Day-1

Java Programming

Java is Object Oriented programming Language.

1) Class

2) Object

3) Inheritance

4) Polymorphism

5) Abstraction

6) Encapsulation

Types of programming languages

1) Structured: C, Python

2) Object based: Visual Basic, VB Script, Python

3) Object oriented Java, C++, C#, Python

Java -1994 Sun Microsystems

JDK- Java Development Kit

JRE - Java Run Time Environment

JVM - Java Virtual Machine (because of JVM java is system independent. Basically, JVM convert java file into class file which contains Byte code so any system can read it.)

program(.java) ----> .class (byte code)

windows

Linux

Mac

**** Java is platform independent language.**

**** Java is case sensitive language.**

Naming Conventions when you create a class

1) Class name should start with upper case letter

2) Should not start with numbers

3) should not use special characters ~! %&* etc. (is accepted in the middle of class name)

4) Do not give spaces

Day-2

Java Variables and Data Types

Java Operators

Variable? Variable is a container which can hold data.

int x=10; (Statically typed language)

String name="John"

float price=150.50

double price=150.50

char grade='A'

boolean status=true

Data types

1) Primitive Data Types

2) Non-Primitive /Derived Data types

Primitive Data Types

1. short/int/long -----> Integer type data
2. float/double --------> Decimal type of data

3. char ----------> single character (single quotes)
4. boolean ----------> boolean value (true/false)

Wrapper classes

Integer , Double, Character, Boolean

```
// PREMITIVE DATA TYPES
// boolean m; declaring variable
// m = false; initialization
// Warper class Integer, Double, Character, Boolean

boolean pass = true;  // 1 byte       Sores true or false value

byte num1 = 127;       // 1 byte Stores whole numbers from -128 to 127
short num2 = 32767;  // 2 bytes Stores whole numbers from -32,768 to 32,767
int   num3 = 22000;    // 4 bytes Stores whole numbers from -2,147,483,648 to 2,147,483,647
long  num4 = 910000; // 8 bytes Stores whole numbers from -+9,223,372,036,854,775,808

char grade = 'A';          // 2 bytes Stores a single character/letter or ASCII values\
char grade1 = 65;        // same as grade = 'A'

float  GPA = 3.12f;  // 4 bytes Stores fractional numbers. for storing 6 to 7 decimal digits
double gpa = 3.12;   // 8 bytes, Stores fractional numbers. Sufficient for storing 15 decimal digits
```

`i = i++` is a postfix increment operator - it increments `i`, then returns it to its original value (because the `i++` essentially "returns" the value of `i` before it was incremented.)

`i = ++i` would work since it's a prefix increment operator, and would return the value of I after the increment. However, you probably just want to do `i++` there without any extra assignment as you do in the first run - it's (essentially) shorthand as it is for `i = i+1`.

```
// swapping in different way without using third variable
                int j = 10;
                int i = 20;
                System.out.println("Approach 1");
                System.out.println("Before Swapping  j and i are " + j +" " + i);
                j = j + i;
                i = j - i;
                j = j - i;
                System.out.println("After  Swapping  j and i are " + j +" " + i);

                j = 10; i = 20;
                System.out.println("Approach 2");

                System.out.println("Before Swapping  j and i are " + j +" " + i);
                i = j + i - (j=i); // right to left
                System.out.println("After  Swapping  j and i are " + j +" " + i);

                j = 10; i = 20;
                System.out.println("Approach 3");
                System.out.println("Before Swapping  j and i are " + j +" " + i);
                i= i^j;
                j= i^j;
                i= i^j;      // using bitwise xor operation
                System.out.println("After  Swapping  j and i are " + j +" " + i);
```

```
switch(variable)
{
case value:  statements; break;
case value:  statements; break;
case value:  statements; break;
case value:  statements; break;
case value:  statements; break;
case value:  statements; break;
```

default: statements
}

Number’s manipulation

```
int num = 3215870;
int rev = 0;
int count = 0 , sum = 0;
int oddDigits = 0 , evenDigits = 0 ;

System.out.println(num);
while(num != 0)
{
        if((num % 10) % 2 == 0)
                evenDigits++;
        else
                oddDigits++;

        sum = sum + (num % 10);
        rev = rev * 10 + num%10;
        num = num / 10;

        count++;
}
System.out.println(rev);
System.out.println(count);
System.out.println("This is sum of digits" +sum);
System.out.println("even digits in num” + evenDigits);
System.out.println("odd digits in num "+oddDigits);
```

Arrays

Array is collection elements of same data type (Homogenies data).

