Object Oriented Programming Course Code: 19CS1203

Session-1:

Q1) Introduction to Object oriented Paradigm-Principles

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.

OOPs Concepts:

1. Polymorphism
2. Inheritance
3. Encapsulation
4. Abstraction
5. Class
6. Object
7. Method
8. Message Passing

Q2)What is a class?

Class is a template that is used to create objects and to define object's states and behaviour.

Example:

ClassTelevision defines general properties like width, height, screenSize, Volume.

Let object of Television class is SONYTV

State of the Object, SONYTV may have the current state like width =17, height =23, screenSize=21 and Volume=3

The methods defines what objects defined by this class does.

Class Telvision defines general behaviour or methods that object of class Television must have.

Ex: increaseVolume() method

Q3) What is an object?

Object is an instance of class and object is a real world entity

Example: Car has gear and we can change the gear by using a changeGear() method. Here gear is state and changeGear() is behaviour and this method must not change the behaviour of all cars but a specific car

Q4) Difference between Procedural Programming and Object Oriented Programming:

PROCEDURAL ORIENTED PROGRAMMING	OBJECT ORIENTED PROGRAMMING
In procedural programming, program is divided into small parts called ***functions***.	In object oriented programming, program is divided into small parts called ***objects***.
Procedural programming follows ***top down approach***.	Object oriented programming follows ***bottom up approach***.
There is no access specifier in procedural programming.	Object oriented programming have access specifiers like private, public, protected etc.
Adding new data and function is not easy.	Adding new data and function is easy.

PROCEDURAL ORIENTED PROGRAMMING	OBJECT ORIENTED PROGRAMMING
Procedural programming does not have any proper way for hiding data so it is ***less secure***.	Object oriented programming provides data hiding so it is ***more secure***.
In procedural programming, overloading is not possible.	Overloading is possible in object oriented programming.
In procedural programming, function is more important than data.	In object oriented programming, data is more important than function.
Procedural programming is based on ***unreal world***.	Object oriented programming is based on ***real world***.
Examples: C, FORTRAN, Pascal, Basic etc.	Examples: C++, Java, Python, C# etc.

Local variables − Variables defined inside methods
Instance variables − Instance variables are variables within a class but outside any method.
Class variables are variables declared within a class, outside any method, with the static keyword.

Session-2

Q1) Naming conventions for Class names, methods and data members.
Java uses PascalCase for writing names of classes, interfaces
Pascal Case is a naming convention in which the first letter of each word in a compound word is capitalized.
Class name should be a noun and interface name should be adjective.
Java uses CamelCase for writing names of mehods and variables
If the name is combined with two words, the second word will start with uppercase letter always such as actionPerformed(), firstName

Names of Packages should be in lowercase. If the name contains multiple words, it should be separated by dots (.) such as java.util, java.lang.

Names of Constants should be in uppercase letters such as RED, YELLOW.
If the name contains multiple words, it should be separated by an underscore(_) such as MAX_PRIORITY. Q2) Static variables and static methods and static block

Q2) static keyword
Static keyword can be used with class, variable, method and block. Static members belong to block instead of object. Static members can be accessed without creating object.
Non static members are separate for each instance of class.
Static Block
Static block is used for initializing the static variables.This block gets executed when the class is loaded in the memory. A class can have multiple Static blocks, which will execute in the same sequence in which they have been written into the program.

Example 1: Single static block
As you can see that both the static variables were intialized before we accessed them in the main method.

```
class JavaExample{
   static int num;
```

```
  static String mystr;
  static{
    num = 97;
    mystr = "Static keyword in Java";
  }
  public static void main(String args[])
  {
    System.out.println("Value of num: "+num);
    System.out.println("Value of mystr: "+mystr);
  }
}
```

Output:

```
Value of num: 97
Value of mystr: Static keyword in Java
```

Example 2: Multiple Static blocks

Lets see how multiple static blocks work in Java. They execute in the given order which means the first static block executes before second static block. That's the reason, values initialized by first block are overwritten by second block.

```
class JavaExample2{
  static int num;
  static String mystr;
  //First Static block
  static{
    System.out.println("Static Block 1");
    num = 68;
    mystr = "Block1";
  }
 //Second static block
 static{
    System.out.println("Static Block 2");
    num = 98;
    mystr = "Block2";
  }
 public static void main(String args[])
 {
    System.out.println("Value of num: "+num);
    System.out.println("Value of mystr: "+mystr);
  }
}
```

Output:

```
Static Block 1
Static Block 2

Value of num: 98
Value of mystr: Block2
```

Java Static Variables

A static variable is common to all the instances (or objects) of the class because it is a class level variable. In other words you can say that only a single copy of static variable is created and shared among all the instances of the class. Memory allocation for such variables only happens once when the class is loaded in the memory.

Few Important Points:

Static variables are also known as Class Variables.
Unlike non-static variables, such variables can be accessed directly in static and non-static methods.

Session-3:

Q1)Write a Cuboid class with 3 static variables length, breadth and height of type double, and a static method volume (), access them using main () method within the same class.

```
public class Cuboid
{
	public static double l=20,b=30,h=40;
	public static double volume()
	{
		return l*b*h;
	}
	public static void main(String a[])
	{
		System.out.println(Cuboid.volume());
	}
}
```

Q2) Explain the need of a class as a template (Encapsulate data and methods)
Syntax – Define a class
If I write the program as shown in Q1) I cannot reuse the cuboid class in another java program. In order to reuse the Cuboid class in another java program we need to modularize the code.
Also the Cuboid class can be written by one programmer and CuboidDemo class can be written by another programmer. This way the entire project is divided into modules.
Also as user interacts with only CuboidDemo class.

Q3)
Modularize the above Cuboid class
Write a Cuboid class with 3 static variables length, breadth and height of type double, and a static method volume (), access them using main () method within another class CuboidDemo.

Cuboid.java

```
public class Cuboid {
	static double l,b,h;
	static double volume()
	{
		return l*b*h;
	}
}
```

CuboidDemo.java

```
public class CuboidDemo
{
	public static void main(String a[])
	{
		Cuboid.l=20;
		Cuboid.b=10;
		Cuboid.h=5;
		System.out.println(Cuboid.volume());
	}

}
```

Q4) Explain the usage of access modifiers – private and public
if a variable/method is private, it can be accessed within the class
but if it has public scope, it can be accessed outside the class, outside the package.
According to OOP principles, data must be private and so length, breadth and height variables have private access specifier. But if they have private access specifier, they cannot be accessed by another class CuboidDemo. To solve this problem, we create a static method called setDimension() that will give values to length,breadth and height of cuboid.

Q5) Rewrite the Cuboid class with appropriate access specifiers

Cuboid.java

```
package p1;
public class Cuboid
{
	private static double l,b,h;
	public static void setDimensions(double len, double br, double he)
	{
		l=len;
		b=br;
		h=he;
	}
	public static double volume()
	{
		return l*b*h;
	}

}
```

CuboidDemo.java

```
package p1;
public class CuboidDemo
{
	public static void main(String a[])
	{
		Cuboid.setDimensions(10,20,30);
		System.out.println(Cuboid.volume());
	}

}
```

Session 4

Q1) Modularize the Cuboid class to a package level with appropriate access specifiers

Cuboid.java

```
package p1;
public class Cuboid
{
	private static double l,b,h;
	public static void setDimensions(double len, double br, double he)
	{
		l=len;
		b=br;
		h=he;
	}
	public static double volume()
	{
		return l*b*h;
	}

}
```

CuboidDemo.java

```
package p2;
import p1.Cuboid;
public class CuboidDemo
{
	public static void main(String a[])
	{
		Cuboid.setDimensions(10,20,30);
		System.out.println(Cuboid.volume());
	}

}
```

Q2) To the above modularized code, add a method isCube () that returns true if all dimensions are same, else returns false.

Cuboid.java

```
package p1;
public class Cuboid
{
	private static double l,b,h;
	public static void setDimensions(double len, double br, double he)
	{
		l=len;
		b=br;
		h=he;
	}
	public static double volume()
	{
		return l*b*h;
	}
	public static boolean isCube()
	{
		if((l == b) && (l == h))
			return true;
		return false;
	}

}
```

CuboidDemo.java

```
package p2;
import p1.Cuboid;
public class CuboidDemo
{
	public static void main(String a[])
	{
		Cuboid.setDimensions(10,20,30);
		System.out.println("volume ="+Cuboid.volume());
		boolean chk=Cuboid.isCube();
		if(chk == true)
			System.out.println("the given Cuboid is cube");
		else
			System.out.println("the given Cuboid is not a cube");

	}

}
```

SESSION NUMBER: 05

Q1) Create a Cuboid class with 3 instance variables length, breadth and height of type double, and a method volume (). Create 2 objects with different values and print the volume.

Cuboid1.java

```
package p1;
public class Cuboid1
{
	private  double l,b,h;
	public void setDimensions(double len, double br, double he)
	{
		l=len;
		b=br;
		h=he;
	}
	public double volume()
	{
		return l*b*h;
	}
}
```

CuboidDemo1.java

```
package p2;
import p1.Cuboid1;
public class CuboidDemo1
{
	public static void main(String a[])
	{
		Cuboid1 ob1 = new Cuboid1();
		ob1.setDimensions(5,6,7);
		System.out.println("volume of first object ="+ob1.volume());
		Cuboid1 ob2 = new Cuboid1();
		ob2.setDimensions(1,2,3);
		System.out.println("volume of second object="+ob2.volume());

	}

}
```

Q2) Differentiate between instance members and static members and rules of accessing

Q3) Java Byte code, Architectural neutrality and JVM

Q4) Predict the output of the below code:

```
class BoxDemo2 {
public static void main(String args[]) {
Box mybox1 = new Box();
mybox1.width=10;
mybox1.height=20;
Box mybox2 = mybox1;
System.out.println(mybox2.width);
```

Here mybox1 and mybox2 are both pointing to same memory and so the output is 10.

