **can provide the implementation of interface.**	Interface **can't provide the implementation of abstract class.**
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.

6) **Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

8. Explain Access protection with the help of a program?

Java Access Specifiers (also known as Visibility Specifiers) regulate access to classes, fields and methods in Java.

	Private	No Modifier	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package non-subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Program:

A - Notepad

File Edit Format View Help

```
package pack;
class A{
 protected void msg(){System.out.println("Hello");}
}
```

B - Notepad

File Edit Format View Help

```
package MyPack;
import pack.*;
class B extends A{
  public static void main(String args[]){
   B obj = new B();
   obj.msg();
  }
}
```

pexample - Notepad

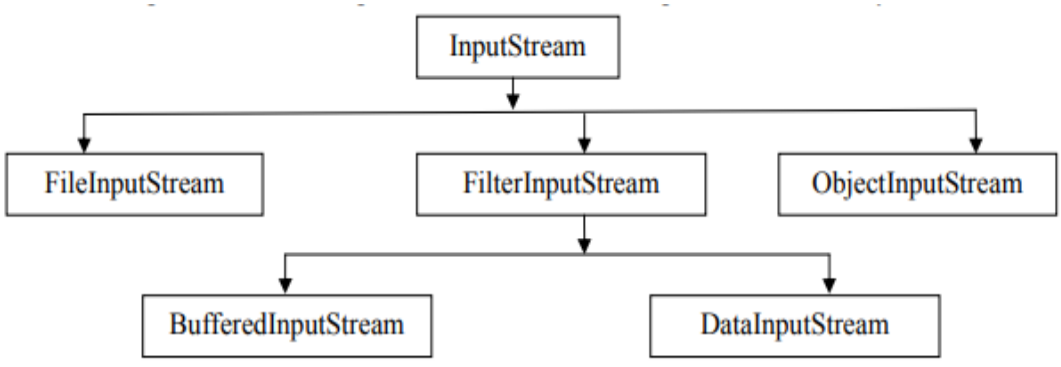
File Edit Format View Help

private

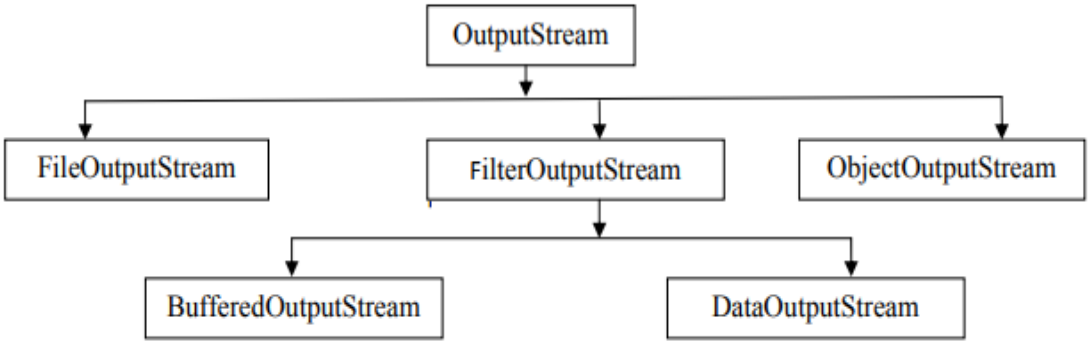
```
class A{
private int data=40;
private void msg(){System.out.println("Hello java");}
}

public class pexample{
 public static void main(String args[]){
   A obj=new A();
   System.out.println(obj.data);//Compile Time Error
   obj.msg();//Compile Time Error
   }
}
```

```
E:\oops>start notepad pexample.java

E:\oops>javac pexample.java
pexample.java:9: error: data has private access in A
   System.out.println(obj.data);//Compile Time Error
                           ^
pexample.java:10: error: msg() has private access in A
   obj.msg();//Compile Time Error
      ^
2 errors
```

9. List the Byte Stream classes?



Byte Stream Classes for Reading Data

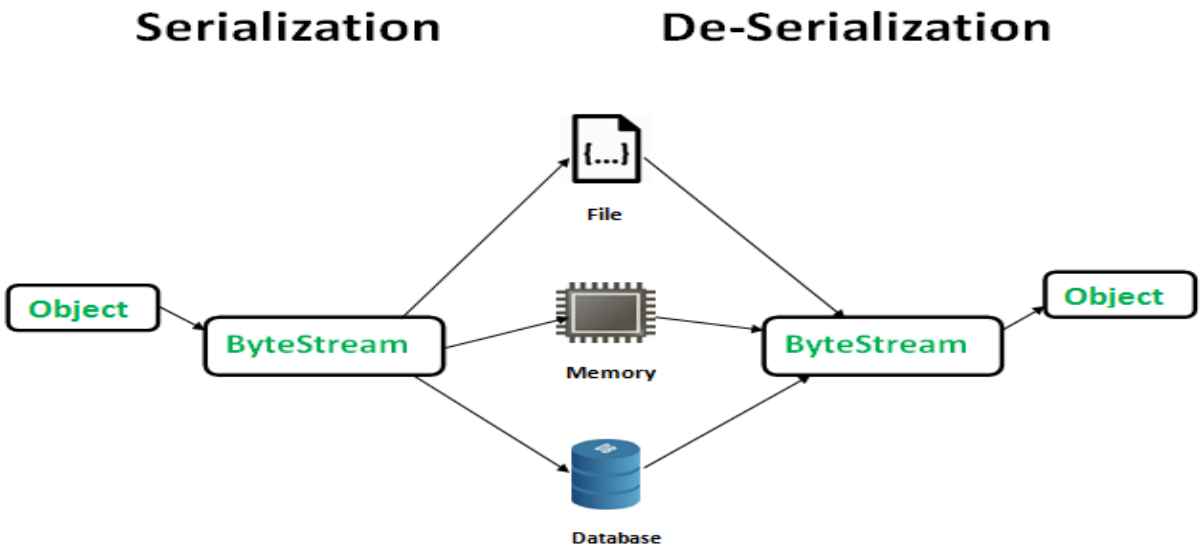


Byte Stream Classes for Writing Data

10. Define Serialization and Deserialization?

Serialization is a mechanism of converting the state of an object into a byte stream.

Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object.



11. Write about CLASSPATH environment variable?

A classpath environment variable is the location from which classes are loaded at runtime by JVM in java.

Classes may may include system classes and user-defined classes.

It is impractical for it to go through every folder on your system and search for it. Thus, using the CLASSPATH variable we provide it the place where we want it to look. We put directories and jars in the CLASSPATH variable.

12. What is the purpose of nested interface?

Inner interface is also called nested interface, which means declare an interface inside of another interface.

There are several compelling reasons for using inner interface:

It is a way of logically grouping interfaces that are only used in one place.

It increases encapsulation.

Nested interfaces can lead to more readable and maintainable code.

13. Write a Java Program to read an entire line through Keyboard?

Program:

```
1  import java.io.*;
2  public class BufferedReaderExample{
3  public static void main(String args[])throws Exception{
4      InputStreamReader r=new InputStreamReader(System.in);
5      BufferedReader br=new BufferedReader(r);
6      System.out.println("Enter your name");
7      String name=br.readLine();
8      System.out.println("Welcome "+name);
9  }
10 }
```

Output:

```
C:\Users\sana mateen\Desktop\oops\streams>start EditPlus BufferedReaderExample.j
ava

C:\Users\sana mateen\Desktop\oops\streams>javac BufferedReaderExample.java

C:\Users\sana mateen\Desktop\oops\streams>java BufferedReaderExample
Enter your name
sana mateen
Welcome sana mateen

C:\Users\sana mateen\Desktop\oops\streams>
```

Long Answer Question for 5 Marks:

1. **What is nested class?Differentiate between static nested classes and non static nested classes?**

   In java, it is possible to define a class within another class, such classes are known as nested classes. They enable you to logically group classes that are only used in one place, thus this increases the use of encapsulation, and create more readable and maintainable code.

   The scope of a nested class is bounded by the scope of its enclosing class. Thus in above example, class NestedClass does not exist independently of class OuterClass.

   A nested class has access to the members, including private members, of the class in which it is nested. However, reverse is not true i.e. the enclosing class does not have access to the members of the nested class.

   A nested class is also a member of its enclosing class.

   As a member of its enclosing class, a nested class can be declared private, public, protected, or package private(default).

   Nested classes are divided into two categories:

   static nested class : Nested classes that are declared static are called static nested classes.

   inner class : An inner class is a non-static nested class.

2. **Difference between static and inner(non-static nested) classes**

   Static nested classes do not directly have access to other members(non-static variables and methods) of the enclosing class because as it is static, it must access the non-static members of its enclosing class through an object. That is, it cannot refer to non-static members of its enclosing class directly. Because of this restriction, static nested classes are seldom used.

   Non-static nested classes (inner classes) has access to all members(static and non-static variables and methods, including private) of its outer class and may refer to them directly in the same way that other non-static members of the outer class do.

3. **Define package?How to access,import a package?Explain with example program?**

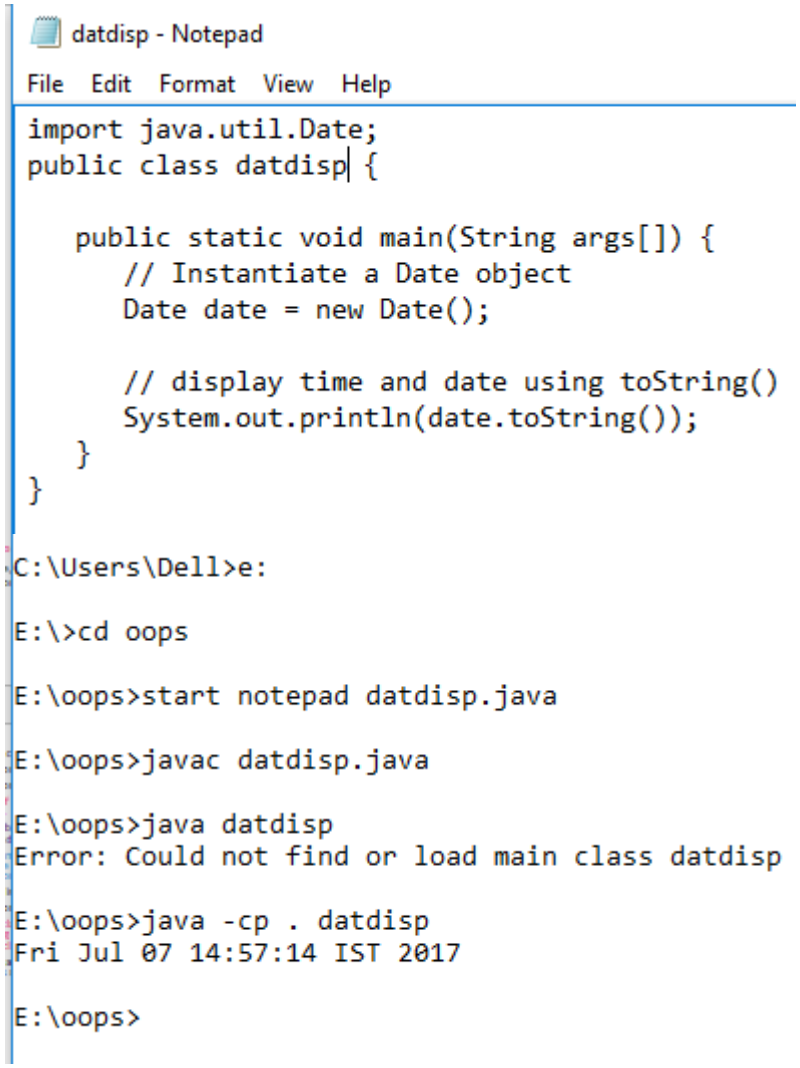
   A java package is a group of similar types of classes, interfaces and sub-packages.

   Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Creating Package:

To create a package, you choose a name for the package and put a package statement with that name at the top of every source file.



4. Explain Generics with an example program?

Generics in Java is similar to templates in C++. The idea is to allow type (Integer, String, … etc and user defined types) to be a parameter to methods, classes and interfaces.

For example, classes like HashSet, ArrayList, HashMap, etc use generics very well. We can use them for any type.

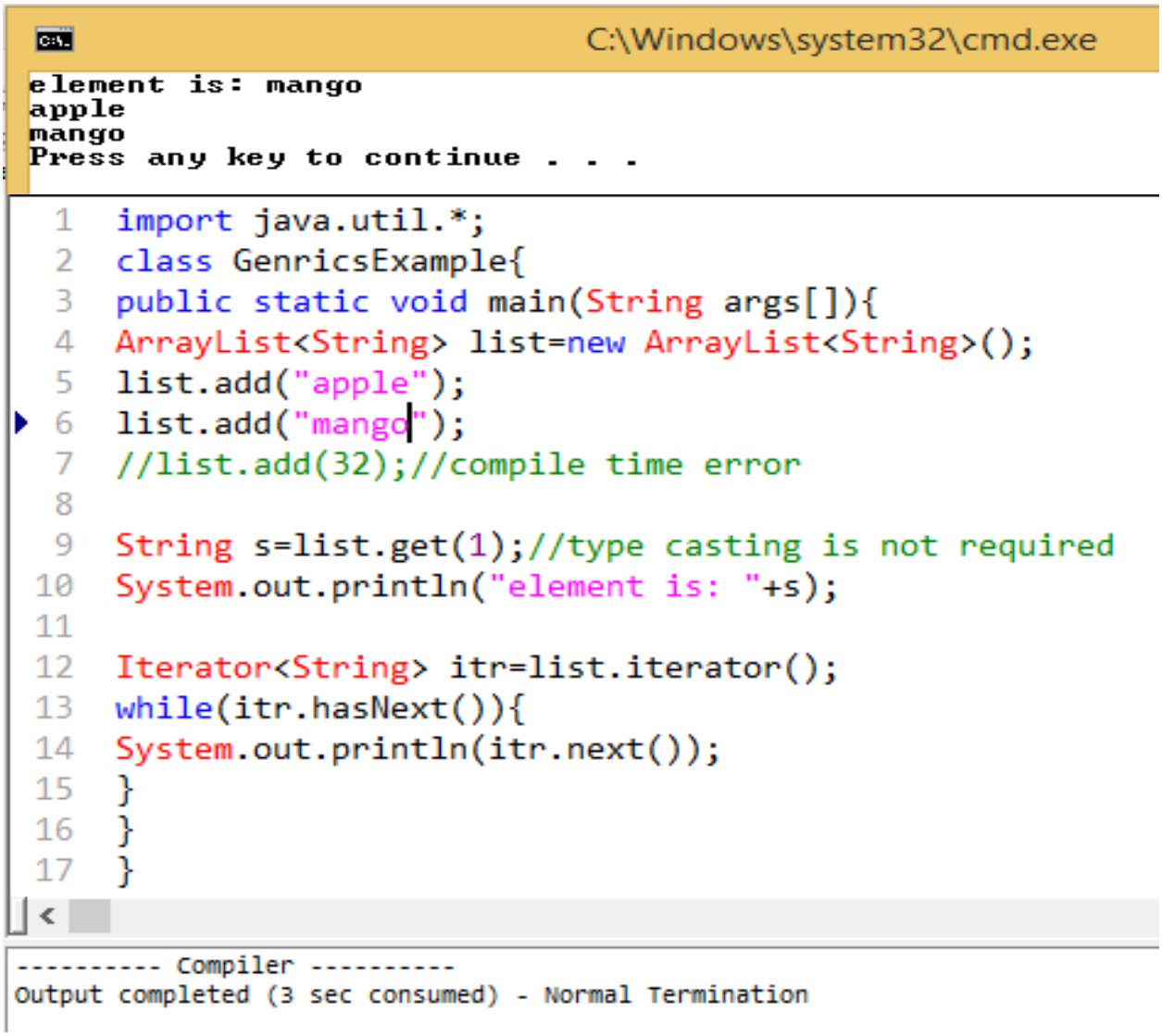
Generic Class

Like C++, we use <> to specify parameter types in generic class creation. To create objects of generic class, we use following syntax.

// To create an instance of generic class

BaseType <Type> obj = new BaseType <Type>()

Note: In Parameter type we can not use primitives like 'int','char' or 'double'.



5. How to design and implement interfacein java?Give an example?

An **interface in java** is a blueprint of a class. It has static constants and abstract methods.

The interface in java is **a mechanism to achieve abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve abstraction and multiple inheritance in Java.

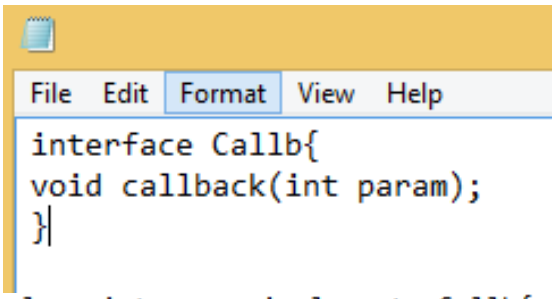
- Why use Java interface?
- There are mainly three reasons to use interface. They are given below.
- It is used to achieve abstraction.

- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

Once an interface has been defined, one or more classes can implement that interface. To implement an interface, include the implements clause in a class definition, and then create the methods required by the interface. The general form of a class that includes the implements clause looks like this:

- class classname [extends superclass] [implements interface [,interface...]] {
- // class-body
- }

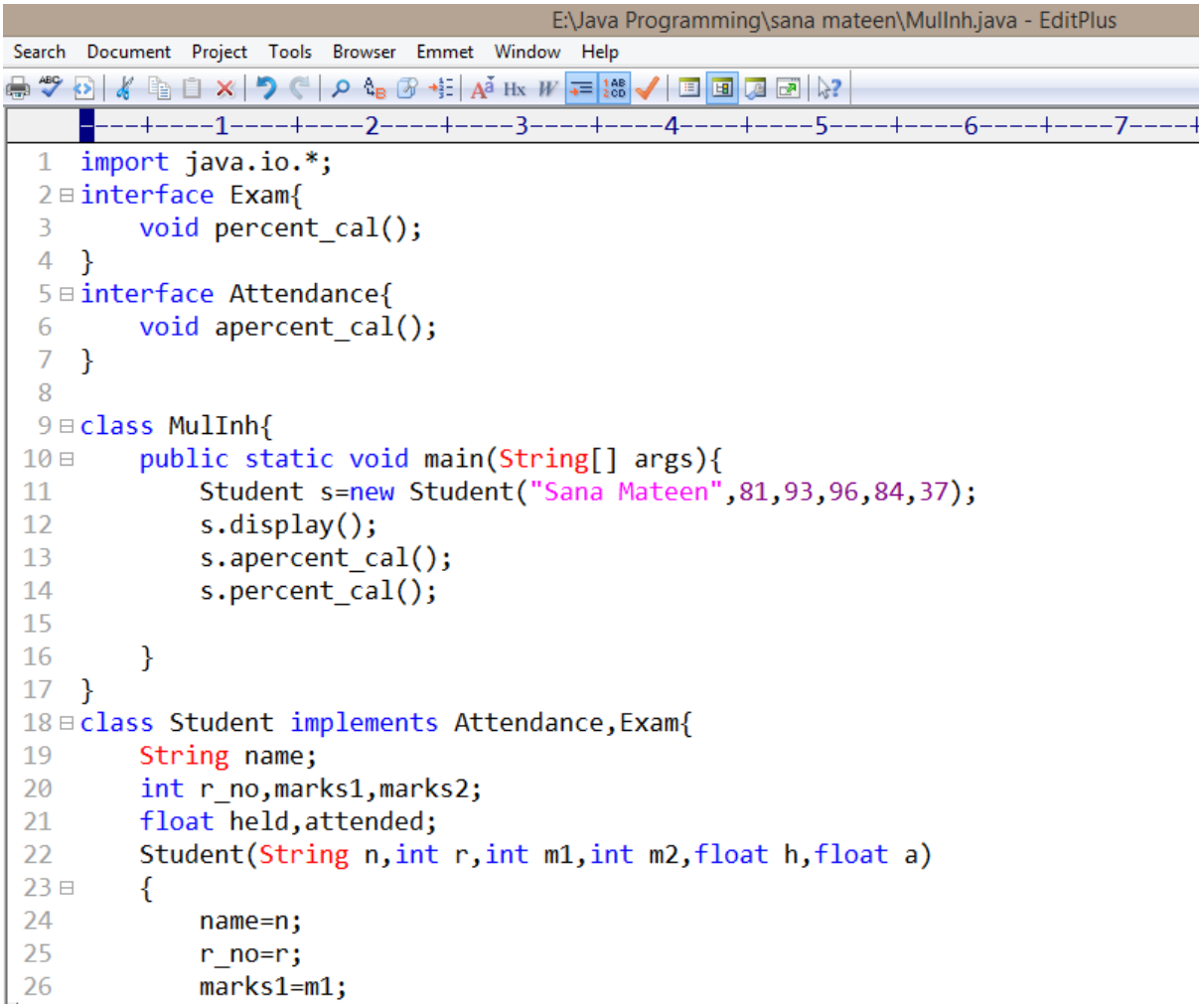
If a class implements more than one interface, the interfaces are separated with a comma. If a class implements two interfaces that declare the same method, then the same method will be used by clients of either interface.



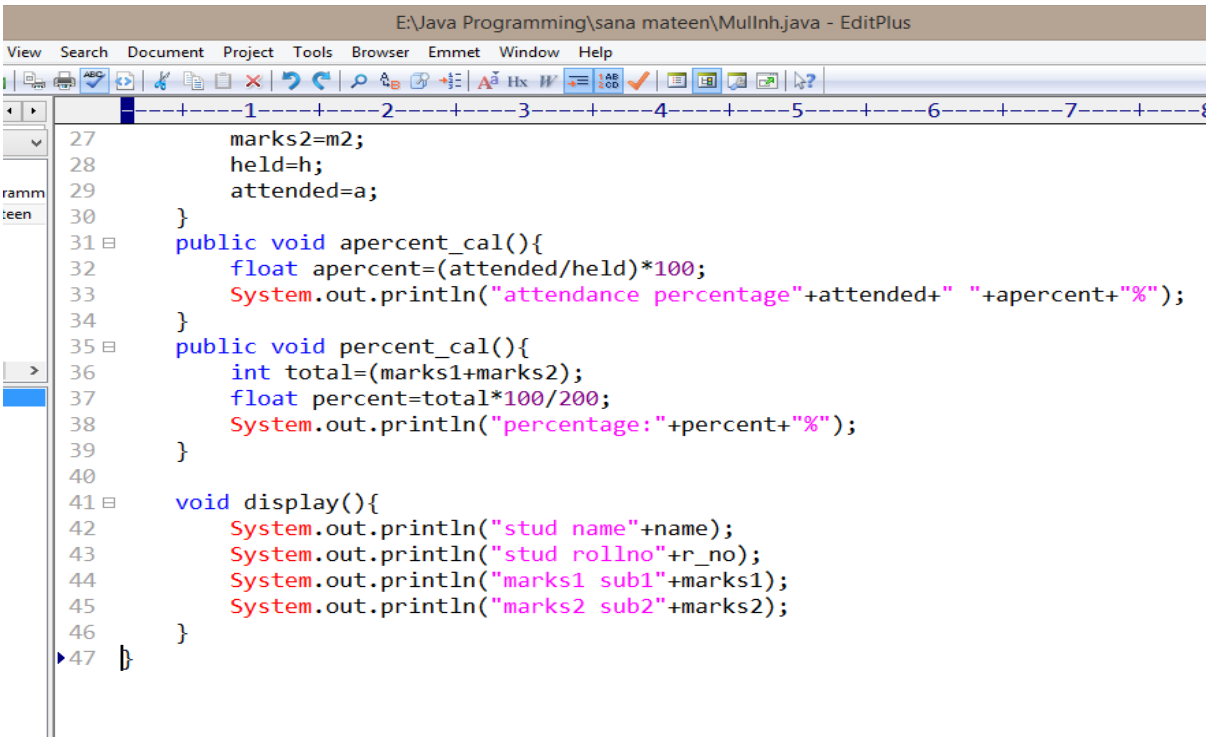

```
class interexam implements Callb{
public void callback(int p){
System.out.println("callback called with"+p);
}
public static void main(String[] args){
Callb b=new interexam();
b.callback(43);
}
}
```

```
C:\Users\sana mateen\Desktop\oops>javac interexam.java

C:\Users\sana mateen\Desktop\oops>java interexam
callback called with43

C:\Users\sana mateen\Desktop\oops>
```

6. Write a java program to implement multiple inheritance?

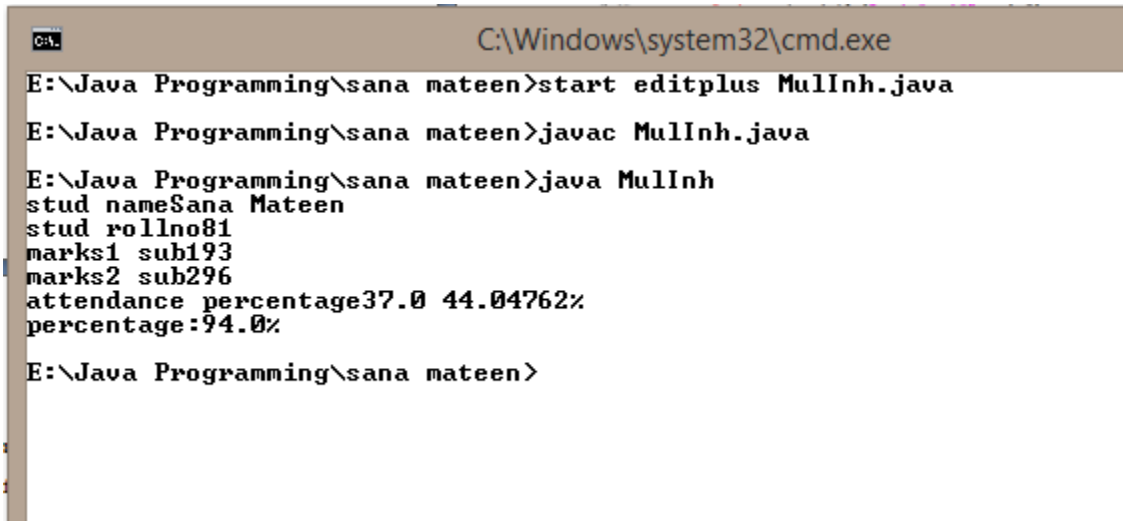


```
import java.io.*;
interface Exam{
    void percent_cal();
}
interface Attendance{
    void apercent_cal();
}

class MulInh{
    public static void main(String[] args){
        Student s=new Student("Sana Mateen",81,93,96,84,37);
        s.display();
        s.apercent_cal();
        s.percent_cal();

    }
}
class Student implements Attendance,Exam{
    String name;
    int r_no,marks1,marks2;
    float held,attended;
    Student(String n,int r,int m1,int m2,float h,float a)
    {
        name=n;
        r_no=r;
        marks1=m1;
```



```
        marks2=m2;
        held=h;
        attended=a;
    }
    public void apercent_cal(){
        float apercent=(attended/held)*100;
        System.out.println("attendance percentage"+attended+" "+apercent+"%");
    }
    public void percent_cal(){
        int total=(marks1+marks2);
        float percent=total*100/200;
        System.out.println("percentage:"+percent+"%");
    }

    void display(){
        System.out.println("stud name"+name);
        System.out.println("stud rollno"+r_no);
        System.out.println("marks1 sub1"+marks1);
        System.out.println("marks2 sub2"+marks2);
    }
}
```

Output:



7. What are the methods available in the Character Streams?Discuss

Character Stream Classes are used to read characters from the source and write characters to destination.

There are two kinds of Character Stream classes - Reader classes and Writer classes.

Reader Classes - These classes are subclasses of an abstract class, Reader and they are used to read characters from a source(file, memory or console).

Writer Classes - These classes are subclasses of an abstract class, Writer and they used to write characters to a destination(file, memory or console).

Reader Class Methods:

Methods Description

int read() This method reads a characters from the input stream.

int read(char[] ch) This method reads a chunk of characters from the input stream and store them in its char array, ch.

close() This method closes this output stream and also frees any system resources connected with it.

Writer Class Methods:

Methods Description

abstract void flush() This method flushes the output steam by forcing out buffered bytes to be written out.

void write(int c) This method writes a characters(contained in an int) to the output stream.

void write(char[] arr) This method writes a whole char array(arr) to the output stream.

abstract void close() This method closes this output stream and also frees any resources connected with this output stream.

8. Distinguish between Byte Stream Classes and Character Stream Classes?

Java defines two types of streams: byte and character.

Byte streams provide a convenient means for handling input and output of bytes.

Byte streams are used, for example, when reading or writing binary data.

Character streams provide a convenient means for handling input and output of characters.

The Byte Stream Classes

Byte streams are defined by using two class hierarchies. At the top are two abstract classes: InputStream and OutputStream.

Each of these abstract classes has several concrete subclasses that handle the differences among various devices, such as disk files, network connections, and even memory buffers.

Remember, to use the stream classes, you must import java.io.

Character-stream class	Description	Corresponding byte class
`Reader`	Abstract class for character-input streams	`InputStream`
`BufferedReader`	Buffers input, parses lines	`BufferedInputStream`
`LineNumberReader`	Keeps track of line numbers	`LineNumberInputStream`
`CharArrayReader`	Reads from a character array	
`InputStreamReader`	Translates a byte stream into a character stream	*(none)*
`FileReader`	Translates bytes from a file into a character stream	`FileInputStream`
`FilterReader`	Abstract class for filtered character input	`FilterInputStream`
`PushbackReader`	Allows characters to be pushed back	`PushbackInputStream`
`PipedReader`	Reads from a `PipedWriter`	`PipedInputStream`
`StringReader`	Reads from a `String`	`StringBufferInputStream`
`Writer`	Abstract class for character-output streams	`OutputStream`
`BufferedWriter`	Buffers output, uses platform's line separator	`BufferedOutputStream`
`CharArrayWriter`	Writes to a character array	`ByteArrayOutputStream`
`FilterWriter`	Abstract class for filtered character output	`FilterOutputStream`
`OutputStreamWriter`	Translates a character stream into a byte stream	*(none)*
`FileWriter`	Translates a character stream into a byte file	`FileOutputStream`
`PrintWriter`	Prints values and objects to a `Writer`	`PrintStream`
`PipedWriter`	Writes to a `PipedReader`	`PipedOutputStream`

9. Explain different types of access specifiers along with example program?

As the name suggests access modifiers in Java helps to restrict the scope of a class, constructor , variable , method or data member. There are four types of access modifiers available in java:

Default – No keyword required
Private
Protected
Public

	Private	No Modifier	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package non-subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Private:

pexample - Notepad

private

File Edit Format View Help

```
class A{
private int data=40;
private void msg(){System.out.println("Hello java");}
}

public class pexample{
 public static void main(String args[]){
   A obj=new A();
   System.out.println(obj.data);//Compile Time Error
   obj.msg();//Compile Time Error
   }
}

E:\oops>start notepad pexample.java

E:\oops>javac pexample.java
pexample.java:9: error: data has private access in A
   System.out.println(obj.data);//Compile Time Error
                           ^
pexample.java:10: error: msg() has private access in A
   obj.msg();//Compile Time Error
      ^
2 errors
```

Default:

A - Notepad

File Edit Format View Help

default

```
package pack;
class A{
  void msg(){System.out.println("Hello");}
}
```

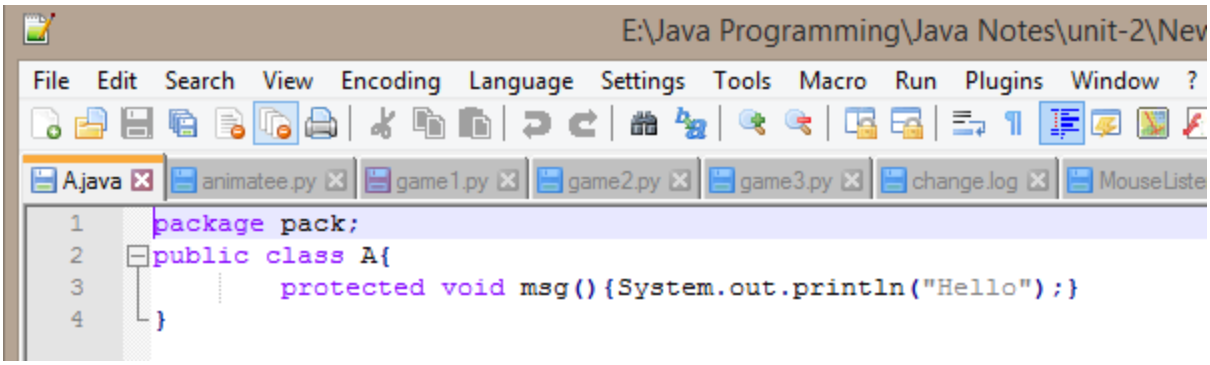
B - Notepad

File Edit Format View Help

```
package MyPack;
import pack.*;
class B{
  public static void main(String args[]){
   A obj = new A();//Compile Time Error
   obj.msg();//Compile Time Error
  }
}
```

```
E:\oops\MyPack>start notepad B.java

E:\oops\MyPack>javac B.java
B.java:2: error: package pack does not exist
import pack.*;
^
B.java:5: error: cannot find symbol
   A obj = new A();//Compile Time Error
   ^
  symbol:   class A
  location: class B
B.java:5: error: cannot find symbol
   A obj = new A();//Compile Time Error
               ^
  symbol:   class A
  location: class B
3 errors
```

Protected:



E:\Java Programming\Java Note

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Wind

B.java | A.java | animatee.py | game1.py | game2.py | game3.py | change

```
1  import pack.A;
2  class B extends A{
3      public static void main(String[] a){
4              B obj=new B();
5              obj.msg();
6      }
7  }
```

"sana mateen"

```
E:\Java Programming\Java Notes\unit-2>javac -d . A.java

E:\Java Programming\Java Notes\unit-2>javac B.java

E:\Java Programming\Java Notes\unit-2>java B
Hello

E:\Java Programming\Java Notes\unit-2>title "sana mateen"
```

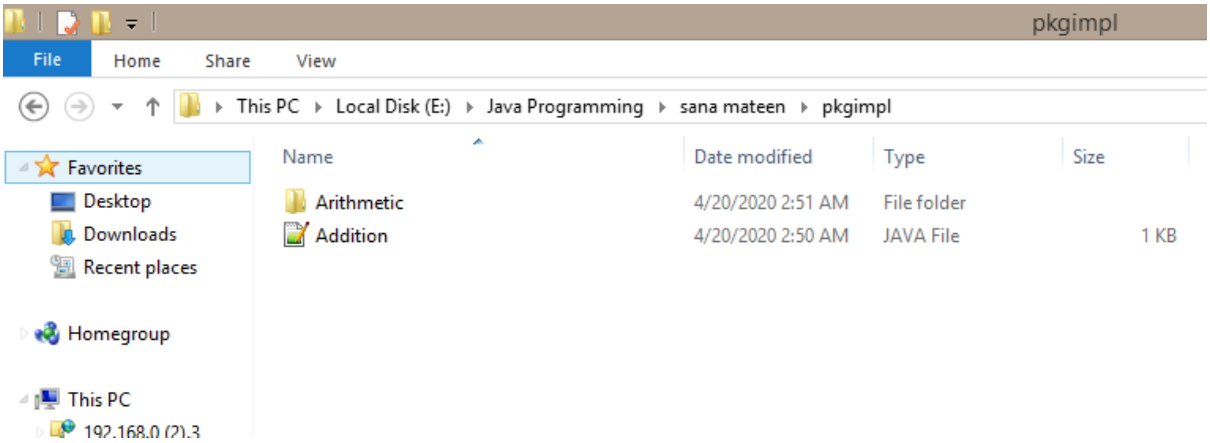
10. With an example program explain how to create user defined package and import in another class?