We can store multiple values into a single variable.

```
int arr[] = new int[5];   This is fixed size Array
int arr2[] = {1,2,3,4,5};

for(int value:arr2) {  //Enhanced for loop is for array or collection
        System.out.println(value);}
```

2D Arrays: Two-dimensional array - rows & columns

```
int arr[][] = new int [3][2];

arr[0][0] = 100;
arr[0][1] = 200;

arr[1][0] = 300;
arr[1][1] = 400;

arr[2][0] = 500;
arr[2][1] = 600;
```

100	200
300	400
500	600

Find size of an 2D array

```
arr.length;   // return number of rows
arr[0].length;// return number of columns
```

Traversing 2D Array Approach 1

```
for(int row=0; row<arr.length;row++) {

        for(int column=0; column<arr[0].length;column++) {

                System.out.print(arr[row][column]);
        }
        System.out.println();
}
```

Traversing 2D Array Approach 2

```
for(int oneDemension[]: arr) {
        for(int value:oneDemension) {
                System.out.print(value);
        }
        System.out.println();}
```

Arrays cons:

Fixed Size, Same type values only (heterogeneous data)

Array of objects

```
System.out.println();
Object b[] = {10,5.6,'A',"Mohammad", false};

for(Object value: b) {
        System.out.println(value);
}
```

```
System.out.println(b.length);
```

Sort Elements in Array

Ref Links:
https://www.youtube.com/watch?v=cJ2eMUiCFy4
https://www.youtube.com/watch?v=3PLtvTUOCpM

```
int a[] = {50,8,7,55,14,-8,20,40,10,1000};
int b[] = {50,8,7,55,14,-8,20,40,10,1000};
Integer c[] = {50,8,7,55,14,-8,20,40,10,1000};

// Using bubble sort
        int sortedElement = 0;
```

```
for(int i=0; i<a.length;i++) {
        for(int j = a.length -1 ; j>=sortedElement; j--){
                if(a[i] > a[j])
                        a[j] = a[i] + a[j] - (a[i] = a[j]);
        }
        sortedElement++;
}
for(int value:a) {
        System.out.print(value +" ");
}
System.out.println("\nB before sort:   "+ Arrays.toString(b));
Arrays.parallelSort(b);
System.out.println("B after sort:    "+Arrays.toString(b));

System.out.println("c before sort:   "+Arrays.toString(c));
Arrays.sort(c);
System.out.println("c after sort:    "+Arrays.toString(c));

Arrays.sort(c,Collections.reverseOrder());//c Must be object to use collection

System.out.println("c after sort:    "+Arrays.toString(c));
```

Strings

String is collection of characters.
String is a predefined class which has many built-in methods.

```
String name="Mohammad";              // This is normal variable
String name1 = new String("Mohammad");  // This will create object

name.length();
name1.concat(name);
name1.trim(); //remove all leading and trilling spaces
name.equals("Mohammad");
name.equalsIgnoreCase("MoHaMmAd");
name.endsWith("ad");
"heiwh".contains("ei"); // true
"Wellcome to Java".replace("Java", "Selenium");
name.toUpperCase();
name.toLowerCase();
String  s = "abc xyz:sdfdfs:sdafasf";
String[] r = s.split(" ");
Arrays.binarySearch(arr, "xyd")
```

DO NOT CONFIUSE length with length();
length: is keyword length of array(how many elements exists in the array)
length(): is a method returns number of characters in a string.