SESSION NUMBER: 06

Q1) Write a Java Program to read set of integers through command line arguments, use for each loop to print the data and sum of all values

```
package p1;
public class CommandLine
{
	public static void main(String args[])
	{
		int sum=0;
		for(String r: args)
```

```
                sum=sum+Integer.parseInt(r);
            System.out.println("sum is "+sum);

      }
}
```

If the user enters 10 20 as command line arguments, 10 will be stored as string "10" in args[0], 20 will be stored as string "20" in args[1]. To convert String to Integer, we use parseInt static method defined in Integer Class.

Ex: int ans=Integer.parseInt("10"); "10" is converted into integer 10 and stored in ans.

For each loop:

```
for(String r: args)
      sum=sum+Integer.parseInt(r);
```

in the above for loop,
In the first iteration of for loop r will hold args[0]
in the next iteration of for loop r will hold args[1] and so on until there are elements in the string array args.

Q2) Explain about Wrapper classes – Byte, Short, Integer, Float, Double, Boolean, Character.

Ans: Primitive types, rather than objects, are used for these quantities for the sake of performance. Using objects for these values would add an unacceptable overhead to even the simplest of calculations. Thus, the primitive types are not part of the object hierarchy, and they do not inherit Object. Despite the performance benefit offered by the primitive types, there are times when you will need an object representation. For example, you can't pass a primitive type by reference to a method. Also, many of the standard data structures implemented by Java operate on objects, which means that you can't use these data structures to store primitive types. To handle these (and other) situations, Java provides type wrappers, which are classes that encapsulate a primitive type within an object.

The process of encapsulating a value within an object is called *boxing*. Thus, in the program, this line boxes the value 100 into an **Integer**:

```
Integer iOb = new Integer(100);
```

The process of extracting a value from a type wrapper is called *unboxing*. For example, the program unboxes the value in **iOb** with this statement:

```
int i = iOb.intValue();
```

Summary:

autoboxing->means converting primitive to object

Ex: int x = 100;

```
Integer iOb = new Integer(x);
```

auto-unboxing ->means converting object to primitive

ex: `int i = iOb.intValue();`

Thus, autoboxing/unboxing might occur when an argument is passed to a method, or when a value is returned by a method automatically.

Q4) Write a Java Program to read Student ID, name, marks of 3 subjects through Scanner, and display the details along with total and percentage obtained.

Student.java

```
package p1;
public class Student
{
      private long id;
      private String name;
      private int m1,m2,m3;
```

```
	public void setValues(long id, String name, int m1, int m2, int m3)
	{
		this.id=id;
		this.name=name;
		this.m1=m1;
		this.m2=m2;
		this.m3=m3;

	}
	public long getId()
	{
		return id;
	}
	public String getName()
	{
		return name;
	}
	public int total()
	{
		return m1+m2+m3;
	}
	public double per()
	{
		return(total()/300);
	}
}
```

StudentDemo.java

```
package p1;
import java.util.Scanner;
public class StudentDemo
{
	public static void main(String a[])
	{
		Student ob = new Student();
		System.out.println("enter id, name, m1,m2,m3");
		Scanner sc = new Scanner(System.in);

	ob.setValues(sc.nextLong(),sc.next(),sc.nextInt(),sc.nextInt(),sc.nextInt());
		System.out.println("total = "+ob.total());
		System.out.println("per = "+ob.per());
		sc.close();
	}
}
```

===(or)

```
package p1;
import java.util.Scanner;
public class StudentDemo
{
	public static void main(String a[])
	{
		Student ob[] = new Student[3];
		System.out.println("enter id, name, m1,m2,m3");
		Scanner sc = new Scanner(System.in);
		for(int i=0;i<3;i++)
		{
			ob[i]=new Student();

	ob[i].setValues(sc.nextLong(),sc.next(),sc.nextInt(),sc.nextInt(),sc.nextInt());
			System.out.println("total = "+ob[i].total());
			System.out.println("per = "+ob[i].per());
		}

	}

}
```

SESSION NUMBER: 07

Q1) Need for accessors and mutators

User interacts only with the class that has main method. The programmer who is writing the main method do not know what variables names have been used and ……

also hacker will not come to know what variables names have been used

Q2) Create a Cuboid class with 3 private instance variables length, breadth and height of type double, and a public method volume of return type double().

Add 3 setter methods with return type boolean (the instance variable is set when the argument is +ve) for each of the 3 instance members

Also add 3 getter methods of return type, which return the value appended with m (meters).

Use toString() method to print the details of Cuboid as

Length : 10.0 m

Breadth : 10.0 m

Height: 10.0 m

Volume : 100.0 cu.m

Access each of these methods through an object of Cuboid from the main() method of Demo class.

Cuboid3.java

```
package p1;
public class Cuboid3
{
	private  double l,b,h;
	/*public boolean setLength(double l)
	{
		if(l<0)
			return false;

		this.l=l;
		return true;

	}
	public boolean setBreadth(double b)
	{
		if(b<0)
			return false;
		this.b=b;
		return true;
	}
	public boolean setHeight(double h)
	{
		if(h<0)
			return false;
		this.h=h;
		return true;
	}*/
	public boolean setDimensions(double l, double b, double h)
	{
		if(l<0 || b <0 || h < 0)
			return false;
		this.l=l;
		this.b=b;
		this.h=h;
		return true;
	}
	public String getLength()
	{
		return l+" meters";

	}
	public String getBreadth()
	{
		return b+" meters";
	}
	public String getHeight()
	{
		return h+" meters";
	}
	public String toString()
```

```
        {
String output=String.format("Length =%.1f Breadth= %.1f Height = %.1f volume= %.1f",l,b,h,volume() );
                return output;
        }
        public double volume()
        {
                return l*b*h;
        }
}
```

CuboidDemo3.java

```
package p1;
public class CuboidDemo3
{
        public static void main(String a[])
        {
                Cuboid3 ob = new Cuboid3();
                boolean chk=ob.setDimensions(5,6,7);
                if(chk == true)
                {
                        System.out.println("length = "+ob.getLength());
                        System.out.println(ob);
                }
                else
                        System.out.println("invalid input");

        }

}
```

Q2) Set the instance variables by obtaining input from console. (Use Scanner)

SESSION NUMBER: 08

Q) Explain the concept of method overloading and its advantages

Method Overloading is a feature that allows a class to have more than one method having the same name.

In order to overload a method, the parameter lists of the methods must differ in either of these:

1. Number of parameters.

For example: This is a valid case of overloading

add(int, int)
add(int, int, int)

2. Data type of parameters.

For example:

add(int, int)
add(int, float)

3. Sequence of Data type of parameters.

For example:

add(int, float)
add(float, int)

Method overloading is an example of Static Polymorphism.

Points to Note:

1. Static Polymorphism is also known as compile time binding or early binding.
2. Static binding happens at compile time. Method overloading is an example of static binding where binding of method call to its definition happens at Compile time.

Add setDimensions(double l, double b, double h) in the Cuboid class which set the instance variables when all the arguments are +ve. Overload this with setDimensions(double s). Both methods return Boolean type.

```
package p1;
public class Cuboid4
{
	private  double l,b,h;

	public boolean setDimensions(double l, double b, double h)
	{
		if(l<0 || b <0 || h < 0)
			return false;
		this.l=l;
		this.b=b;
		this.h=h;
		return true;
	}
	public boolean setDimensions(double l)
	{
		if(l<0)
			return false;
		this.l=l;
		this.b=l;
		this.h=l;
		return true;
	}
	public String toString()
	{
String output=String.format("Length =%.1f Breadth= %.1f Height = %.1f",l,b,h);
		return output;
	}

}
```

Q2) Access the methods through an object of Cuboid from main () method of Demo class.

```
package p1;

public class CuboidDemo4
{
	public static void main(String a[])
	{
		Cuboid4 ob = new Cuboid4();
		boolean chk=ob.setDimensions(5,6,7);
		if(chk == true)
		{
			System.out.println(ob);
		}
		else
			System.out.println("invalid input");

		chk=ob.setDimensions(5);
		if(chk == true)
		{
			System.out.println(ob);
		}
		else
			System.out.println("invalid input");

	}

}
```

Q3) Modify the Demo class to store the details of 10 cuboid references in an array and print them

Cuboid4 ob[] = **new** Cuboid4[10]; →creates an array of 10 reference variables ob[0],ob[1]…ob[9]

ob[i] = **new** Cuboid4(); →creates an object and assigns them to the reference variable ob[i]

```
package p1;

public class CuboidDemo5
{
	public static void main(String a[])
	{
		Cuboid4 ob[] = new Cuboid4[10]; //creates an array of 10 reference variables
		for(int i=0;i<10;i++)
		{
			ob[i] = new Cuboid4();
			boolean chk=ob[i].setDimensions(5,6,7);
			if(chk == true)
			{
				System.out.println(ob[i]);
			}
			else
				System.out.println("invalid input");


		}

	}
}
```

Session 9

Writing to a file

```
/* writes them to an output file, and stop processing when the user inputs "DONE."*/
import java.util.Scanner;
import java.io.*;

public class Write1
{
	public static void main(String [] args) throws IOException
	{
		Scanner scan = new Scanner(System.in);

		PrintWriter pw = new PrintWriter("test.txt");
		while(true)
		{
			String input = scan.nextLine();			// read the data from user
			if("DONE".equalsIgnoreCase(input.trim()))
			break;

			pw.println(input);   // write to file test.txt

		}
		scan.close();
		pw.close();
	}
}
```

input :
10 20 30
1 2 3

Q) Write a java program to read length, breadth and height of 3 cuboids and compute the volume of each cuboid and display each cuboid volume on console

```
import java.io.PrintWriter;
import java.io.FileNotFoundException;
import java.util.Scanner;
import java.io.File;
public class Write
{
```

```
public static void main(String a[]) throws FileNotFoundException
        {
                File f = new File("input.txt");
                Scanner sc = new Scanner(f);

                while(sc.hasNextDouble())
                {
                        double vol=sc.nextDouble()*sc.nextDouble()*sc.nextDouble();
                        System.out.println("volume ="+vol);
                }
        }
}
```