Addition - Notepad

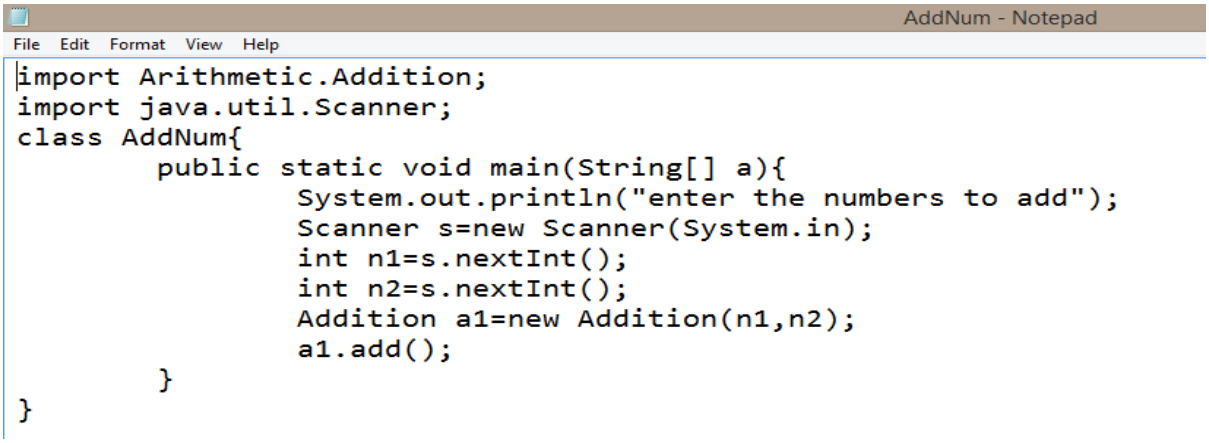
File Edit Format View Help

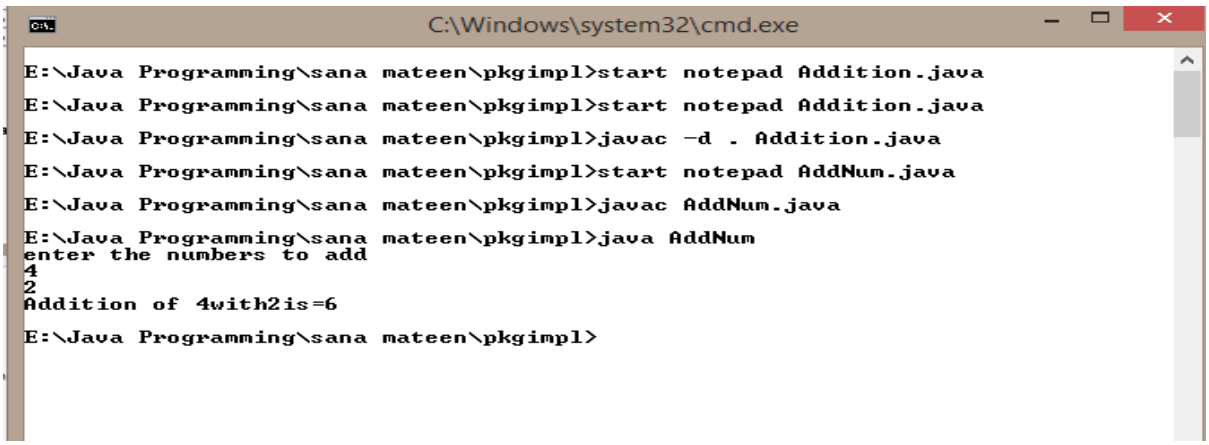
```
package Arithmetic;
public class Addition{
        int n1,n2;
        public Addition(int a,int b){
                this.n1=a;
                this.n2=b;
        }
        public void add(){
                System.out.println("Addition of "+n1+"with"+n2+"is="+(n1+n2));
        }
}
```

```
E:\Java Programming\sana mateen>mkdir pkgimpl

E:\Java Programming\sana mateen>cd pkgimpl

E:\Java Programming\sana mateen\pkgimpl>start notepad Addition.java

E:\Java Programming\sana mateen\pkgimpl>javac -d . Addition.java

E:\Java Programming\sana mateen\pkgimpl>
```


pkgimpl
File
Home
Share
View
This PC ▸ Local Disk (E:) ▸ Java Programming ▸ sana mateen ▸ pkgimpl
Favorites
Desktop
Downloads
Recent places
Homegroup
This PC
Name
Date modified
Type
Size
Arithmetic
4/20/2020 2:51 AM
File folder
Addition
4/20/2020 2:50 AM
JAVA File
1 KB


Arithmetic
Home
Share
View
This PC ▸ Local Disk (E:) ▸ Java Programming ▸ sana mateen ▸ pkgimpl ▸ Arithmetic
Desktop
Downloads
Recent places
Name
Date modified
Type
Size
Addition.class
4/20/2020 2:51 AM
CLASS File
1 KB


AddNum - Notepad
File Edit Format View Help
import Arithmetic.Addition;
import java.util.Scanner;
class AddNum{
public static void main(String[] a){
System.out.println("enter the numbers to add");
Scanner s=new Scanner(System.in);
int n1=s.nextInt();
int n2=s.nextInt();
Addition a1=new Addition(n1,n2);
a1.add();
}
}


C:\Windows\system32\cmd.exe
E:\Java Programming\sana mateen\pkgimpl>start notepad Addition.java
E:\Java Programming\sana mateen\pkgimpl>start notepad Addition.java
E:\Java Programming\sana mateen\pkgimpl>javac -d . Addition.java
E:\Java Programming\sana mateen\pkgimpl>start notepad AddNum.java
E:\Java Programming\sana mateen\pkgimpl>javac AddNum.java
E:\Java Programming\sana mateen\pkgimpl>java AddNum
enter the numbers to add
4
2
Addition of 4with2is=6
E:\Java Programming\sana mateen\pkgimpl>

11. Write a java program to extend one interface from another interface?

When a class implements an interface that inherits another interface, it must provide implementations for all methods required by the interface inheritance chain. Following is an example:

```
interface A {
void meth1();
void meth2();
}
// B now includes meth1() and meth2() -- it adds meth3().
interface B extends A {
void meth3();
}
// This class must implement all of A and B
class MyClass implements B {
public void meth1() {
System.out.println("Implement meth1().");
 }
  public void meth2() {
System.out.println("Implement meth2().");
}
  public void meth3() {
System.out.println("Implement meth3().");
}
}
class IFExtend {
  public static void main(String arg[]) {
MyClass ob = new MyClass();
 ob.meth1();
ob.meth2();
ob.meth3();
}
}
```

```
C:\Users\sana mateen\Desktop\oops>start notepad IFExtend.java

C:\Users\sana mateen\Desktop\oops>javac IFExtend.java

C:\Users\sana mateen\Desktop\oops>java IFExtend
Implement meth1().
Implement meth2().
Implement meth3().
```

12. Explain in detail about Console Class?

It is used to read from and write to the console, if one exists. It implements the Flushable interface.

Console is primarily a convenience class because most of its functionality is available through System.in and System.out.

Console supplies no constructors.

Instead, a Console object is obtained by calling System.console( ), which is shown here:

static Console console( )

If a console is available, then a reference to it is returned. Otherwise, null is returned.

A console will not be available in all cases. Thus, if null is returned, no console I/O is possible.

Notice that the input methods, such as readLine( ), throw IOError if an input error occurs. IOError is a subclass of Error. It indicates an I/O failure that is beyond the control of your program. Thus, you will not normally catch an IOError.

Frankly, if an IOError is thrown while accessing the console, it usually means there has been a catastrophic system failure.

Method	Description
void flush( )	Causes buffered output to be written physically to the console.
Console format(String *fmtString*, Object...*args*)	Writes *args* to the console using the format specified by *fmtString*.
Console printf(String *fmtString*, Object...*args*)	Writes *args* to the console using the format specified by *fmtString*.
Reader reader( )	Returns a reference to a **Reader** connected to the console.
String readLine( )	Reads and returns a string entered at the keyboard. Input stops when the user presses ENTER. If the end of the console input stream has been reached, **null** is returned. An **IOError** is thrown on failure.
String readLine(String *fmtString*, Object...*args*)	Displays a prompting string formatted as specified by *fmtString* and *args*, and then reads and returns a string entered at the keyboard. Input stops when the user presses ENTER. If the end of the console input stream has been reached, **null** is returned. An **IOError** is thrown on failure.
char[ ] readPassword( )	Reads a string entered at the keyboard. Input stops when the user presses ENTER. The string is not displayed. If the end of the console input stream has been reached, **null** is returned. An **IOError** is thrown on failure.
char[ ] readPassword(String *fmtString*, Object... *args*)	Displays a prompting string formatted as specified by *fmtString* and *args*, and then reads a string entered at the keyboard. Input stops when the user presses ENTER. The string is not displayed. If the end of the console input stream has been reached, **null** is returned. An **IOError** is thrown on failure.
PrintWriter writer( )	Returns a reference to a **Writer** connected to the console.

```java
import java.io.Console;
class ConsoleDemo{
public static void main(String args[]){
Console c=System.console();
System.out.println("Enter your name: ");
String n=c.readLine();
System.out.println("Welcome "+n);
System.out.println("Enter password: ");
char[] ch=c.readPassword();
String pass=String.valueOf(ch);//converting char array into string
System.out.println("Password is: "+pass);
}
}
```

```
C:\Users\sana mateen\Desktop\oops\streams>start EditPlus ConsoleDemo.java

C:\Users\sana mateen\Desktop\oops\streams>javac ConsoleDemo.java

C:\Users\sana mateen\Desktop\oops\streams>java ConsoleDemo
Enter your name:
sana mateen
Welcome sana mateen
Enter password:

Password is: sana@
```

13. Explain the process of reading console input and writing console output with an example program?

Reading Console Input:

- **Connect the keyboard to an input stream object. Here, we can use InputStreamReader that can read data from the keyboard.**
- InputSteamReader obj = new InputStreamReader (System.in);
- **Connect InputStreamReader to BufferReader, which is another input type of stream.**
- We are using BufferedReader as it has got methods to read data properly, coming from the stream.
- BufferedReader br = new BufferedReader (obj);
- The above two steps can be combined and rewritten in a single statement as:
- BufferedReader br = new BufferedReader (new InputStreamReader (System.in));

Now, we can read the data coming from the keyboard using read () and readLine () methods available in BufferedReader class.

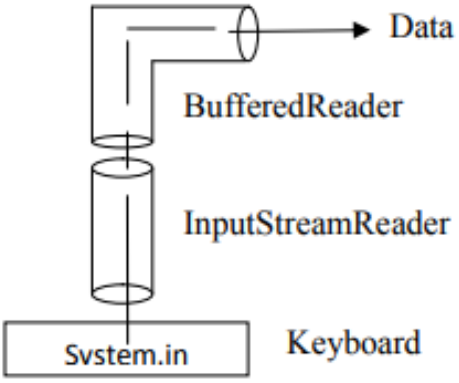


Figure: Reading data from keyboard

```
1  import java.io.*;
2  public class BufferedReaderExample{
3  public static void main(String args[])throws Exception{
4      InputStreamReader r=new InputStreamReader(System.in);
5      BufferedReader br=new BufferedReader(r);
6      System.out.println("Enter your name");
7      String name=br.readLine();
8      System.out.println("Welcome "+name);
9  }
10 }
```

```
C:\Users\sana mateen\Desktop\oops\streams>start EditPlus BufferedReaderExample.j
ava

C:\Users\sana mateen\Desktop\oops\streams>javac BufferedReaderExample.java

C:\Users\sana mateen\Desktop\oops\streams>java BufferedReaderExample
Enter your name
sana mateen
Welcome sana mateen

C:\Users\sana mateen\Desktop\oops\streams>
```

Writing Console Output:

- Console output is most easily accomplished with print( ) and println( ).
- **These methods are defined by the class PrintStream (which is the type of object referenced by System.out).**
- Even though System.out is a byte stream, using it for simple program output is still acceptable.
- Because PrintStream is an output stream derived from OutputStream, it also implements the low-level method write( ).

- Thus, write( ) can be used to write to the console.
- The simplest form of write( ) defined by PrintStream is shown here:
- void write(int byteval)
- This method writes the byte specified by byteval.
- Although byteval is declared as an integer, only the low-order eight bits are written. Here is a short example that uses write( ) to output the character "A" followed by a newline to the screen:

```
class ConsoleOutput {
    public static void main(String args[]) {
        int b;
        b = 'A';
        System.out.write(b);
        System.out.write('\n');
    }
}
```

```
C:\Users\sana mateen\Desktop\oops\streams>start EditPlus ConsoleOutput.java

C:\Users\sana mateen\Desktop\oops\streams>javac ConsoleOutput.java

C:\Users\sana mateen\Desktop\oops\streams>java ConsoleOutput
A

C:\Users\sana mateen\Desktop\oops\streams>
```

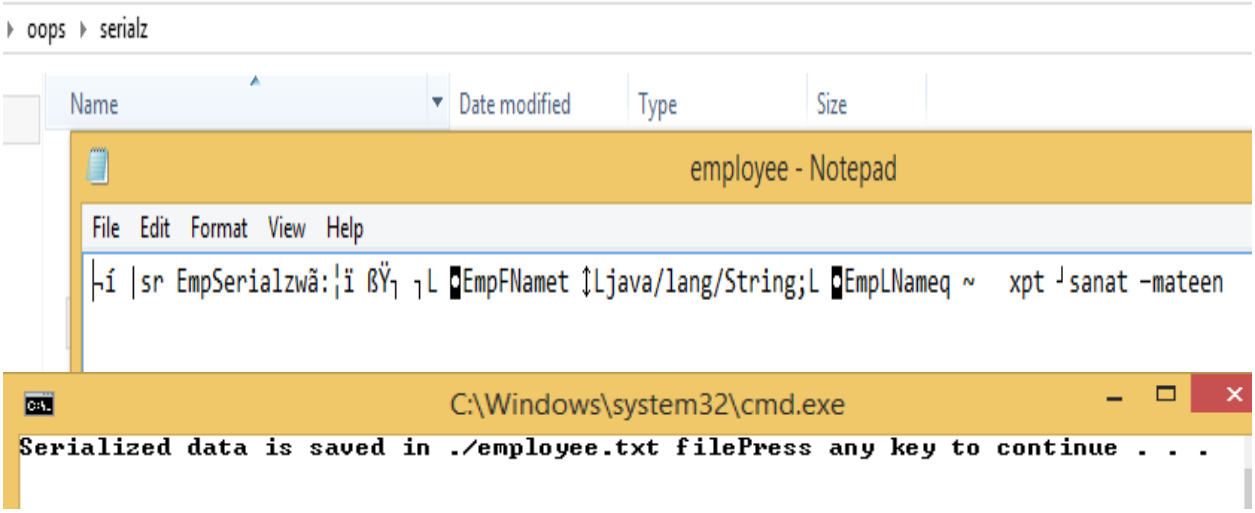
14. Write a java program to implement serialization and deserialization?

Serialization is a mechanism of converting the state of an object into a byte stream.

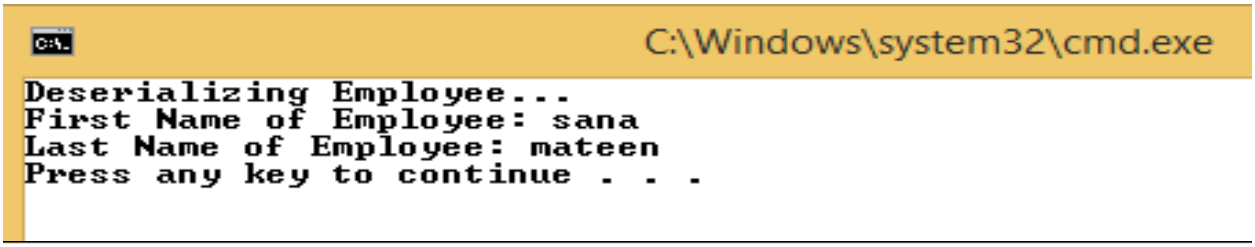
Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object.

```java
import java.io.Serializable;
public class EmpSerialz implements Serializable
{
    public String EmpFName;
    public String EmpLName;
}
```

```java
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
public class EmpSerialzImpl {
  public static void main(String[] args) {
    EmpSerialz emp = new EmpSerialz();
    emp.EmpFName = "sana";
    emp.EmpLName = "mateen";
    try {
      FileOutputStream fileOut = new FileOutputStream("./employee.txt");
      ObjectOutputStream out = new ObjectOutputStream(fileOut);
      out.writeObject(emp);
      out.close();
      fileOut.close();
      System.out.printf("Serialized data is saved in ./employee.txt file");
    } catch (IOException i) {
      i.printStackTrace();
    }
  }
}
```



Deserialization




```java
import java.io.*;
public class EmpDeSerialzImpl {
  public static void main(String[] args) {
    EmpSerialz emp = null;
    try {
      FileInputStream fileIn = new FileInputStream("./employee.txt");
      ObjectInputStream in = new ObjectInputStream(fileIn);
      emp = (EmpSerialz) in.readObject();
      in.close();
      fileIn.close();
    } catch (IOException i) {
      i.printStackTrace();
      return;
    } catch (ClassNotFoundException c) {
      System.out.println("Employee class not found");
      c.printStackTrace();
      return;
    }
    System.out.println("Deserializing Employee...");
    System.out.println("First Name of Employee: " + emp.EmpFName);
    System.out.println("Last Name of Employee: " + emp.EmpLName);
  }
}
```

15. Explain autoboxing in detail with an example program?

Autoboxing: The automatic conversion of primitive data types into its equivalent Wrapper type is known as boxing.

For example, converting int to Integer class. The Java compiler applies autoboxing when a primitive value is:

Passed as a parameter to a method that expects an object of the corresponding wrapper class.

Assigned to a variable of the corresponding wrapper class.

```java
import java.io.*;

class AutoBoxAndUnBox
{
    public static void main (String[] args)
    {
        // creating an Integer Object
        // with value 10.
        Integer i = new Integer(10);

        // unboxing the Object
        int i1 = i;

        System.out.println("Value of i: " + i);
        System.out.println("Value of i1: " + i1);

        //Autoboxing of char
        Character gfg = 'a';

        // Auto-unboxing of Character
        char ch = gfg;
        System.out.println("Value of ch: " + ch);
        System.out.println("Value of gfg: " + gfg);

    }
}
```

```
---------- Compiler ----------
Output completed (2 sec consumed) - Normal Termination
```

C:\Windows\system32\cmd.exe

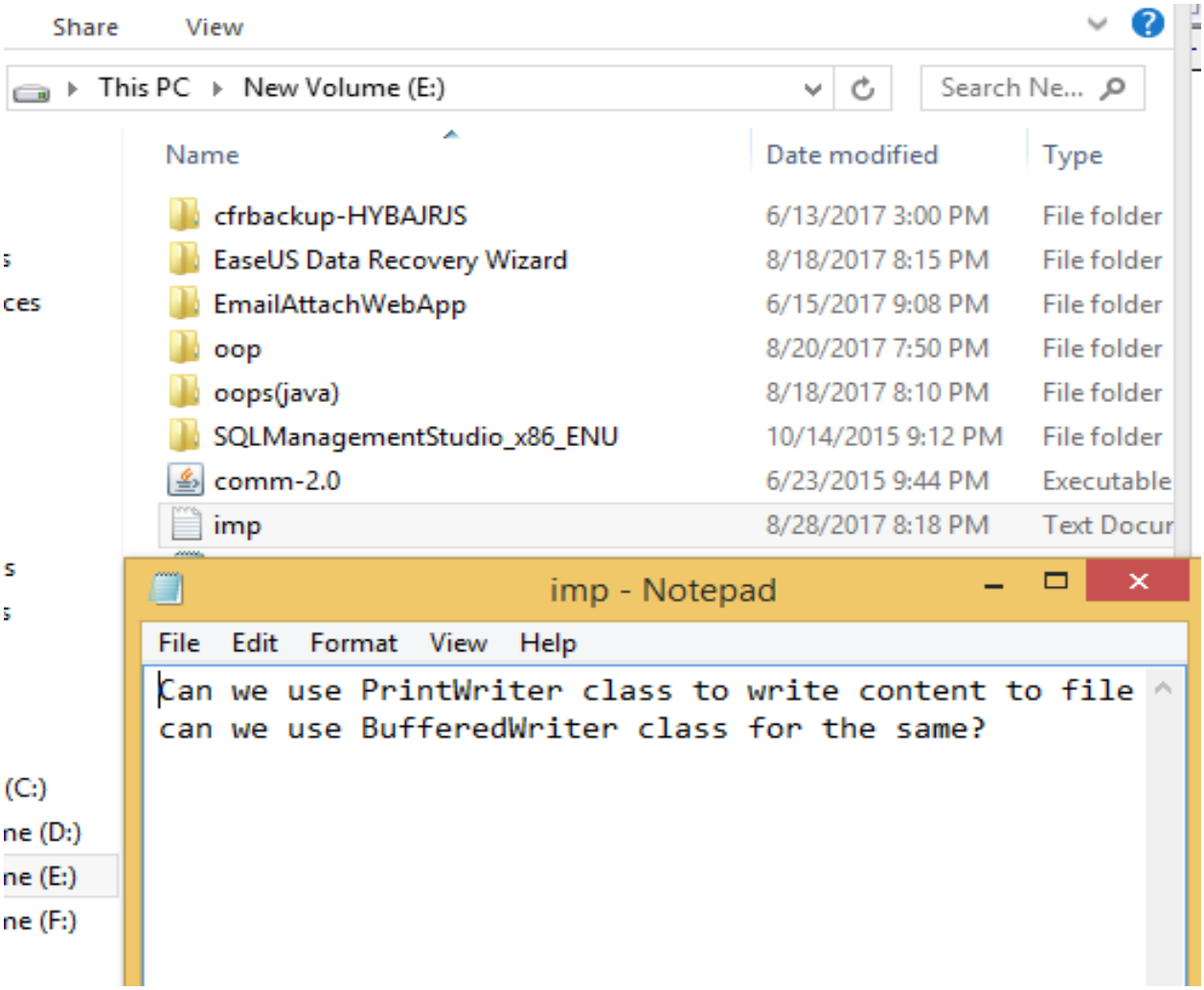
```
Value of i: 10
Value of i1: 10
Value of ch: a
Value of gfg: a
Press any key to continue . . .
```

16. Write a java program to read from and write content to a file?

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.Scanner;
class writereadfile
{
    public static void main(String[] a){
        File f=new File("E:\\","imp.txt");
        try{
            PrintWriter p=new PrintWriter(f);
            p.println("Can we use PrintWriter class to write content to file?");
            p.println("can we use BufferedWriter class for the same?");
            p.close();
            if(f.isFile()){
                System.out.println("successful creation");
                System.out.println("contents of the file are:");
                Scanner sc=new Scanner(f);
                String s1=sc.nextLine();
                String s2=sc.nextLine();
                System.out.println(s1+"\n"+s2);
            }else{
                System.out.println("try again");
            }
        }
        catch(FileNotFoundException e){
            e.printStackTrace();
        }
    }
}
```

```
C:\Users\sana mateen\Desktop\oops\fileexamp>javac writereadfile.java

C:\Users\sana mateen\Desktop\oops\fileexamp>java writereadfile
successful creation
contents of the file are:
Can we use PrintWriter class to write content to file?
```

```
C:\Users\sana mateen\Desktop\oops\fileexamp>javac writereadfile.java

C:\Users\sana mateen\Desktop\oops\fileexamp>java writereadfile
successful creation
contents of the file are:
Can we use PrintWriter class to write content to file?
can we use BufferedWriter class for the same?
```



Long Answer Questions for 10 Marks

17. Explain File class in detail along with an example program?

File class of java.io package provides some methods to know the properties of a file or a directory. We can create the File class object by passing the filename or directory name to it.

Java File class represents the files and directory pathnames in an abstract manner. This class is used for creation of files and directories, file searching, file deletion, etc.

The File object represents the actual file/directory on the disk. Following is the list of constructors to create a File object.

File obj = new File (filename);

File obj = new File (directoryname);

File obj = new File ("path", filename);

File obj = new File ("path", directoryname);

File class Methods:

Methods	Description
boolean isFile ()	Returns true if the File object contains a filename, otherwise false
boolean isDirectory ()	Returns true if the File object contains a directory name
boolean canRead ()	Returns true if the File object contains a file which is readable
boolean canWrite ()	Returns true if the File object contains a file which is writable
boolean canExecute ()	Returns true if the File object contains a file which is executable
Boolean exists ()	Returns true when the File object contains a file or directory which physically exists in the computer.
String getParent ()	Returns the name of the parent directory
String getPath ()	Gives the name of directory path of a file or directory
String getAbsolutePath ()	Gives the fully qualified path
long length ()	Returns a number that represents the size of the file in bytes
boolean delete ()	Deletes the file or directory whose name is in File object
boolean createNewFile ()	Automatically crates a new, empty file indicated by File object, if and only if a file with this name does not yet exist.

```
2  import java.io.File;
3  import java.io.IOException;
4  public class createfile
5  {
6      public static void main(String[] a){
7          try{
8              File f=new File("F:\\viva.txt");
9              if(f.createNewFile()){
10                 System.out.println("file created successfully");
11             } else {
12                 System.out.println("file not created");
13             }
14         }
15         catch(IOException e){
16             e.printStackTrace();
17         }
18
19
20     }
21 }
```

```
C:\Users\sana mateen\Desktop\oops\fileexamp>start EditPlus createfile.java

C:\Users\sana mateen\Desktop\oops\fileexamp>javac createfile.java

C:\Users\sana mateen\Desktop\oops\fileexamp>java createfile
file created successfully
```

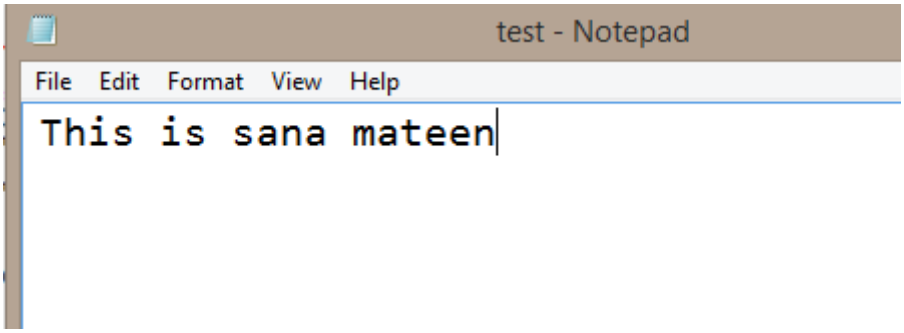
18. Write a java program to implement DataInputStream and DataOutputStream?

DataInputStream

DataInput - Notepad

File Edit Format View Help

```
import java.io.*;
public class DataInput {
  public static void main(String[] args) throws IOException {
    InputStream input = new FileInputStream("E:\\Java Programming\\test.txt");
    DataInputStream inst = new DataInputStream(input);
    int count = input.available();
    byte[] ary = new byte[count];
    inst.read(ary);
    for (byte bt : ary) {
      char k = (char) bt;
      System.out.print(k+"-");
    }
  }
}
```




```
E:\Java Programming>javac DataInput.java

E:\Java Programming>java DataInput
T-h-i-s- -i-s- -s-a-n-a- -m-a-t-e-e-n-
E:\Java Programming>
```

DataOutputStream:

DataOutput - Notepad

File Edit Format View Help

```
import java.io.*;
public class DataOutput {
    public static void main(String[] args) throws IOException {
        FileOutputStream file = new FileOutputStream("E:\\Java Programming\\testout.txt");
        DataOutputStream data = new DataOutputStream(file);
        data.writeChars("This is sana mateen");
        data.flush();
        data.close();
        System.out.println("Succcess...");
    }
}
```

```
E:\Java Programming>javac DataOutput.java

E:\Java Programming>java DataOutput
Succcess...

E:\Java Programming>
```

testout - Notepad

File Edit Format View Help

```
T h i s   i s   s a n a   m a t e e n
```

UNIT-3

Short Answer Questions for 2 Marks

1.What is thread based preemptive multitasking?

Thread Based Multitasking: **Executing different parts of the same program simultaneously with the help of a thread is called thread based multitasking.**

The operating system's scheduler decides which thread or program gets to use the CPU next and for how much time

2.How do we set priorities for threads?

- **To set a thread's priority, use the setPriority( ) method, which is a member of Thread.**
- This is its general form:
- **final void setPriority(int level)**
- Here, level specifies the new priority setting for the calling thread.
- **The value of level must be within the range MIN_PRIORITY and MAX_PRIORITY.**
- **Currently, these values are 1 and 10, respectively.**
- **To return a thread to default priority, specify NORM_PRIORITY, which is currently 5**.
- These **priorities are defined as static final variables** within Thread.

3.Explain the difference between process and threads

Process	**Thread**
Process means a program is in execution.	Thread means a segment of a process.
The process is not Lightweight.	Threads are Lightweight.
Communication between processes needs more time compared to thread.	Communication between threads requires less time compared to processes.
Process consume more resources.	Thread consume fewer resources.
The process is mostly isolated.	Threads share memory.

4.Explain how a multiple catch statement works?

A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler. So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.

- At a time only one exception occurs and at a time only one catch block is executed.
- All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception.

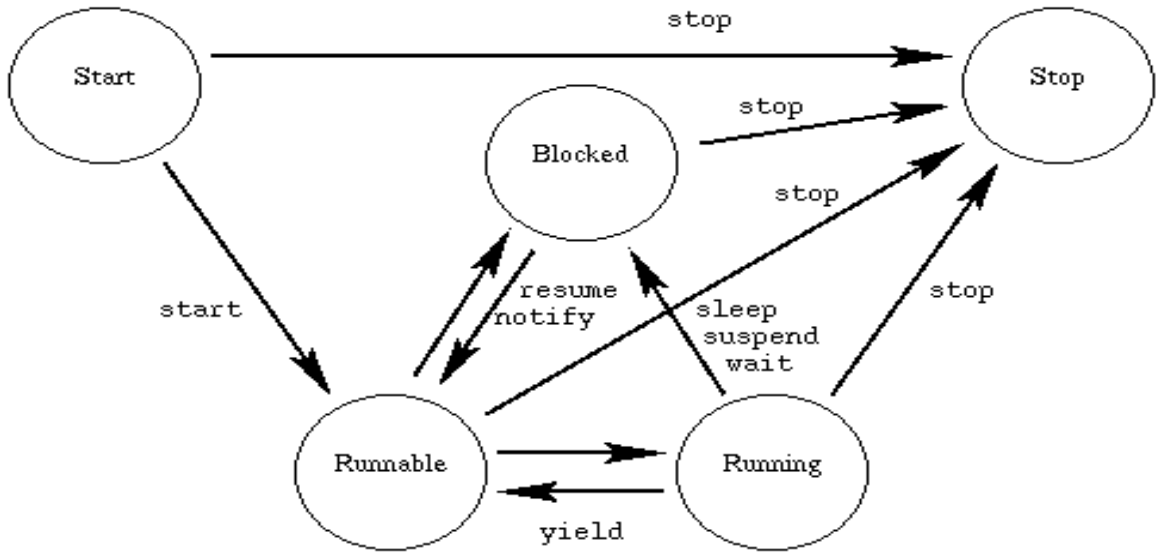
5.How do we start and stop thread?

Thread class:

Thread class provide constructors and methods to create and perform operations on a thread.Thread class extends Object class and implements Runnable interface.

- **public void start(): starts the execution of the thread.JVM calls the run() method on the thread.**
- **public void stop(): is used to stop the thread(depricated).**

6.Write the complete life cycle of a thread?



- **Java provides built-in support for multithreaded programming. A multithreaded program contains two or more parts that can run concurrently.**
- **Each part of such a program is called a thread.**

- **Thread is a smallest unit of code. Thread is also defined as a subprocess.**
- **A Thread sometimes called an execution context or a light weight process.**

7.List any six built-in exceptions in Java?

Built-in exceptions are the exceptions which are available in Java libraries.

- Arithmetic exception
- ArrayIndexOutOfBounds Exception
- ClassNotFoundException
- ClassNotFoundException
- IOException
- InterruptedException

8.What is Termination or Resumptive Model?

There are two basic models in exception-handling theory.

In termination the error is so critical there's no way to get back to where the exception occurred.

The alternative is called resumption.

It means that the exception handler is expected to do something to rectify the situation and then the faulty method is retried, presuming success the second time .

If you want resumption, it means you still hope to continue execution after the exception is handled.

Short Answer Questions for 3 Marks:

9.Differentiate between compiletime error and runtime error?

CompileTime Exception	**RunTime Exception**
The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException, SQLException etc.	**The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc.**
Checked exceptions are checked at compile-time.	**Unchecked exceptions are not checked at compile-time rather they are checked at runtime.**

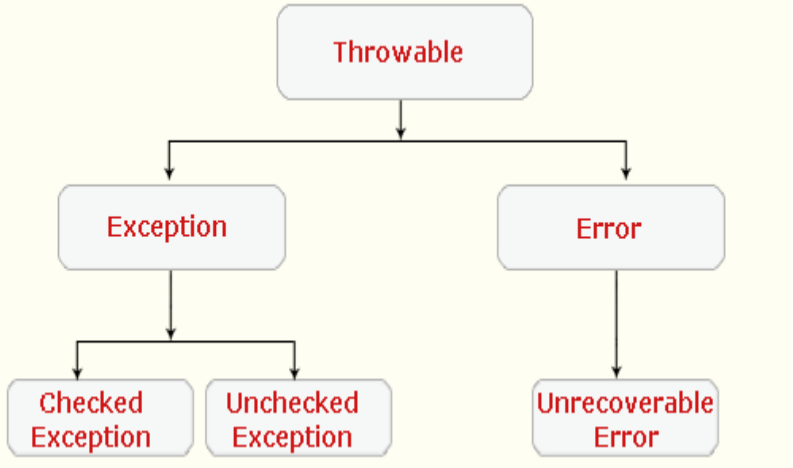
10.Differentiate between throw and throws clause?

throw	**throws**
The Java throw keyword is used to explicitly throw an exception.	"**Throws keyword**" is mainly used for handling checked exception as using throws we can declare multiple exceptions in one go.
The syntax of java throw keyword is given below. throw exception;	The throws keyword is used in method declaration, in order to explicitly specify the exceptions that a particular method might throw.
throw new IOException("sorry device error);	**The throws keyword is used in method declaration, in order to explicitly specify the exceptions that a particular method might throw.**

11.Explain the types of Exception?

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. The sun microsystem says there are three types of exceptions:

- Checked Exception
- Unchecked Exception
- Error



12.Which class is the super class for all types of errors and exceptions in Java?

Throwable class: Only objects that are instances of this

class (or one of its subclasses) are thrown by the Java Virtual Machine or can be thrown by the Java {@code throw} statement. Similarly, only this class or one of its subclasses can be the argument type in a {@code catch} clause.

13.What are the two ways of creating thread?

Threads can be created by using two mechanisms :
1. Extending the Thread class
2. Implementing the Runnable Interface

Thread creation by extending the Thread class

We create a class that extends the java.lang.Thread class.

This class overrides the run() method available in the Thread class.

A thread begins its life inside run() method.

We create an object of our new class and call start() method to start the execution of a thread.

Start() invokes the run() method on the Thread object.

Thread creation by implementing the Runnable Interface

We create a new class which implements java.lang.Runnable interface and override run() method. Then we instantiate a Thread object and call start() method on this object.

14.What is Thread State?

A thread in Java at any point of time exists in any one of the following states. A thread lies only in one of the shown states at any instant:

1.New
2.Runnable
3.Blocked
4.Waiting
5.Timed Waiting
6.Terminated

15.List any four Thread constructors?

Thread class:

Thread class provide constructors and methods to create and perform operations on a thread.Thread class extends Object class and implements Runnable interface.

Commonly used Constructors of Thread class:

Thread()

Thread(String name)

Thread(Runnable r)

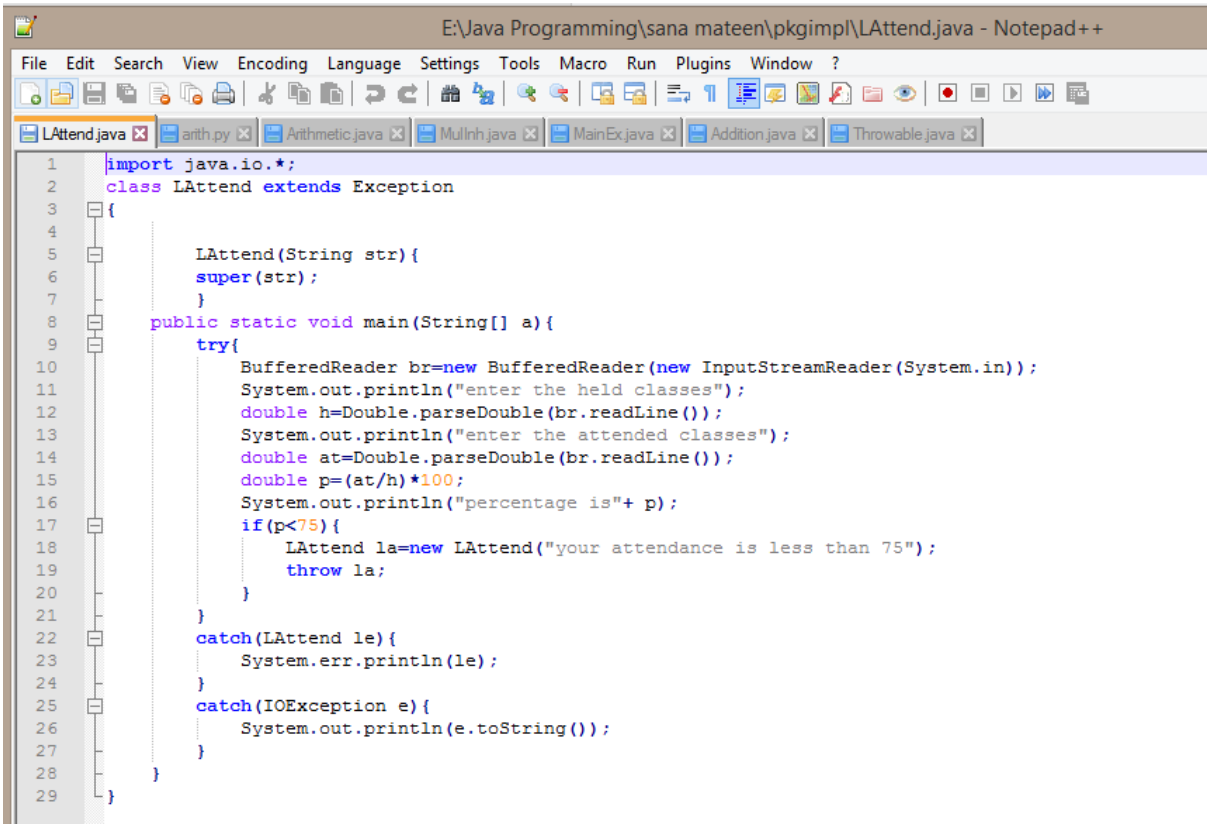
Thread(Runnable r,String name)

16.What is synchronization?

- Synchronization in java is the capability *to control the access of multiple threads to any shared resource*.Java Synchronization is better option where we want to allow only one thread to access the shared resource.
- Why use Synchronization
  - The synchronization is mainly used to
  - To prevent thread interference.
  - To prevent consistency problem.

Long Answer Questions for 5 Marks:

1.Explain user-defined exception with an example program?



Output:

```
E:\Java Programming\sana mateen\pkgimpl>javac LAttend.java

E:\Java Programming\sana mateen\pkgimpl>java LAttend
enter the held classes
65
enter the attended classes
35
percentage is53.84615384615385
LAttend: your attendance is less than 75
```

2.What is meant by re-throwing exception?Explain with an example program?

An exception can be rethrown in a catch block. This action will cause the exception to be passed to the calling method. If the rethrow operation occurs in the main method then the exception is passed to the JVM and displayed on the console. The purpose of the rethrow operation is to get the attention of the outside world that an exception has occurred and at the same time perform any contingency logic (such as logging) in the catch block. To learn how to rethrow an exception in Java, follow these four steps.

Program:

```
how-to-topics.txt | how-to-rethrow-an-exception-in-java.html | CheckObjectType.java | ThrowAnException.java | DivideByZeroException.java | HandleExceptions.java | Ret
1  public class RethrowAnException {
2      public static void main (String[] args) {
3          try {
4              String inputString=args[0];
5              System.out.println("Received: " + inputString);
6          }
7          catch (ArrayIndexOutOfBoundsException ex) {
8              System.out.println("Log entry: Array index out of bounds exception raised! Rethrowing exception...");
9              throw ex;
10         }
11     }
12 }
```

Output:

```
Command Prompt

c:\JavaStuff\how-tos\rethrow-an-exception>javac RethrowAnException.java

c:\JavaStuff\how-tos\rethrow-an-exception>
```

```
Command Prompt

c:\JavaStuff\how-tos\rethrow-an-exception>java RethrowAnException
Log entry: Array index out of bounds exception raised! Rethrowing exception...
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
        at RethrowAnException.main(RethrowAnException.java:4)

c:\JavaStuff\how-tos\rethrow-an-exception>
```

3.Does java support thread priorities?Justify your answer?

- **Java assigns to each thread a priority that determines how that thread should be treated with respect to the others.**
- Thread priorities are integers that specify the relative priority of one thread to another.
- As an absolute value, a priority is meaningless; a higher-priority thread doesn't run any faster than a lower-priority thread if it is the only thread running.
- Instead, a thread's priority is used to decide when to switch from one running thread to the next.
- This is called a context switch.
- The rules that determine when a context switch takes place are simple:
- **A thread can voluntarily relinquish control**. This is done by explicitly yielding, sleeping, or blocking on pending I/O. In this scenario, all other threads are examined, and the highest-priority thread that is ready to run is given the CPU.
- **A thread can be preempted by a higher-priority thread. In this case, a lower-priority thread that does not yield the processor is simply preempted—no matter what it is doing— by a higher-priority thread.**
- Basically, as soon as a higher-priority thread wants to run, it does. This is called preemptive multitasking.
- **To set a thread's priority, use the setPriority( ) method, which is a member of Thread.**
- This is its general form:
- **final void setPriority(int level)**
- Here, level specifies the new priority setting for the calling thread.
- **The value of level must be within the range MIN_PRIORITY and MAX_PRIORITY.**
- **Currently, these values are 1 and 10, respectively.**
- **To return a thread to default priority, specify NORM_PRIORITY, which is currently 5**.
- These **priorities are defined as static final variables** within Thread.
- You can obtain the current priority setting by calling the getPriority( ) method of Thread, shown here:

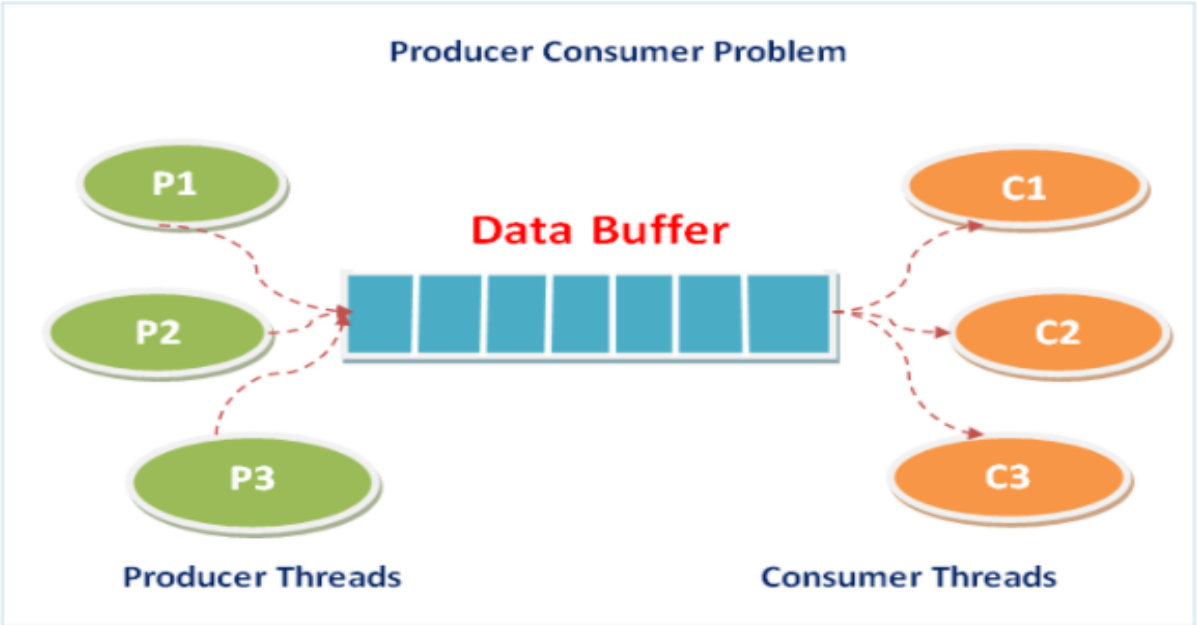
- **final int getPriority( )**
- Implementations of Java may have radically different behavior when it comes to scheduling.
- Most of the inconsistencies arise when you have threads that are relying on preemptive behavior, instead of cooperatively giving up CPU time.

4.Describe producer-consumer problem using inter-thread communication?

The Producer-Consumer problem is a classic problem this is used for multi-process synchronization i.e. synchronization between more than one processes.

In the producer-consumer problem, there is one Producer that is producing something and there is one Consumer that is consuming the products produced by the Producer. The producers and consumers share the same memory buffer that is of fixed-size.

The job of the Producer is to generate the data, put it into the buffer, and again start generating data. While the job of the Consumer is to consume the data from the buffer.



Java includes an elegant interprocess communication mechanism via the wait( ), notify( ), and notifyAll( ) methods.

These methods are implemented as final methods in Object, so all classes have them.

All three methods can be called only from within a synchronized context.

The rules for using these methods are actually quite simple:

wait( ) tells the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls notify( ) or notifyAll( ).

notify( ) wakes up a thread that called wait( ) on the same object.

notifyAll( ) wakes up all the threads that called wait( ) on the same object. One of the threads will be granted access.

These methods are declared within Object, as shown here:

final void wait( ) throws InterruptedException

final void notify( )

final void notify All( )

```
1  class Q
2  {
3      int num;
4      synchronized int get()
5      {
6          System.out.println("Got : " + num);
7          return num;
8      }
9      synchronized void put(int num)
10     {
11         this.num = num;
12         System.out.println("Put : " + num);
13     }
14 }
```

```
15  class Producer implements Runnable
16 {
17      Q que;
18      Producer(Q que)
19      {
20          this.que = que;
21          new Thread(this, "Producer").start();
22      }
23      public void run()
24      {
25          int n = 0;
26          while(true)
27          {
28              que.put(n++);
29          }
30      }
31  }

32  class Consumer implements Runnable
33 {
34      Q que;
35      Consumer(Q que)
36      {
37          this.que = que;
38          new Thread(this, "Consumer").start();
39      }
40      public void run()
41      {
42          while(true)
43          {
44              que.get();
45          }
46      }
47  }
```

```
48  class PC
49  {
50      public static void main(String args[])
51      {
52          Q que = new Q();
53          new Producer(que);
54          new Consumer(que);
55          System.out.println("Press Control-C to stop...");
56      }
57  }
```

Output:

```
Put : 0
Put : 1
Put : 2
Put : 3
Press Control-C to stop...
Put : 4
Put : 5
Put : 6
Put : 7
Put : 8
Put : 9
Put : 10
Put : 11
Put : 12
Put : 13
Put : 14
Put : 15
Put : 16
Put : 17
Put : 18
Put : 19
Put : 20
Put : 21
Put : 22
Put : 23
Put : 24
Put : 25
Put : 26
Put : 27
Put : 28
Put : 29
Put : 30
Put : 31
Put : 32
```

5.What is an Exception?Explain Exception Handling mechanism with an example program?

The **exception handling in java** is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained.

What is exception?

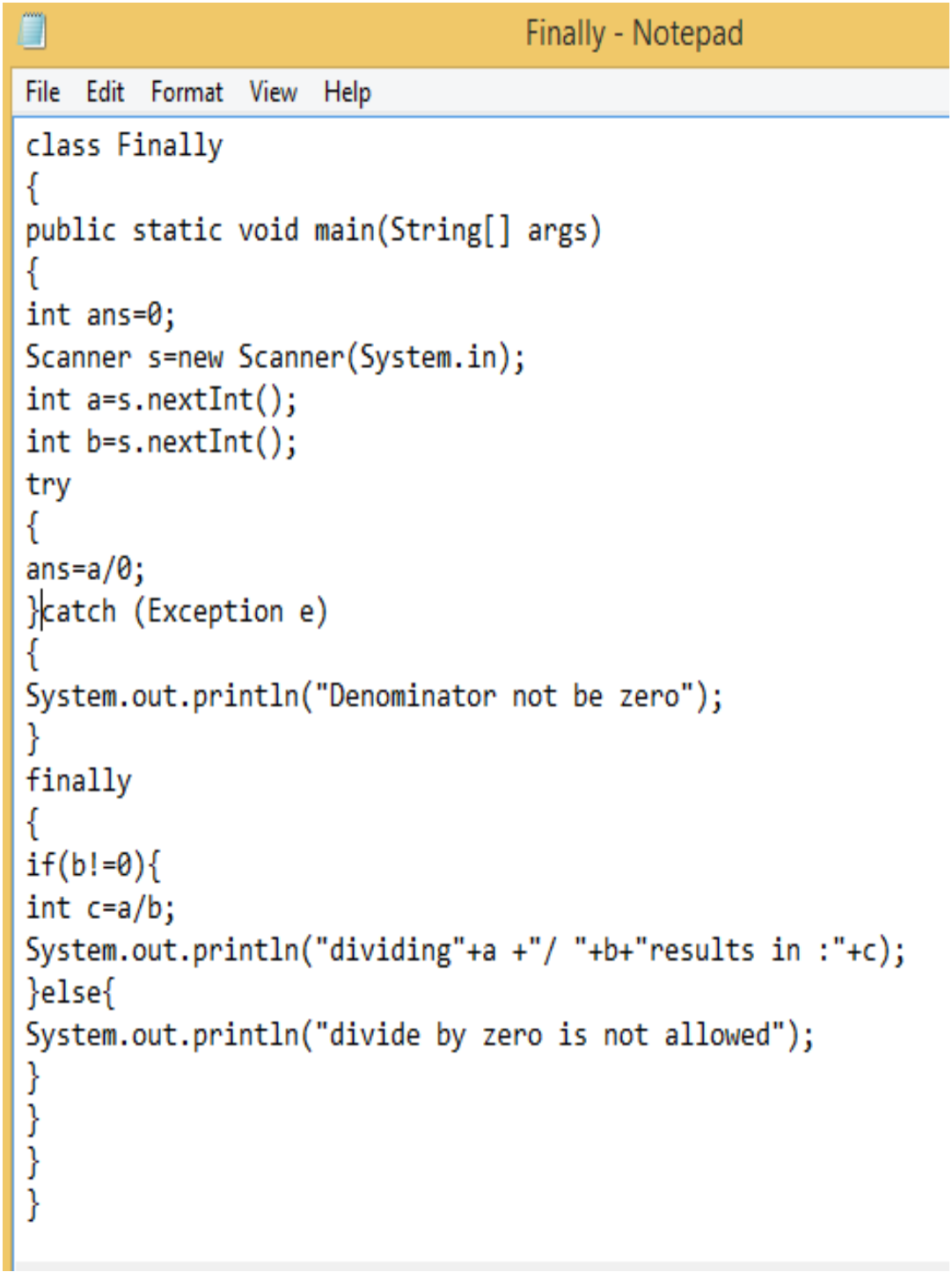
Meaning: Exception is an abnormal condition.

In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

What is exception handling?

Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc.

Program:


Finally - Notepad
File Edit Format View Help
class Finally
{
public static void main(String[] args)
{
int ans=0;
Scanner s=new Scanner(System.in);
int a=s.nextInt();
int b=s.nextInt();
try
{
ans=a/0;
}catch (Exception e)
{
System.out.println("Denominator not be zero");
}
finally
{
if(b!=0){
int c=a/b;
System.out.println("dividing"+a +"/ "+b+"results in :"+c);
}else{
System.out.println("divide by zero is not allowed");
}
}
}
}

Output:

```
C:\Users\sana mateen\Desktop\oops>javac Finally.java

C:\Users\sana mateen\Desktop\oops>java Finally
8
2
Denominator not be zero
dividing8/ 2results in :4

C:\Users\sana mateen\Desktop\oops>
```

6.Write a java program that illustrates the application of multiple catch statements?

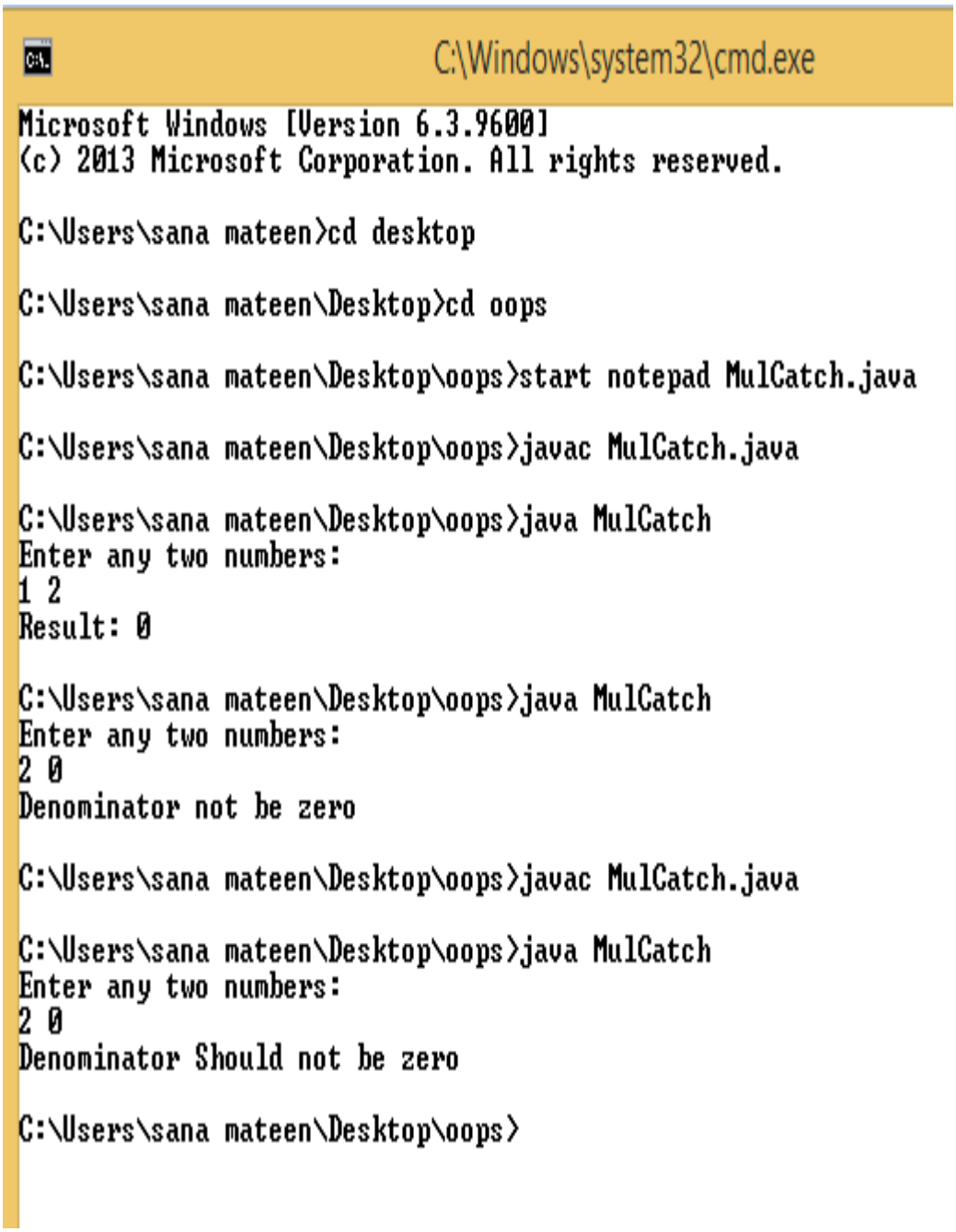
Program

MulCatch - Notepad

File Edit Format View Help

```
import java.util.*;
class MulCatch
{
public static void main(String[] args)
{
int a, b, ans=0;
Scanner s=new Scanner(System.in);
System.out.println("Enter any two numbers: ");
try
{
        a=s.nextInt();
        b=s.nextInt();
        ans=a/b;
        System.out.println("Result: "+ans);
}
catch(ArithmeticException ae)
{
System.out.println("Denominator Should not be zero");
}
catch(Exception e)
{
System.out.println("Enter valid number");
}
}
}
```

Output


C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.
C:\Users\sana mateen>cd desktop
C:\Users\sana mateen\Desktop>cd oops
C:\Users\sana mateen\Desktop\oops>start notepad MulCatch.java
C:\Users\sana mateen\Desktop\oops>javac MulCatch.java
C:\Users\sana mateen\Desktop\oops>java MulCatch
Enter any two numbers:
1 2
Result: 0
C:\Users\sana mateen\Desktop\oops>java MulCatch
Enter any two numbers:
2 0
Denominator not be zero
C:\Users\sana mateen\Desktop\oops>javac MulCatch.java
C:\Users\sana mateen\Desktop\oops>java MulCatch
Enter any two numbers:
2 0
Denominator Should not be zero
C:\Users\sana mateen\Desktop\oops>

7.Write a program that creates two threads First thread prints the numbers from 1 to 100 and other thread prints the numbers from 100 to 1?

Program:

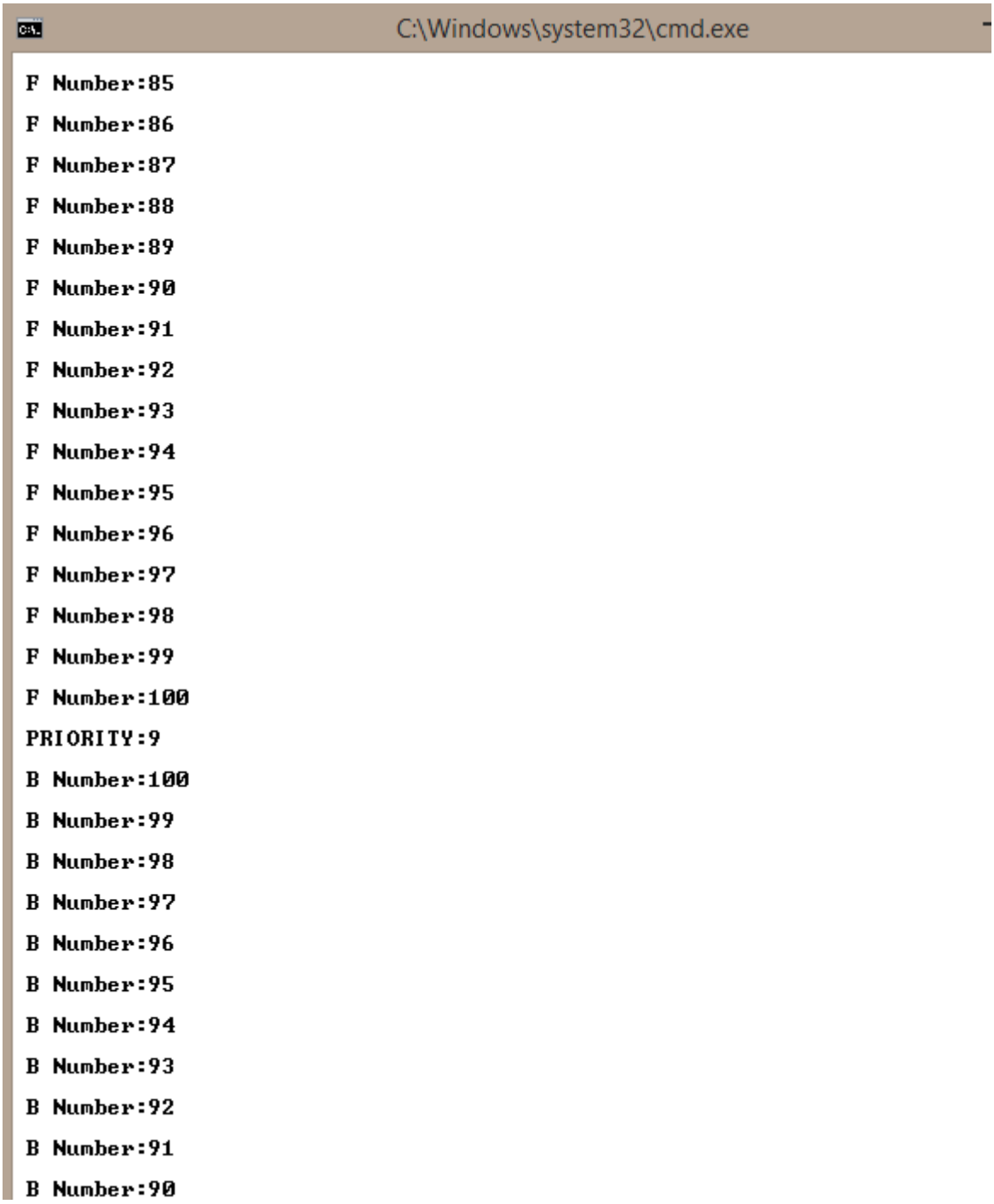
Threadnum - Notepad

File Edit Format View Help

```
class Threadnum{
        public static void main(String[] ar) throws InterruptedException{
                ThreadImpl im1=new ThreadImpl();
                ThreadImpl im2=new ThreadImpl();
                Thread t1=new Thread(im1);
                Thread t2=new Thread(im2);
                t1.setPriority(10);
                t1.start();
                t1.sleep(4000);
                t2.setPriority(9);
                t2.start();
        }
}
```

```
class ThreadImpl implements Runnable{

        public void run(){
                        Thread t=Thread.currentThread();
                        int n=t.getPriority();
                        System.out.println("\n PRIORITY:"+n);
                        if(n==10){
                        for(int i=1;i<=100;i++){
                                System.out.println("\n F Number:"+i);
                        }
                        }else if(n==9){
                        for(int i=100;i>=1;i--){
                                System.out.println("\n B Number:"+i);
                        }
                        }else{
                                System.out.println("\n F Nu");
                        }
                }

}
```

Output:



Long Answer Questions for 10 Marks:

8.Explain thread creation along with an example program?

Threads can be created by using two mechanisms :
1. Extending the Thread class
2. Implementing the Runnable Interface

Thread creation by extending the Thread class

We create a class that extends the **java.lang.Thread** class.

This class overrides the run() method available in the Thread class.

A thread begins its life inside run() method.

We create an object of our new class and call start() method to start the execution of a thread.

Start() invokes the run() method on the Thread object.

Thread creation by implementing the Runnable Interface

We create a new class which implements java.lang.Runnable interface and override run() method. Then we instantiate a Thread object and call start() method on this object.

```
// Main Class
public class Multi
{
    public static void main(String[] args)
    {
        int n = 8; // Number of threads
        for (int i=0; i<8; i++)
        {
            MultithreadingDemo object = new MultithreadingDemo();
            object.start();
        }
    }
}
```

```
class MultithreadingDemo extends Thread
{
    public void run()
    {
        try
        {
            // Displaying the thread that is running
            System.out.println ("Thread " +
                  Thread.currentThread().getId() +
                  " is running");

        }
        catch (Exception e)
        {
            // Throwing an exception
            System.out.println ("Exception is caught");
        }
    }
}
```

```
Thread 11 is running
Thread 13 is running
Thread 12 is running
Thread 10 is running
Thread 17 is running
Thread 16 is running
Thread 15 is running
Thread 14 is running
Press any key to continue . . .
```

Thread Creation by implementing Runnable Interface:

```
1  class Multithread implements Runnable{
2  public void run(){
3  System.out.println("thread is running...");
4  }
5
6  public static void main(String args[]){
7  Multithread m1=new Multithread();
8  Thread t1 =new Thread(m1);
9  t1.start();
10  }
11  }
```

```
C:\Windows\syste
thread is running...
Press any key to continue . . .
```

9.Explain thread synchronization with the help of example program?

When a thread is acting on an object preventing other threads from acting on the same object is called Thread Synchronization or Thread Safe.

The Object on which the Threads are synchronized is called synchronized object or Mutex (Mutually Exclusive Lock).

Thread synchronization is done in two ways: ·

Using synchronized block we can synchronize a block of statements.

e.g.: synchronized (obj){

statements;

}

To synchronize an entire method code we can use synchronized word before method name e.g.:

synchronized void method () {

}

```
class Safe {
    public static void main(String args[]){
        Reserve obj = new Reserve (1);
        Thread t1 =new Thread (obj);
        Thread t2 = new Thread (obj);
        t1.setName ("First Person");
        t2.setName ("Second Person");
        t1.start ();
        t2.start ();
    }
}
```

```
class Reserve implements Runnable {
    int available = 1;
    int wanted;
    Reserve (int i)  {
        wanted = i;
        }
    public void run()  {
        synchronized(this){
            System.out.println ("Number of berths available: " + available);
            if ( available >= wanted){
                String name = Thread.currentThread ().getName ();
                System.out.println (wanted + " berths alloted to: " + name);
                try{
                    Thread.sleep (2000); // wait for priniting the ticket
                    available = available - wanted;
                }catch (InterruptedException ie){
                          ie.printStackTrace ();
                        }
            }else{
                System.out.println ("Sorry, no berths available");
            }
        }
    }
}
```

```
Number of berths available: 1
1 berths alloted to: First Person
Number of berths available: 0
Sorry, no berths available
Press any key to continue . . .
```

10.Write a java program to generate multiplication of two numbers using thread?

Program:

```
class Table{
    void printTable(int n){//method not synchronized
        for(int i=1;i<=5;i++){
        System.out.println(n*i);
        try{
            Thread.sleep(400);
        }catch(Exception e){System.out.println(e);}
        }
    }
}

class MyThread1 extends Thread{
    Table t;
    MyThread1(Table t){
        this.t=t;
    }
    public void run(){
        t.printTable(5);
    }
}
class MyThread2 extends Thread{
    Table t;
    MyThread2(Table t){
        this.t=t;
    }
    public void run(){
        t.printTable(100);
    }
}
class TestSynchronization1{
    public static void main(String args[]){
    Table obj = new Table();//only one object
    MyThread1 t1=new MyThread1(obj);
    MyThread2 t2=new MyThread2(obj);
    t1.start();
    t2.start();
    }
}
```

Output:

```
5
100
200
10
15
300
20
400
25
500
Press any key to continue . . .
```

11.Explain thread class in detail along with an example?

Thread class:

Thread class provide constructors and methods to create and perform operations on a thread.Thread class extends Object class and implements Runnable interface.

Commonly used Constructors of Thread class:

Thread()

Thread(String name)

Thread(Runnable r)

Thread(Runnable r,String name)

Commonly used methods of Thread class:

public void run(): is used to perform action for a thread.

public void start(): starts the execution of the thread.JVM calls the run() method on the thread.

public void sleep(long miliseconds): Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.

public void join(): waits for a thread to die.

public void join(long miliseconds): waits for a thread to die for the specified miliseconds.

public int getPriority(): returns the priority of the thread.

public int setPriority(int priority): changes the priority of the thread

public String getName(): returns the name of the thread.

public void setName(String name): changes the name of the thread.

public Thread currentThread(): returns the reference of currently executing thread.

public int getId(): returns the id of the thread.

public Thread.State getState(): returns the state of the thread.

public boolean isAlive(): tests if the thread is alive.

public void yield(): causes the currently executing thread object to temporarily pause and allow other threads to execute.

public void suspend(): is used to suspend the thread(depricated).

public void resume(): is used to resume the suspended thread(depricated).

public void stop(): is used to stop the thread(depricated).

public boolean isDaemon(): tests if the thread is a daemon thread.

public void setDaemon(boolean b): marks the thread as daemon or user thread.

public void interrupt(): interrupts the thread.

public boolean isInterrupted(): tests if the thread has been interrupted.

public static boolean interrupted(): tests if the current thread has been interrupted.

```
1  class Theatre extends Thread {
2      String str;
3      Theatre (String str)  {
4          this.str = str;
5          }
6      public void run()  {
7          for (int i = 1; i <= 10 ; i++)
8          {
9              System.out.println (str + " : " + i);
10             try{
11                 Thread.sleep (2000);
12             }catch (InterruptedException ie){
13                 ie.printStackTrace ();
14             }
15         }
16     }
17 }
18 class TDemo1 {
19     public static void main(String args[]){
20         Theatre obj1 = new Theatre ("Cut Ticket");
21         Theatre obj2 = new Theatre ("Show Chair");
22         Thread t1 = new Thread (obj1);
23         Thread t2 = new Thread (obj2);
24         t1.start ();
25         t2.start ();
26         }
27     }
```

```
Cut Ticket : 1
Show Chair : 1
Cut Ticket : 2
Show Chair : 2
Cut Ticket : 3
Show Chair : 3
Cut Ticket : 4
Show Chair : 4
Cut Ticket : 5
Show Chair : 5
Cut Ticket : 6
Show Chair : 6
Cut Ticket : 7
Show Chair : 7
Cut Ticket : 8
Show Chair : 8
Cut Ticket : 9
Show Chair : 9
Cut Ticket : 10
Show Chair : 10
```

12.Write a java program to implement nested try catch blocks?