System.***out***.println(1 + 1 + name1 + 1 + 2); output: 2 Mohammad 12

```
String s1 = "Mohammad";
String s2 = "Mohammad"; // it is value

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

String s3 = new String("Mohammad");
String s4 = new String("Mohammad"); // Objects

System.out.println(s3 == s4); // false here compares the objects/instances
System.out.println(s3.equals(s4)); //true compares the value of the objects
```

Search string in a Array

```
String  s = "abc:xyz:sdfdfs:sdafasf";
//search in Array
        String []arr = s.split(":");

        boolean found = false;

        for(String value:arr) {
                if (value.equals("abc")) {
                        found = true;
                        break;
                }
                else
                        System.out.println("Not found");
        }
        if(found){
                System.out.println("found");
        }
System.out.println(Arrays.binarySearch(arr, "xyd"));
```

Revers String

```
        String str = "Mohammad Ameri";
        int sizeStr = str.length();
        String revStr = "";
        for(int i= sizeStr-1; i >= 0; i--) {
                System.out.print(str.charAt(i));
                revStr = revStr+str.charAt(i);
        }
        System.out.println("\n"+revStr);
```

Note: length() returns the length of the string, but last index is length– 1;

OOP Concepts

Class: is consists of two things Attributes and Methods.

Class is logical Entity (blue print). and it doesn't have a physical existence, so it will not occupy any space in a MEMORY.

Object: is an instance of a class. It has a physical existence, so it needs space in memory. Objects are independent and we can Instantiate as many objects as we need.

OBJECT	CLASS
Object is an instance ofa class.	Class is a blue print from which objects are created
Object is a real world entity such as chair, pen. table, laptop etc.	Class is a group of similar objects.
Object is a physical entity.	Class is a logical entity.
Object is created many times as per requirement.	Class is declared once.
Object allocates memory when it is created.	Class doesn't allocated memory when it is created.
Object is created through new keyword. Employee ob = new Employee();	Class is declared using class keyword. class Employee{}

Inside a method we cannot create another method.

Inside a class we cannot create another class.

Constructor method: same as the class name, no return, no void keyword, may take Arguments.

Constructor invokes at the time of object creation.

Passing reference via method we passing an object;

```
cal v1 = new cal();
v1.x = 17;
v1.add(v1);
System.out.println(v1.x);
```

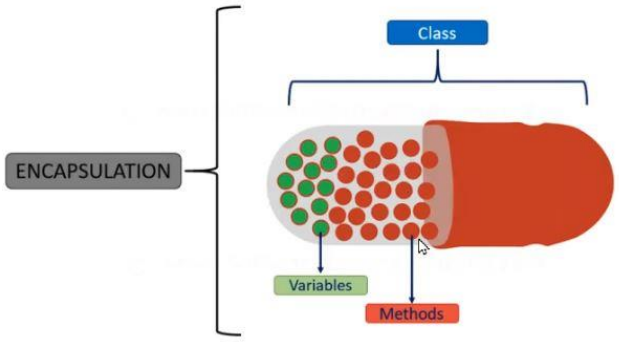
NOTE: Object name is a reference (pointer) to the location of the object in memory

Encapsulation: Wrapping up of variables and methods into a single unite. Is required for security purposes. All attributes must be private, and only attributes are accessible via seters and getters.

Encapsulation

- Encapsulation : wrapping up of data & methods into one class.
- for example, a capsule which is mixed of several medicines.

```
class X
{
variables
methods   ( setters & getters)

}
```




```
public class Account {
        private int acc_no;
        private String name;
        private double amount;

        public int getAcc_no() {
                return acc_no;
        }
        public void setAcc_no(int acc_no) {
                this.acc_no = acc_no;
```

```
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public double getAmount() {
		return amount;
	}
	public void setAmount(double amount) {
		this.amount = amount;
	}



	@Override
	public String toString() {

	return("Name is: " + this.getName() +" Account number is: " + this.getAcc_no() +
	"Account Amount:"+ this.getAmount());
	}
}
```

Polymorphism: One thing having many forms. Ploy(many).