If input.txt contains

```
10 20 5
1 2 5
3 2 4
```

Output:

```
volume = 1000.0
volume = 10.0
volume = 24.0
```

Q) Develop the main method of Demo class such that it reads length, breadth and height of 10 cuboids from a file and outputs the volume details to another file

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
import java.io.PrintWriter;
public class Write
{

   public static void main(String args[]) throws FileNotFoundException
   {

      File text = new File("input.txt");

      Scanner sc = new Scanner(text); // to read from file
      PrintWriter pw = new PrintWriter("vol.txt");
      double vol;
      while(sc.hasNextDouble())
          {
         vol=sc.nextDouble()*sc.nextDouble()*sc.nextDouble();
             pw.println(vol);
      }
          sc.close();
          pw.close();

   }
 }
```

Q) Develop a JFrame to illustrate the operation of basic calculator (Accept 2 integer inputs, perform either addition/subtraction and display the result)

```
import javax.swing.*;
import java.awt.event.*;
public class TextFieldExample implements ActionListener
{
        JTextField tf1,tf2,tf3;
        JButton b1,b2;
        TextFieldExample()
```

```
	{
		JFrame f= new JFrame();
		tf1=new JTextField();
		tf1.setBounds(50,50,150,20);
		tf2=new JTextField();
		tf2.setBounds(50,100,150,20);
		tf3=new JTextField();
		tf3.setBounds(50,150,150,20);
		tf3.setEditable(false);
		b1=new JButton("+");
		b1.setBounds(50,200,50,50);
		b2=new JButton("-");
		b2.setBounds(120,200,50,50);
		b1.addActionListener(this);
		b2.addActionListener(this);
		f.add(tf1);f.add(tf2);f.add(tf3);f.add(b1);f.add(b2);
		f.setSize(300,300);
		f.setLayout(null);
		f.setVisible(true);
	}
	public void actionPerformed(ActionEvent e)
	 {
		String s1=tf1.getText();
		String s2=tf2.getText();
		int a=Integer.parseInt(s1);
		int b=Integer.parseInt(s2);
		int c=0;
		if(e.getSource()==b1){
			c=a+b;
		}
		else if(e.getSource()==b2){
			c=a-b;
		}
		String result=String.valueOf(c);
		tf3.setText(result);
	}
	public static void main(String[] args)
	{
		new TextFieldExample();
	}
}
```

Notes:

Scanner class is part of java.util package. so import statements

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
   public static void main(String args[]) throws FileNotFoundException {
```

=>To open a file in read mode,

```
File f = new File("input.txt");
Scanner sc = new Scanner(f);
```

=>To read a line from file

```
while(sc.hasNextLine()){
   String line = scnr.nextLine();
   System.out.println( line);}
```

The **hasNextLine**() is a method of **Java Scanner** class which is used to check if there is another line in the input of this **scanner**. It returns true if it finds another line, otherwise returns false.

List of some ofscanner methods available are

Modifier & Type	Method	Description
void	close()	It is used to close this scanner.
boolean	hasNext()	It returns true if this scanner has another token in its input.
boolean	hasNextLine()	It is used to check if there is another line in the input of this scanner or not.
boolean	hasNextLong()	It is used to check if the next token in this scanner's input can be interpreted as a Long using the nextLong() method or not.
boolean	hasNextShort()	It is used to check if the next token in this scanner's input can be interpreted as a Short using the nextShort() method or not.
String	next()	It is used to get the next complete token from the scanner which is in use.
BigDecimal	nextBigDecimal()	It scans the next token of the input as a BigDecimal.
BigInteger	nextBigInteger()	It scans the next token of the input as a BigInteger.
boolean	nextBoolean()	It scans the next token of the input into a boolean value and returns that value.
byte	nextByte()	It scans the next token of the input as a byte.
double	nextDouble()	It scans the next token of the input as a double.
float	nextFloat()	It scans the next token of the input as a float.
int	nextInt()	It scans the next token of the input as an Int.
String	nextLine()	It is used to get the input string that was skipped of the Scanner object.
long	nextLong()	It scans the next token of the input as a long.
short	nextShort()	It scans the next token of the input as a short.

Writing to file

Print Writer class: is part of java.io package throws FileNotFoundException

```
import java.io.PrintWriter;
public static void main(String args[]) throws FileNotFoundException
```

=> To open/create a file in write mode:

PrintWriter pw = new PrintWriter(“vol.txt”);

=> To write to a file

pw.println(“this is new line”);

String s=”First”;

pw.printf("This is a %s program", s);

If the file vol.txt does not exists, a new file is created. If the file exists, the contents of file are erased and the current data will be written to file.

Other methods available in PrintWriter class

Method	**Description**
void println(boolean x)	It is used to print the boolean value.
void println(char[] x)	It is used to print an array of characters.
void println(int x)	It is used to print an integer.
boolean checkError()	It is used to flushes the stream and check its error state.
protected void setError()	It is used to indicate that an error occurs.
protected void clearError()	It is used to clear the error state of a stream.
PrintWriter format(String format, Object... args)	It is used to write a formatted string to the writer using specified arguments and format string.
void print(Object obj)	It is used to print an object.
void flush()	It is used to flushes the stream.
void close()	It is used to close the stream.

Java Exception and Errors

What is an Exception?

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.

Error vs Exception

Error: Problems due to system. Error is irrecoverable. An Error indicates serious problem that a reasonable application should not try to catch.

Exception: Exception indicates conditions that a reasonable application might try to catch.

Unchecked vs Checked Exceptions

The exceptions which are checked by compiler for smooth execution of the program at run time are called Checked Exceptions. The compiler is like mother asking to check whether taking hall ticket or not? for smooth writing of exam

Compiler will check whether we are handling exception . If the programmer is not handling, we get compile time error. **If a code block throws a checked exception then the method must handle the exception or it must specify the exception using throws keyword.**

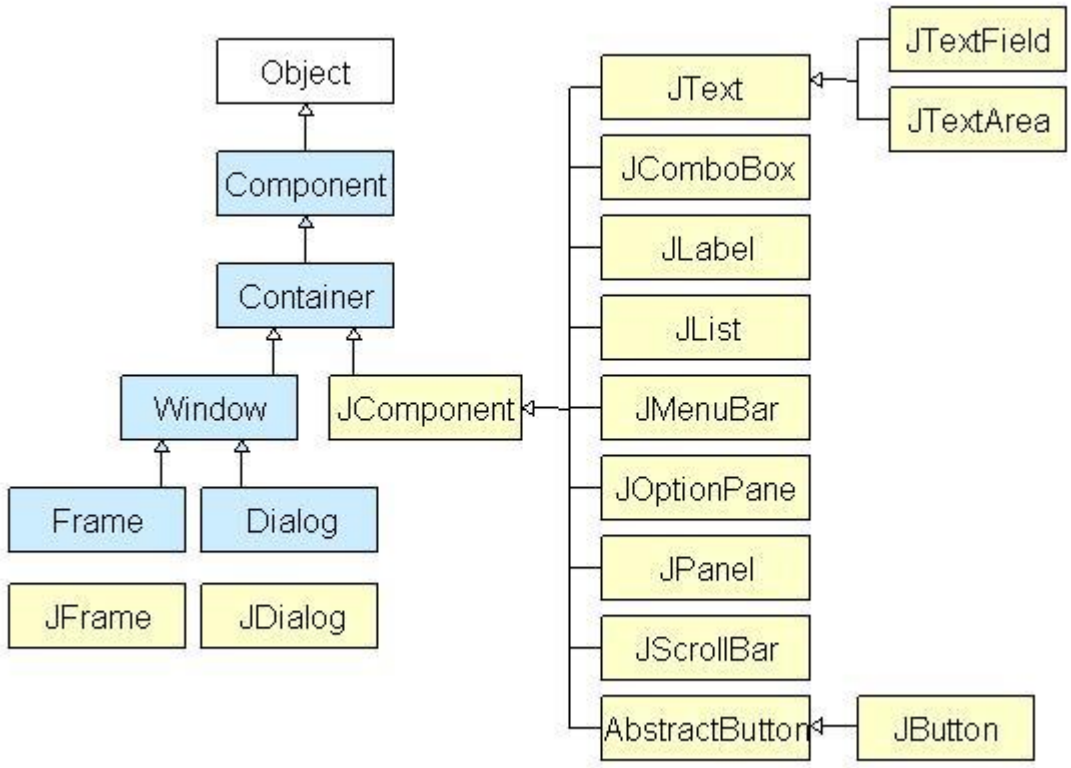
Unchecked Exception : mother will not ask how you will handle a bomb blast at your exam center…until run time

ArithmeticException
e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc.
Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

Whether exception is checked or unchecked, compulsory it will occur only at run time. There is no chance of exception at compile time.

RunTimeException and its child classes , error and its child classes are unchecked except this remaining are checked exceptions
Java Exception Keywords
There are 5 keywords which are used in handling exceptions in Java.

Keyword	**Description**
try	The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.



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**.
The methods of Component class are widely used in java swing that are given below.

Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component in pixels.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

For creating applications, JFrame is used. The pane with which your application will interact the most is the content pane, because this is the pane to which you will add visual components. In other words, when you add a component, such as a button, to a top-level container, you will add it to the content pane. By default, the content pane is an opaque instance of **JPanel**.