```
class Nest{
   public static void main(String args[]){
         //Parent try block
     try{
        //Child try block1
         try{
            System.out.println("Inside block1");
            int b =45/0;
            System.out.println(b);
         }
         catch(ArithmeticException e1){
            System.out.println("Exception: e1"+e1);
         }
         //Child try block2
         try{
            System.out.println("Inside block2");
            int b =45/0;
            System.out.println(b);
         }
         catch(ArrayIndexOutOfBoundsException e2){
            System.out.println("Exception: e2"+e2);
         }
        System.out.println("Just other statement");
    }
    catch(ArithmeticException e3){
         System.out.println("Arithmetic Exception"+e3);
         System.out.println("Inside parent try catch block");
    }
    catch(ArrayIndexOutOfBoundsException e4){
        System.out.println("ArrayIndexOutOfBoundsException");
         System.out.println("Inside parent try catch block");
    }
    catch(Exception e5){
        System.out.println("Exception");
         System.out.println("Inside parent try catch block");     }
     System.out.println("Next statement..");
  }
}
```

```
C:\Users\sana mateen\Desktop\oops>javac Nest.java

C:\Users\sana mateen\Desktop\oops>java Nest
Inside block1
Exception: e1java.lang.ArithmeticException: / by zero
Inside block2
Arithmetic Exceptionjava.lang.ArithmeticException: / by zero
Inside parent try catch block
Next statement..
```

13.Differentiate between throw and throws clause along with an example?

No.	throw	throws
1)	Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
2)	Checked exception cannot be propagated using throw only.	Checked exception can be propagated with throws.
3)	Throw is followed by an instance.	Throws is followed by class.
4)	Throw is used within the method.	Throws is used with the method signature.
5)	You cannot throw multiple exceptions.	You can declare multiple exceptions e.g. public void method()throws IOException,SQLException.

throw implementation:

```
public class Examplethrow {
   static void checkEligibilty(int stuage, int stuweight){
      if(stuage<12 && stuweight<40) {
         throw new ArithmeticException("Student is not eligible for registration");
      }
      else {
         System.out.println("Entries Valid!!");
      }
   }

   public static void main(String args[]){
     System.out.println("Welcome to the Registration process!!");
    Examplethrow.checkEligibilty(10, 39);
     System.out.println("Have a nice day..");
  }
}
```

```
C:\Users\sana mateen\Desktop\oops>start notepad Examplethrow.java

C:\Users\sana mateen\Desktop\oops>javac Examplethrow.java

C:\Users\sana mateen\Desktop\oops>java Examplethrow
Welcome to the Registration process!!
Exception in thread "main" java.lang.ArithmeticException: Student is not eligibl
e for registration
        at Examplethrow.checkEligibilty(Examplethrow.java:4)
        at Examplethrow.main(Examplethrow.java:13)
```

throws implementation:

```
import java.util.Scanner;
public class Examplethrows{
static void divide(int a,int b)throws ArithmeticException,IndexOutOfBoundsException{
int c=a/b;
System.out.println("result is"+c);
int[] arn=new int[2];
for(int i=0;i<=arn.length;i++){
arn[i]=i;
}
for(int i:arn){
System.out.println("elements of array are"+arn[i]);
}
}
public static void main(String[] args){
Scanner sc=new Scanner(System.in);
int x=sc.nextInt();
int y=sc.nextInt();
divide(x,y);
}
}
```

```
C:\Users\sana mateen\Desktop\oops>java Examplethrows
4
2
result is2
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 2
        at Examplethrows.divide(Examplethrows.java:8)
        at Examplethrows.main(Examplethrows.java:18)
```

UNIT-4

Short Answer Questions for 2 Marks

1.What is Java Priority queue?

A priority queue in Java is a special type of queue wherein all the elements are ordered as per their natural ordering or based on a custom Comparator supplied at the time of creation. ... The Priority Queue class is part of Java's collections framework and implements the Queue interface.

2.What is Collection Class?Give an example?

Collections class in java is a useful utility class to work with collections in java. The java.util.Collections class directly extends the Object class and exclusively consists of the static methods that operate on Collections or return them.

3.Explain the use of String tokenizer with an example?

StringTokenizer class is used for creating tokens in Java. It allows an application to break or split into small parts. Each split string part is called Token.

A StringTokennizer in Java, object keeps the string in the present position as it is to be tokenized. By taking a substring of the string a token can return that utilize to make the StringTokenizer protest.

4.Explain any three methods defined by Iterator?

No.	Method	Description
1	public boolean hasNext()	It returns true if iterator has more elements.
2	public Object next()	It returns the element and moves the cursor pointer to the next element.
3	public void remove()	It removes the last elements returned by the iterator. It is rarely used.

5.What is the benefit of Generics in Collections Framework?

Generics allow us to provide the type of Object that a collection can contain, so if you try to add any element of other type it throws compile time error.

This avoids ClassCastException at Runtime because you will get the error at compilation. Also Generics make code clean since we don't need to use casting and instanceof operator. It

also adds up to runtime benefit because the bytecode instructions that do type checking are not generated.

6..Differentiate between Enumeration and iterator interface?

Enumeration	iterator
Enumeration is not a universal cursor as it applies only to legacy classes.	Iterator is a universal cursor as it is applicable for all the collection classes.
Using Enumeration you can only read the elements in the collection.	Using Iterator you can read and remove the elements in the collection.

public boolean hasMoreElements();
public object nextElement();
	public boolean hasnext();
	public objects next();
	public void remove();

7.Differentiate between Arrays and Vector?

Arrays	Vector
The length of an array is fixed once it is created and elements cannot be added or removed prior to its creation	A vector is a resizable array which works by reallocating the storage and copying the elements of the old array to the new array
Arrays are not synchronized	A vector is synchronized
An array has length property which gives its size.	To find the size of vector call its size()

8.Differentiate between ArrayList and LinkedList?

ArrayList	LinkedList
ArrayList internally uses a **dynamic array** to store the elements	LinkedList internally uses a **doubly linked list** to store the elements.
Manipulation with ArrayList is slow because it internally uses an array. If any element is removed from the array, all the bits are shifted in memory.	Manipulation with LinkedList is faster than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory.

9.Write about HashSet and TreeSet class?

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.

Java TreeSet class implements the Set interface that uses a tree for storage. It inherits AbstractSet class and implements NavigableSet interface. The objects of TreeSet class are stored in ascending order.

Short Answer Questions for 3 Marks:

10.Write a Java Program for traversing through ArrayList?

```
package collectionsexample;
import java.util.ArrayList;
import java.util.Iterator;
public class ArrayListImpl{
 public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();//Creating arraylist
  ArrayList<Integer> rl=new ArrayList<>();
  list.add("Pranathi");//Adding object in arraylist
  list.add("Shanoor");
  list.add("Shivani");
  list.add("Pravalika");
  list.add(4,"Sowmya");
  //Traversing list through Iterator
   System.out.println("size of collection 1"+list.size());
  Iterator itr=list.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
    }
  rl.add(61);
  rl.add(86);
  rl.add(89);
  rl.add(63);
  rl.add(4,65);
  System.out.println("size of collection 2"+rl.size());
   for(int obj:rl)  {
       System.out.println("Roll numbers are:");
    System.out.println(obj);  }
   rl.remove(4);
```

```
    Iterator itr1=rl.iterator();
   while(itr1.hasNext()){
   System.out.println(itr1.next());
  }
   System.out.println("size of collection 2"+rl.size());
   rl.retainAll(list);
   System.out.println("size of collection 2"+rl.size());

 }
}
```

```
run:
size of collection one5
Pranathi
Shancor
Shivani
Pravalika
Soumya
size of collection two5
Roll numbers are:
61
Roll numbers are:
66
Roll numbers are:
69
Roll numbers are:
63
Roll numbers are:
65
61
66
69
63
size of collection two after removing4
size of collection two0
BUILD SUCCESSFUL (total time: 0 seconds)
```

11.Define Map?Mention the different type of Map classes?

The Map interface of the Java collections framework provides the functionality of the map data structure.

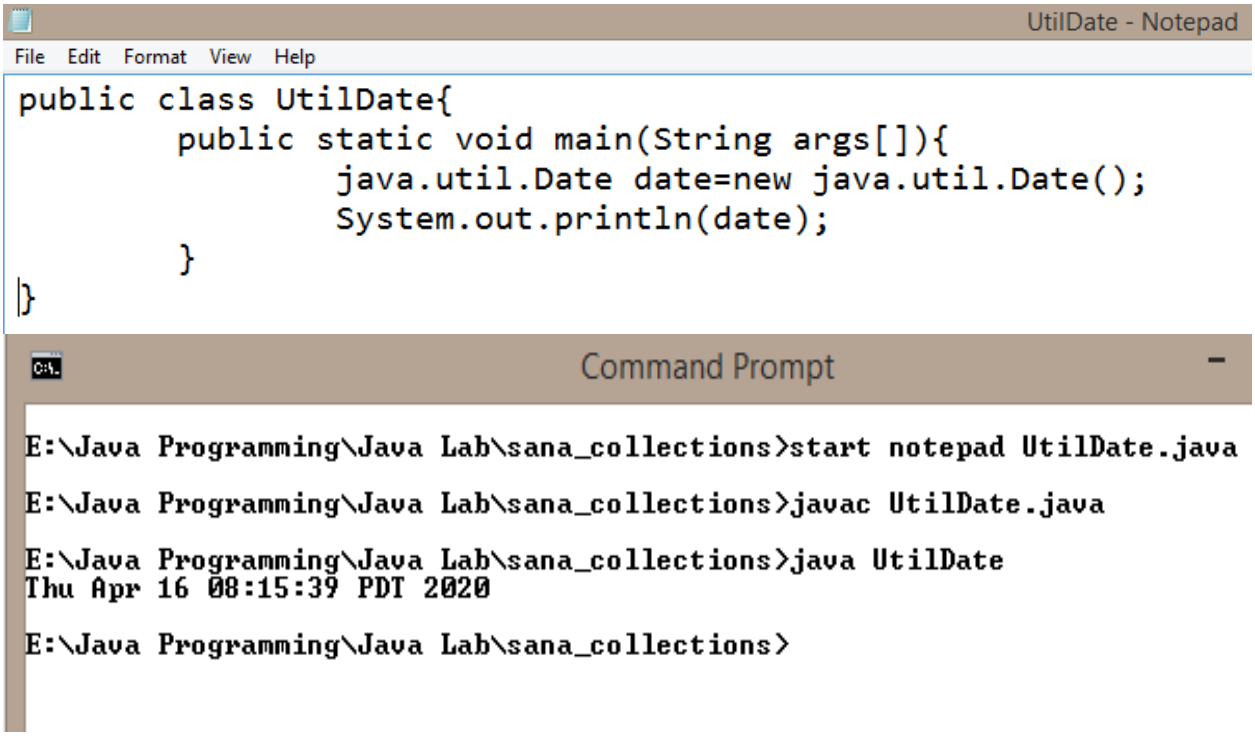
Classes that implement Map

Since Map is an interface, we cannot create objects from it.

In order to use functionalities of the Map interface, we can use these classes:

- HashMap
- EnumMap
- LinkedHashMap
- WeakHashMap
- TreeMap

12.Write a Java Program to implement Date Class?



13.What is ArrayDeque?Specify its features?

The ArrayDeque class provides **the facility of using deque and resizable-array**. It **inherits AbstractCollection class** and implements the Deque interface.

Unlike Queue**, we can add or remove elements from both sides.**

Null elements are not allowed in the ArrayDeque.

ArrayDeque is not thread safe, in the absence of external synchronization.

ArrayDeque has no capacity restrictions.

ArrayDeque is faster than LinkedList and Stack.

14.List the HashTable constructors?

Hashtable class has four constructors.

Hashtable(): It is the default constructor. It constructs a new, empty hashtable with a default initial capacity (11) and load factor (0.75).

Hashtable(int size): It constructs a new, empty hashtable of specified initial size.

Hashtable(int size, float fillRatio): It constructs a new, empty hashtable of specified initial size and fill ratio.

Hashtable(Map m): It constructs a hashtable that is initialized with the key-value pairs in specified map.

15.Differentiate between HashMap andTreeMap?

HashMap	TreeMap
Java HashMap class implements the map interface by using a hash table. It inherits AbstractMap class and implements Map interface.	**TreeMap is Red-Black tree based NavigableMap implementation**. It is sorted according to the natural ordering of its keys. **TreeMap class implements Map interface similar to HashMap class.**
Java HashMap class contains values based on the key. **Java HashMap class contains only unique keys.**	The main difference between them is that **HashMap** is an unordered collection while **TreeMap is sorted in the ascending order of its keys.**
Java HashMap class may have one null key and multiple null values. **Java HashMap class is non synchronized**. Java HashMap class maintains no order	**TreeMap is unsynchronized collection class which means it is not suitable for thread-safe operations until unless synchronized explicitly.**

16.What are benefits of Collection framework?

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

Interfaces and its implementations i.e. classes

Algorithm

The Java Collections Framework provides the following benefits:

Reduces programming effort: By providing useful data structures and algorithms, the Collections Framework frees you to concentrate on the important parts of your program rather than on the low-level "plumbing" required to make it work. By facilitating interoperability among unrelated APIs, the Java Collections Framework frees you from writing adapter objects or conversion code to connect APIs.

Increases program speed and quality: This Collections Framework provides high-performance, high-quality implementations of useful data structures and algorithms. The various implementations of each interface are interchangeable, so programs can be easily tuned by switching collection implementations. Because you're freed from the drudgery of writing your own data structures, you'll have more time to devote to improving programs' quality and performance.

Allows interoperability among unrelated APIs: The collection interfaces are the vernacular by which APIs pass collections back and forth. If my network administration API furnishes a collection of node names and if your GUI toolkit expects a collection of column headings, our APIs will interoperate seamlessly, even though they were written independently.

Reduces effort to learn and to use new APIs: Many APIs naturally take collections on input and furnish them as output. In the past, each such API had a small sub-API devoted to manipulating its collections. There was little consistency among these ad hoc collections sub-APIs, so you had to learn each one from scratch, and it was easy to make mistakes when using them. With the advent of standard collection interfaces, the problem went away.

Reduces effort to design new APIs: This is the flip side of the previous advantage. Designers and implementers don't have to reinvent the wheel each time they create an API that relies on collections; instead, they can use standard collection interfaces.

Fosters software reuse: New data structures that conform to the standard collection interfaces are by nature reusable. The same goes for new algorithms that operate on objects that implement these interfaces.

17.What is collection?

A collection — sometimes called a container — is simply an object that groups multiple elements into a single unit. Collections are used to store, retrieve, manipulate, and communicate aggregate data. Typically, they represent data items that form a natural group, such as a poker hand (a collection of cards), a mail folder (a collection of letters), or a telephone directory (a mapping of names to phone numbers). If you have used the Java programming language — or just about any other programming language — you are already familiar with collections.

What Is a Collections Framework?

A collections framework is a unified architecture for representing and manipulating collections. All collections frameworks contain the following:

Interfaces: These are abstract data types that represent collections. Interfaces allow collections to be manipulated independently of the details of their representation. In object-oriented languages, interfaces generally form a hierarchy.

Implementations: These are the concrete implementations of the collection interfaces. In essence, they are reusable data structures.

Algorithms: These are the methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces. The algorithms are said to be polymorphic: that is, the same method can be used on many different implementations of the appropriate collection interface. In essence, algorithms are reusable functionality.

18.What is the difference between Collection and Collections?

Collection	Collections
It is an interface in Java collection framework	It is a utility class in Collection framework
It doesn't has all static methods	It has all static method
It is used to store list of object in a single object	It is used to operate on collection.

Long Answer Questions for 5 Marks

1.Write a java program to implement Stack class?

Program:

```
package collectionsexample;
import java.util.*;
public class StackDemo {
    static void showpush(Stack st, int a) {
        st.push(new Integer(a));
        System.out.println("push(" + a + ")");
        System.out.println("stack: " + st);
    }
    static void showpop(Stack st) {
        System.out.print("pop -> ");
        Integer a = (Integer) st.pop();
        System.out.println(a);
        System.out.println("stack: " + st);
    }
    public static void main(String args[]) {
        Stack st = new Stack();
        System.out.println("stack: " + st);
        showpush(st, 42);
        showpush(st, 66);
        showpush(st, 99);
        showpop(st);
        showpop(st);
        showpop(st);
        try {
            showpop(st);
        } catch (EmptyStackException e) {
            System.out.println("empty stack");
        }
    }
}
```

Output:

```
stack: []
push(42)
stack: [42]
push(66)
stack: [42, 66]
push(99)
stack: [42, 66, 99]
pop -> 99
stack: [42, 66]
pop -> 66
stack: [42]
pop -> 42
stack: []
pop -> empty stack
```

2.Explain Random collection class?

Java Random class is used to generate a stream of pseudorandom numbers.

The algorithms implemented by Random class use a protected utility method than can supply up to 32 pseudorandomly generated bits on each invocation.

Methods	Description
doubles()	Returns an unlimited stream of pseudorandom double values.
ints()	Returns an unlimited stream of pseudorandom int values.
longs()	Returns an unlimited stream of pseudorandom long values.
next()	Generates the next pseudorandom number.
nextBoolean()	Returns the next uniformly distributed pseudorandom boolean value from the random number generator's sequence
nextByte()	Generates random bytes and puts them into a specified byte array.
nextDouble()	Returns the next pseudorandom Double value between 0.0 and 1.0 from the random number generator's sequence
nextFloat()	Returns the next uniformly distributed pseudorandom Float value between 0.0 and 1.0 from this random number generator's sequence

Program:

RandomGen - Notepad

File Edit Format View Help

```
import java.util.Random;
public class RandomGen{
        public static void main(String[] a){
                int n=10;
                Random num=new Random();
                for(int i=0;i<n;i++){
                        int value=num.nextInt(10);
                        System.out.println("Random Number"+(i+1)+" "+value);
                }
        }
}
```

Output:

Command Prompt

```
E:\Java Programming\Java Lab\sana_collections>java RandomGen
Random Number1 9
Random Number2 2
Random Number3 4
Random Number4 5
Random Number5 8
Random Number6 6
Random Number7 7
Random Number8 9
Random Number9 7
Random Number10 9

E:\Java Programming\Java Lab\sana_collections>java RandomGen
Random Number1 1
Random Number2 8
Random Number3 2
Random Number4 3
Random Number5 5
Random Number6 6
Random Number7 2
Random Number8 1
Random Number9 6
Random Number10 5
```

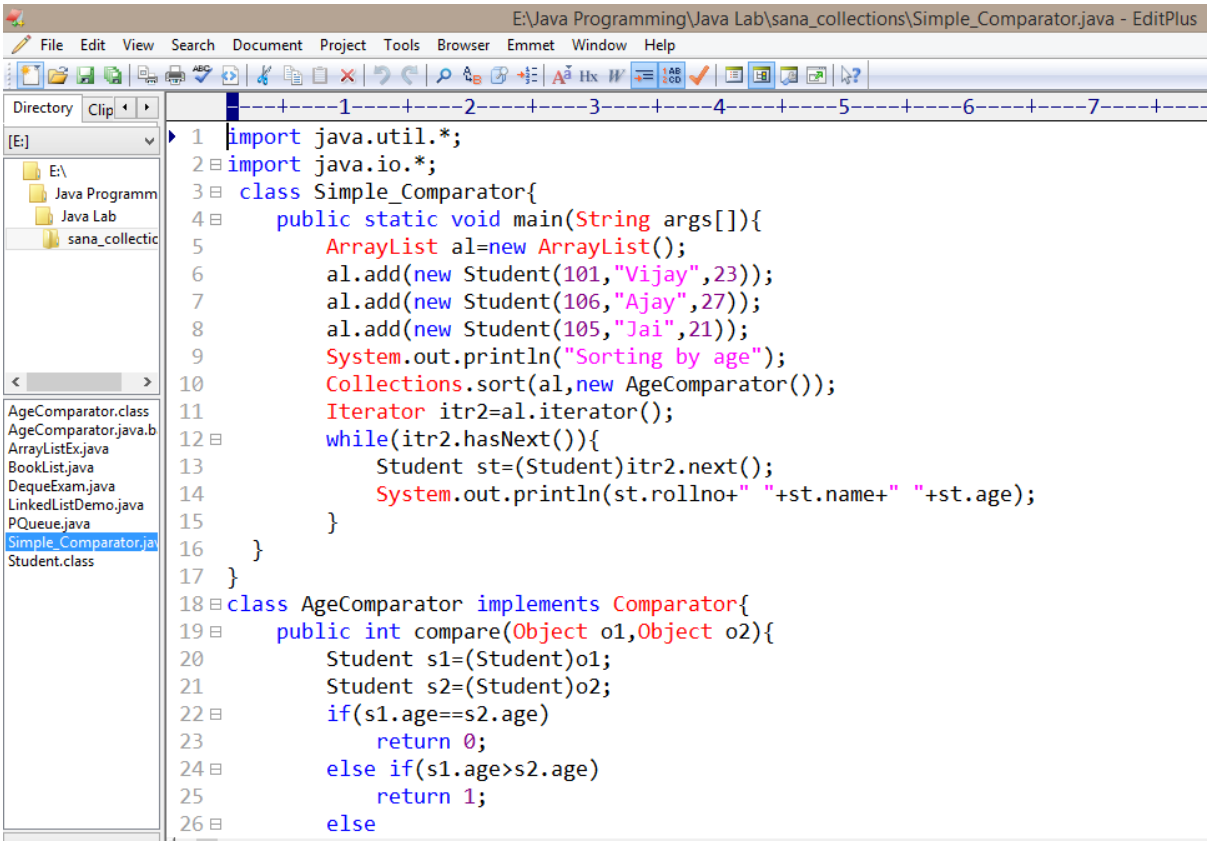
3.Differentiate between ArrayList and Vector?Explain

ArrayList	Vector
ArrayList is not synchronized .	In Vector class all the methods are synchronized .Thats why the Vector object is already synchronized when it is created .
In comparison ArrayList is fast as it is non synchronized . Thus in ArrayList two or more threads can access the code at the same time .	Vector is slow as it is thread safe .while Vector is limited to one thread at a time.
By default ArrayList size is 10 . It checks whether it reaches the last element then it will create the new array ,copy the new data of last array to new array ,then old array is garbage collected by the Java Virtual Machine (JVM) .	A Vector defaults to doubling size of its array . While when you insert an element into the ArrayList , it increases its Array size by 50% .
ArrayList does not define the increment size	. Vector defines the increment size
While ArrayList can only use Iterator for traversing an ArrayList .	Other than Hashtable ,Vector is the only other class which uses both Enumeration and Iterator .

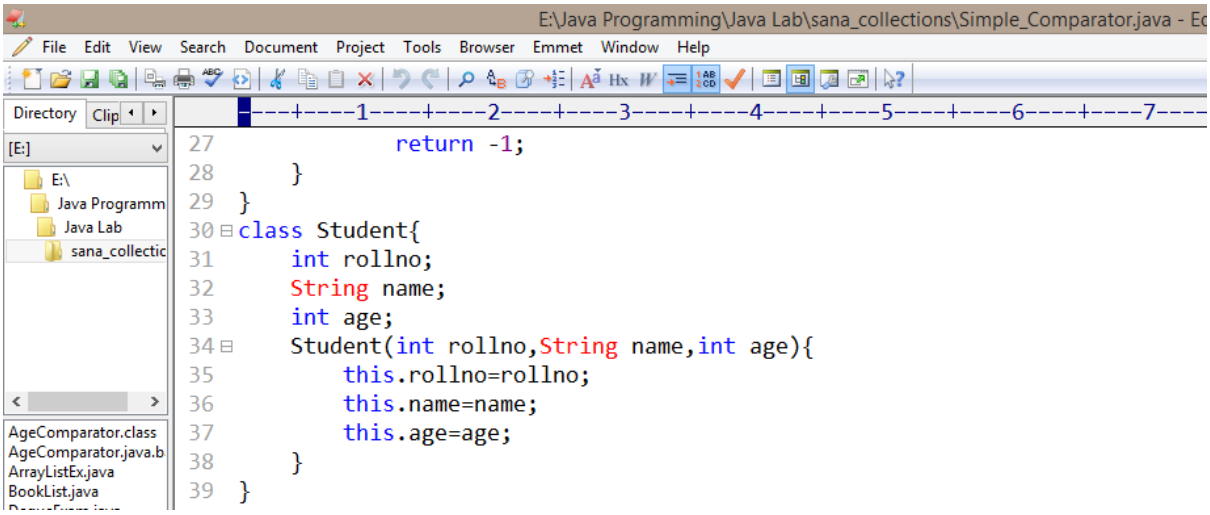
4.What is Comparable and Comparator interface?Differentiate between them?

Comparable	Comparator
Comparable provides a **single sorting sequence**. In other words, we can sort the collection on the basis of a single element such as id, name, and price.	The Comparator provides **multiple sorting sequences**. In other words, we can sort the collection on the basis of multiple elements such as id, name, and price etc.
Comparable affects the original class, i.e., the actual class is modified.	Comparator **doesn't affect the original class**, i.e., the actual class is not modified.
Comparable provides **compareTo() method** to sort elements.	Comparator provides **compare() method** to sort elements.
Comparable is present in **java.lang** package.	A Comparator is present in the **java.util** package.
We can sort the list elements of Comparable type by **Collections.sort(List)** method.	We can sort the list elements of Comparator type by **Collections.sort(List, Comparator)** method.

Program:

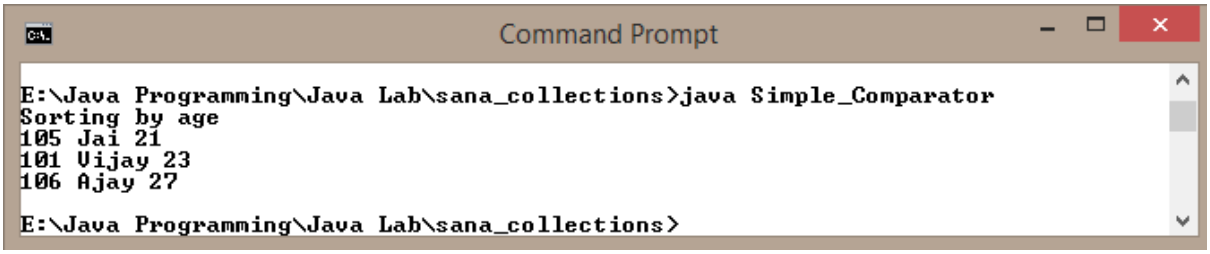


```
import java.util.*;
import java.io.*;
 class Simple_Comparator{
    public static void main(String args[]){
        ArrayList al=new ArrayList();
        al.add(new Student(101,"Vijay",23));
        al.add(new Student(106,"Ajay",27));
        al.add(new Student(105,"Jai",21));
        System.out.println("Sorting by age");
        Collections.sort(al,new AgeComparator());
        Iterator itr2=al.iterator();
        while(itr2.hasNext()){
            Student st=(Student)itr2.next();
            System.out.println(st.rollno+" "+st.name+" "+st.age);
        }
   }
 }
class AgeComparator implements Comparator{
    public int compare(Object o1,Object o2){
        Student s1=(Student)o1;
        Student s2=(Student)o2;
        if(s1.age==s2.age)
            return 0;
        else if(s1.age>s2.age)
            return 1;
        else
```



```
            return -1;
    }
 }
class Student{
    int rollno;
    String name;
    int age;
    Student(int rollno,String name,int age){
        this.rollno=rollno;
        this.name=name;
        this.age=age;
    }
 }
```

Output:



```
Command Prompt

E:\Java Programming\Java Lab\sana_collections>java Simple_Comparator
Sorting by age
105 Jai 21
101 Vijay 23
106 Ajay 27

E:\Java Programming\Java Lab\sana_collections>
```

5.Explain about StringTokenizer in detail?

- The **java.util.StringTokenizer** class allows you to break a string into tokens. It is simple way to break string.
- It doesn't provide the facility to differentiate numbers, quoted strings, identifiers etc. like StreamTokenizer class.

Constructor	Description
StringTokenizer(String str)	creates StringTokenizer with specified string.
StringTokenizer(String str, String delim)	creates StringTokenizer with specified string and delimeter.
StringTokenizer(String str, String delim, boolean returnValue)	creates StringTokenizer with specified string, delimeter and returnValue. If return value is true, delimiter characters are considered to be tokens. If it is false, delimiter characters serve to separate tokens.

Public method	Description
boolean hasMoreTokens()	checks if there is more tokens available.
String nextToken()	returns the next token from the StringTokenizer object.
String nextToken(String delim)	returns the next token based on the delimeter.
boolean hasMoreElements()	same as hasMoreTokens() method.
Object nextElement()	same as nextToken() but its return type is Object.
int countTokens()	returns the total number of tokens.

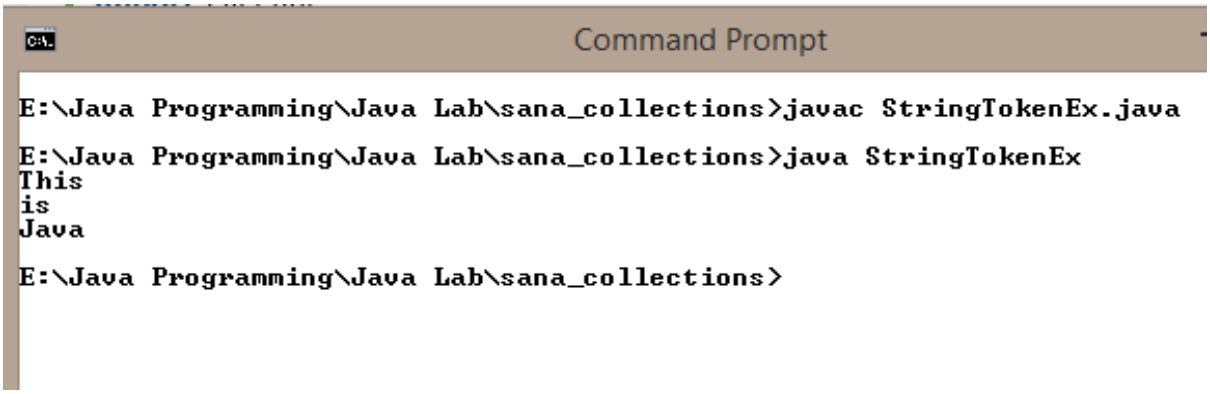
Program:

StringTokenEx - Notepad

File Edit Format View Help

```
import java.util.StringTokenizer;
public class StringTokenEx{
 public static void main(String args[]){
   StringTokenizer st = new StringTokenizer("This is Java"," ");
     while (st.hasMoreTokens()) {
         System.out.println(st.nextToken());
     }
   }
}
```

Output:



Long Answer Question for 10 Marks:

6.What is the difference between Iterator and ListIterator?Explain different ways to iterate over a list?

Iterator	ListIterator
Iterator is used for traversing List and Set both.	We can use ListIterator to traverse List only, we cannot traverse Set using ListIterator.
We can traverse in only forward direction using Iterator.	Using ListIterator, we can traverse a List in both the directions (forward and Backward).
We cannot obtain indexes while using Iterator	We can obtain indexes at any point of time while traversing a list using ListIterator. The methods nextIndex() and previousIndex() are used for this purpose.
We cannot add element to collection while traversing it using Iterator, it throws ConcurrentModificationException when you try to do it.	We can add element at any point of time while traversing a list using ListIterator.
We cannot replace the existing element value when using Iterator.	By using set(E e) method of ListIterator we can replace the last element returned by next() or previous() methods.
Methods of Iterator: hasNext() next() remove()	Methods of ListIterator: add(E e) hasNext() hasPrevious() next()

	nextIndex()

There are 7 ways you can iterate through List.

Simple For loop

Enhanced For loop

Iterator

ListIterator

While loop

Iterable.forEach() util

Stream.forEach() util

Program:

```
package collectionsexample;
import java.util.ArrayList;
import java.util.Iterator;
public class ArrayListImpl{
 public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();//Creating arraylist
  ArrayList<Integer> rl=new ArrayList<>();
  list.add("Pranathi");//Adding object in arraylist
  list.add("Shanoor");
  list.add("Shivani");
  list.add("Pravalika");
  list.add(4,"Sowmya");
  //Traversing list through Iterator
   System.out.println("size of collection 1"+list.size());
  Iterator itr=list.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
    }
  rl.add(61);
  rl.add(86);
  rl.add(89);
  rl.add(63);
  rl.add(4,65);
  System.out.println("size of collection 2"+rl.size());
   for(int obj:rl)  {
       System.out.println("Roll numbers are:");
    System.out.println(obj);  }
   rl.remove(4);
```

```
    Iterator itr1=rl.iterator();
   while(itr1.hasNext()){
   System.out.println(itr1.next());
  }
   System.out.println("size of collection 2"+rl.size());
   rl.retainAll(list);
   System.out.println("size of collection 2"+rl.size());

 }
}
```

Output:

```
run:
size of collection one5
Pranathi
Shanoor
Shivani
Pravalika
Sowmya
size of collection two5
Roll numbers are:
61
Roll numbers are:
86
Roll numbers are:
89
Roll numbers are:
63
Roll numbers are:
65
61
86
89
63
size of collection two after removing4
size of collection two0
BUILD SUCCESSFUL (total time: 0 seconds)
```

7.What are the best practices related to Java Collection Framework?Discuss

Use non synchronized Collection: Use ArrayList, HashMap as opposed to Vector, Hashtable etc. Thus we can avoid the synchronization overhead. Even more better is to use an array where possible. If multiple threads are concurrently access a collection and at least one of the threads either adds or deletes an entry into the collection, then the collection must be externally synchronized. This can be achieved by:

Map aMap = Collections.synchronizedMap (aMap);

List aList = Collections.synchronizedList (aList);

Set the initial capacity of a collection appropriately: Set the initial capacity of a collection appropriately (e.g. ArrayList, HashMap etc) whenever possible. As because collection classes like ArrayList, HashMap etc must grow periodically to accommodate new elements. But if you have a very large number of elements to process, and you know the size in advance then you can speed things up by setting the initial size appropriately of the Collection.

For example: HashMaps/Hashtables needs to be created with sufficiently large capacity to minimize rehashing (which happens every time the table grows). HashMap has two parameters initial capacity and load factor that affect its performance and space requirements. Higher load factor values (default load factor of 0.75 provides a good tradeoff between performance and space) will reduce the space cost but will increase the lookup cost of aMap.get(…) and aMap.put(…) methods. When the number of entries in the HashMap exceeds the (current capacity * loadfactor) then the capacity of the HashMap is roughly doubled by calling the rehash function. It is also very important not to set the initial capacity too high or load factor too low if iteration performance or reduction in space is important.

Program in terms of interface not implementation: For example, you might decide a LinkedList is the best choice for some application, but then later decide ArrayList might be a better choice for performance reason.

// Use:

List list = new ArrayList(100);

//program in terms of interface & set the initial capacity. Instead of:

ArrayList list = new ArrayList();

Use generic type and diamond operator: Earlier we were used to declare a Collection of generic type in the following way:

List<Student> studentList = new ArrayList<Student>();

Since Java 7, the compiler can infer the generic type on the right side from the generic type declared on the left side, so you can write:

List<Student> studentList = new ArrayList<>();

The <> is informally called the diamond operator. This operator is quite useful. Specially when we use Collection to define some Complex data structure.Consider we are storing result of every student by department. In that case we will use following data structure:

Map<Department,Map<Student,Result>> studentResultByDepartment=new HashMap<Department,Map<Student,Result>>();

But using the diamond operator the code is more clean to read.

Map<Department,Map<Student,Result>> studentResultByDepartment = new HashMap<>();

Prefer isEmpty() over size(): Avoid checking the emptiness of a collection like this:

```
if (myList.size() > 0) {
   // do something if the list is not empty
}
```

Instead, you should use the isEmpty() method:

```
if (!myList.isEmpty()) {
   // do something if the list is not empty
}
```

There's no performance difference between isEmpty() and size(). The reason is for the readability of the code. More over you can use Apache's Collection util, which has even more readable methods : Use org.apache.commons.collections4.CollectionUtils

```
// Null-safe check if the specified collection is empty.
// If Null returns true.
public static boolean isEmpty(Collection coll)

// Null-safe check if the specified collection is not empty.
```

// If Null returns false.

public static boolean isNotEmpty(Collection coll)

Avoid to use the classic for loop: Using classic for loop with Collection will not have any performance impact. But it will decrease the readability of the code and can introduce potential bug in your code. Consider the following example:

```
for (int i = 0; i < itemList.size(); i++) {
    Item item = itemList.get(i);
    // do something with item
}
```

There is nothing wrong with the code. However, this is considered as bad practice because using the counter variable i may lead to potential bugs if it is altered somewhere inside the loop. Also this kind of loop is not object-oriented, since every collection has its own iterator. So it's recommended to use an iterator like the following code:

```
Iterator<Item> iterator = aList.iterator();

while (iterator.hasNext()) {
    Item item = iterator.next();
     // do something with item
}

// More over You also can use ListIterator which support both forward and backward iteration
listItr = aList.listIterator();
System.out.println("Elements in forward directiton");
while(listItr.hasNext()){
```

```
        System.out.println(listItr.next());
}
System.out.println("Elements in backward directiton");
while(listItr.hasPrevious()){
        System.out.println(listItr.previous());
}
```

Also the iterator may throw ConcurrentModificationException if the collection is modified by another thread after the iterator is created, which eliminates potential bugs.

Now, it’s better to use the enhanced for loop like this:

```
for (Item item: itemList) {
   // do something with an item
}
```

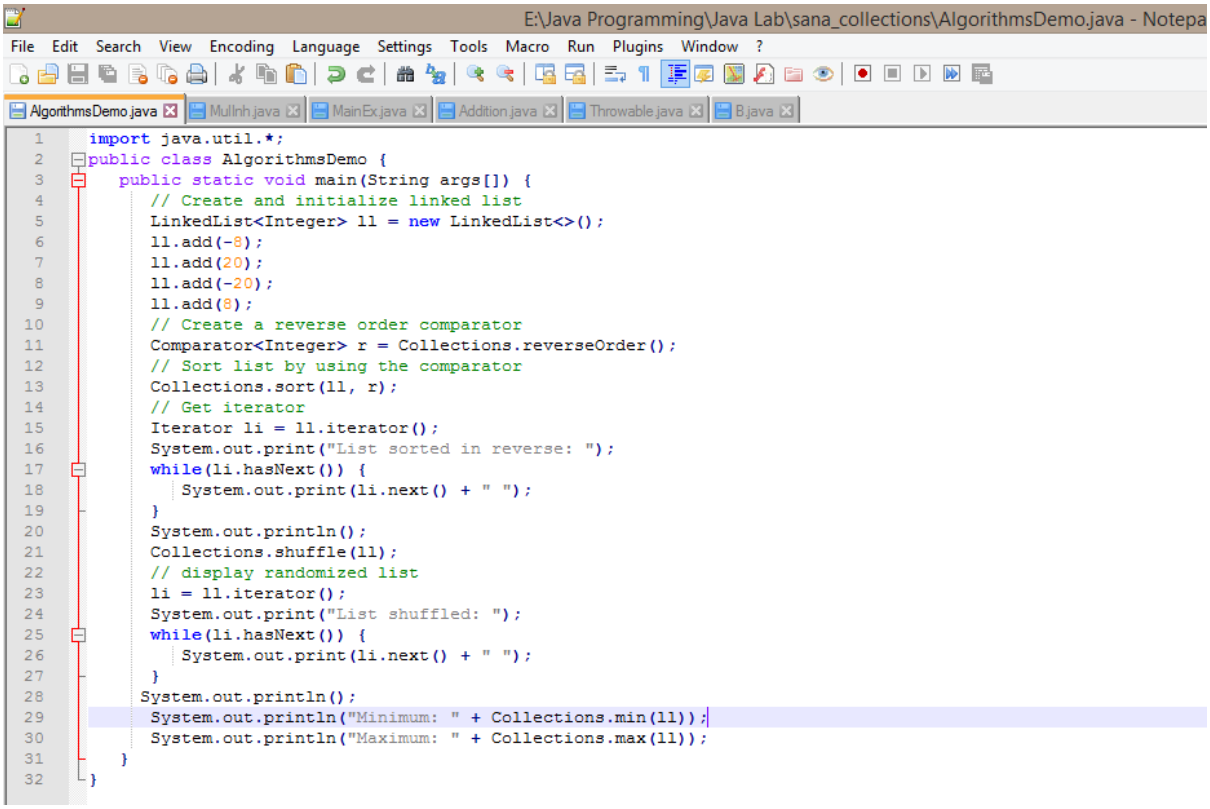
As you can see, the enhanced for loop is more succinct and readable though it uses an iterator behind the scenes.

8.Explain Collection algorithms in detail along with an example program?

The collections framework defines several algorithms that can be applied to collections and maps.

These algorithms are defined as static methods within the Collections class. Several of the methods can throw a ClassCastException, which occurs when an attempt is made to compare incompatible types, or an UnsupportedOperationException, which occurs when an attempt is made to modify an unmodifiable collection.

Program:



Output:

```
E:\Java Programming\Java Lab\sana_collections>javac AlgorithmsDemo.java

E:\Java Programming\Java Lab\sana_collections>java AlgorithmsDemo
List sorted in reverse: 20 8 -8 -20
List shuffled: -20 -8 20 8
Minimum: -20
Maximum: 20
```

9.Explain Vector class in detail?

Vector implements a dynamic array. It is similar to ArrayList, but with two differences −

Vector is synchronized.

Vector contains many legacy methods that are not part of the collections framework.

Vector proves to be very useful **if you don't know the size of the array in advance** or you just need one that can change sizes over the lifetime of a program.

Sr.No.	Constructor & Description

1	**Vector( )** **This constructor creates a default vector, which has an initial size of 10.**
2	**Vector(int size)** **This constructor accepts an argument that equals to the required size, and creates a vector whose initial capacity is specified by size.**
3	**Vector(int size, int incr)** **This constructor creates a vector whose initial capacity is specified by size and whose increment is specified by incr. The increment specifies the number of elements to allocate each time that a vector is resized upward.**
4	**Vector(Collection c)** **This constructor creates a vector that contains the elements of collection c.**

Program:

```
package collectionsexample;

import java.util.*;

public class VectorClass {

   public static void main(String args[]) {
      /* Vector of initial capacity(size) of 2 */
      Vector<String> vec = new Vector<String>(2);

      /* Adding elements to a vector*/
      vec.addElement("Apple");
      vec.addElement("Orange");
      vec.addElement("Mango");
      vec.addElement("Fig");

      /* check size and capacityIncrement*/
      System.out.println("Size is: "+vec.size());
      System.out.println("Default capacity increment is: "+vec.capacity());
```

```
        vec.addElement("fruit1");
        vec.addElement("fruit2");
        vec.addElement("fruit3");

        /*size and capacityIncrement after two insertions*/
        System.out.println("Size after addition: "+vec.size());
        System.out.println("Capacity after increment is: "+vec.capacity());

        /*Display Vector elements*/
        Enumeration en = vec.elements();
        System.out.println("\nElements are:");
        while(en.hasMoreElements())
            System.out.print(en.nextElement() + " ");
    }
}
```

Output:

```
Size is: 4
Default capacity increment is: 4
Size after addition: 7
Capacity after increment is: 8

Elements are:
Apple Orange Mango Fig fruit1 fruit2 fruit3
```

10.Explain Dictionary Class along with an example program?

Dictionary is an abstract class that represents a key/value storage repository and operates much like Map.

Given a key and value, you can store the value in a Dictionary object.

Once the value is stored, you can retrieve it by using its key. Thus, like a map, a dictionary can be thought of as a list of key/value pairs.

Sr.No.	Method & Description
1	**Enumeration elements( )** Returns an enumeration of the values contained in the dictionary.
2	**Object get(Object key)** Returns the object that contains the value associated with the key. If the key is not in the dictionary, a null object is returned.
3	**boolean isEmpty( )** Returns true if the dictionary is empty, and returns false if it contains at least one key.
4	**Enumeration keys( )** Returns an enumeration of the keys contained in the dictionary.
5	**Object put(Object key, Object value)** Inserts a key and its value into the dictionary. Returns null if the key is not already in the dictionary; returns the previous value associated with the key if the key is already in the dictionary.

Program:

DictionaryEx - Notepad

File Edit Format View Help

```
import java.util.*;
class DictionaryEx{
        public static void main(String[] a){
                Dictionary<Integer,String> dict=new Hashtable<>();
                dict.put(1,"windows");
                dict.put(2,"linux");
                dict.put(3,"mac os");
                System.out.println("items in dictionary are"+dict);
                System.out.println("retrieve elements from dictionary wih key 3"+dict.get(3));
                System.out.println("remove elements from dictionary wih key 2"+dict.remove(2));
                System.out.println("items in dictionary after removing are"+dict);
        }
}
```

Output:

```
E:\Java Programming\Java Lab\sana_collections>javac DictionaryEx.java

E:\Java Programming\Java Lab\sana_collections>java DictionaryEx
items in dictionary are{3=mac os, 2=linux, 1=windows}
retrieve elements from dictionary wih key 3mac os
remove elements from dictionary wih key 2linux
items in dictionary after removing are{3=mac os, 1=windows}
```

11.Explain TreeMap in detail along with an example?

TreeMap is Red-Black tree based NavigableMap implementation. It is sorted according to the natural ordering of its keys.

TreeMap class implements Map interface similar to HashMap class.

The main difference between them is that **HashMap** is an unordered collection while **TreeMap is sorted in the ascending order of its keys.**

TreeMap is unsynchronized collection class which means it is not suitable for thread-safe operations until unless synchronized explicitly.

Program:

```java
import java.util.TreeMap;
import java.util.Set;
import java.util.Iterator;
import java.util.Map;

public class TreeMapEx {

   public static void main(String args[]) {

      /* This is how to declare TreeMap */
      TreeMap<Integer, String> tmap =
            new TreeMap<Integer, String>();

      /*Adding elements to TreeMap*/
      tmap.put(1, "JP");
      tmap.put(23, "DBMS");
      tmap.put(70, "OS");
      tmap.put(4, "DM");
      tmap.put(2, "BEFA");

      /* Display content using Iterator*/
      Set set = tmap.entrySet();
      Iterator iterator = set.iterator();
      while(iterator.hasNext()) {
         Map.Entry mentry = (Map.Entry)iterator.next();
         System.out.print("key is: "+ mentry.getKey() + " & Value is: ");
         System.out.println(mentry.getValue());
      }

   }
}
```

Output:

```
E:\Java Programming\Java Lab\sana_collections>start notepad TreeMapEx.java

E:\Java Programming\Java Lab\sana_collections>javac TreeMapEx.java

E:\Java Programming\Java Lab\sana_collections>java TreeMapEx
key is: 1 & Value is: JP
key is: 2 & Value is: BEFA
key is: 4 & Value is: DM
key is: 23 & Value is: DBMS
key is: 70 & Value is: OS

E:\Java Programming\Java Lab\sana_collections>
```

12.Explain TreeSet class in detail along with an example?

Java TreeSet class implements the Set interface that uses a tree for storage. It inherits AbstractSet class and implements NavigableSet interface. The objects of TreeSet class are stored in ascending order.

The important points about Java TreeSet class are:

Contains unique elements only like HashSet.

Access and retrieval times are quiet fast.

Maintains ascending order.

Hierarchy of TreeSet class

Java TreeSet class implements NavigableSet interface. The NavigableSet interface extends SortedSet, Set, Collection and Iterable interfaces in hierarchical order.

TreeSet class declaration

Let's see the declaration for java.util.TreeSet class.

public class TreeSet<E> extends AbstractSet<E> implements NavigableSet<E>, Cloneable, Serializable

Constructor	Description
TreeSet()	**It is used to construct an empty tree set that will be sorted in an ascending order according to the natural order of the tree set.**

TreeSet(Collection c)	**It is used to build a new tree set that contains the elements of the collection c.**
TreeSet(Comparator comp)	**It is used to construct an empty tree set that will be sorted according to given comparator.**
TreeSet(SortedSet ss)	**It is used to build a TreeSet that contains the elements of the given SortedSet.**

Method:

Method	Description
boolean addAll(Collection c)	**It is used to add all of the elements in the specified collection to this set.**
boolean contains(Object o)	**It is used to return true if this set contains the specified element.**
boolean isEmpty()	**It is used to return true if this set contains no elements.**

Program:

```
package collectionsexample;

import java.util.TreeSet;

public class TreeSetExample {

    public static void main(String[] args) {
        //Creating a TreeSet without supplying any Comparator
        TreeSet<Integer> set = new TreeSet<Integer>();
        //Adding elements to TreeSet
        set.add(23);
        set.add(11);
        set.add(41);
        set.add(7);
        set.add(69);
        set.add(18);
        set.add(38);
        //printing elements of TreeSet
        System.out.println(set);       //Output : [7, 11, 18,
    }
}
```

Output:

```
run:
[7, 11, 18, 23, 38, 41, 69]
BUILD SUCCESSFUL (total time: 1 second)
```

13.Explain HashSet class in detail along wih an example?

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.

Difference between List and Set

List can contain duplicate elements whereas Set contains unique elements only.

Hierarchy of HashSet class

The HashSet class extends AbstractSet class which implements Set interface. The Set interface inherits Collection and Iterable interfaces in hierarchical order.

HashSet class declaration

Let's see the declaration for java.util.HashSet class.

public class HashSet<E> **extends** AbstractSet<E> **implements** Set<E>, Cloneable, Serializa ble

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.**
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.**

Methods:

Method	Description
void clear()	**It is used to remove all of the elements from this set.**
boolean contains(Object o)	**It is used to return true if this set contains the specified element.**
boolean add(Object o)	**It is used to adds the specified element to this set if it is not already present.**
boolean isEmpty()	**It is used to return true if this set contains no elements.**
boolean remove(Object o)	**It is used to remove the specified element from this set if it is present.**
Object clone()	**It is used to return a shallow copy of this HashSet instance: the elements themselves are not cloned.**
Iterator iterator()	**It is used to return an iterator over the elements in this set.**
int size()	**It is used to return the number of elements in this set.**

Program:

```java
package collectionsexample;

import java.util.HashSet;
import java.util.Iterator;

class Student
{
    String name;

    int rollNo;

    String department;

    public Student(String name, int rollNo, String department)
    {
        this.name = name;

        this.rollNo = rollNo;

        this.department = department;
    }

    @Override
    public int hashCode()
    {
        return rollNo;
    }

    @Override
    public boolean equals(Object obj)
    {
        Student student = (Student) obj;

        return (rollNo == student.rollNo);
    }

    @Override
    public String toString()
    {
        return rollNo+", "+name+", "+department;
    }
}

public class MainClass
{
```

```
        set.add(new Student("Samson", 565, "Civil"));

        //Iterating through HashSet

        Iterator<Student> it = set.iterator();

        while (it.hasNext())
        {
            Student student = (Student) it.next();

            System.out.println(student);
        }
    }
}
```

```
public static void main(String[] args)
{
    HashSet<Student> set = new HashSet<Student>();

    //Adding elements to HashSet

    set.add(new Student("Avinash", 121, "ECE"));

    set.add(new Student("Bharat", 101, "EEE"));

    set.add(new Student("Malini", 151, "Civil"));

    set.add(new Student("Suresh", 200, "IT"));

    set.add(new Student("Vikram", 550, "CS"));

    set.add(new Student("Bharat", 301, "IT"));

    set.add(new Student("Amit", 301, "IT"));

    set.add(new Student("Bhavya", 872, "ECE"));

    set.add(new Student("Naman", 301, "CS"));
```

Output:

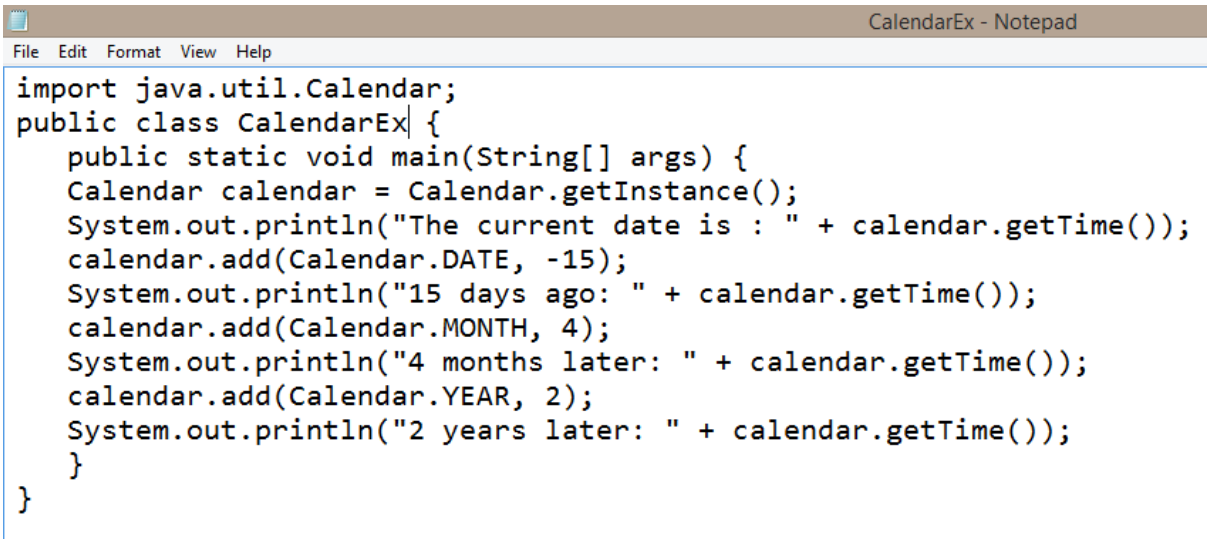
```
101, Bharat, EEE
565, Samson, Civil
550, Vikram, CS
151, Malini, Civil
200, Suresh, IT
872, Bhavya, ECE
121, Avinash, ECE
301, Bharat, IT
```

14.Explain calendar class in detail?

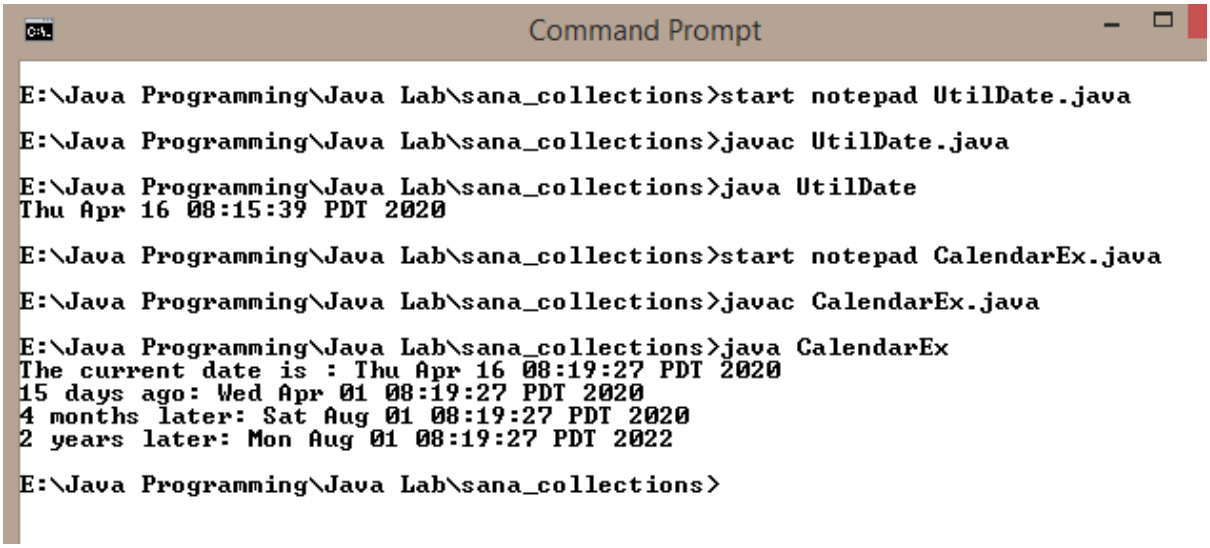
Java Calendar class is an abstract class that provides methods for converting date between a specific instant in time and a **set of calendar fields such as MONTH, YEAR, HOUR, etc.** It inherits Object class and implements the Comparable interface.

public abstract class Calendar **extends** Object **implements** Serializable, Cloneable, Compa rable<Calendar>

Program:



Output:


Command Prompt
E:\Java Programming\Java Lab\sana_collections>start notepad UtilDate.java
E:\Java Programming\Java Lab\sana_collections>javac UtilDate.java
E:\Java Programming\Java Lab\sana_collections>java UtilDate
Thu Apr 16 08:15:39 PDT 2020
E:\Java Programming\Java Lab\sana_collections>start notepad CalendarEx.java
E:\Java Programming\Java Lab\sana_collections>javac CalendarEx.java
E:\Java Programming\Java Lab\sana_collections>java CalendarEx
The current date is : Thu Apr 16 08:19:27 PDT 2020
15 days ago: Wed Apr 01 08:19:27 PDT 2020
4 months later: Sat Aug 01 08:19:27 PDT 2020
2 years later: Mon Aug 01 08:19:27 PDT 2022
E:\Java Programming\Java Lab\sana_collections>

UNIT-5

Short Answer Questions for 2 Marks

1.What is Swing in Java?How it differs from Applet?

Swing is a GUI widget toolkit for Java. It is part of Oracle's Java Foundation Classes (JFC)

Swings are called light weight component because swing components sits on the top of AWT components and do the work.

Applets will be downloaded at the client web browser and executed locally where as swing has a set of APIs for developing GUI components and can act as stand alone applications.

2.How do applets differ from application program?

Applets	**Application**
A program that runs within the scope of a dedicated widget engine or a larger program often as a plug-in	**A standalone program that is designed torun on a standalone machine to accomplish a task**
initialized through init()	**starts from main()**

3.Explain the use of layout managers?

a layout manager automatically arranges your controls within a window by using some type of algorithm

Each Container object has a layout manager associated with it. **A layout manager is an instance of any class that implements the LayoutManager interface. (panel,window or frame)**

The layout manager is set by the setLayout( ) method. If no call to setLayout( ) is made, then the default layout manager is used.

Whenever a container is resized (or sized for the first time), the layout manager is used to position each of the components within it.

The setLayout( ) method has the following general form:

void setLayout(LayoutManager layoutObj)

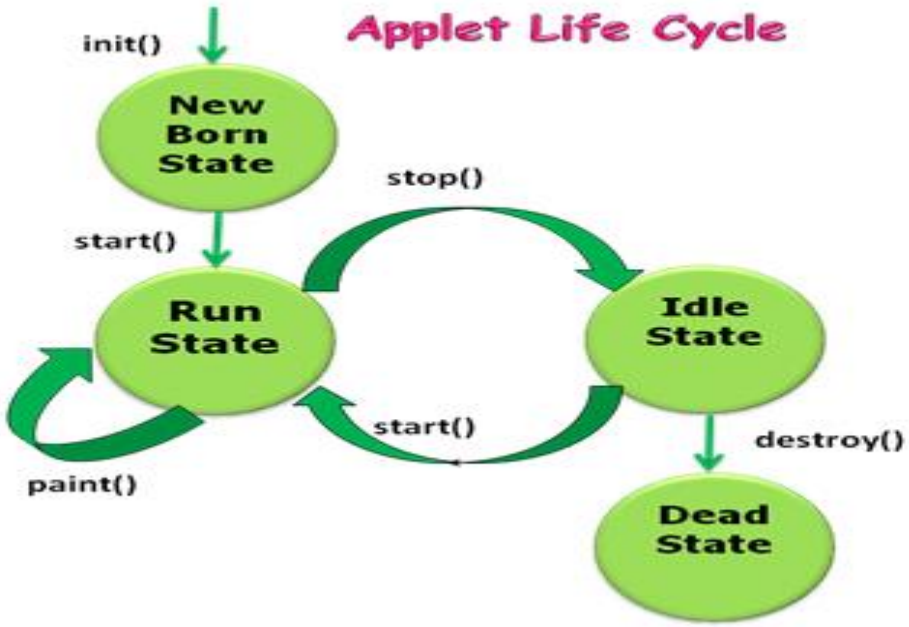
4.Explain the life cycle of an applet?

Four methods in the Applet class gives you the framework on which you build any serious applet −

It is important to understand the order in which the various methods shown in the skeleton are called.

When an applet begins, the following methods are called, in this sequence: **1. init( ) 2. start( ) 3. paint( )**

When an applet is terminated, the following sequence of method calls takes place: **1. stop( ) 2. destroy( )**



5.What are the limitations of AWT?

- The AWT defines a basic set of controls, windows, and dialog boxes that support a usable, but limited graphical interface.
- One reason for the limited nature of the AWT is that it translates its various visual components into their corresponding, platform-specific equivalents, or peers.
- **This means that the look and feel of a component is defined by the platform, not by Java.**
- Because the AWT components use native code resources, they are referred to as heavyweight. The use of native peers led to several problems.

6.Why do Applet classes need to be declared as public?

A java applet is a small program that can be placed within a web page, like a small game or a calculator.

-> When ever we declare the class as public, it means it is possible to access this class inside the same package and outside of the current package.

-> The server needs to access some of the applet's attributes, so it is declared as public. If the applet was private, nobody would be able to access it.

-> Also because classes from other packages such as sun.applet.AppletPanel cannot access it.

7.Mention the situation in which anonymous inner class is used?

A class that have no name is known as anonymous inner class in java. It should be used if you have to override method of class or interface

8.What is adapter class?

An adapter class provides the default implementation of all methods in an event listener interface. Adapter classes are very useful when you want to process only few of the events that are handled by a particular event listener interface

9.What is the relationship between an event listener interface and an event adapter class?

An event-listener interface allows describing the methods which must be implemented by one of the event handler for a specific event.

An event-adapter allows default implementations of an event-listener interface of a specific event.

10.What is AWT?

AWT stands for Abstract Window Toolkit. It is a platform dependent API for creating Graphical User Interface (GUI) for java programs. Java AWT calls native platform (Operating systems) subroutine for creating components such as textbox, checkbox, button etc.

11.What is container and component?

a container is a special type of component that is designed to hold other components..

a component is an independent visual control, such as a push button or slider.

In general, **Swing components are derived from the JComponent class**.

12.What are inner classes?Specify any two properties of inner class?

Java inner class or nested class is a class which is declared inside the class or interface.

We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.

Code Optimization: It requires less code to write.

Short Answer Questions for 3 Marks:

13.What are the containers available in swing?

a container is a special type of component that is designed to hold other components..

Swing defines two types of containers.

The first are top-level containers: JFrame, JApplet, JWindow, and JDialog. These containers do not inherit JComponent. They do, **however, inherit the AWT classes Component and Container**. Unlike Swing's other components, which are lightweight, the **top-level containers are heavyweight**. This makes the top-level containers a special case in the Swing component library.

The second type of containers supported by Swing are lightweight containers. Lightweight containers do inherit JComponent. An example of a lightweight container is JPanel, which is a general-purpose container. Lightweight containers are often used to organize and manage groups of related components because a lightweight container can be contained within another container. Thus, you can use lightweight containers such as JPanel to create subgroups of related controls that are contained within an outer container.

14.What are the advantages of event delegation Model?

There are two advantages of event-delegation model over event-inheritance model.

First – Event-delegation enables the handling of events by objects other than the ones which generate the events. It is a clean separation between design and usage of a component.

Second – It's performance is much better in applications in which many events are generated. This improvement of performance is due to the fact that the unhandled events need not be repeatedly processed, which is the case of event-inheritance model.

15.What is the function of SetLayout and FlowLayout?

The setLayout(...) method allows you to set the layout of the container, often a JPanel, to say FlowLayout, BorderLayout, GridLayout, null layout, or whatever layout desired. The layout manager helps lay out the components held by this container.

FlowLayout is the default layout manager.

This is the layout manager that the preceding examples have used. **FlowLayout implements a simple layout style, which is similar to how words flow in a text editor.**

16.What is an Applet?

An **applet** is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.

Advantage of Applet

There are many advantages of applet. They are as follows:

It works at client side so less response time.

Secured

17.What is JPanel Object?

JPanel is a Swing's lightweight container which is used to group a set of components together. JPanel is a pretty simple component which, normally, does not have a GUI (except when it is being set an opaque background or has a visual border)

The JPanel class resides in the package javax.swing and it's a subclass of the javax.swing.JComponent class.

Normally we create new JPanel object as simple as follows:

JPanel newPanel = new JPanel();

18.Differentiate between GridBagLayout and GridLayout?

GridBagLayout	**GridLayout**
The GridBagLayout class is a flexible layout manager that aligns components vertically and horizontally, without requiring that the components be of the same size	GridLayout class lays all components in a rectangular grid like structure of container.
. Each GridBagLayout object maintains a dynamic, rectangular grid of cells, with each component occupying one or more cells, called its display area.	The container is divided into an equal sized rectangles and each component is placed inside a rectangle.

Long Answer Questions for 10 Marks

1.What is the significance of layout managers?Discuss briefly various layout managers?

A layout manager is an object that controls the size and position of the components in the container. Every container object has a layout manager object that controls its layout.

Actually, layout managers are used to arrange the components in a specific manner. It is an interface that is implemented by all the classes of layout managers. There are some classes that represent the layout managers.

The Abstract Windowing Toolkit (AWT) has the following five layout managers:

java.awt.BorderLayout

java.awt.FlowLayout

java.awt.GridLayout

java.awt.CardLayout

java.awt.GridBagLayout

The layout manager is set by the setLayout( ) method. If no call to setLayout( ) is made, then the default layout manager is used.

Whenever a container is resized (or sized for the first time), the layout manager is used to position each of the components within it.

The setLayout( ) method has the following general form:

void setLayout(LayoutManager layoutObj)

Here, layoutObj is a reference to the desired layout manager.

If you wish to disable the layout manager and position components manually, pass null for layoutObj.

If you do this, you will need to determine the shape and position of each component manually, using the setBounds( ) method defined by Component

FlowLayout:

- **FlowLayout is the default layout manager.**
- This is the layout manager that the preceding examples have used. **FlowLayout implements a simple layout style, which is similar to how words flow in a text editor.**
- The direction of the layout is governed by the **container's component orientation property**, which, by default, **is left to right, top to bottom.**

- Therefore, by default, components are laid out line-by-line beginning at the upper-left corner.
- In all cases, when a line is filled, layout advances to the next line.
- A small space is left between each component, above and below, as well as left and right.
- Here are the constructors for FlowLayout:
- **FlowLayout( )**
- **FlowLayout(int how)**
- **FlowLayout(int how, int horz, int vert)**

BorderLayout:

- The BorderLayout class implements a common layout style for top-level windows.
- It has four narrow, fixed-width components at the edges and one large area in the center.
- The four sides are referred to as north, south, east, and west. The middle area is called the center.
- Here are the constructors defined by BorderLayout:
- BorderLayout( )
- BorderLayout(int horz, int vert)
- The first form creates a default border layout.
- The second allows you to specify the horizontal and vertical space left between components in horz and vert, respectively.
- BorderLayout defines the following constants that specify the regions:
- BorderLayout.CENTER,BorderLayout.SOUTH,BorderLayout.EAST BorderLayout.WEST ,BorderLayout.NORTH

GridLayout:

- **GridLayout lays out components in a two-dimensional grid.**
- When you instantiate a GridLayout, you define the number of rows and columns.
- The constructors supported by GridLayout are shown here:
- GridLayout( )

- GridLayout(int numRows, int numColumns)
- GridLayout(int numRows, int numColumns, int horz, int vert)
- **The first form creates a single-column grid layout.**
- The second form creates a grid layout with the specified number of rows and columns.
- **The third form allows you to specify the horizontal and vertical space left between components in horz and vert, respectively.**
- Either numRows or numColumns can be zero.
- **Specifying numRows as zero allows for unlimitedlength columns.**
- Specifying numColumns as zero allows for unlimited-length rows

CardLayout:

- **The CardLayout class is unique among the other layout managers in that it stores several different layouts.**
- Each layout can be thought of as being on a separate index card in a deck that can be shuffled so that any card is on top at a given time.
- This can be useful for user interfaces with optional components that can be dynamically enabled and disabled upon user input.
- You can prepare the other layouts and have them hidden, ready to be activated when needed.
- CardLayout provides these two constructors:
- **CardLayout( )**
- **CardLayout(int horz, int vert)**
- The first form creates a default card layout.
- The second form allows you to specify the horizontal and vertical space left between components in horz and vert, respectively.

GridBagLayout:

- A good way to do this is to use a grid bag layout, which is specified by the GridBagLayout class.
- **What makes the grid bag useful is that you can specify the relative placement of components by specifying their positions within cells inside a grid.**

- The key to the grid bag is that each component can be a different size, and each row in the grid can have a different number of columns.
- **This is why the layout is called a grid bag. It's a collection of small grids joined together.**
- The location and size of each component in a grid bag are determined by a set of constraints linked to it.
- **The constraints are contained in an object of type GridBagConstraints.**
- Constraints include the height and width of a cell, and the placement of a component, its alignment, and its anchor point within the cell.

2.Explain about JButton class along with an example?

- The class **JButton** is an implementation of a push button. This component has a label and generates an event when pressed. It can also have an Image.
- **Class Declaration**
- Following is the declaration for **javax.swing.JButton** class −
- **public class JButton extends AbstractButton implements Accessible**

Sr.No.	Constructor & Description
1	**JButton()** Creates a button with no set text or icon.
2	**JButton(Action a)** Creates a button where properties are taken from the Action supplied.
3	**JButton(Icon icon)** Creates a button with an icon.
4	**JButton(String text)** Creates a button with the text.
5	**JButton(String text, Icon icon)** Creates a button with an initial text and an icon.

Methods:

Methods	Description
void setText(String s)	It is used to set specified text on button
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object.

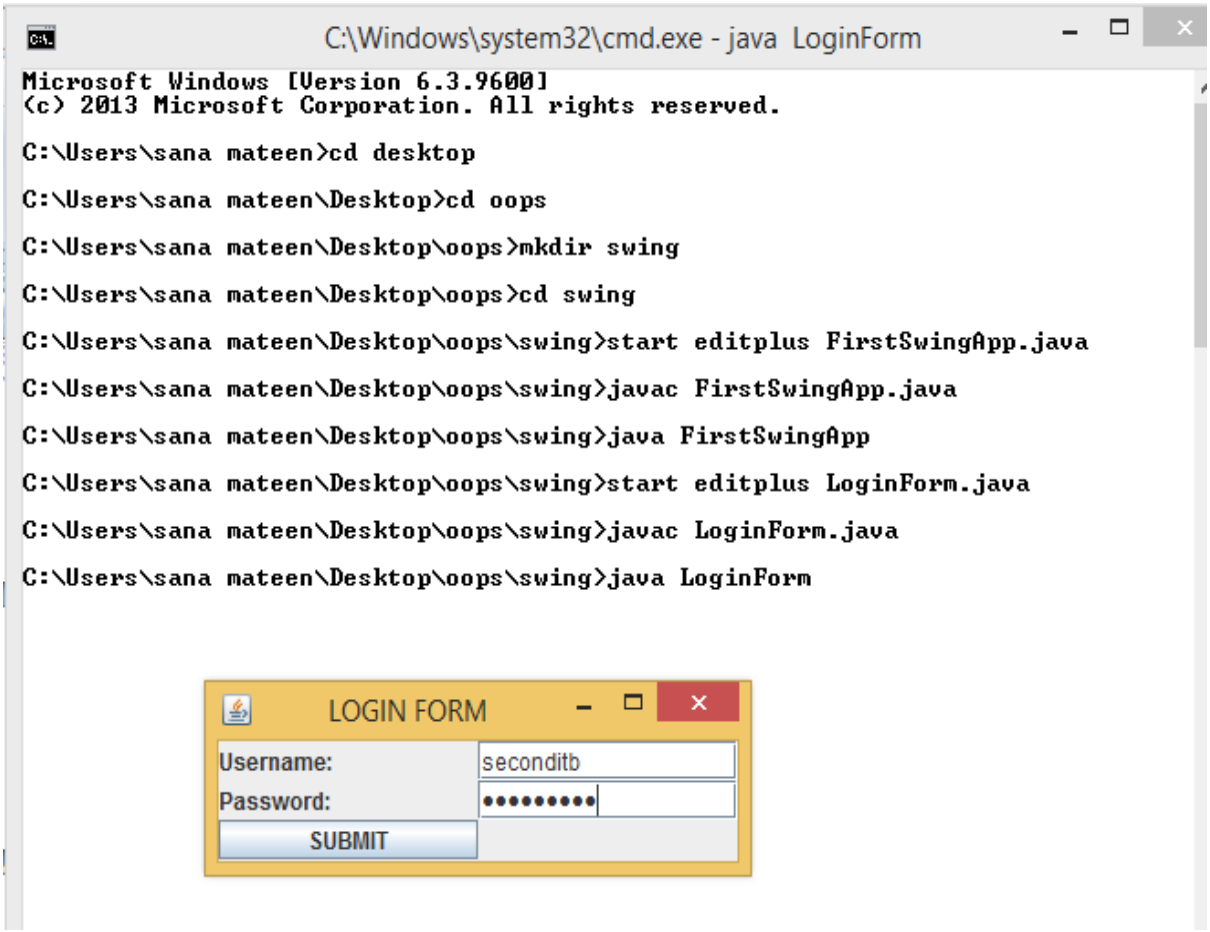
Program:

```
50      class  LoginForm
51      {
52            public static void main(String arg[])
53            {
54                try
55                {
56                    Login frame=new Login();
57                    frame.setSize(300,100);
58                    frame.setVisible(true);
59                }
60                catch(Exception e)
61                {
62                    JOptionPane.showMessageDialog(null,  e.getMessage());
63                }
64            }
65      }
66     class NextPage extends JFrame
67     {
68        NextPage()
69       {
70           setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
71           setTitle("Welcome");
72           setSize(400, 200);
73        }
74   }
```

```
1       import javax.swing.*;
2       import java.awt.*;
3       import java.awt.event.*;
4
5       class Login extends JFrame implements ActionListener
6        {
7             JButton SUBMIT;
8             JPanel panel;
9             JLabel label1,label2;
10            final JTextField  text1,text2;
11            Login()
12            {
13                      label1 = new JLabel();
14                      label1.setText("Username:");
15                      text1 = new JTextField(15);
16
17                      label2 = new JLabel();
18                      label2.setText("Password:");
19                      text2 = new JPasswordField(15);
20
21                      SUBMIT=new JButton("SUBMIT");
22
23                      panel=new JPanel(new GridLayout(3,1));
24                      panel.add(label1);
25                      panel.add(text1);
26                      panel.add(label2);
27                      panel.add(text2);
28                      panel.add(SUBMIT);
29                      add(panel,BorderLayout.CENTER);
30                      SUBMIT.addActionListener(this);
```

```
31                  setTitle("LOGIN FORM");
32              }
33            public void actionPerformed(ActionEvent ae)
34            {
35                  String value1=text1.getText();
36                  String value2=text2.getText();
37                  if (value1.equals("seconditb") && value2.equals("seconditb")) {
38                      NextPage page=new NextPage();
39                      page.setVisible(true);
40                      JLabel label = new JLabel("Welcome:"+value1);
41                      page.getContentPane().add(label);
42                  }
43                  else{
44                      System.out.println("enter the valid username and password");
45                      JOptionPane.showMessageDialog(this,"Incorrect login or password",
46                      "Error",JOptionPane.ERROR_MESSAGE);
47                  }
48           }
49      }
```

Output:



3.Explain delegation event model?

The event model is based on the Event Source and Event Listeners. Event Listener is an object that receives the messages / events. The Event Source is any object which creates the message / event. The Event Delegation model is based on – The Event Classes, The Event Listeners, Event Objects.

There are three participants in event delegation model in Java;

- Event Source – the class which broadcasts the events

- Event Listeners – the classes which receive notifications of events

- Event Object – the class object which describes the event.

An event occurs (like mouse click, key press, etc) which is followed by the event is broadcasted by the event source by invoking an agreed method on all event listeners. The event object is passed as argument to the agreed-upon method. Later the event listeners respond as they fit, like submit a form, displaying a message / alert etc.

Steps involved in Event Handling:

- The User clicks the button and the event is generated.
- Now the object of concerned event class is created automatically and information about the source and the event get populated with in same object.
- ✓ Event object is forwarded to the method of registered listener class.
- the method now gets executed and returns.
- Points to remember about listener
- In order to design a listener class we have to develop some listener interfaces.These Listener interfaces forecast some public abstract callback methods which must be implemented by the listener class.
- If you do not implement the any if the predefined interfaces then your class can not act as a listener class for a source object.

Event Class	Description
ActionEvent	Generated when a button is pressed, a list item is double-clicked, or a menu item is selected.
AdjustmentEvent	Generated when a scroll bar is manipulated.
ComponentEvent	Generated when a component is hidden, moved, resized, or becomes visible.
ContainerEvent	Generated when a component is added to or removed from a container.
FocusEvent	Generated when a component gains or loses keyboard focus.
InputEvent	Abstract superclass for all component input event classes.
ItemEvent	Generated when a check box or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected.
KeyEvent	Generated when input is received from the keyboard.
MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.
MouseWheelEvent	Generated when the mouse wheel is moved.
TextEvent	Generated when the value of a text area or text field is changed.
WindowEvent	Generated when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

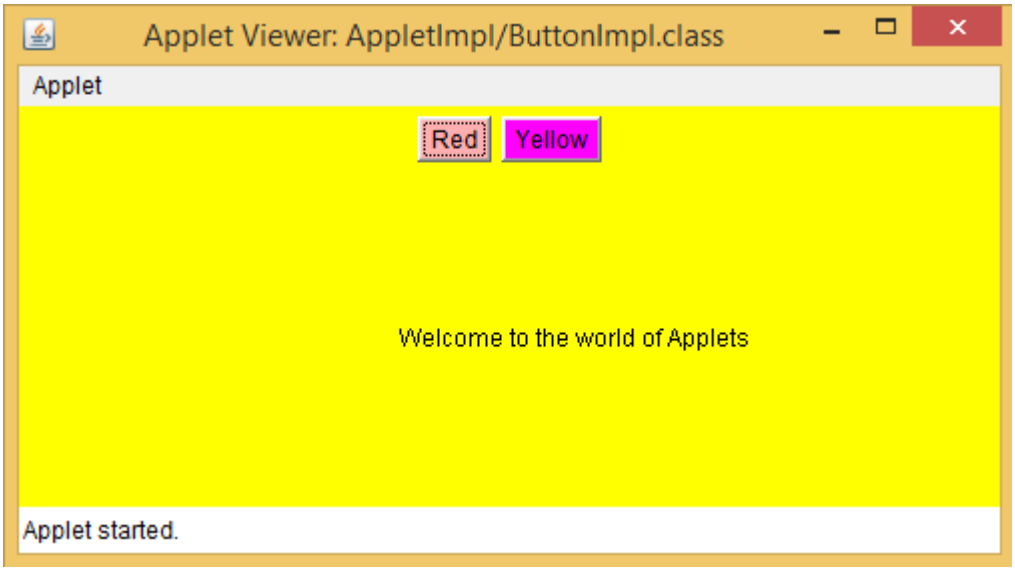
Interface	Description
ActionListener	Defines one method to receive action events.
AdjustmentListener	Defines one method to receive adjustment events.
ComponentListener	Defines four methods to recognize when a component is hidden, moved, resized, or shown.
ContainerListener	Defines two methods to recognize when a component is added to or removed from a container.
FocusListener	Defines two methods to recognize when a component gains or loses keyboard focus.
ItemListener	Defines one method to recognize when the state of an item changes.
KeyListener	Defines three methods to recognize when a key is pressed, released, or typed.
MouseListener	Defines five methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released.
MouseMotionListener	Defines two methods to recognize when the mouse is dragged or moved.
MouseWheelListener	Defines one method to recognize when the mouse wheel is moved.
TextListener	Defines one method to recognize when a text value changes.
WindowFocusListener	Defines two methods to recognize when a window gains or loses input focus.
WindowListener	Defines seven methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

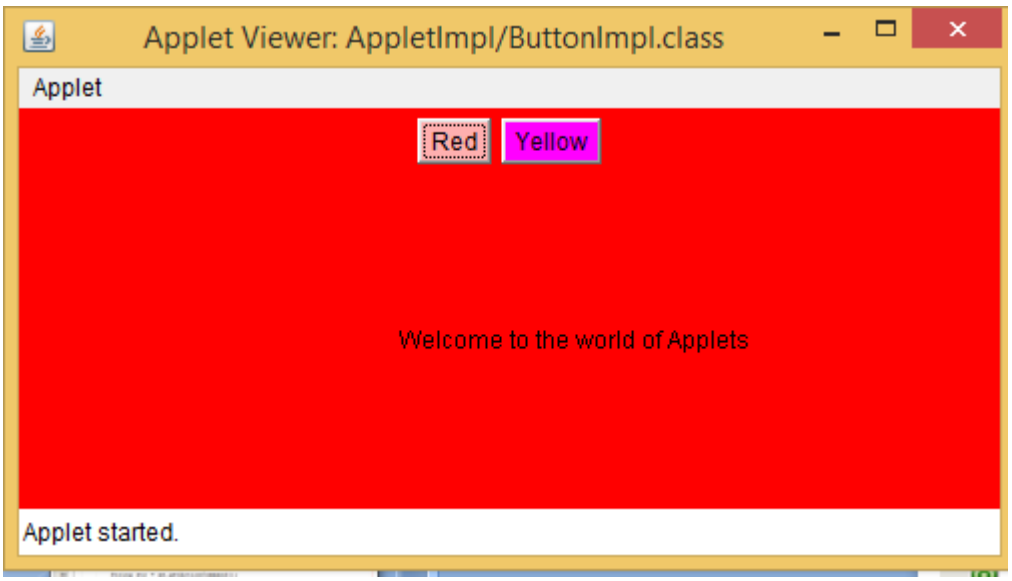
Program:

```
1   package AppletImpl;
2   import java.applet.Applet;
3   import java.awt.*;
4   import java.awt.event.ActionEvent;
5   import java.awt.event.ActionListener;
6
7   /**
8    *
9    * @author sana mateen
10   */
11  public class ButtonImpl extends Applet implements ActionListener{
12  Button b1,b2;
13      /**
14       * Initialization method that will be called after the applet is loaded into
15       * the browser.
16       */
17  @Override
        public void init() {
19          b1=new Button("Red");
20          b2=new Button("Yellow");
21          b1.setBackground(Color.pink);
22          b2.setBackground(Color.MAGENTA);
23          add(b1);
24          add(b2);
25          b1.addActionListener(this);
26          b2.addActionListener(this);
27      }
28  @Override
        public void actionPerformed(ActionEvent ae){
30          String str = ae.getActionCommand();
31          if (str.equals("Red"))
32              setBackground(Color.red);
```

```
33              if(str.equals("Yellow"))
34                  setBackground(Color.yellow);
35
36              repaint();
37          }
38      @Override
            public void paint(Graphics g) {
40       g.drawString("Welcome to the world of Applets", 190, 120);
41       }
42
43
44      }
```

Output:





4.Write an applet to draw a smiley picture and accept user name as a parameter and display welcome message?

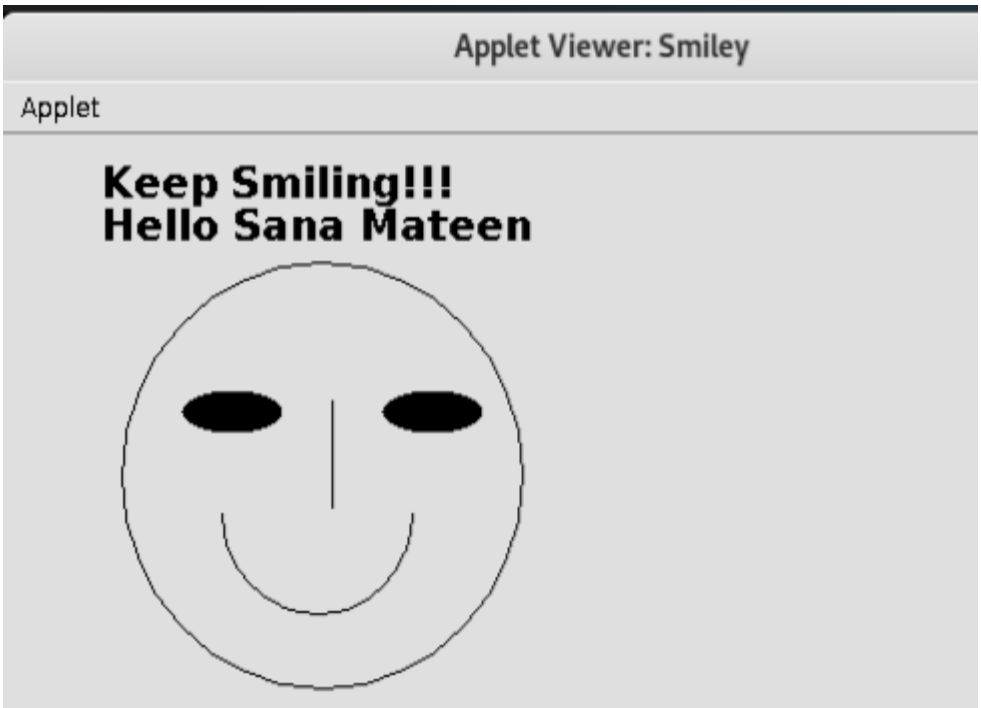
Program:

Open

Smiley.java
~/jtuts

```
import java.awt.*;
import java.applet.*;
/*<applet code="Smiley" width=600 height=500>
<param name="Username" value="Sana Mateen">
</applet>*/
public class Smiley extends Applet{
        String msg="";
        public void init()
        {
                 msg = getParameter("Username");
                 if(msg == null)
                {
                        msg = "Java";
                 }
                 msg = "Hello " + msg;
        }
        public void paint(Graphics g){
                Font f = new Font("Helvetica", Font.BOLD,20);
                g.setFont(f);
                g.drawString("Keep Smiling!!!", 50, 30);
                g.drawOval(60, 60, 200, 200);
                g.fillOval(90, 120, 50, 20);
                g.fillOval(190, 120, 50, 20);
                g.drawLine(165, 125, 165, 175);
                g.drawArc(110, 130, 95, 95, 0, -180);
                g.drawString(msg,50,50);
        }
}
```

Output:



5.What is the difference between init() and start() methods in an Applet?Explain through an example program?

an Applet also comes with a life cycle. Different states in which an applet exist between its object creation and object garbage collection is known as life cycle. The methods involved in the life cycle of applet are init(), start(), paint(), stop() and destroy(). These methods are known as life cycle methods. All the methods are callback methods.

The Browser is the tool that executes the applet. It is the browser's responsibility to provide the execution environment for applet. Browser is also should take care to call the life cycle methods at appropriate times.

Because all the life cycle methods defined in java.applet.Applet are concrete (non-abstract) methods, the Programmer need not override all and can override (use) what ever he thinks fit for his code. Generally, Programmer uses init() to initialize variables, paint() to display on applet window and destroy() to free the system resources like closing file streams and socket handles etc. Let us summarize start vs init differences.

init() and start() and other methods usage with complete tutorial along figures can be read from Life Cycle of Applet that goes in 3 pages.

1.init() Method

In this stage, the applet starts its journey. This method is called when the applet is loaded by the Browser for execution. In this method, the applet object is created by the applet. Here, the Programmer can initialize variables, create objects, load imags etc. The init() method is equivalent to a constructor of an application. This method is called only once in the life cycle. Here, the PARAM tags of HTML file are read with getParameter() method of Applet.

2. start() Method

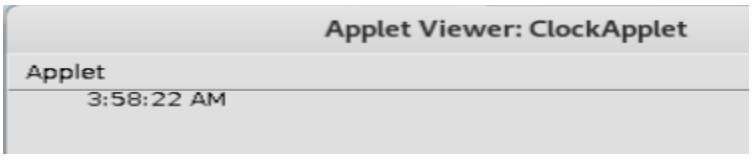
The start() method is called by init() method. For browser, there is another reason to call the start() method. When applet is brought back from minimize (seen in status bar) position. In minimize position, applet looses focus of the viewer. When the applet get focus, Browser calls first start() and then paint(). This method is called and executed multiple times whenever the applet gains focus (after loosing earlier).

Program:

```
import java.applet.Applet;
import java.awt.*;
import java.util.Date;
import java.text.DateFormat;
/*<applet code="ClockApplet" width=500 height=600></applet>*/
/** An Applet to display the current time */
public class ClockApplet extends Applet implements Runnable {
  /** A Thread to run the timer */
  protected Thread timerThread;
  /** The date object */
  Date date = new Date();
  /** The date format */
  protected DateFormat format = DateFormat.getTimeInstance();

  /* Applet Lifestyle Methods */
  @Override
  public void start() {
    timerThread = new Thread(this, "Clock");
    timerThread.start();
  }

  @Override
  public void stop() {
    if (timerThread == null)
      return;
    timerThread = null;
  }
```

```
  /** Show the time, and wait a while. */
  @Override
  public void run() {
    while (timerThread != null) {
      repaint();   // request a redraw
      try {
        Thread.sleep(1000);
      } catch (InterruptedException e){ /* do nothing*/ }
    }
  }

  /** Display the time.
   * @param g */
  @Override
  public void paint(Graphics g) {
    date.setTime(System.currentTimeMillis());
    g.drawString(format.format(date), 40, 10);
  }
}
```

Output:

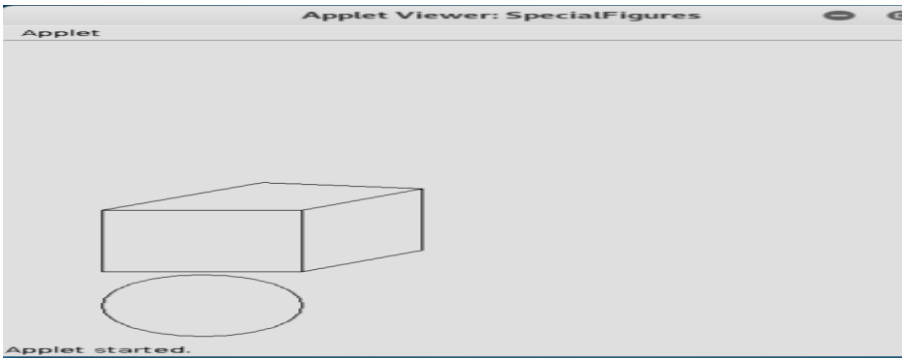


6.Write the applets to draw Cube and Circle shapes?

Program:

```
Open                                                    *SpecialFigure
                                                        ~/jtuts
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*@sana mateen*/
/*<applet code="SpecialFigures" width=500 height=600></applet>*/
public class SpecialFigures extends Applet
{

  public void paint(Graphics g)
  {
                          // to draw cube
    g.drawRect(50,275,100,100);
    g.drawLine(50,275,130,230);
    g.drawLine(150,275,210,240);
    g.drawLine(130,230,210,240);
    g.drawLine(210,240,210,340);
    g.drawLine(150,375,210,340);
                          // to draw circle
    g.drawOval(50,380,100,100);
  }

}
```

Output:



7.Write a program to create a frame windows that responds to mouse clicks?

Program:

import java.awt.Frame;

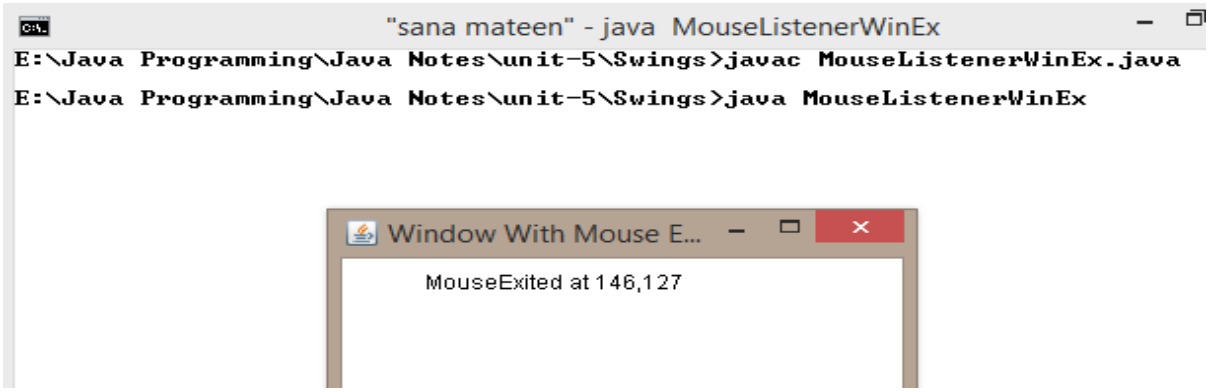
import java.awt.Graphics;

```
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class MouseListenerWinEx extends Frame implements MouseListener{
        int x=0, y=0;
        String strEvent = "";
        MouseListenerWinEx(String title){
        //call superclass constructor with window title
        super(title);
        //add window listener
        addWindowListener(new MyWindowAdapter(this));
        //add mouse listener
        addMouseListener(this);
        //set window size
        setSize(300,300);
        //show the window
        setVisible(true);
 }
 public void mouseClicked(MouseEvent e) {
        strEvent = "MouseClicked";
        x = e.getX();
        y = getY();
        repaint();
 }
 public void mousePressed(MouseEvent e) {
```

```
            strEvent = "MousePressed";
            x = e.getX();
            y = getY();
            repaint();
    }
    public void mouseReleased(MouseEvent e) {
            strEvent = "MouseReleased";
            x = e.getX();
            y = getY();
            repaint();
    }
    public void mouseEntered(MouseEvent e) {
            strEvent = "MouseEntered";
            x = e.getX();
            y = getY();
            repaint();
    }
    public void mouseExited(MouseEvent e) {
            strEvent = "MouseExited";
            x = e.getX();
            y = getY();
            repaint();
   }
   public void paint(Graphics g){
            g.drawString(strEvent + " at " + x + "," + y, 50,50);
    }
    public static void main(String[] args) {
```

```
        MouseListenerWinEx myWindow =  new MouseListenerWinEx("Window With Mouse Events Example");
 }


}
 class MyWindowAdapter extends WindowAdapter{
        MouseListenerWinEx myWindow = null;
        MyWindowAdapter(MouseListenerWinEx myWindow){
                this.myWindow = myWindow;
        }

        public void windowClosing(WindowEvent we){
                myWindow.setVisible(false);
        }
}
```

Output:



8.Explain Frame class along with an example program?

The class **Frame** is a top level window with border and title. It uses BorderLayout as default layout manager.

public class Frame extends Window implements MenuContainer

Frame()

Constructs a new instance of Frame that is initially invisible.

Frame(String title)

Constructs a new, initially invisible Frame object with the specified title.

void setResizable(boolean resizable)

Sets whether this frame is resizable by the user.

void setTitle(String title)

Sets the title for this frame to the specified string.

String getTitle()

Gets the title of the frame.

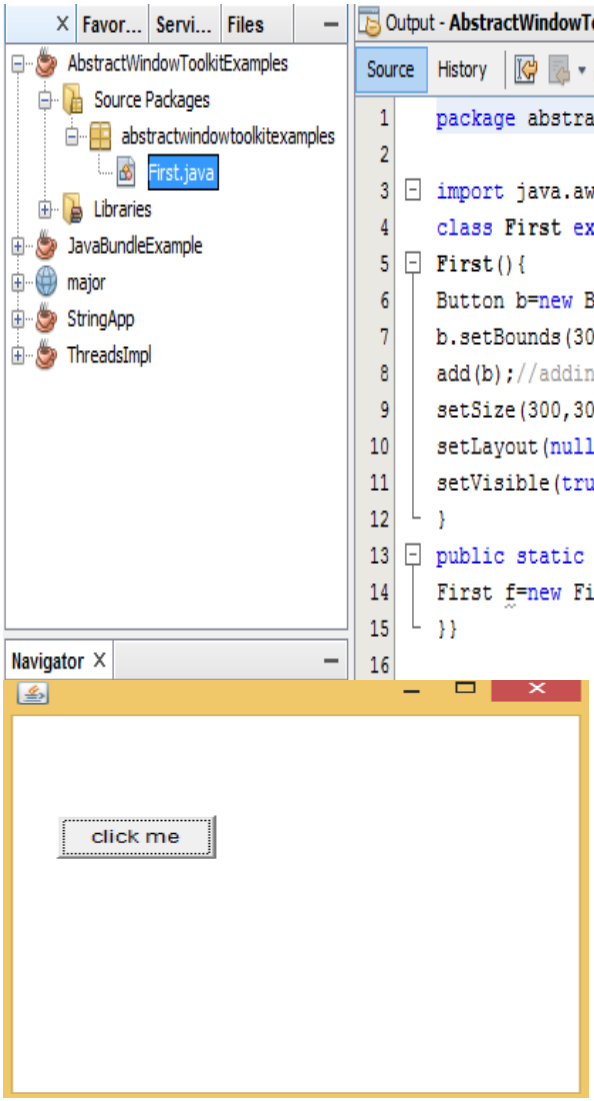
Void setVisible(boolean visible)

Set the frame visible or not

A frame can be created by two ways:

By extending Frame class

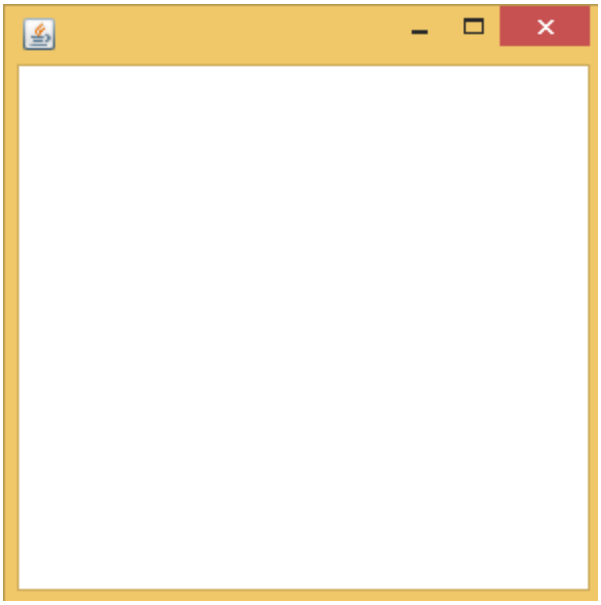
By creating an instance of Frame class


Favor...
Servi...
Files
AbstractWindowToolkitExamples
Source Packages
abstractwindowtoolkitexamples
First.java
Libraries
JavaBundleExample
major
StringApp
ThreadsImpl
Navigator
Output - AbstractWindowToolkitExamples (run)
First.java
Source
History
1 package abstractwindowtoolkitexamples;
2
3 import java.awt.*;
4 class First extends Frame{
5 First(){
6 Button b=new Button("click me");
7 b.setBounds(30,100,80,30);// setting button position
8 add(b);//adding button into frame
9 setSize(300,300);//frame size 300 width and 300 height
10 setLayout(null);//no layout manager
11 setVisible(true);//now frame will be visible, by default not visible
12 }
13 public static void main(String args[]){
14 First f=new First();
15 }}
16
click me

Example program:

```
1  /*
2   * To change this license header, choose License Header
3   * To change this template file, choose Tools | Templat
4   * and open the template in the editor.
5   */
6  package abstractwindowtoolkitexamples;
7  import java.awt.*;
8  /**
9   *
10  * @author sana mateen
11  */
12 public class Second {
13     public static void main(String[] args){
14         Frame fr=new Frame();
15         fr.setSize(300,300);
16         fr.setVisible(true);
17     }
18
19 }
```

Output:



9.**Differentiate between AWT and Swings?**

AWT stands for Abstract windows toolkit.	**Swing is also called as JFC's (Java Foundation classes).**
AWT components are called Heavyweight component.	**Swings are called light weight component because swing components sits on the top of AWT components and do the work.**
AWT components require java.awt package.	**Swing components require javax.swing package.**
AWT components are platform dependent.	**Swing components are made in purely java and they are platform independent.**
This feature is not supported in AWT.	**We can have different look and feel in Swing.**
These feature is not available in AWT.	**Swing has many advanced features like JTabel, Jtabbed pane which is not available in AWT. Also. Swing components are called "lightweight" because they do not require a native OS object to implement their functionality. JDialog and JFrame are heavyweight, because they do have a peer. So components like JButton, JTextArea, etc., are lightweight because they do not have an OS peer.**
With AWT, you have 21 "peers" (one for each control and one for the dialog itself). A "peer" is a widget provided by the operating system, such as a button object or an entry field object.	**With Swing, you would have only one peer, the operating system's window object. All of the buttons, entry fields, etc. are drawn by the Swing package on the drawing surface provided by the window object. This is the reason that Swing has more code. It has to draw the button or other control and implement its behavior instead of relying on the host operating system to perform those functions**
AWT is a thin layer of code on top of the OS.	**Swing is much larger. Swing also has very much richer functionality.**

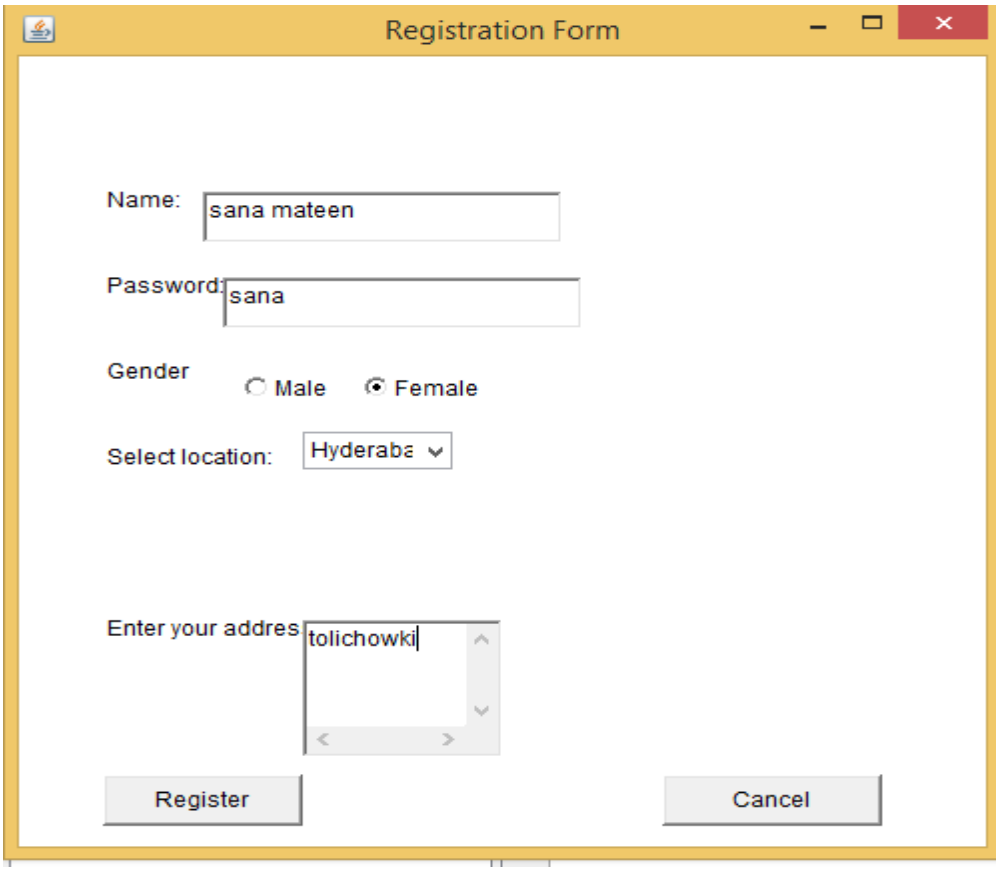
10.Write a java program to create a registration form using AWT?

Program:

```
1   package abstractwindowtoolkitexamples;
2
3   /**
4    * @author sana mateen
5    */
6   import java.awt.*;
7
8   class LoginForm {
9
10      public static void main(String args[]) {
11          Frame f = new Frame("Registration Form");
12          Label l1, l2, l3, l4,l5;
13          l1 = new Label("Name:");
14          l1.setBounds(50, 100, 100, 30);
15          TextField t1, t2;
16          t1 = new TextField("Please enter your name");
17          t1.setBounds(100, 110, 180, 30);
18          f.add(t1);
19          l2 = new Label("Password:");
20          l2.setBounds(50, 150, 100, 30);
21          t2 = new TextField("Please enter your password");
22          t2.setBounds(110, 160, 180, 30);
23          l3 = new Label("Gender");
24          l3.setBounds(50, 200, 100, 30);
25          CheckboxGroup cbg = new CheckboxGroup();
26          Checkbox checkbox1 = new Checkbox("Male", cbg, false);
27          checkbox1.setBounds(120, 200, 50, 50);
28          Checkbox checkbox2 = new Checkbox("Female", cbg, true);
29          checkbox2.setBounds(180, 200, 60, 50);
30          l4 = new Label("Select location:");
31          l4.setBounds(50, 250, 100, 30);
32          Choice c = new Choice();
```