Overloading is a technical term for Polymorphism.

Overloading is writhing multiple methods with the same name, but Parameters should be different.

Ex. Shape: circle, square, rectangle.

```
	public void add(int c) {
		x = c + 5;
	}

	public void add(float b, int c) {
		x = c + 5;
	}
	public void add(int c , float b) {
```

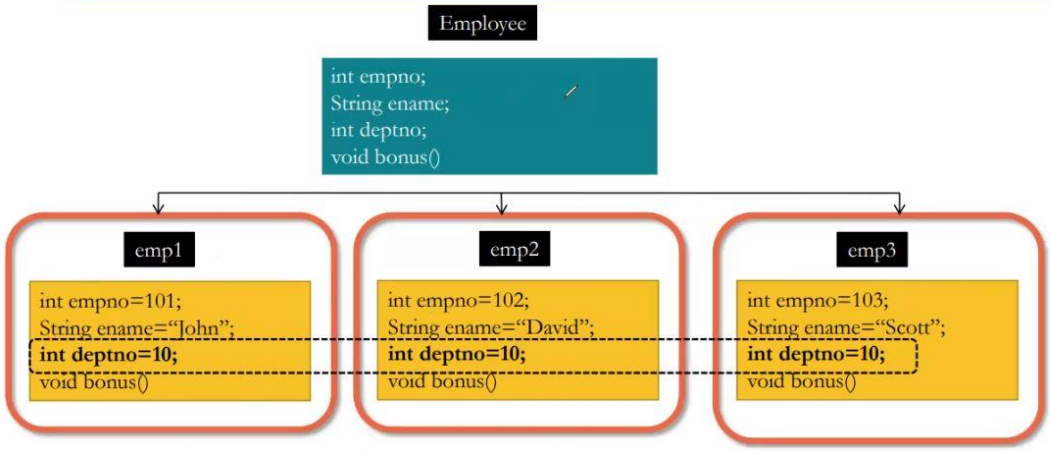
```
        x = c + 5;
    }
    public void add(int c , int b) {
        x = c + 5;
    }
```

Arguments types and orders are considered, return type not.

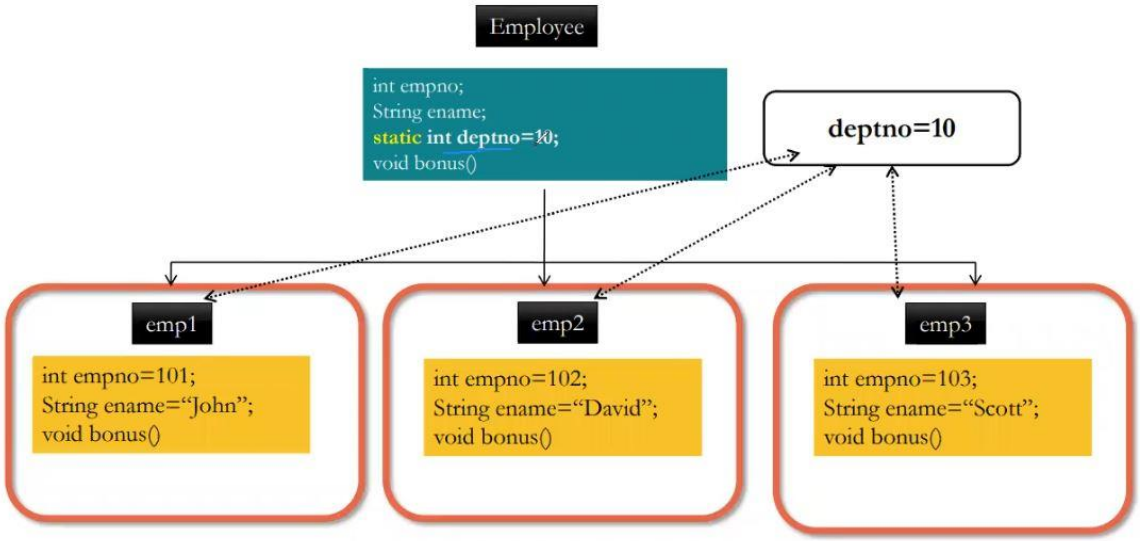
```
Method:
1) Method name can be anything
2) Method may or maynot return a value
3) If method is not returnign any value then specify void
4) Method can take parameters/arguments
5) We have to invoke/call methods explicitely through object
6) used for specifying logic

Constructor:
1) Constructor name should be same as class name.
2) Constructor will never return a value
3) We dont specify the void
4) Constructor can take parameters/arguments
5) Constructor automatically invoked at the time of object creation.
6) used for initilizing the values of the variables.
```