To create a container called jfrm that defines a rectangular window complete with a title bar; close, minimize, maximize, and restore buttons; and a system menu use the below statement.
JFrame jfrm = new JFrame("A Simple Swing Application");

To create a TextField that allow to you to enter data TextField tf=**new** TextField();
You can set the position and height and width of text field using setBounds(int xaxis, int yaxis, int width, int height) ex: tf.setBounds(50,50,150,20);
tf.setEditable(**false**); This statement will not allow user to enter data into the textfield.
To add the TextField to JFrame, jfrm.add(tf);

jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
After this call executes, closing the window causes the entire application to terminate.

Creating a container without a layout manager involves the following steps.

1. Set the container's layout manager to null by calling `setLayout(null)`.
2. Call the `Component` class's `setbounds` method for each of the container's children.
3. Call the `Component` class's `repaint` method.

However, creating containers with absolutely positioned containers can cause problems if the window containing the container is resized. So it is better to use layout managers.

The LayoutManagers are used to arrange components in a particular manner.
LayoutManager is an interface that is implemented by all the classes of layout managers.
Refer https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html. There are following classes that represents the layout managers:

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout
6. javax.swing.BoxLayout
7. javax.swing.GroupLayout
8. javax.swing.ScrollPaneLayout

9. javax.swing.SpringLayout etc.

Java GridLayout

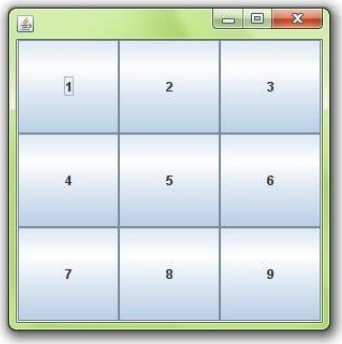
The GridLayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle.

Constructors of GridLayout class

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

Example of GridLayout class

```
1.  import java.awt.*;
2.  import javax.swing.*;
3.
4.  public class MyGridLayout{
5.  JFrame f;
6.  MyGridLayout(){
7.      f=new JFrame();
8.
9.      JButton b1=new JButton("1");
10.     JButton b2=new JButton("2");
11.     JButton b3=new JButton("3");
12.     JButton b4=new JButton("4");
13.     JButton b5=new JButton("5");
14.         JButton b6=new JButton("6");
15.         JButton b7=new JButton("7");
16.     JButton b8=new JButton("8");
17.         JButton b9=new JButton("9");
18.
19.     f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
20.     f.add(b6);f.add(b7);f.add(b8);f.add(b9);
21.
22.     f.setLayout(new GridLayout(3,3));
23.     //setting grid layout of 3 rows and 3 columns
24.
25.     f.setSize(300,300);
26.     f.setVisible(true);
27. }
28. public static void main(String[] args) {
29.     new MyGridLayout();
30. }
31. }
```



Java ActionListener Interface

The Java ActionListener is notified whenever you click on the button or menu item. It is notified against ActionEvent. The ActionListener interface is found in java.awt.event package. It has only one method: actionPerformed().

actionPerformed() method

The actionPerformed() method is invoked automatically whenever you click on the registered component.

1. **public abstract void** actionPerformed(ActionEvent e);

How to write ActionListener

The common approach is to implement the ActionListener. If you implement the ActionListener class, you need to follow 3 steps:

1) Implement the ActionListener interface in the class:

```
1. public class ActionListenerExample Implements ActionListener
```

2) Register the component with the Listener:

```
1. component.addActionListener(instanceOfListenerclass);
```

3) Override the actionPerformed() method:

```
1. public void actionPerformed(ActionEvent e){
2.         //Write the code here
3. }
```

Session 10

Q) Define constructors, rules and types

Constructors are used to perform initialization of object at the time of object creation.
Rules for constructors

1. **name of constructor is same as class name**.
2. **no return type allowed for constructors** even void as the constructor is called at the time of object creation. if you write return type it becomes a method even though method is having the same name as class name.
3. public, private, default(within package) and protected(other packages also but only in child classes) access specifier are allowed for constructors.
4. If the constructor is private then the object can be created only in singleton classes(where we are allowed to create only single object)

Q) Implicit vs Explicit

Explicit means done by the programmer. **Implicit** means done by the JVM or the tool , not the Programmer. For Example: Java will provide us default **constructor implicitly**. Even if the programmer didn't write code for **constructor**, he can call default **constructor**.
For every object, JVM will provide default value like studentName=null and rollno=0;
When ever object is created constructor is automatically called.

Q) No-argument, parameterized constructor, copy constructors

if the constructor has no arguments and is explicitly written by programmer then it is called no-argument constructor.
Parameterized constructor has parameters that can be passed to initialize the instance variables at the time of object creation.
C**opy Constructors are used to prepare a duplicate of an existing object of a class**. In such situations, we have to use a special type of constructor, known as copy constructor. This constructor can take only one parameter, which is a reference to an object of the same class.
When we perform deep copy then both the objects will exist as two different objects. But if we do shallow copy then both the objects point to same memory.
Animation: https://hajsoftutorial.com/java-copy-constructors/

Q) Explain destructors and garbage collection

A destructor is a special method that gets called automatically as soon as the life-cycle of an object is finished. A destructor is called to de-allocate and free memory. A garbage collector is a program that runs on the Java virtual machine to recover the memory by deleting the objects which are no longer in use. A few advantages of garbage collection in Java:

- It automatically deletes the unused objects that are unreachable to free up the memory
- Garbage collection makes Java memory efficient
- It need not be explicitly called since the implementation lives in the JVM
- Garbage collection has become an important and standard component of many programming languages.

It becomes fairly difficult for any developer to force the execution of a garbage collector, but there is an alternative to this. We can use the object.finalize() method which works exactly like a destructor in Java. An Object.finalize() method is inherited in all Java objects. It is not a destructor but is used to make sure or provide additional security to ensure the use of external resources like closing the file, etc before the program shuts down. You can call it by using the method itself or system.runFinalizersOnExit(true).

The use of finalize() method is highly not recommended since it can be very unsafe and in some cases used incorrectly.

Q**) Define the no-argument, parameterized constructor and copy constructors for the Cuboid class. The no-argument constructor will set the instance variables to 0.**

The below program is an example of Constructor overloading.

```
package p1;
public class ConCuboid {
	private  double l,b,h;
	public ConCuboid()
	{
		l=b=h=0;
	}
	public ConCuboid(double le, double br, double he)
	{
		l=le;
		b=br;
		h=he;
	}
	public ConCuboid(ConCuboid ob)
	{
		l=ob.l;
		b=ob.b;
		h=ob.h;
	}
	public String toString()
	{
		String output="Length = "+l+" Breadth = "+b+" Height = "+h;
		return output;
	}
}
```

Session 11

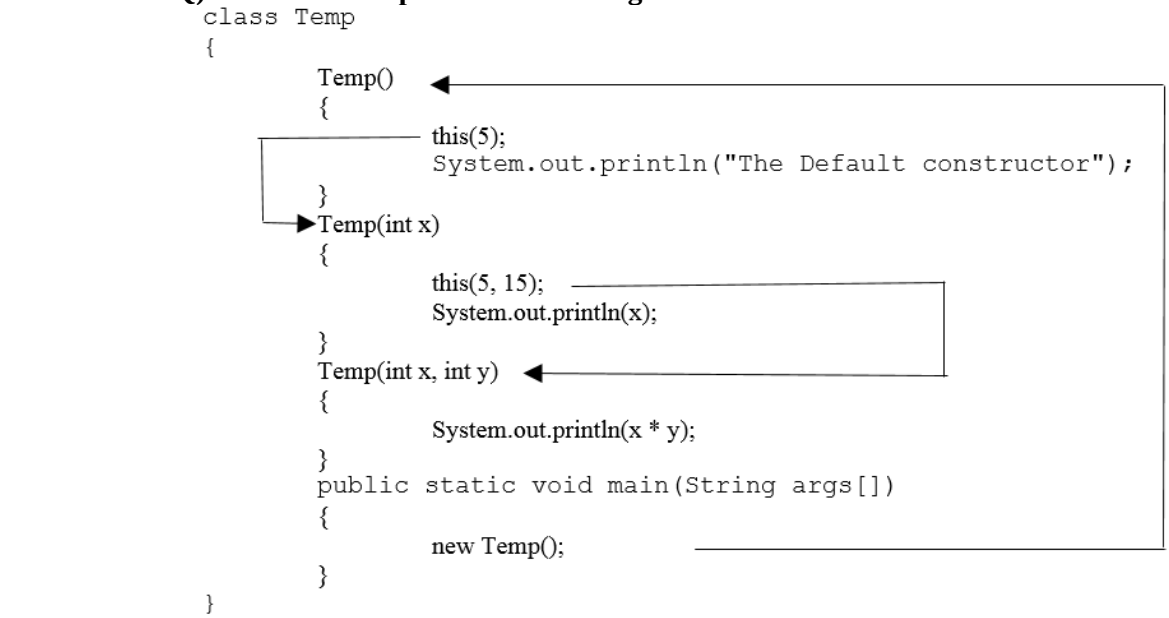
Q) Overload the Cuboid class with all 3 types of constructors and create 3 objects each invoking a different type of constructor from the main method of Demo class.

```
package p1;
public class ConCuboidDemo {
	public static void main(String args[])
	{
		ConCuboid ob[] = new ConCuboid[3];
		ob[0] = new ConCuboid();
		System.out.println(ob[0]);
		ob[1] = new ConCuboid(10,20,5);
		System.out.println(ob[1]);
		ob[2] = new ConCuboid(ob[1]);
		System.out.println(ob[2]);
	}

}
```

Output:
Length = 0.0 Breadth = 0.0 Height = 0.0
Length = 10.0 Breadth = 20.0 Height = 5.0
Length = 10.0 Breadth = 20.0 Height = 5.0

Q) Predict the output of the following



Output:
75
5
The Default constructor

Q)Enhance the above code by chaining the constructors Illustrate the importance of constructor chaining

Constructor chaining is the process of calling one constructor from another constructor with respect to current object.
Constructor chaining can be done in two ways:

Within same class: It can be done using this() keyword for constructors in same class
From base class: by using super() keyword to call constructor from the base class.
This process is used when we want to perform multiple tasks in a single constructor rather than creating a code for each task in a single constructor we create a separate constructor for each task and make their chain which makes the program more readable.