```
33          c.setBounds(150, 250, 75, 75);
34          c.add("Hyderabad");
35          c.add("secunderabad");
36          c.add("chennai");
37          c.add("Ajmer");
38          c.add("Punjab");
39          l5 = new Label("Enter your address:");
            l5.setBounds(50, 350, 120, 30);
41          TextArea area=new TextArea();
42          area.setBounds(150,360, 100,80);
43          f.add(area);
44          f.add(c);
45          f.add(checkbox1);
46          f.add(checkbox2);
47          f.add(t2);
48          f.add(l1);
49          f.add(l2);
50          f.add(l3);
51          f.add(l4);
52          f.add(l5);
53          f.setSize(400, 400);
54          Button b1 = new Button();
55          b1.setLabel("Register");
56          b1.setBounds(50, 450, 100, 30);// setting button position
57          f.add(b1);//adding button into frame
58          Button b2 = new Button();
59          b2.setLabel("Cancel");
60          b2.setBounds(330, 450, 110, 30);// setting button position
61          f.add(b2);//adding button into frame
62          f.setSize(500, 500);//frame size 300 width and 300 height
63          f.setLayout(null);
```

Output:



11.Explain MVC Architecture in detail?

In general, a visual component is a composite of three distinct aspects:

• The way that the **component looks when rendered** on the screen

• The way that the **component reacts** to the user

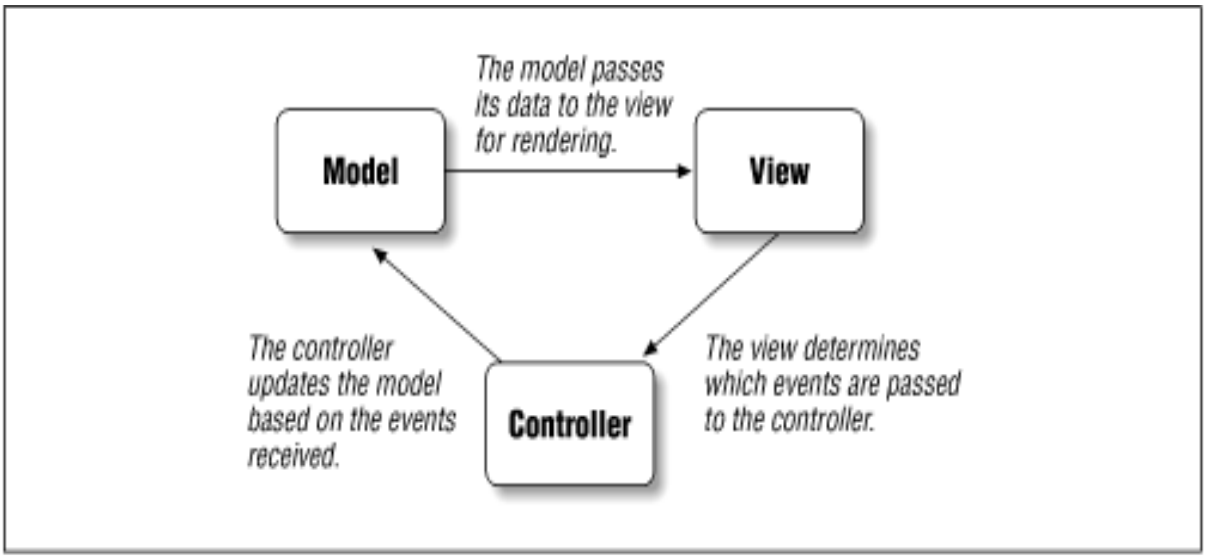
• **The state information associated with the component** No matter what architecture is used to implement a component, it must implicitly contain these three parts.

Over the years, one component architecture has proven itself to be exceptionally effective: Model-View-Controller, or MVC for short.

In MVC terminology, **the model corresponds to the state information** associated with the component.

For example**, in the case of a check box, the model contains a field that indicates if the box is checked or unchecked.**

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.

For example, when the user clicks a check box, the controller reacts by changing the model to reflect the user's choice (checked or unchecked).

This then results in the view being updated.

By separating a component into a model, a view, and a controller, the specific implementation of each can be changed without affecting the other two.

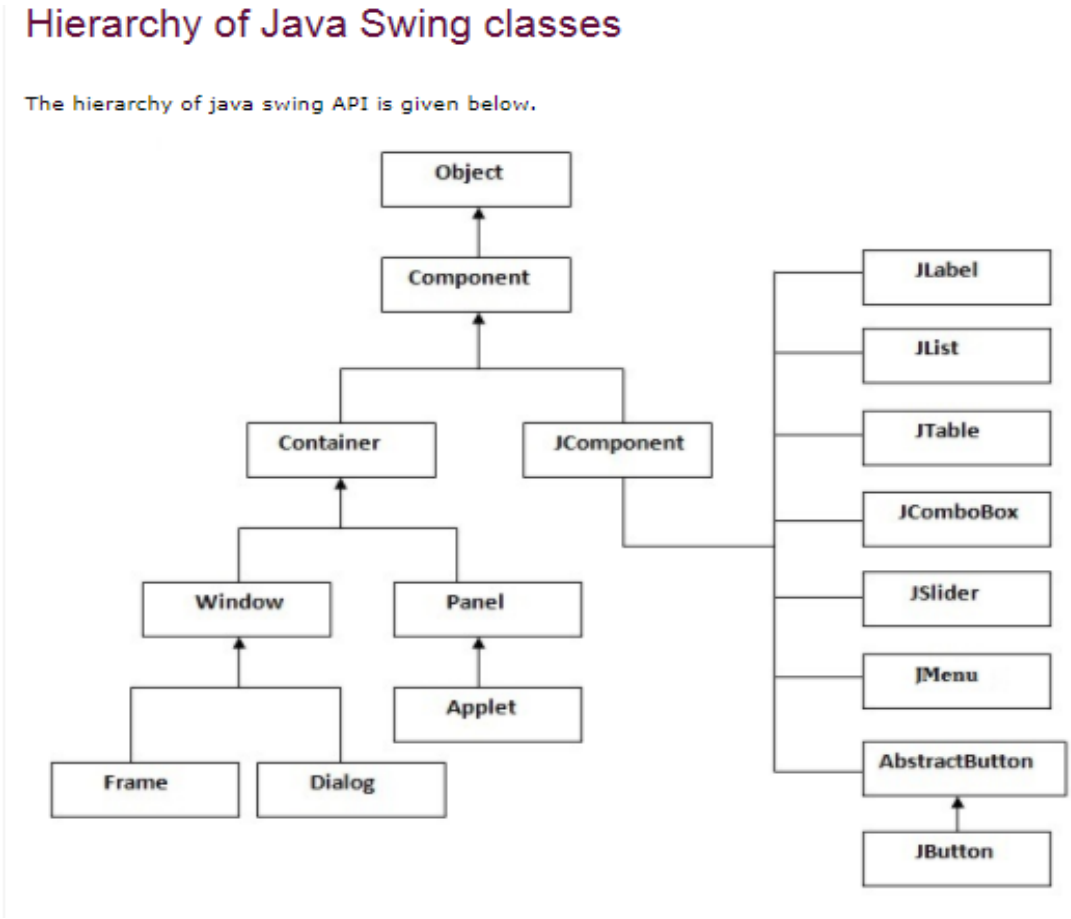
For instance, different view implementations can render the same component in different ways without affecting the model or the controller.

Swing uses a modified version of MVC that combines the view and the controller into a single logical entity called the UI delegate

For this reason, Swing's approach is called either the Model-Delegate architecture or the Separable Model architecture.

Therefore, although Swing's component architecture is based on MVC, it does not use a classical implementation of it.

12.Briefly Explain hierarchy of swing components?



In general, Swing components except the top-level containers extend the JComponent class which extends Container class. So, any swing component that extends JComponent is light-weight. The top-level containers which are discussed previously, are heavy-weight because they must depend on the OS. In this tutorial you will learn:

General class hierarchy of light-weight components.

What is containment hierarchy? What is a root?

What is a content pane and a root pane?

Containment Hierarchy

This might seem quite weird, but we can kill it! You already knew that a top-level container contains all the [components] or [light-weight containers (for example, JPanel) which again contain components]. This hierarchy that contains all the components is called as Containment Hierarchy. Every containment hierarchy has its root which is a top-level container. So, how many containment hierarchies can be in an application? Well, the answer is it is equal to the no.of top-level containers in the application. For example, if the application contains 1 JFrame and 4 dialogs, then it has got 5 top-level containers, so 5 containment hierarchies.

Note that containment hierarchies are different for applets and stand alone applications as their roots differ. In applets, the root is JApplet and for stand alone apps it is JFrame.

What is a content pane?

A content pane is an abstract panel on a top-level container that holds components. You already knew that a component cannot be displayed if it isn't add to a container. So, in order to add components to a container, we need to get its content pane and add to it. Programmers, previously wrote

myFrame.getContentPane().add(new JButton("Button"));

to write add a simple JButton to a JFrame however, we no longer require call to this method except in the case of setting background for JFrame.Only top-level containers in Swing has this method.

The great thing in Swing is every component is a container. Yes, because every component, including the top-level container has the add() method. Well, another reason that will strongly support this argument is JComponent is a sub-class of Container.

The Root Pane

Root Pane is an abstract layer on a top-level container. This root pane contains several panes in it like the Layered pane, content pane and menu bar and glass pane. Though, we don't use it typically, we can.

Now you might be clear about the general class hierarchy of light weight swing components and the containment hierarchy.

13.What is an event? Mention the different event classes and event listener interfaces and write a java program to implement ActionListener interface?

Change in the state of an object is known as event i.e. event describes the change in state of source. Events are generated as result of user interaction with the graphical user interface components. For example, **clicking on a button**, **moving the mouse**, **entering a character through keyboard**, **selecting an item from list**, **scrolling the page** are the activities that causes an event to happen.

Event Classes:

Event Class	Description
ActionEvent	Generated when a button is pressed, a list item is double-clicked, or a menu item is selected.
AdjustmentEvent	Generated when a scroll bar is manipulated.
ComponentEvent	Generated when a component is hidden, moved, resized, or becomes visible.
ContainerEvent	Generated when a component is added to or removed from a container.
FocusEvent	Generated when a component gains or loses keyboard focus.
InputEvent	Abstract superclass for all component input event classes.
ItemEvent	Generated when a check box or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected.
KeyEvent	Generated when input is received from the keyboard.
MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.
MouseWheelEvent	Generated when the mouse wheel is moved.
TextEvent	Generated when the value of a text area or text field is changed.
WindowEvent	Generated when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

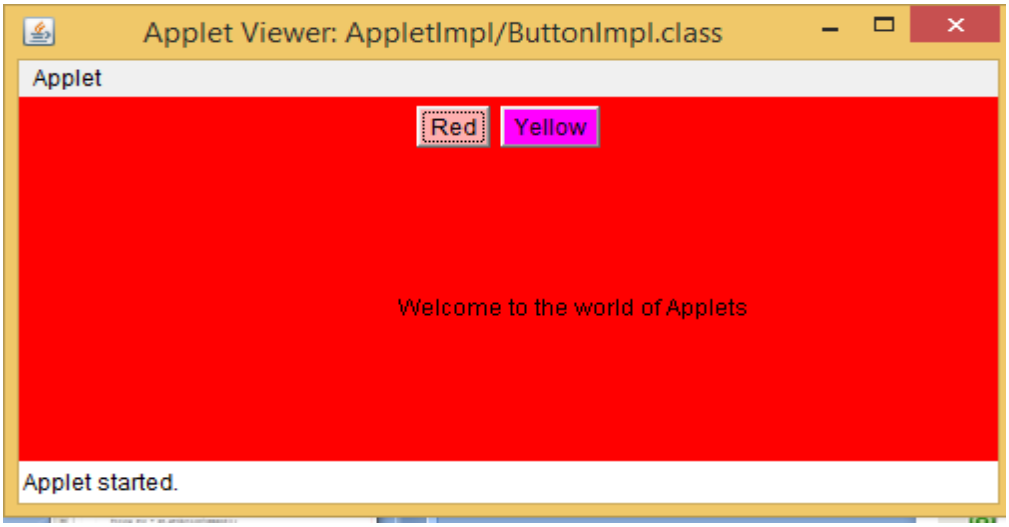
Event Interfaces:

Interface	Description
ActionListener	Defines one method to receive action events.
AdjustmentListener	Defines one method to receive adjustment events.
ComponentListener	Defines four methods to recognize when a component is hidden, moved, resized, or shown.
ContainerListener	Defines two methods to recognize when a component is added to or removed from a container.
FocusListener	Defines two methods to recognize when a component gains or loses keyboard focus.
ItemListener	Defines one method to recognize when the state of an item changes.
KeyListener	Defines three methods to recognize when a key is pressed, released, or typed.
MouseListener	Defines five methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released.
MouseMotionListener	Defines two methods to recognize when the mouse is dragged or moved.
MouseWheelListener	Defines one method to recognize when the mouse wheel is moved.
TextListener	Defines one method to recognize when a text value changes.
WindowFocusListener	Defines two methods to recognize when a window gains or loses input focus.
WindowListener	Defines seven methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Program:

```java
package AppletImpl;
import java.applet.Applet;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 *
 * @author sana mateen
 */
public class ButtonImpl extends Applet implements ActionListener{
Button b1,b2;
    /**
     * Initialization method that will be called after the applet is loaded into
     * the browser.
     */
@Override
    public void init() {
        b1=new Button("Red");
        b2=new Button("Yellow");
        b1.setBackground(Color.pink);
        b2.setBackground(Color.MAGENTA);
        add(b1);
        add(b2);
        b1.addActionListener(this);
        b2.addActionListener(this);
    }
@Override
    public void actionPerformed(ActionEvent ae){
        String str = ae.getActionCommand();
        if (str.equals("Red"))
            setBackground(Color.red);
        if(str.equals("Yellow"))
            setBackground(Color.yellow);

        repaint();
    }
  @Override
      public void paint(Graphics g) {
   g.drawString("Welcome to the world of Applets", 190, 120);
   }


  }
```

Output:



14.Explain AdjustmentEvent class and its usage along with an example program?

An AdjustmentEvent is generated by a scroll bar. There are five types of adjustment events.

The AdjustmentEvent class defines integer constants that can be used to identify them.

The constants and their meanings are shown here:

BLOCK_DECREMENT The user clicked inside the scroll bar to decrease its value.

BLOCK_INCREMENT The user clicked inside the scroll bar to increase its value.

TRACK The slider was dragged.

UNIT_DECREMENT The button at the end of the scroll bar was clicked to decrease its value.

UNIT_INCREMENT The button at the end of the scroll bar was clicked to increase its value.

In addition, there is an integer constant, **ADJUSTMENT_VALUE_CHANGED, that indicates that a change has occurred.**

Here is one AdjustmentEvent constructor:

AdjustmentEvent(Adjustable src, int id, int type, int val)

Here, src is a reference to the object that generated this event.

The id specifies the event.

The type of the adjustment is specified by type, and its associated value is val.

The getAdjustable( ) method returns the object that generated the event. Its form is shown here:

Adjustable getAdjustable( )

The type of the adjustment event may be obtained by the **getAdjustmentType( ) method.**

It returns one of the constants defined by AdjustmentEvent.

The general form is shown here:

int getAdjustmentType( )

The amount of the adjustment can be obtained from the getValue( ) method, shown here:

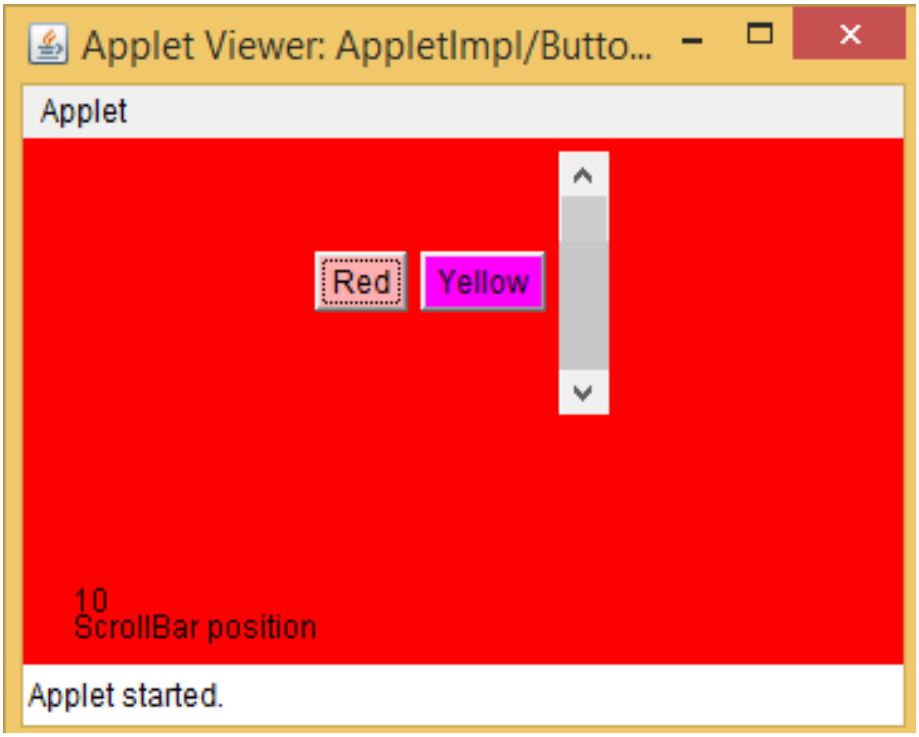
int getValue( ) For example, when a scroll bar is manipulated, this method returns the value represented by that change.

Program:

```
1   package AppletImpl;
2
3   import java.applet.Applet;
4   import java.awt.*;
5   import java.awt.event.ActionEvent;
6   import java.awt.event.ActionListener;
7   import java.awt.event.AdjustmentEvent;
8   import java.awt.event.AdjustmentListener;
9
10  /**
11   *
12   * @author sana mateen
13   */
14  public class ButtonImpl extends Applet implements ActionListener, AdjustmentListener {
15
16      Button b1, b2;
17      Scrollbar sl;
18      String msg = "";
19
20      @Override
22      public void init() {
23          b1 = new Button("Red");
24          b2 = new Button("Yellow");
25          b1.setBackground(Color.pink);
26          b2.setBackground(Color.MAGENTA);
27          b1.setSize(400, 700);
28          add(b1);
29          add(b2);
30          sl = new Scrollbar(Scrollbar.VERTICAL, 0, 10, 10, 100);
31          sl.setBounds(100, 100, 50, 100);
32          sl.setPreferredSize(new Dimension(20, 100));
            add(sl);
```

```
33              b1.addActionListener(this);
34              b2.addActionListener(this);
35              sl.addAdjustmentListener(this);
36
37              sl.setVisible(true);
38          }
39          @Override
41          public void actionPerformed(ActionEvent ae) {
42              String str = ae.getActionCommand();
43              if (str.equals("Red")) {
44                  setBackground(Color.red);
45              }
46              if (str.equals("Yellow")) {
47                  setBackground(Color.yellow);
48              }
49
50              repaint();
51          }
53          @Override
54          public void adjustmentValueChanged(AdjustmentEvent ae) {
55              repaint();
57          }
58          @Override
59          public void paint(Graphics g) {
60
61              g.drawString("ScrollBar position", 20, 190);
62              msg += sl.getValue();
63              g.drawString(msg, 20, 180);
64              msg = "";
          }

      }
```

Output:



15.Write a java program to implement Mouse Events?

Program:

```
1   package AppletImpl;
2
3   // Demonstrate the mouse event handlers.
4   import java.awt.*;
5   import java.awt.event.*;
6   import java.applet.*;
7   /*
8    <applet code="MouseEvents" width=300 height=100>
9    </applet>
10  */
11  public class MouseEvents extends Applet
12   implements MouseListener, MouseMotionListener {
13   String msg = "";
14   int mouseX = 0, mouseY = 0; // coordinates of mouse
15   @Override
16   public void init() {
17   addMouseListener(this);
18   addMouseMotionListener(this);
19   }
20   // Handle mouse clicked.
21   @Override
22   public void mouseClicked(MouseEvent me) {
23   // save coordinates
24   mouseX = 0;
25   mouseY = 10;
26   msg = "Mouse clicked.";
27   repaint();
28   }
29   // Handle mouse entered.
30   @Override
31   public void mouseEntered(MouseEvent me) {
```

```
32   // save coordinates
33   mouseX = 0;
34   mouseY = 10;
35   msg = "Mouse entered.";
36   repaint();
37   }
38   // Handle mouse exited.
39   @Override
40   public void mouseExited(MouseEvent me) {
41   // save coordinates
42   mouseX = 0;
43   mouseY = 10;
44   msg = "Mouse exited.";
45   repaint();
46   }
47   // Handle button pressed.
48   @Override
49   public void mousePressed(MouseEvent me) {
50   // save coordinates
51   mouseX = me.getX();
52   mouseY = me.getY();
53   msg = "Down";
54   repaint();
55   }
56   // Handle button released.
57   @Override
58   public void mouseReleased(MouseEvent me) {
59   // save coordinates
60   mouseX = me.getX();
61   mouseY = me.getY();
62   msg = "Up";
```

```
63      repaint();
64      }
65      // Handle mouse dragged.
66      @Override
        public void mouseDragged(MouseEvent me) {
68      // save coordinates
69      mouseX = me.getX();
70      mouseY = me.getY();
71      msg = "*";
72      showStatus("Dragging mouse at " + mouseX + ", " + mouseY);
73      repaint();
74      }
75      // Handle mouse moved.
76      @Override
        public void mouseMoved(MouseEvent me) {
78      // show status
79      showStatus("Moving mouse at " + me.getX() + ", " + me.getY());
80      }
81      // Display msg in applet window at current X,Y location.
82      @Override
        public void paint(Graphics g) {
84      g.drawString(msg, mouseX, mouseY);
85      }
86  }
```

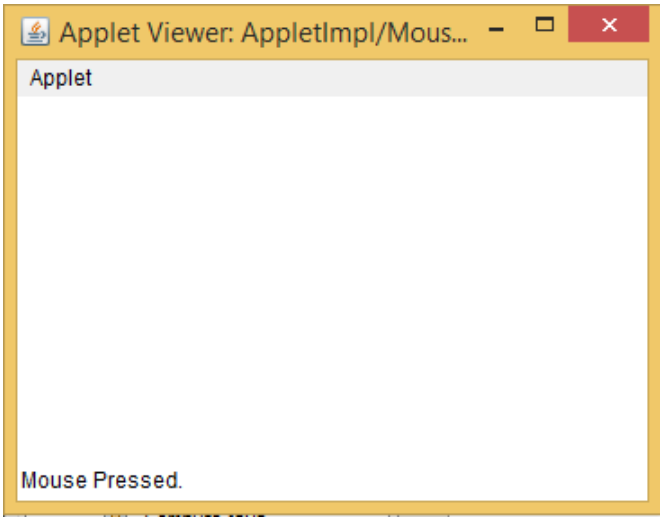
16.Write a java program to implement Adapter Class?

Program:

```
package AppletImpl;
// Inner class demo.
import java.applet.*;
import java.awt.event.*;
/*
 <applet code="InnerClassDemo" width=200 height=100>
 </applet>
*/
public class InnerClassDemo extends Applet {
 @Override
 public void init() {
 addMouseListener(new MyMouseAdapter());
 }
 class MyMouseAdapter extends MouseAdapter {
 @Override
 public void mousePressed(MouseEvent me) {
 showStatus("Mouse Pressed");
 }
 }
}
```

```
package AppletImpl;

import java.applet.*;
import java.awt.event.*;
/*
 <applet code="MousePressedDemo" width=200 height=100>
 </applet>
*/
public class MousePressedDemo extends Applet {
 @Override
 public void init() {
 addMouseListener(new MyMouseAdapt(this));
 }
}
class MyMouseAdapt extends MouseAdapter {
 MousePressedDemo mousePressedDemo;
 public MyMouseAdapt(MousePressedDemo mousePressedDemo) {
 this.mousePressedDemo = mousePressedDemo;
 }
 @Override
 public void mousePressed(MouseEvent me) {
 mousePressedDemo.showStatus("Mouse Pressed.");
 }
}
```

Output:



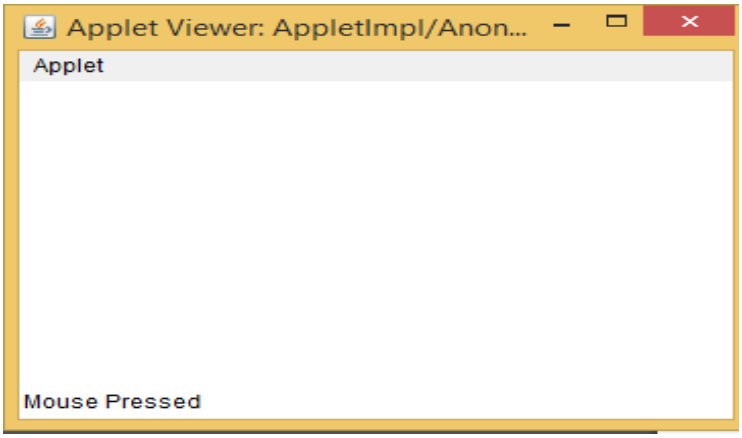
17.Write a java program to implement Anonymous Inner class?

An anonymous inner class is one that is not assigned a name. This section illustrates **how an anonymous inner class can facilitate the writing of event handlers**. Consider the applet shown in the following listing. As before, **its goal is to display the string "Mouse Pressed" in the status bar of the applet viewer or browser when the mouse is pressed.**

Program:

```
1   package AppletImpl;
2   // Anonymous inner class demo.
3   import java.applet.*;
4   import java.awt.event.*;
5   /*
6    <applet code="AnonymousInnerClassDemo" width=200 height=100>
7    </applet>
8   */
9   public class AnonymousInnerClass extends Applet {
10   @Override
11   public void init() {
12   addMouseListener(new MouseAdapter() {
13   @Override
14   public void mousePressed(MouseEvent me) {
15   showStatus("Mouse Pressed");
16   }
17   });
18   }
19  }
20
```

Output:



1. There is one top-level class in this program: AnonymousInnerClassDemo.
2. The init( ) method calls the addMouseListener( ) method.
3. Its argument is an expression that defines and instantiates an anonymous inner class.

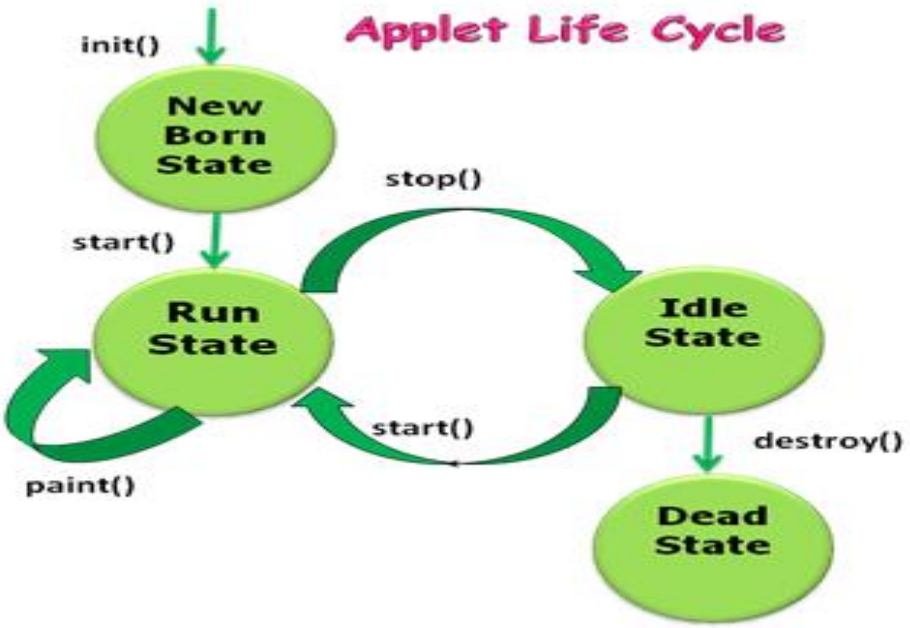
Let's analyze this expression carefully.

1. The syntax new MouseAdapter(){...} indicates to the compiler that the code between the braces defines an anonymous inner class.
2. Furthermore, that class extends MouseAdapter. This new class is not named, but it is automatically instantiated when this expression is executed.
3. Because this anonymous inner class is defined within the scope of AnonymousInnerClassDemo, it has access to all of the variables and methods within the scope of that class.

Therefore, it can call the showStatus( ) method directly.

18.Explain the life cycle methods of Applet with an example program?

- Four methods in the Applet class gives you the framework on which you build any serious applet −
- It is important to understand the order in which the various methods shown in the skeleton are called.
- When an applet begins, the following methods are called, in this sequence: 1. init( ) 2. start( ) 3. paint( )
- When an applet is terminated, the following sequence of method calls takes place: 1. stop( ) 2. destroy( )



Init()

The init( ) method is the first method to be called.

This is where you should initialize variables.

This method is called only once during the run time of your applet.

```
public void init() {
        // initialization
        }
```

Start()

The start( ) method is called after init( ).

It is also called to restart an applet after it has been stopped.

Whereas init( ) is called once—the first time an applet is loaded—

start( ) is called each time an applet's HTML document is displayed onscreen.

So, if a user leaves a web page and comes back, the applet resumes execution at start( ).

```
public void start() {
// start or resume execution
 }
```

Paint()

The paint( ) method is called each time an AWT-based applet's output must be redrawn.

The applet window may be minimized and then restored. paint( ) is also called when the applet begins execution.

Whatever the cause, whenever the applet must redraw its output, paint( ) is called.

The paint( ) method has one parameter of type Graphics.

This parameter will contain the graphics context, which describes the graphics environment in which the applet is running.

This context is used whenever output to the applet is required

public void paint(Graphics g) {

// redisplay contents of window

}

Stop()

The stop( ) method is called when a web browser leaves the HTML document containing the applet—when it goes to another page, for example.

When stop( ) is called, the applet is probably running. You should use stop( ) to suspend threads that don't need to run when the applet is not visible.

You can restart them when start( ) is called if the user returns to the page.

public void stop() {

// suspends execution

}

Destroy()

- The destroy( ) method is called when the environment determines that your applet needs to be removed completely from memory.
- A**t this point, you should free up any resources the applet may be using.**
- The stop( ) method is always called before destroy( ).
- public void destroy() {
- // perform shutdown activities
- }

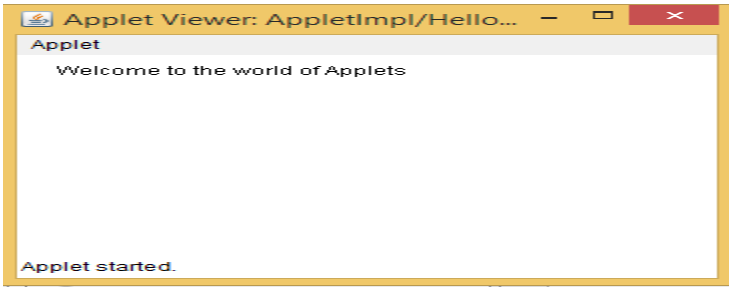
Structure:

```
// An Applet skeleton.
import java.awt.*;
import java.applet.*;
/*
<applet code="AppletSkel" width=300 height=100>
</applet>
*/

public class AppletSkel extends Applet {
  // Called first.
  public void init() {
    // initialization
  }

  /* Called second, after init().  Also called whenever
     the applet is restarted. */
  public void start() {
    // start or resume execution
  }

  // Called when the applet is stopped.
  public void stop() {
    // suspends execution
  }

  /* Called when applet is terminated.  This is the last
     method executed. */
  public void destroy() {
    // perform shutdown activities
  }

  // Called when an applet's window must be restored.
  public void paint(Graphics g) {
    // redisplay contents of window
  }
}
```

Example Program:

```
6   package AppletImpl;
7
8   import java.applet.Applet;
9   import java.awt.Graphics;
10
11  /**
12   *
13   * @author sana mateen
14   */
15  public class Hello extends Applet {
16   @Override
17   public void paint(Graphics g) {
18   g.drawString("Welcome to the world of Applets", 20, 20);
19   }
20  }
21
```

Output:



19.Write a java program to display an image using ImageIcon class?

Program:

```
1
2   package swingsexamples;
3   // Demonstrate JLabel and ImageIcon.
4   import java.lang.reflect.InvocationTargetException;
5   import javax.swing.*;
6   /*
7    <applet code="JLabelDemo" width=250 height=200>
8    </applet>
9    */
10  public class JLabelDemo extends JApplet {
11      @Override
12          public void init() {
13           try {
14               SwingUtilities.invokeAndWait(
15                       new Runnable() {
16                   @Override
17                   public void run() {
18                       makeGUI();
19                   }
20               }
21               );
22           } catch (InterruptedException | InvocationTargetException exc) {
23               System.out.println("Can't create because of " + exc);
24           }
25      }
26      private void makeGUI() {
27          // Create an icon.
28          ImageIcon ii = new ImageIcon("imgs/photo.jpg");
29          // Create a label.
30          JLabel jl = new JLabel("Sana Mateen", ii, JLabel.CENTER);
31          // Add the label to the content pane.
32              add(jl);
33          }
34      }
```

Output

:

20.Explain JScrollPane along with an example program?

A JscrollPane is used to make scrollable view of a component. When screen size is limited, we use a scroll pane to display a large component or a component whose size can change dynamically.

Constructor:

JScrollPane() It creates a scroll pane. The Component parameter, when present, sets the scroll pane's client. The two int parameters, when present, set the vertical and horizontal scroll bar policies (respectively).

Methods:

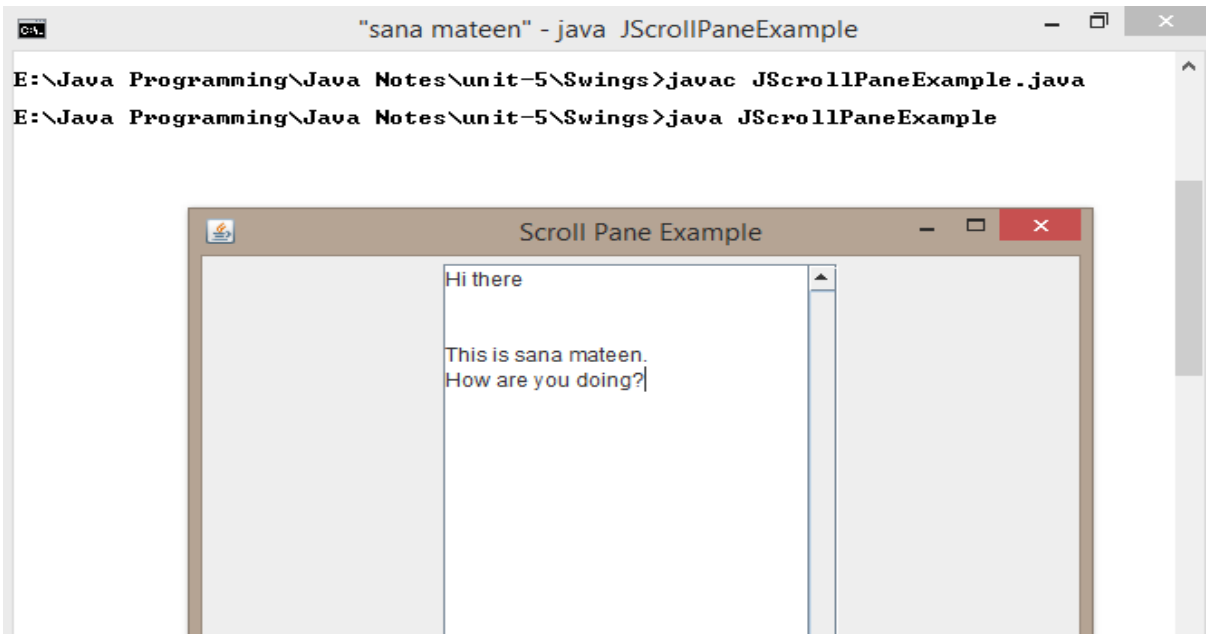
void setColumnHeaderView(Component) It sets the column header for the scroll pane.

void setRowHeaderView(Component) It sets the row header for the scroll pane.

Program:

```java
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
 /*

 @SANA MATEEN

 */
public class JScrollPaneExample {
    private static final long serialVersionUID = 1L;

    private static void createAndShowGUI() {

        // Create and set up the window.
        final JFrame frame = new JFrame("Scroll Pane Example");

        // Display the window.
        frame.setSize(500, 500);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // set flow layout for the frame
        frame.getContentPane().setLayout(new FlowLayout());

        JTextArea textArea = new JTextArea(20, 20);
        JScrollPane scrollableTextArea = new JScrollPane(textArea);

        scrollableTextArea.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
        scrollableTextArea.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);

        frame.getContentPane().add(scrollableTextArea);
    }

    public static void main(String[] args) {

        javax.swing.SwingUtilities.invokeLater(new Runnable() {

            public void run() {
                createAndShowGUI();
            }
        });
    }
}
```

Output:



21.Explain JTabbedPane along with an example program?

- The JTabbedPane class is used to switch between a group of components by clicking on a tab with a given title or icon. It inherits JComponent class.
- JTabbedPane class declaration
- Let's see the declaration for javax.swing.JTabbedPane class.
- **public class** JTabbedPane **extends** JComponent **implements** Serializable, Accessible, SwingConstants

Constructor	Description
JTabbedPane()	**Creates an empty TabbedPane with a default tab placement of JTabbedPane.Top.**
JTabbedPane(int tabPlacement)	**Creates an empty TabbedPane with a specified tab placement.**
JTabbedPane(int tabPlacement, int tabLayoutPolicy)	**Creates an empty TabbedPane with a specified tab placement and tab layout policy.**

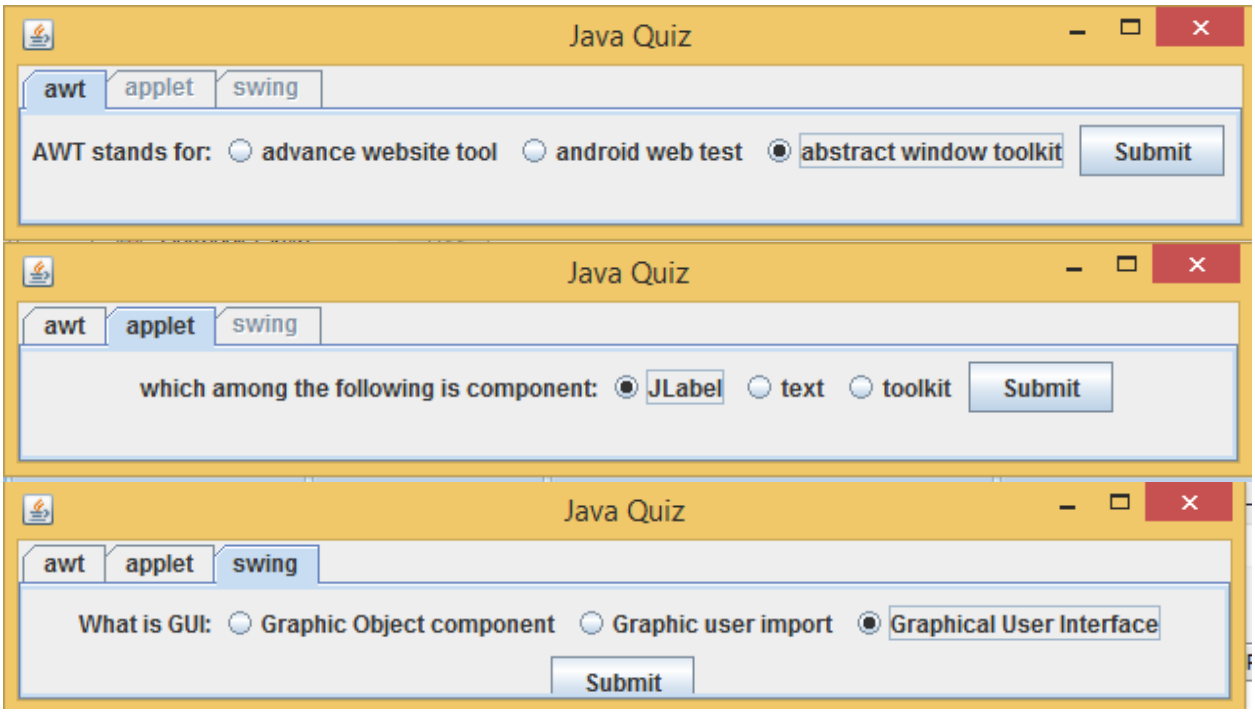
Program:

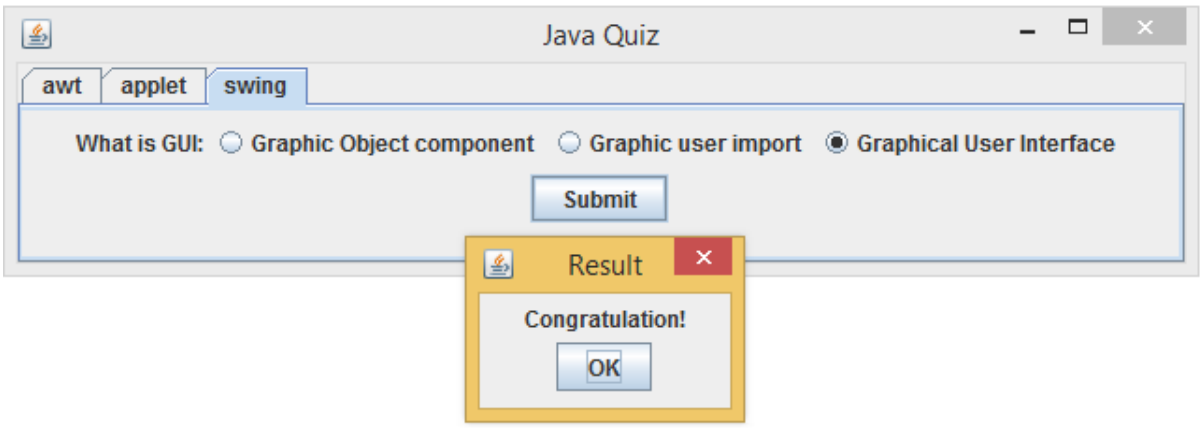
```
import java.awt.Container;//Container Holds other components

import java.awt.FlowLayout;/*FlowLayout implements a simple layout style, which is similar to
how words flow in a text editor. */

import java.awt.event.ActionEvent;/*An ActionEvent is generated when a button is pressed, a list item is
double-clicked, or
a menu item is selected. */
import java.awt.event.ActionListener;/*This interface defines the actionPerformed( ) method that is in
voked when an action event occurs*/

import javax.swing.ButtonGroup;/*Button groups ( javax.swing.ButtonGroup ) are used in combination with
radio buttons to ensure that
only one radio button in a group of radio buttons is selected.*/
import javax.swing.JButton;/*The JButton class is used to create a labeled button that
has platform independent implementation. */

import javax.swing.JDialog;/*The JDialog control represents a top level window with a border and a title used to
take some form of input from the user. */
import javax.swing.JFrame;/*JFrame works like the main window where components like labels, buttons,
textfields are added to create a GUI.*/

import javax.swing.JLabel;//The class JLabel can display either text, an image, or both
import javax.swing.JPanel;/*The JPanel is a simplest container class. It provides space in which an application
can attach any other component. */
import javax.swing.JRadioButton;/*The JRadioButton class is used to create a radio button.
It is used to choose one option from multiple options. */
import javax.swing.JTabbedPane;/*The JTabbedPane class is used to switch between a group of components by
clicking on a tab with a given title or icon. It inherits JComponent class.*/


public class Quiz extends JFrame {

    JTabbedPane tp;
    JPanel panel1, panel11, panel2, panel3;
    ButtonGroup g1;
    JRadioButton rb1a, rb1b, rb1c, rb1d, rb1e, rb1f;
    ButtonGroup g2;
    JRadioButton rb2a, rb2b, rb2c;
    ButtonGroup g3;
    JRadioButton rb3a, rb3b, rb3c;
    JButton b1, b1a, b2, b3;

    public static void main(String[] args) {
        Quiz qz = new Quiz();
    }

    public Quiz() {

        super("Java Quiz");//calling super class constructor JFrame(String title)
        setSize(640, 120);//set size of frame
        panel1 = new JPanel();//creating panel object
        panel11 = new JPanel();
        panel2 = new JPanel();
        panel3 = new JPanel();
        tp = new JTabbedPane();
        JLabel l1 = new JLabel("AWT stands for:");
        g1 = new ButtonGroup();
        rb1a = new JRadioButton("advance website tool", false);
        rb1b = new JRadioButton("android web test", false);
        rb1c = new JRadioButton("abstract window toolkit", false);
        g1.add(rb1a);
```

```
g1.add(rb1b);
g1.add(rb1c);
panel1.add(l1);
panel1.add(rb1a);
panel1.add(rb1b);
panel1.add(rb1c);
b1 = new JButton("Submit");
panel1.add(b1);
tp.addTab("awt", panel1);
JLabel l2 = new JLabel("which among the following is component:");
g2 = new ButtonGroup();
rb1d = new JRadioButton("JLabel", false);
rb1e = new JRadioButton("text", false);
rb1f = new JRadioButton("toolkit", false);
g2.add(rb1d);
g2.add(rb1e);
g2.add(rb1f);
panel2.add(l2);
panel2.add(rb1d);
panel2.add(rb1e);
panel2.add(rb1f);
b1a = new JButton("Submit");
panel2.add(b1a);
tp.addTab("applet", panel2);
panel3 = new JPanel();
JLabel l3 = new JLabel("What is GUI:");
g3 = new ButtonGroup();
rb2a = new JRadioButton("Graphic Object component", false);
rb2b = new JRadioButton("Graphic user import", false);
rb2c = new JRadioButton("Graphical User Interface", false);
g3.add(rb2a);
g3.add(rb2b);
    g3.add(rb2c);
    panel3.add(l3);
    panel3.add(rb2a);
    panel3.add(rb2b);
    panel3.add(rb2c);
    b2 = new JButton("Submit");
    panel3.add(b2);

    tp.addTab("swing", panel3);
    tp.setEnabledAt(0, true);//Sets whether or not the tab at index is enabled.
    tp.setEnabledAt(1, false);
    tp.setEnabledAt(2, false);
    add(tp);
    tp.setSize(600, 100);

    ButtonHand handler = new ButtonHand();
    b1.addActionListener(handler);
    b1a.addActionListener(handler);
    b2.addActionListener(handler);
    getContentPane().add(tp);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true);
}

class ButtonHand implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent ae) {
        if (ae.getSource() == b1 && rb1c.isSelected()) {
            tp.setEnabledAt(1, true);
            tp.setSelectedIndex(1);
        }
        if (ae.getSource() == b1a && rb1d.isSelected()) {
```

```
                tp.setEnabledAt(2, true);
                 tp.setSelectedIndex(2);
            }
            if (ae.getSource() == b2 && rb2c.isSelected()) {
                CongratDialog cd = new CongratDialog(null);
                cd.setVisible(true);
            }

        }
    }

    class CongratDialog extends JDialog {

        public CongratDialog(JFrame parent) {
            super(parent, "Result", true);
            Container cp = getContentPane();/*The getContentPane() method retrieves the content pane layer
            so that you can add an object to it.*/
            cp.setLayout(new FlowLayout());
            cp.add(new JLabel("Congratulation!"));
            JButton ok = new JButton("OK");
            ok.addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e) {
                    dispose();
                }
            });
            cp.add(ok);
            setSize(150, 100);
        }
    }
}
```

Output:





22.Discuss in detail about MenuBars,Menus and Dialog boxes?

A menu is a group of commands located in a menubar. A toolbar has buttons with some common commands in the application.

Use the following components:

JMenuBar — implements a menu bar.

JMenu — implements a menu, a popup window containing JMenuItems that is displayed when the user selects an item on the JMenuBar.

JMenuItem — implements an item in a menu. It is selected by the user to perform an action.

JSeparator — provides a general purpose component for implementing divider lines.

JCheckBoxMenuItem -— implements a menu that can be selected or deselected.

JRadioButtonMenuItem -— implements a radio button menu item, used for mutually exclusive selection.

ButtonGroup — creates a multiple-exclusion scope for a set of buttons.

JPopupMenu — implements a popup menu, a small window that pops up and displays a series of

choices.

JToolBar — implements a toolbar, a component that is useful for displaying commonly used Action

or controls

Program:

E:\Java Programming\Java Notes\unit-5\Swin

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

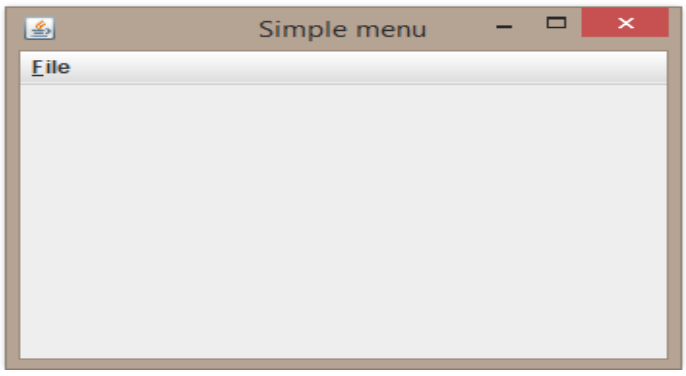
MenuBarEx.java | animatee.py | game1.py | game2.py | game3.py | change.log

```
2  import java.awt.event.KeyEvent;
3  import javax.swing.ImageIcon;
4  import javax.swing.JFrame;
5  import javax.swing.JMenu;
6  import javax.swing.JMenuBar;
7  import javax.swing.JMenuItem;
8  public class MenuBarEx extends JFrame {
9      public MenuBarEx() {
10         initUI();
11     }
12     private void initUI() {
13         createMenuBar();
14         setTitle("Simple menu");
15         setSize(350, 250);
16         setLocationRelativeTo(null);
17         setDefaultCloseOperation(EXIT_ON_CLOSE);
18     }
19     private void createMenuBar() {
20         var menuBar = new JMenuBar();
21         var exitIcon = new ImageIcon("src/resources/exit.png");
22         var fileMenu = new JMenu("File");
23         fileMenu.setMnemonic(KeyEvent.VK_F);
24         var eMenuItem = new JMenuItem("Exit", exitIcon);
25         eMenuItem.setMnemonic(KeyEvent.VK_E);
26         eMenuItem.setToolTipText("Exit application");
27         eMenuItem.addActionListener((event) -> System.exit(0));
28         fileMenu.add(eMenuItem);
29         menuBar.add(fileMenu);
30         setJMenuBar(menuBar);
31     }
32     public static void main(String[] args) {
33         EventQueue.invokeLater(() -> {
34         var ex = new MenuBarEx();
35         ex.setVisible(true);
36         });
37     }
38 }
```

Output:

"sana mateen" - java MenuBarEx

```
E:\Java Programming\Java Notes\unit-5\Swings>javac MenuBarEx.java
E:\Java Programming\Java Notes\unit-5\Swings>java MenuBarEx
E:\Java Programming\Java Notes\unit-5\Swings>java MenuBarEx
E:\Java Programming\Java Notes\unit-5\Swings>java MenuBarEx
```



23.Explain about JAPPLET in detail?

A Swing applet extends JApplet rather than Applet.

JApplet is derived from Applet.

Thus, JApplet includes all of the functionality found in Applet and adds support for Swing.

JApplet is a top-level Swing container, which means that it is not derived from JComponent.

Because JApplet is a toplevel container, it includes the various panes described earlier.

This means that all components are added to JApplet's content pane in the same way that components are added to JFrame's content pane. Swing applets use the same four life-cycle methods as init( ), start( ), stop( ), and destroy( ).

Of course, you need override only those methods that are needed by your applet. Painting is accomplished differently in Swing than it is in the AWT, and **a Swing applet will not normally override the paint( ) method**.

One other point: **All interaction with components in a Swing applet must take place on the event dispatching thread**

Swing event handling code runs on a special thread known as the event dispatch thread. Most code that invokes Swing methods also runs on this thread.

This is necessary because most Swing object methods are not "thread safe": invoking them from multiple threads risks thread interference or memory consistency errors.

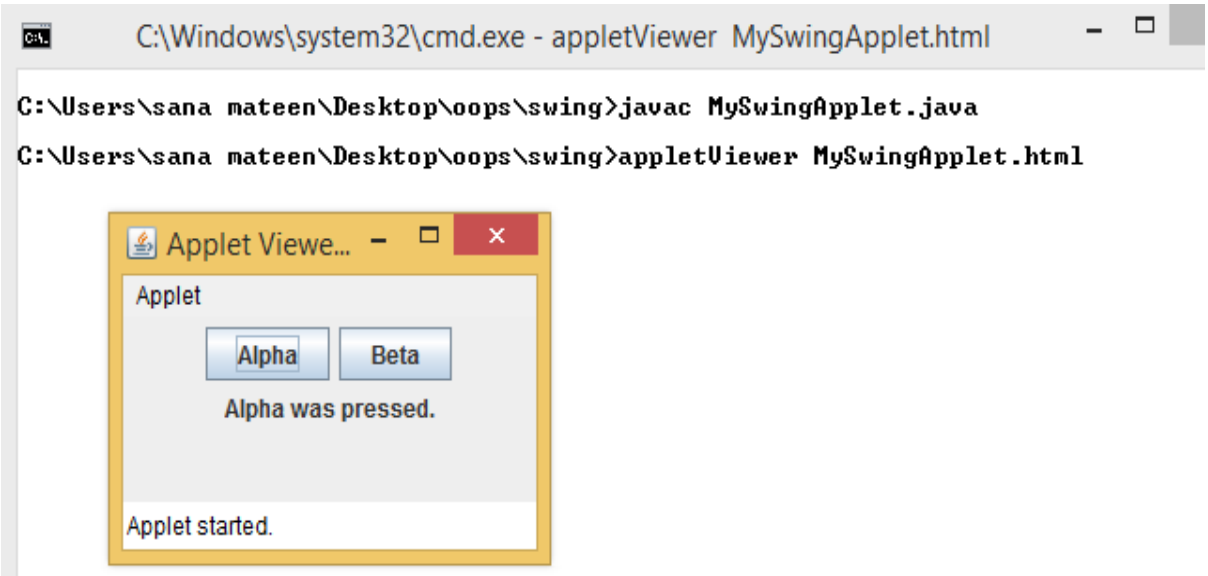
Some Swing component methods are labelled "thread safe" in the API specification; these can be safely invoked from any thread.

All other Swing component methods must be invoked from the event dispatch thread. Programs that ignore this rule may function correctly most of the time, but are subject to unpredictable errors that are difficult to reproduce.

Program:

```
1       import javax.swing.*;
2       import java.awt.*;
3       import java.awt.event.*;
4       /*
5       This HTML can be used to launch the applet:
6       <applet code="MySwingApplet" width=220 height=90>
7       </applet>
8       */
9       public class MySwingApplet extends JApplet {
10           JButton jbtnAlpha;
11           JButton jbtnBeta;
12           JLabel jlab;
13           // Initialize the applet.
14           public void init() {
15               try {
16                   SwingUtilities.invokeAndWait(new Runnable () {
17                       public void run() {
18                       makeGUI(); // initialize the GUI
19                       }
20                   });
21                  } catch(Exception exc) {
22                   System.out.println("Can't create because of "+ exc);
23                   }
24              }
25            // This applet does not need to override start(), stop(),
26           // or destroy().
27           // Set up and initialize the GUI.
28           private void makeGUI() {
29               // Set the applet to use flow layout.
30               setLayout(new FlowLayout());
31               // Make two buttons.
32               jbtnAlpha = new JButton("Alpha");
33               jbtnBeta = new JButton("Beta");
34               // Add action listener for Alpha.
35               jbtnAlpha.addActionListener(new ActionListener() {
36                   public void actionPerformed(ActionEvent le) {
37                       jlab.setText("Alpha was pressed.");
38                   }
39               });
40               // Add action listener for Beta.
41               jbtnBeta.addActionListener(new ActionListener() {
42                   public void actionPerformed(ActionEvent le) {
43                       jlab.setText("Beta was pressed.");
44                   }
45               });
46               // Add the buttons to the content pane.
47               add(jbtnAlpha);
48               add(jbtnBeta);
49               // Create a text-based label.
50               jlab = new JLabel("Press a button.");
51               // Add the label to the content pane.
52               add(jlab);
53           }
54       }
```

Output:



24.Explain the steps involved increating JCheckBox,JLabel,JRadioButton,JButton?

1.import java.awt, javax.swing, java.awt.event, javax.swing.event.

2.create GUI class for top-level container, i.e. usually extends JFrame

3.declare private variable of 2nd-level container, i.e. usually JPanel p

4.declare private variables of components, i.e. JButton, JCheckBox, JRadioButton, JTextField, JLabel, JSlider, JComboBox, etc.

5.construct the GUI class by:

◦assign 2nd-level container variable by referencing top-level container's content panel. i.e. p = (JPanel)this.getContentPane();

◦setLayout of 2nd-level container, i.e. p.setLayout(null);

◦create instances of all component variables declared above

◦set all necessary properties of components, i.e. setBounds(....), setValue(....), etc.

◦add action/event listener object, if any (use either anonymous inner class or named inner class)

◦add all component instance variables into 2nd-level container panel, i.e. p.add(..)

(these steps 6-8 can be same with GUI class or at separate driver class)

6.create main method

7.create instance of GUI class

8.setTitle("..."), setSize(..), setVisible(true), setResizable(false), of main frame or container

(these steps 9-10 can be same with GUI class or at separate public class)

9.create named inner class by implementing event listener interface

10.implement all necessary action and event handling methods

```
//Frame1.java
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class Frame1 extends JFrame{
	private JPanel p;
	private JButton button;
	public Frame1()  {
	p = (JPanel)this.getContentPane();
	p.setLayout(null);
	button = new JButton("Click me");
	button.setBounds(30,30,120,30);
	button.addActionListener(new WidgetListener());
	p.add(button);
	}
	public static void main(String []args)  {
		Frame1 f1 = new Frame1();
		f1.setTitle("GUI Demo");
		f1.setSize(200,150);
		f1.setVisible(true);
		f1.setResizable(false);
	}
	private class WidgetListener implements ActionListener  {
		public void actionPerformed(ActionEvent e)    {
			button.setText("Thank you");
		}
	}
}
```

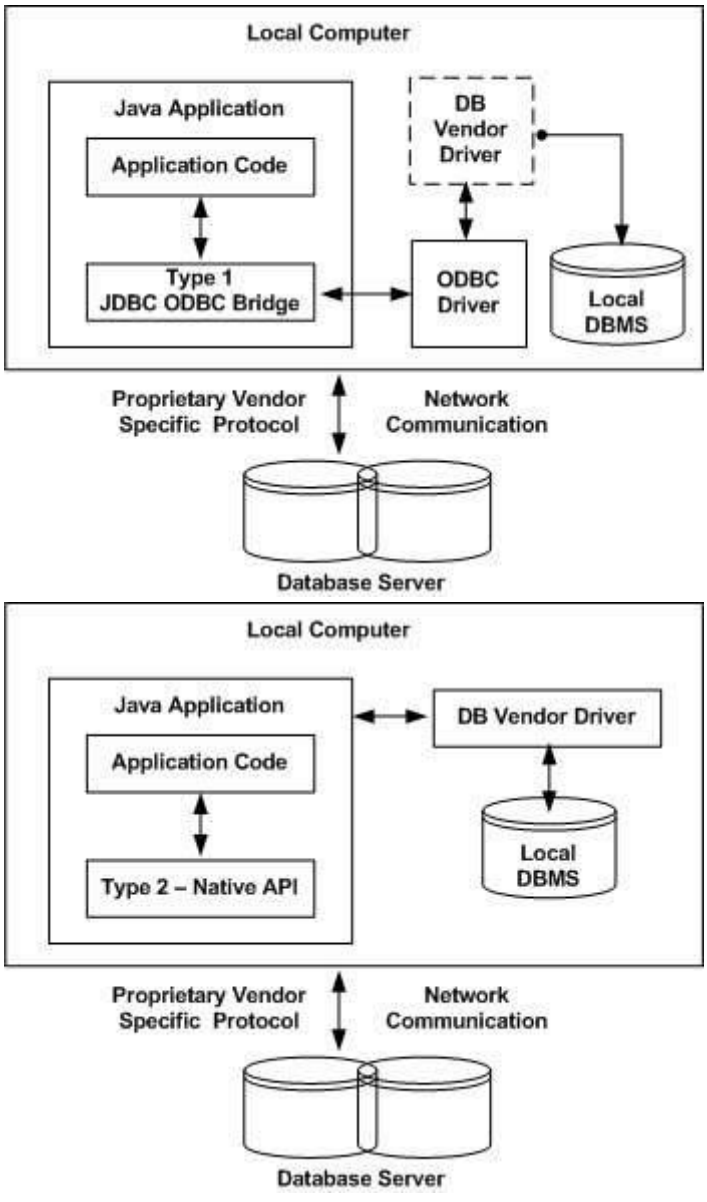
Beyond Syllabus Topics with materials:

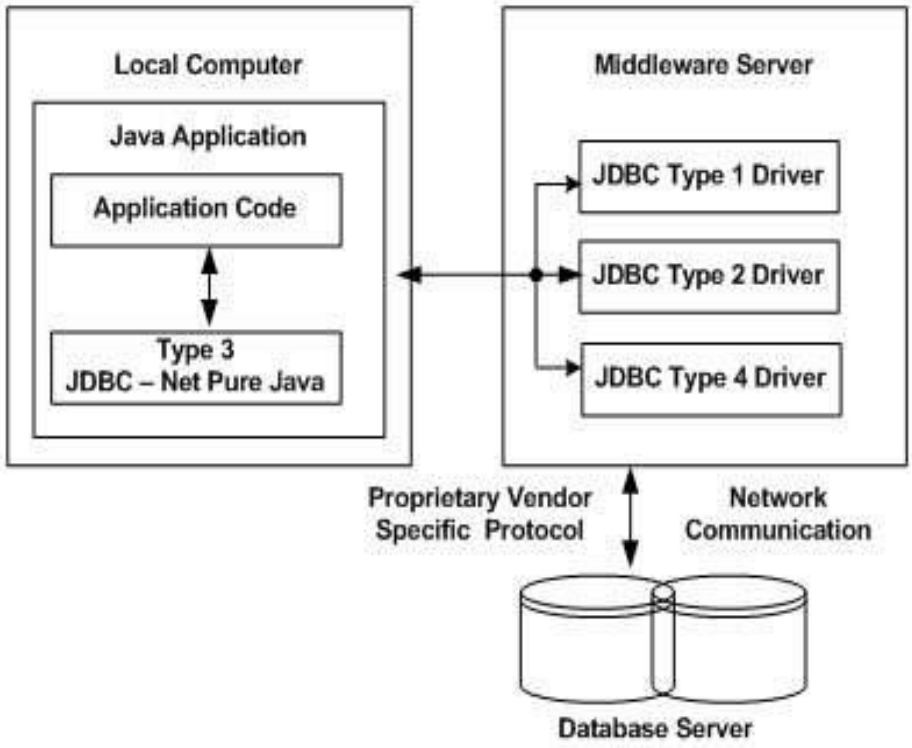
What is JDBC Driver?

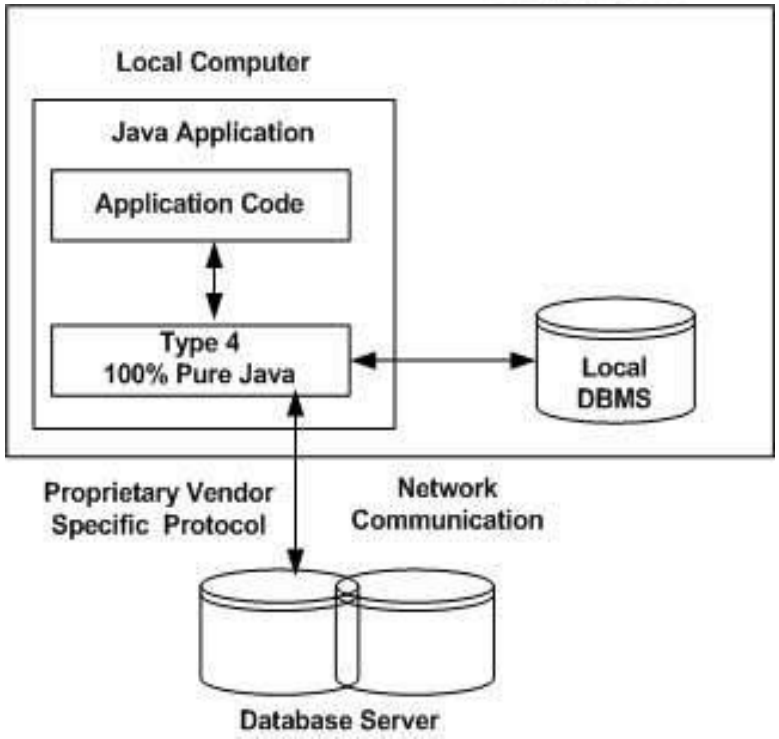
JDBC drivers implement the defined interfaces in the JDBC API, for interacting with your database server.

JDBC Drivers Types

JDBC driver implementations vary because of the wide variety of operating systems and hardware platforms in which Java operates. Sun has divided the implementation types into four categories, Types 1, 2, 3, and 4, which is explained below –







Which Driver should be Used?

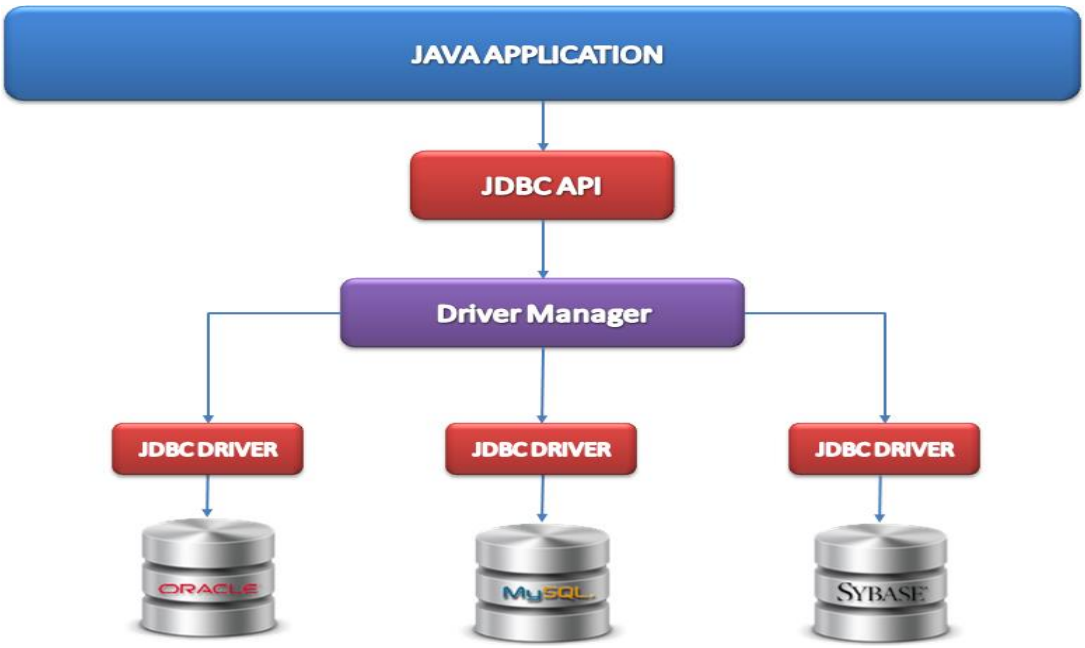
If you are accessing one type of database, such as Oracle, Sybase, or IBM, the preferred driver type is 4.

If your Java application is accessing multiple types of databases at the same time, type 3 is the preferred driver.

Type 2 drivers are useful in situations, where a type 3 or type 4 driver is not available yet for your database.

The type 1 driver is not considered a deployment-level driver, and is typically used for development and testing purposes only.

JDBC API is a Java API that can access any kind of tabular data, especially data stored in a Relational Database. JDBC works with Java on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.



The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.

Making a connection to a database.

Creating SQL or MySQL statements.

Executing SQL or MySQL queries in the database.

Viewing & Modifying the resulting records.

JDBC - Environment Setup:

Make sure you have done following setup:

Core JAVA Installation

SQL or MySQL Database Installation

Apart from the above you need to setup a database which you would use for your project.

Assuming this is EMP and you have created on table Employees within the same database.

There are six steps involved in building a JDBC application:

Import the packages:

This requires that you include the packages containing the JDBC classes needed for database programming.

Most often, using import java.sql.* will suffice as follows:

```
//STEP 1. Import required packages
 import java.sql.*;
```

Register the JDBC driver:

This requires that you initialize a driver so you can open a communications channel with the database.

```
//STEP 2: Register JDBC driver
 Class.forName("com.mysql.jdbc.Driver");
```

This requires using the DriverManager.getConnection() method to create a Connection object, which represents a physical connection with the database as follows:

```
//STEP 3: Open a connection
// Database credentials
static final String USER = "username";
static final String PASS = "password";
System.out.println("Connecting to database...");
conn = DriverManager.getConnection(DB_URL,USER,PASS);
```

This requires using an object of type Statement or PreparedStatement for building and submitting an SQL statement to the database as follows:

```
//STEP 4: Execute a query
System.out.println("Creating statement...");
stmt = conn.createStatement();
String sql;
sql = "SELECT id, first, last, age FROM Employees";
ResultSet rs = stmt.executeQuery(sql);
```

If there is an SQL UPDATE,INSERT or DELETE statement required, then following code snippet would be required:

```
//STEP 4: Execute a query
System.out.println("Creating statement...");
stmt = conn.createStatement();
String sql = "DELETE FROM Employees";
ResultSet rs = stmt.executeUpdate(sql);
```

This step is required in case you are fetching data from the database. You can use the appropriate ResultSet.getXXX() method to retrieve the data from the result set as follows:

```
//STEP 5: Extract data from result set
while(rs.next()){
//Retrieve by column name
int id = rs.getInt("id");
int age = rs.getInt("age");
String first = rs.getString("first");
String last = rs.getString("last");
System.out.print("ID: " + id);
System.out.print(", Age: " + age);
System.out.print(", First: " + first);
 System.out.println(", Last: " + last); }
```

You should explicitly close all database resources versus relying on the JVM's garbage collection as follows:

```
//STEP 6: Clean-up environment
rs.close();
stmt.close();
conn.close();
```