Static Keyword

static



- Every object occupy certain amount of space in memory.
- Objects are independent. So if modify a variable value in obj1 , that will not reflect other objects variable values.




```
static
------------
1) static methods can access only static stuff(variables & methods)- Direct ( without creating object)
2) static methods can also access Non-static stuff but through object.
3) Non-static methods can access everything directly.
```

Note: When we create static variable or method, we can access that directly by Classname.method() or Classname.variable (If the main function is in the different file use classname.method) without creating any objects.

There advantages of static are avoiding DUBLICATIOAN and REUSABILITY, also direct access.

```
int m;
static int n;

public static void  n1() {
        System.out.println("this is static method");
        n = 200;
        m = 400; // Cannot access Directly a non-static method
        m1();    // Cannot access Directly a non-static variable
}
public void m1() {
        System.out.println("this is non-static method");
        n = 100;
        m = 200;
        n1();
```

methods	static variable	static method	non static variable	non static method
static method	YES	YES	YES(Through object)	YES(Through object)
non-static method	YES	YES	YES	YES

```
System.out.println()
-------------------------
class Test
{
static String s="Welcome";
}

Test.s.length();  //7


class System
{
static PrintStream out;
}


System.out.println("dsdsds");
```

I the left picture we see how important static keyword is.

System is a class that has a variable out type of PrintStream which contains a method println();

It is exactly same as Test class which has a variable s type of String that contains length() method. We can use this method directly (without creating any object)

Basic fundamental concept: JVM runs the main function and it look for the specific pattern for the main function which is: public static void main(String[] args).

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


JDK, JRE & JVM

JDK - Java Developement Kit ( JRE & JVM) - used for developing java applications/softwares
JRE - Java Run Time Environment (JVM) - it provided environment to run/install java based applications.
JVM - Java Virtual Machine - responsible for executing java programs.
```

```
static
------------
1) static methods can access only static stuff(variables & methods)- Direct ( without creating object)
2) static methods can also access Non-static stuff but through object.
3) Non-static methods can access everything directly.
```

Note: When we create static variable or method, we can access that directly by Classname.method() or Classname.variable (If the main function is in the different file use classname.method) without creating any objects.

There advantages of static are avoiding DUBLICATIOAN and REUSABILITY, also direct access.

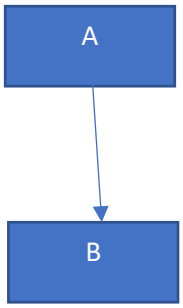
```
        int m;
        static int n;