Rules of constructor chaining :

- The this() expression should always be the first line of the constructor.
- There should be at-least be one constructor without the this() keyword (constructor 3 in above example).
- Constructor chaining can be achieved in any order.

```
package p1;
public class Chaining {
		private  double l,b,h;
		public Chaining()
		{
			l=b=h=10;
		}
		public Chaining(double le, double br, double he)
		{
			this();
			setDimensions(le,br,he);

		}
		public boolean setDimensions(double le, double br, double he)
		{
			if(le >= 0 && br>=0 && he>=0)
			{
				l=le;
				b=br;
				h=he;
				return true;
			}
			return false;
		}

		public String toString()
		{
			String output="Length = "+l+" Breadth = "+b+" Height = "+h;
			return output;
		}
		public static void main(String args[])
		{
			Chaining ob = new Chaining(-10,2,3);
			System.out.println(ob);

	}
}
```

output:

Length = 10.0 Breadth = 10.0 Height = 10.0

Session 12

Passing an object as an argument to a method

Practical session 1:

Q1) Create a class Test with equals () method which compares two objects for equality and returns the result i.e., either true or false (Note: equals () methods should take an object as an argument)

Ans: Using Objects as Parameters

So far, we have only been using simple types as parameters to methods. However, it is both correct and common to pass objects to methods. For example, consider the following short program:

```
// Objects may be passed to methods.
class Test
{
	int a, b;
	Test(int i, int j)
	{
		a = i;
		b = j;
	}
	// return true if o is equal to the invoking object
	boolean equals(Test o)
	 {
		if(o.a == a && o.b == b) return true;
			else return false;
	}
}
class PassOb
{
	public static void main(String args[])
	{
		Test ob1 = new Test(100, 22);
		Test ob2 = new Test(100, 22);
		Test ob3 = new Test(-1, -1);
		System.out.println("ob1 == ob2: " + ob1.equals(ob2));
		System.out.println("ob1 == ob3: " + ob1.equals(ob3));
	}
}
```

This program generates the following output:

```
ob1 == ob2: true
ob1 == ob3: false
```

As you can see, the equals( ) method inside Test compares two objects for equality and returns the result. That is, it compares the invoking object with the one that it is passed. If they contain the same values, then the method returns true. Otherwise, it returns false. Notice that the parameter o in equals( ) specifies Test as its type. Although Test is a class type created by the program, it is used in just the same way as Java's built-in types.

Q) Call by value vs Call by reference

In general, there are two ways Java can pass an argument to a function.
The first way is *call-by-value* where values are passed as an argument to function
The second way an argument can be passed is *call-by-reference* where address is passed as an argument to function.

In Java, when you pass a primitive type to a method, it is passed by value. For example, consider the following program:

```
// Primitive types are passed by value.
class Test
{
	void meth(int i, int j)
```

```
        {
                i *= 2;
                j /= 2;
        }
}
class CallByValue
{
        public static void main(String args[])
        {
                Test ob = new Test();
                int a = 15, b = 20;
                System.out.println("a and b before call: " +a + " " + b);
                ob.meth(a, b);
                System.out.println("a and b after call: " +a + " " + b);
        }
}
```

The output from this program is shown here:

```
a and b before call: 15 20
a and b after call: 15 20
```

As you can see, the operations that occur inside **meth( )** have no effect on the values of **a** and **b** used in the call; their values here did not change to 30 and 10.

Call by Reference

For example, consider the following program:

```
package p1;

class Test {
   int a,b;
        public Test(int x,int y)
        {
                a=x;
                b=y;
        }

        public void changes(Test ob)
        {
           ob.a=100;
           ob.b=400;
        }
}
public class CallByRefernce
 {
        public static void main(String args[]) {
                Test ob1 = new Test(10,20);
                Test ob2 = new Test(20,10);

                System.out.println("a and b before call: " +ob2.a + " " + ob2.b);
                ob1.changes(ob2);
                System.out.println("a and b after call: " +      ob2.a + " " + ob2.b);
                }
}
```

This program generates the following output:
a and b before call: 20 10
a and b after call: 100 400

REMEMBER *When a primitive type is passed to a function, it is done by use of call-by-value. Objects are implicitly passed by use of call-by-reference.*

Q) Returning object from a method
Practical session 2:
Create a class Test with incrByTen () method which returns an object after incrementing the value by 10 than the value in the invoking object.
Ans:

A method can return any type of data, including class types that you create. For example, in the following program, the **incrByTen( )** method returns an object in which the value of **a** is ten greater than it is in the invoking object.

```
// Returning an object.
class Test
 {
        int a;
        Test(int i)
        {
                a = i;
        }
        Test incrByTen()
        {
                Test temp = new Test(a+10);
                return temp;
        }
}
class RetOb
{
        public static void main(String args[])
         {
                Test ob1 = new Test(2);
                Test ob2;
                ob2 = ob1.incrByTen();
                System.out.println("ob1.a: " + ob1.a);
                System.out.println("ob2.a: " + ob2.a);
                ob2 = ob2.incrByTen();
                System.out.println("ob2.a after second increase: "+ ob2.a);
        }
}
```

The output generated by this program is shown here:
ob1.a: 2
ob2.a: 12
ob2.a after second increase: 22

Session 13

Q) Explain about nested and inner classes

A class declared inside another class is known as nested class.

The scope of a nested class is bounded by the scope of its outer class.

Thus, if class B is defined within class A, then B **does not exist independently** of A.

A nested class has access to the members, including private members, of the outer class

However, the outer class does not have access to the members of the nested class.

A nested class that is declared directly within its outer class scope is a member of its outer class.

It is also possible to declare a nested class that is local to a block.

Nested classes are divided into two categories:

1. **static nested class** : Nested classes that are declared static are called static nested classes.
2. **inner class** : An inner class is a non-static nested class. inner classes are two types : Local classes and anonymous classes.

   Real time example of inner class:

   DebitCard is part of Account and DebitCard does not exist independently.

   ```
   class Account
   {  // Account is outer class.
      .......
      class DebitCard
     { // DebitCard is inner class.
            .......
      }
   }
   ```

Static nested classes

As with class methods and variables, a static nested class is associated with its outer class. And like static class methods, a static nested class cannot refer directly to instance variables or methods defined in its enclosing class: it can use them only through an object reference.

They are accessed using the enclosing class name.

OuterClass.StaticNestedClass

For example, to create an object for the static nested class, use this syntax:

OuterClass.StaticNestedClass nestedObject = new OuterClass.StaticNestedClass();

Example:

```
public class MotherBoard
{
    static class USB
    {
        void display()
        {
            System.out.println("inside inner inside display");
        }
    }
}
public class Main {
    public static void main(String[] args) {
        MotherBoard.USB usb = new MotherBoard.USB();
        System.out.println("Total Ports = " + usb.display());
    }
}
```

Q) Predict the output of the following:

```
class Outer
{
        static int temp1 = 1;
        static int temp2 = 2;
        int temp3 = 3;
        int temp4 = 4;
        public static class Inner
        {
                private static int temp5 = 5;
                private static int getSum()
                {
                        return (temp1 + temp2 + temp3 + temp4 + temp5); //error
                }
        }
        public static void main(String[] args)
        {
                Outer.Inner obj = new Outer.Inner();
                System.out.println(obj.getSum());  // an inner class is implicitly associated with
                                                   //an object of its outer class
        }
}
```

Q) Predict the output of the following

```
public class Outer
{
                static int data = 10;
                static int LocalClass()
                {
                                class Inner
                                {
                                                int data = 20;
                                                 int getData()
                                                {
                                                       return data;
                                                 }
                                };
                                Inner inner = new Inner();
                                return inner.getData();
                }
                public static void main(String[] args)
                {
                                System.out.println(data * LocalClass());
                }
}
```

Output:
200

Non Static Nested class or Inner class Example

```
class CPU {

    class Processor{
        double cores;
        String manufacturer;
        double getCache(){
            return 4.3;
        }
    }
public class Main
 {
    public static void main(String[] args) {
        CPU cpu = new CPU();
        CPU.Processor processor = cpu.new Processor();
        System.out.println("Processor Cache = " + processor.getCache());
}}
```

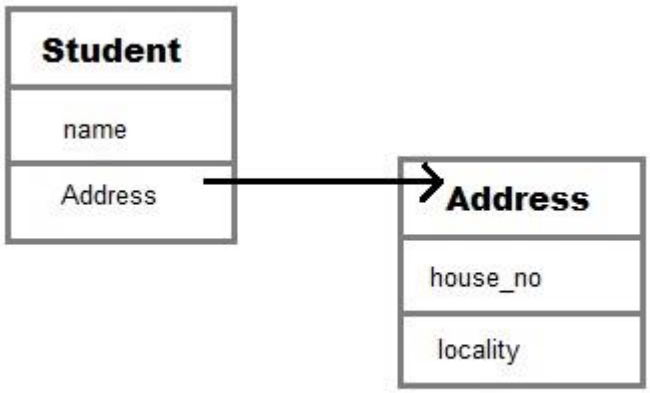
Aggregation (HAS-A relationship) in Java

HAS-A relationship is based on usage, rather than inheritance. In other words, class A *has-a* relationship with class B, if code in class A has a reference to an instance of class B.

Let's take an example:

```
class Student
{
  String name;
  Address ad;
}
```

Here in the above code, you can say that **Student** has-a **Address**.



The Student class has an instance variable of type Address. As we have a variable of type Address in the Student class, it can use Address reference which is ad in this case, to invoke methods of the Address class.

Aggregation allows you to design classes that follow good Object Oriented practices. It also provide code reusability.

Example of Aggregation in Java

Let's take an example of aggregation in Java.

```
class Author
{
  String authorName;
  int age;
  String place;

  // Author class constructor
  Author(String name, int age, String place)
  {
    this.authorName = name;
    this.age = age;
    this.place = place;
  }

  public String getAuthorName()
  {
    return authorName;
  }
  public int getAge()
  {
    return age;
  }
  public String getPlace()
  {
    return place;
  }
}

class Book
{
  String name;
  int price;
  // author details
  Author auth;
  Book(String n,int p,Author at)
  {
    this.name = n;
     this.price = p;
    this.auth = at;
  }
  public void showDetail()
  {
    System.out.println("Book is" + name);
    System.out.println("price " + price);
    // using Author class funtion to get the name value
    System.out.println("Author is " + auth.getAuthorName());
  }
}

class Test
```

```
{
  public static void main(String args[])
  {
    Author auth = new Author("John", 22, "India");
    Book b = new Book("Java", 550, auth);
    b.showDetail();
 }
}
```

Book is Java.
price is 550.
Author is John.

Q. What is Composition in Java?
Composition is a more restricted form of Aggregation. Composition can be described as when one class which includes another class, is dependent on it in such a way that it cannot functionally exist without the class which is included. For example a class Car cannot exist without Engine, as it won't be functional anymore.
Hence the word **Composition** which means the items something is made of and if we change the composition of things they change, similarly in Java classes, one class including another class is called a composition if the class included provides core functional meaning to the outer class.

```
class Car
{
  private Engine engine;
  Car(Engine en)
  {
    engine = en;
  }
}
```

Session 14 and Session 15

String class
in Java, a string is an object that represents a sequence of characters. The *java.lang.String* class is used to create string object.

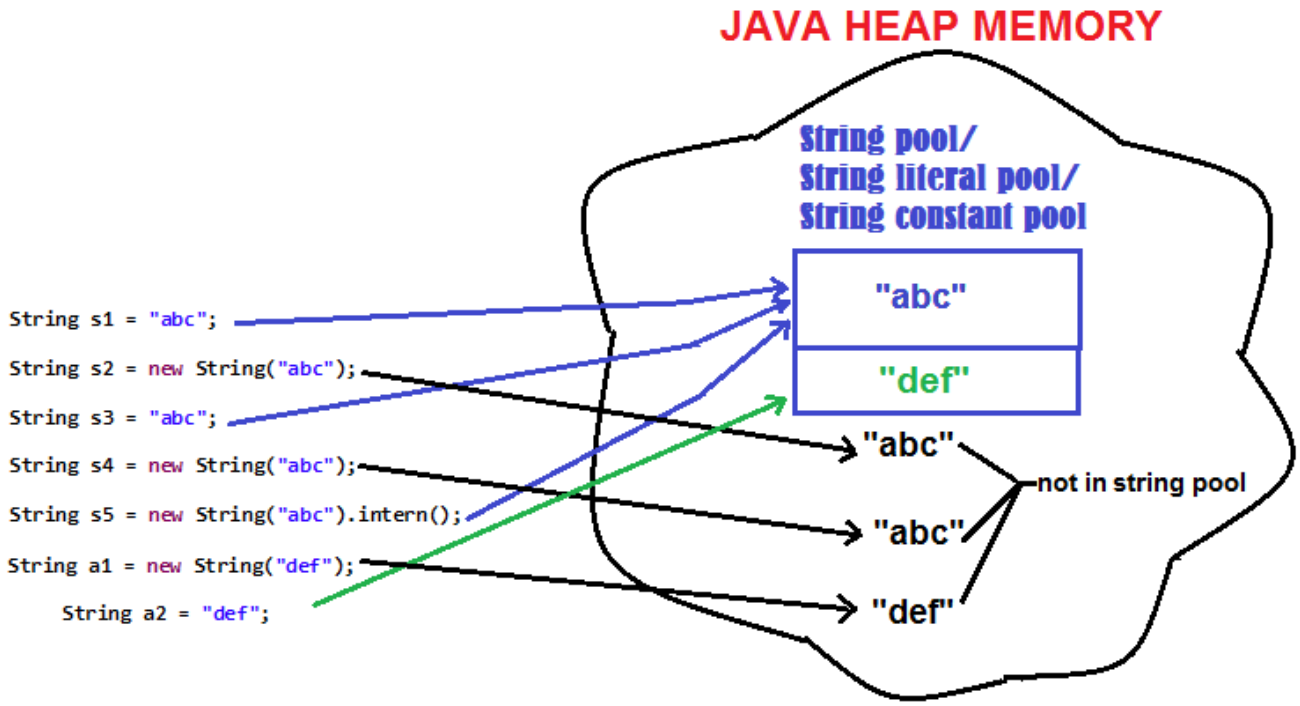
There are two ways to create a String object:

1. **By string literal** : Java String literal is created by using double quotes. For Example: String s=“Welcome”;
2. **By new keyword** : Java String is created by using a “new” keyword.
   For example: String s=new String(“Welcome”);
   It creates two objects (in String pool and in heap) and one reference variable where the variable ‘s’ will refer to the object in the heap.

Now, let us understand the concept of Java String pool.

Java String Pool: Java String pool refers to collection of Strings which are stored in heap memory. In this, whenever a new object is created, String pool first checks whether the object

is already present in the pool or not. If it is present, then same reference is returned to the variable else new object will be created in the String pool and the respective reference will be returned. Refer to the diagrammatic representation for better understanding:



Difference between == and .equals()

```
String s1="abc";

String s2= new String("abc");

System.out.println(s1.equals(s2)); // true

System.out.println(s1==s2); //false
```

In simple words, == checks if both objects point to the same memory location whereas

.equals() checks if values in both the objects are same or not.

The main difference between String and StringBuffer is String is immutable while StringBuffer is mutable means you can modify a StringBuffer object once you created it without creating any new object. This mutable property makes StringBuffer an ideal choice for dealing with Strings in Java.

Constructors of String class

1. String(byte[] byte_arr)
   Example :
   byte[] b_arr = {71, 101, 101, 107, 115};
   String s_byte =new String(b_arr); //Geeks
2. String(char[] char_arr)
   Example:
   char char_arr[] = {'G', 'e', 'e', 'k', 's'};
   String s = new String(char_arr); //Geeks
3. String(StringBuffer s_buffer)

Example:

```
StringBuffer s_buffer = new StringBuffer("Geeks");
String s = new String(s_buffer); //Geeks
```

4. String(StringBuilder s_builder)

Example:

```
StringBuilder s_builder = new StringBuilder("Geeks");
String s = new String(s_builder); //Geeks
```

Java String Methods

Method	**Description**	**Return Type**	**Example**
charAt()	Returns the character at the specified index (position)	Char	Ex: String str = "Hello"; System.out.println (str.charAt(0)); **Output:** **H**
isEmpty()	Checks whether a string is empty or not	boolean	String myStr2 = ""; System.out.println(myStr2.isEmpty()); // true

concat()	Appends a string to the end of another string	String	String f = "Hello ";
			String l = "World";
			String full=f.concat(l);
			System.out.println(f); // Hello
			System.out.println(full); //Hello World

contains()	Checks whether a string contains a sequence of characters	Boolean	Ex: String str = "Hello"; System.out.println (str.contains("Hel")); **Output**: true

endsWith()	Checks whether a string ends with the specified character(s)	boolean	String str = "Hello";
			System.out.println (str.endsWith("llo")); // true
			System.out.println (str.endsWith("o")); // true

startsWith()	Checks whether a string starts with specified characters	boolean	String str = "Hello"; System.out.println (str.startsWith("Hel")); // true

replace()	Searches a string for a specified value, and returns a new string where the specified values are replaced	String	String str = "Hello";
			String n= str.replace('l', 'p');
			String n1= str.replace("He", "p");
			System.out.println (n); //Heppo
			System.out.println (n1); //pllo

substring()	Extracts the characters from a string, beginning at a specified start position, and through the specified number of character	String	String str = "Hello";
			String n = str.substring(1, 4);
			System.out.println (n);
			
			Output:
			ell
			// prints the substring from index 1 till index 3
			Example2:
			"kluniversity".substring(3); // returns niversity

Method	Description	Return Type	Example
split()	Splits a string into an array of substrings	String[]	String str = "Iam klu student"; String[] arr = str.split(" "); for (String a : arr) System.out.println(a); Output: Iam klu student
replaceFirst()	Replaces the first occurrence of a substring that matches the given regular expression with the given replacement	String	

replaceAll()	Replaces each substring of this string that matches the given regular expression with the given replacement	String	String s1="Klu is good. Canteen is good";
			String rep=s1.replaceAll("is","was");
			//replaces all occurrences of "is" to "was"
			System.out.println(rep);

toLowerCase()	Converts a string to lower case letters	String	
toUpperCase()	Converts a string to upper case letters	String	

trim()	Removes whitespace from both ends of a string	String	Read data from user until user enters DONE
			Scanner sc = new Scanner(System.in);
			while(true)
			{
			String input = sc.next();
			input = input.trim();
			input = input.toUpperCase();
			if("DONE".equals(input))
			break;
			}

indexOf()	Returns the position of the first found occurrence of specified characters in a string
	Returns -1 if could not find
		int	String txt = "klustu";
			S.o.p(txt.indexOf("stu")); // Outputs 3
			System.out.println(txt.indexOf('a')); // returns -1 since 'a' is not present in klustu
			
			String s = "Learn Share Learn";
			int output = s.indexOf("ea",3);// returns 13
			Searches for string ea from index3 till end in ***s***

lastIndexOf()	Returns the position of the last found occurrence of specified characters in a string	int	String str = "klu means kluniversity"; System.out.println(str.lastIndexOf("klu")); Output: 10
valueOf()	Returns the primitive value of a String object	String	**boolean** b1=**true**; **byte** b2=11; **short** sh = 12; **int** i = 13; **long** l = 14L; **float** f = 15.5f; **double** d = 16.5d;

Method	Description	Return Type	Example
			char chr[]={'j','a','v','a'}; String s1 = String.valueOf(b1); String s2 = String.valueOf(b2); String s3 = String.valueOf(sh); String s4 = String.valueOf(i); String s5 = String.valueOf(l); String s6 = String.valueOf(f); String s7 = String.valueOf(d); String s8 = String.valueOf(chr); String s9 = String.valueOf(obj);

getChars()	Copies characters from a string to an array of chars	void	String str="Hello World";
			char[] ch = new char[3];
			str.getChars(2, 5, ch, 0);
			System.out.println(ch); //llo

toCharArray()	converts this string into character array. It returns a newly created character array, its length is similar to this string and its contents are initialized with the characters of this string.	char []	String s1="hello"; **char**[] ch=s1.toCharArray(); for(int i=0;i<ch.length;i++) System.out.print(ch[i]);
regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)	regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)	regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)	regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)
compareTo()	Compares two strings lexicographically	int	
compareToIgnoreCase()	Compares two strings lexicographically, ignoring case differences	int	
intern()	Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index	String	
toString()	Returns the value of a String object	String	

StringTokenizer class is deprecated now. It is recommended to use split() method of String class or regex (Regular Expression).

The **java.util.StringTokenizer** class allows you to break a string into tokens.

Below Program tokenizes a string "my name is khan" on the basis of whitespace.

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

Output:my
name
is
khan

StringBuffer class in Java

StringBuffer is a peer class of String that provides much of the functionality of strings. String represents fixed-length, immutable character sequences while StringBuffer represents growable and writable character sequences.

StringBuffer may have characters and substrings inserted in the middle or appended to the end. It will automatically grow to make room for such additions and often has more characters preallocated than are actually needed, to allow room for growth.

StringBuffer Constructors

StringBuffer( ): It reserves room for 16 characters without reallocation.
StringBuffer s=new StringBuffer();

StringBuffer(int size)It accepts an integer argument that explicitly sets the size of the buffer.
StringBuffer s=new StringBuffer(20);

StringBuffer(String str): It accepts a String argument that sets the initial contents of the StringBuffer object and reserves room for 16 more characters without reallocation.
StringBuffer s=new StringBuffer("kluniversity");

Methods:

1) length( ) and capacity( ): The length of a StringBuffer can be found by the length( ) method, while the total allocated capacity can be found by the capacity( ) method.

Code Example:

```
StringBuffer s = new StringBuffer("kluniversity");
int p = s.length();
int q = s.capacity();
```

2) append( ): It is used to add text at the end of text. Here are a few of its forms:

StringBuffer append(String str)
StringBuffer append(int num)

Example:

```
StringBuffer s = new StringBuffer("klu");
s.append("university");
System.out.println(s); // returns kluuniversity
s.append(1);
System.out.println(s); // returns kluuniversity1
```

3) insert( ): It is used to insert text at the specified index position. These are a few of its forms:

StringBuffer insert(int index, String str)
StringBuffer insert(int index, char ch)
Here, index specifies the index at which point the string will be inserted

Example:

```
StringBuffer s = new StringBuffer("klu for me");
s.insert(4, "is ");
System.out.println(s); // returns klu is for me

s.insert(5, 41.35d);
System.out.println(s); // returns klu i41.35s for me

char arr[] = { 'p', 'a', 'w', 'a', 'n' };
s.insert(3, arr);
System.out.println(s); // klupawan i41.35s for me
```

4) delete( ) : The delete( ) method deletes a sequence of characters from the invoking object. Here, start Index specifies the index of the first character to remove, and end Index specifies an index one past the last character to remove. Thus, the substring deleted runs from start Index to endIndex–1. The resulting StringBuffer object is returned.

Syntax:

```
StringBuffer delete(int startIndex, int endIndex)
```

Example:

```
StringBuffer s = new StringBuffer("kluniversity");
s.delete(0, 2);
System.out.println(s); // returns university
```

5) deleteCharAt( ): The deleteCharAt( ) method deletes the character at the index specified by loc. It returns the resulting StringBuffer object.

Syntax:

```
StringBuffer deleteCharAt(int loc)
```

Example:

```
StringBuffer s = new StringBuffer("kluniversity");
s.deleteCharAt(2);
System.out.println(s); // returns klniversity
```

6) void setCharAt(int index, char ch): The character at the specified index is set to ch.

Syntax:

```
public void setCharAt(int index, char ch)
```

7) replace()

Syntax:

```
StringBuffer replace(int start, int end, String str)
```

Example:

```
StringBuffer s = new StringBuffer("Klu is good.");
s.replace(4, 6, "WAS");
System.out.println(s); // returns Klu WAS good.
```

8)reverse()

Write a program to reverse the given string

```
StringBuffer sbf = new StringBuffer("Welcome to KLU");
sbf.reverse();
System.out.println("String after reversing = " + sbf);
```

9) char charAt(int index)
This method returns the char value in this sequence at the specified index.
10) void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)
This method characters are copied from this sequence into the destination character array dst.
11) int lastIndexOf(String str)
This method returns the index within this string of the rightmost occurrence of the specified substring.
12) void trimToSize()
This method attempts to reduce storage used for the character sequence.

Comparison between functions in String and String Buffer class methods.

String f = "Hello ";
f=f.concat("World");
System.out.println(f); // Hello World
	StringBuffer s = new StringBuffer("Hello");
	s.append("World");
	System.out.println(s); // returns kluuniversity

Session-16

Q) Develop a student class with the following fields: ID, name, DOB (Use java.util.Data), gender. Use appropriate getters and setters, toString() method and constructors

```
package p1;
import java.text.SimpleDateFormat;
import java.util.Date;

public class StudentDate
{
        private long id;
        private String name;
        private Date dob;
        public StudentDate(long tid, String tname, Date tdob)
        {
                id=tid;
                name=tname;
                dob=tdob;
        }
        public String toString()
        {
```

```
                SimpleDateFormat df = new SimpleDateFormat("dd/MM/yyyy");
                return "regd id= "+id+" name = "+name+" dob=" +df.format(dob);
        }
}
package p1;

import java.util.Calendar;
import java.util.Scanner;

public class StudentDateDemo {
        public static void main(String[] args)
        {
                Scanner sc = new Scanner(System.in);
                System.out.println("enter id, name ,day,month, year");
                long tid =sc.nextLong();
                sc.nextLine();
                String tname =sc.nextLine();
                int day =sc.nextInt();
                int month = sc.nextInt();
                int year =sc.nextInt();
                Calendar c = Calendar.getInstance();
                c.set(year, month-1,day);
                StudentDate ob = new StudentDate(tid,tname,c.getTime());
                System.out.println(ob);
        }
}
```

When an object of Date class is created it will contain today's date.When we print the date object, it will give date as follows

Tue Jan 07 03:04:28 GMT 2020

To convert date into format like 07/01/2020, we can use SimpleDateFormat class available in java.text package.

SimpleDateFormat constructor accepts pattern in which date has to be displayed. The pattern is of String datatype like "dd/MM/yyyy.

Example:

```
SimpleDateFormat df = new SimpleDateFormat("dd/MM/yyyy");
```

Now we can use df.format() method to format the date object.

To create an object (*here c*) of Calendar class, we need to call getInstance() static method. By default when the object is created, it will store today date. Inorder to set the date to the date of birth of the student we call *c.set()*. The arguments to set method is year, month and day. Since months are zero indexed, i.e, month zero means January and so we pass month-1 to the set() method.

Here c.getTime() is used to convert canlendar to date.

Q) Enhance the above code to store the marks of 6 subjects(array) as a private instance member and corresponding mutator and accessor, modify the toString() method

```
package p1;
import java.util.Arrays;
public class StudentDate {
        private long id;
```

```
	private String name;
	private int marks[];
	public StudentDate(long tid, String tname,int tmarks[])
	{
		id=tid;
		name=tname;
		marks=tmarks;
	}
	public String toString()
	{
		return "regd id= "+id+" name = "+name+" dob=" + " marks in 6"+Arrays.toString(marks);
	}
}
package p1;
import java.util.Scanner;

public class StudentDateDemo
{
	public static void main(String[] args)
	{
		Scanner sc = new Scanner(System.in);
		System.out.println("enter id, name, marks in 6 subjects");
		long tid =sc.nextLong();
		sc.nextLine();
		String tname =sc.nextLine();
		int tmarks[] = new int[6];
		for(int i=0;i<tmarks.length;i++)
			tmarks[i]=sc.nextInt();

		StudentDate ob = new StudentDate(tid,tname,tmarks);
		System.out.println(ob);


	}
}
```

Session 17

Q) Enhance the main () method of Demo class to display a menu of operations as follows:

1. Add new Student
2. Print details of all students
3. Search a student based on ID
4. Search based on name
5. Modify name based on ID
6. Sort based on ID
7. Sort based on total
8. Exit

The program must store details of 10 students

```
import java.util.Scanner;
import java.util.Arrays;

class Student
{
	private long id;
```

```
	private String name;
	private int marks[],total;
	Student(long tid, String tname,int tmarks[],int ttotal )
	{
		id=tid;name=tname; marks=tmarks;total=ttotal;
	}
	public void setName(String tname)
	{
		name=tname;
	}
	public String toString()
	{
		return "regd id = "+ id+" name = "+name + "\n marks in 6 subjects = "+Arrays.toString(marks) +" Total ="+total;
	}
	public long getId()
	{
		return id;
	}
	public String getName()
	{
		return name;
	}
	public int[] getMarks()
	{
		return marks;
	}
	public int getTotal()
	{
		return total;
	}
}
class SearchSort
{
	public static void sortById(Student st[], int n)
	{

		for(int i=0;i<n-1;i++)
		{
			for(int j=0;j<n-i-1;j++)
			{
				if(st[j].getId() > st[j+1].getId())
				{
					Student temp = st[j];
					st[j]=st[j+1];
					st[j+1]=temp;
				}
			}
		}
		System.out.println("After sorting based on id, Student details are");
		for(int i=0;i<n;i++)
			System.out.println(st[i]);
	}
	public static void sortByTotal(Student st[],int n)
	{
		for(int i=0;i<n-1;i++)
		{
			for(int j=0;j<n-i-1;j++)
			{
				if(st[j].getTotal() > st[j+1].getTotal())
				{
					Student temp = st[j];
					st[j]=st[j+1];
					st[j+1]=temp;
				}
			}
		}
		System.out.println("After sorting based on total, Student details are");
```

```
		for(int i=0;i<n;i++)
			System.out.println(st[i]);
	}
	public static int linearSearch(Student st[], int n, int key)
	{
		int flag=0;
		for(int i=0;i<n;i++)
		{
			if(st[i].getId() == key)
				return i;
		   }
		return -1;
	}
	public static int linearSearch(Student st[], int n, String key)
	{
		int flag=0;
		for(int i=0;i<n;i++)
		{
			if(st[i].getName().equalsIgnoreCase(key))
				return i;
		   }
		return -1;
	}
}
public class StudentDemo
{
	public static void main(String args[])
	{
		System.out.println(" 1. Add New Student \n 2. Print details of all students  \n 3. Search a student based on id \n 4. Search based on name \n 5. Modify name based on id \n 6. Sort based on id \n 7. Sort based on total \n 8. Exit");
		Scanner sc = new Scanner(System.in);
		int choice = sc.nextInt();
		Student st[] = new Student[100];
		int nos=0;
		while(true){
		switch(choice)
		{
			case 1:
				System.out.println("enter id, name");
				int tid=sc.nextInt();
				String tname=sc.next();
				int tmarks[]=new int[6];
				int ttotal=0;
				for(int i=0;i<tmarks.length;i++)
				{
					System.out.println("enter marks of  subject "+i);
					tmarks[i]=sc.nextInt();
					ttotal=ttotal+tmarks[i];
				}
				st[nos]  = new Student(tid,tname,tmarks,ttotal);
				nos++;
				break;
			case 2:
				for(int i=0;i<nos;i++)
					System.out.println(st[i]);
				break;
			case 3:	System.out.println("enter id to search");
				int key=sc.nextInt();
				int index = SearchSort.linearSearch(st,nos,key);
				if(index == -1)
				   System.out.println("search id not found");
				else
				   System.out.println("search element found at index " + index +" student details are " +st[index]);
				break;

			case 4:
				System.out.println("enter Student name to search");
```

```
                    String key1=sc.next();
                    index = SearchSort.linearSearch(st,nos,key1);
                     if(index == -1)
                        System.out.println("search id not found");
                    else
                        System.out.println("search element found at index " + index +" student details are " +st[index]);
                    break;
                case 5:  System.out.println("enter id whose name to be modified");
                            int key2=sc.nextInt();
                            index = SearchSort.linearSearch(st,nos,key2);
                     if(index == -1)
                        System.out.println("search id not found");
                    else
                    {
                            System.out.println("enter Student name ");
                            st[index].setName(sc.next());
                            System.out.println(" student details after modifying name = " +st[index]);
                    }
                    break;
                case 6:
                    SearchSort.sortById(st,nos);
                    break;
                case 7:
                    SearchSort.sortByTotal(st,nos);
                    break;
                case 8:
                    System.exit(0);
            }
            System.out.println(" 1. Add New Student \n 2. Print details of all students  \n 3. Search a student based
on id \n 4. Search based on name \n 5. Modify name based on id \n 6. Sort based on id \n 7. Sort based on total \n 8. Exit");
            choice = sc.nextInt();
            }
        }
}
```

Supporting notes for Session-16

The java.util.Calendar class is an abstract class and you create an object of Calendar class by using the getInstance() static method of Calendar class. This method generally returns an instance of GregorianCalendar class

The Calendar class provides support for extracting date and time fields from a Date e.g. YEAR, MONTH, DAY_OF_MONTH, HOUR, MINUTE, SECOND, MILLISECOND; as well as manipulating these fields e.g. by adding and subtracting days from a date and calculating next or previous day, month or year

How to extract day, month and other fields from Calendar

- Once you set the Date to Calendar class, you can use its various get() method to extract different fields e.g. Calendar.get(Calendar.DAY_OF_MONTH) to get the current day. Here is a list of Calendar field you can use with get() and add() method:

  get(Calendar.DAY_OF_WEEK): returns 1 (Calendar.SUNDAY) to 7 (Calendar.SATURDAY).
- get(Calendar.YEAR): year
- get(Calendar.MONTH): returns 0 (Calendar.JANUARY) to 11 (Calendar.DECEMBER).
- get(Calendar.DAY_OF_MONTH), get(Calendar.DATE): 1 to 31
- get(Calendar.HOUR_OF_DAY): 0 to 23
- get(Calendar.MINUTE): 0 to 59
- get(Calendar.SECOND): 0 to 59

- get(Calendar.MILLISECOND): 0 to 999
- get(Calendar.HOUR): 0 to 11, to be used together with Calendar.AM_PM.
- get(Calendar.AM_PM): returns 0 (Calendar.AM) or 1 (Calendar.PM).
- get(Calendar.DAY_OF_WEEK_IN_MONTH): DAY_OF_MONTH 1 through 7 always correspond to DAY_OF_WEEK_IN_MONTH 1; 8 through 14 correspond to DAY_OF_WEEK_IN_MONTH 2, and so on.
- get(Calendar.DAY_OF_YEAR): 1 to 366

Calendar has these setters and operations:

- void set(int calendarField, int value)
- void set(int year, int month, int date)
- void set(int year, int month, int date, int hour, int minute, int second)
- void add(int field, int amount): Adds or subtracts the specified amount of time to the given calendar field, based on the calendar's rules.

Other frequently-used methods are:

- Date getTime(): return a Date object based on this Calendar's value.
- void setTime(Date date)

Conversion between Calendar and Date

You can use getTime() and setTime() to convert between Calendar and Date.
Date getTime(): Returns a Date object representing this Calendar's time value
void setTime(Date aDate): Sets this Calendar's time with the given Date instance

How to add days, month and year to Date

You can use the add() method of Calendar to add or subtract a day, month, year, hour, a minute or any other field from Date in Java. Yes, the same method is used for adding and subtracting, you just need to pass a negative value for subtraction e.g. -1 to subtract one day and get the yesterday's day as shown in the following example:

```
Calendar cal2 = Calendar.getInstance();
cal2.add(Calendar.DAY_OF_MONTH, 1);
Date d = cal2.getTime();
System.out.println("date after adding one day (tomorrow) : " + d);

cal2.add(Calendar.DAY_OF_MONTH, -2);
d = cal2.getTime();
System.out.println("date after subtracting two day (yesterday) : " + d);
```

You can see that the value of date returned by the Calendar is different after adding and subtracting two days from calendar instance using the add() method. Just remember, to subtract a day, pass a negative value e.g. to subtract two days we passed -2. If you want to subtract just one day, pass -1

Java Calendar Example

Here is our Java program to demonstrate some more example of Calendar class in Java.

```
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;

public class Main {

public static void main(String[] args) {

Calendar cal = Calendar.getInstance(); // returns GregorianCalendar except Japan and Thailand

// Example 1 - get the year from Calendar
System.out.println("Year : " + cal.get(Calendar.YEAR));

// Example 2 - get the month from Calendar
System.out.println("Month : " + cal.get(Calendar.MONTH));

// Example 3 - get the Day of month from Date and Calendar
System.out.println("Day of Month : " + cal.get(Calendar.DAY_OF_MONTH));

// Example 4 - get the Day of Week from Date
System.out.println("Day of Week : " + cal.get(Calendar.DAY_OF_WEEK));

// Example 5 - get the Day of year from Date
System.out.println("Day of Year : " + cal.get(Calendar.DAY_OF_YEAR));

// Example 6 - get the Week of Year from Calendar
System.out.println("Week of Year : " + cal.get(Calendar.WEEK_OF_YEAR));

// Example 7 - get the Week of Month from Date
System.out.println("Week of Month : " + cal.get(Calendar.WEEK_OF_MONTH));

// Example 8 - get the hour from time
System.out.println("Hour : " + cal.get(Calendar.HOUR));

// Example 9 - get the AM PM from time
System.out.println("AM PM : " + cal.get(Calendar.AM_PM));
```

```
// Example 10 - get the hour of the day from Calendar
System.out.println("Hour of the Day : " + cal.get(Calendar.HOUR_OF_DAY));

// Example 11 - get the minute from Calendar
System.out.println("Minute : " + cal.get(Calendar.MINUTE));

// Example 12 - get the Second from Calendar
System.out.println("Second : " + cal.get(Calendar.SECOND));
System.out.println();

// Example 13 - converting Date to Calendar
Date date= new Date();
Calendar cal1 = Calendar.getInstance();
cal.setTime(date);

// Example 14 - Creating GregorianCalendar of specific date
Calendar cal2 = new GregorianCalendar(2016, Calendar.JUNE, 21);

// Example 15 - Converting Calendar to Date
Date d = cal2.getTime();
System.out.println("date from Calendar : " + d);

// Example 16 - adding 1 day into date
cal2.add(Calendar.DAY_OF_MONTH, 1);
d = cal2.getTime();
System.out.println("date after adding one day (tommorrow) : " + d);

// Example 17 - subtracting a day from day
cal2.add(Calendar.DAY_OF_MONTH, -2);
d = cal2.getTime();
System.out.println("date after subtracting two day (yesterday) : " + d);
}
}
```