        public static void  n1() {
                System.out.println("this is static method");
                n = 200;
                m = 400; // Cannot access Directly a non-static method
                m1();    // Cannot access Directly a non-static variable
        }
        public void m1() {
                System.out.println("this is non-static method");
                n = 100;
                m = 200;
                n1();
}
```

Inheritance

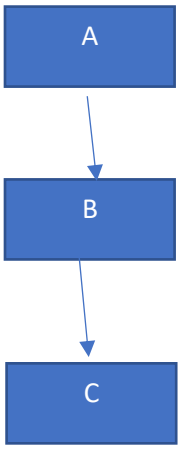
Parent child relationship. Child class will inherit all method and variables from the parent class.

Single Inheritance: child has a one parent.

class B extends A{} B is the child and A is the parent.



Multi-level Inheritance



Class B extends A{}

Class C extends B{}

Class C will contain A and B methods And Variable

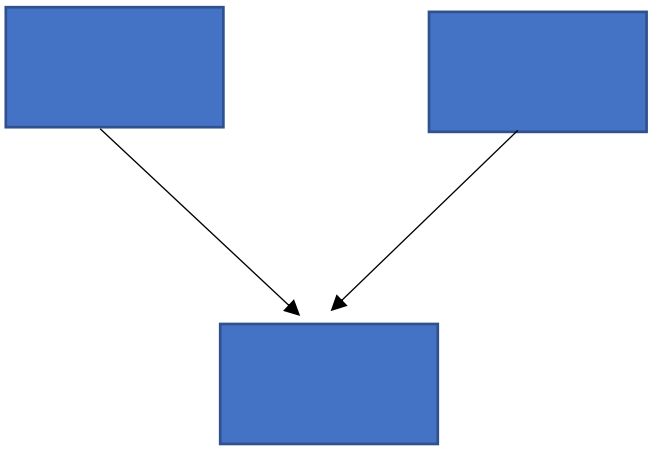
Multiple Inheritance

This type of inheritance is

Not allowed in java.

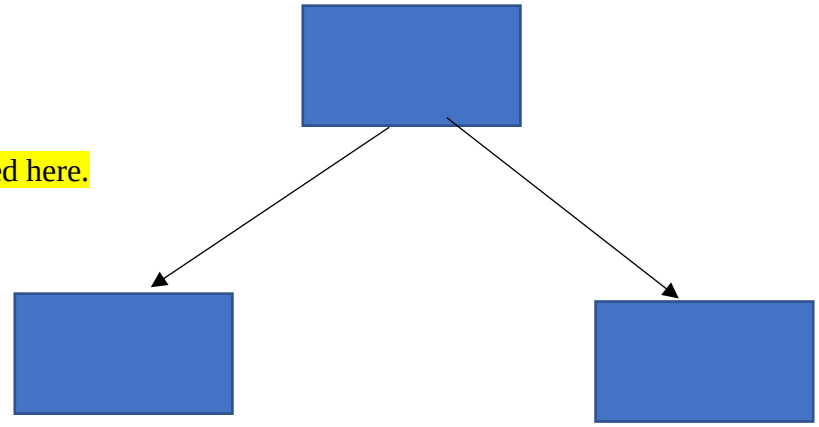
But we may use Interface

To achieve this pattern.



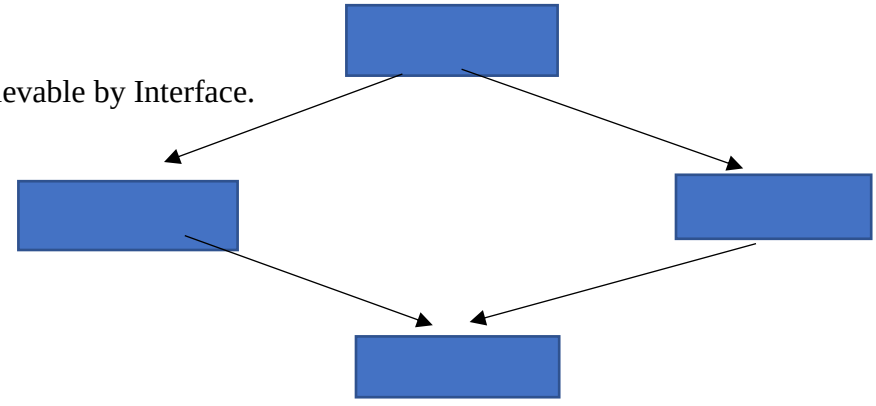
Hierarchy Inheritance

Method Overriding is used here.



Hybrid Inheritance: consist of Hierarchy and multiple Inheritance.

This pattern is only Achievable by Interface.



Method Overriding: we use method overriding to change a parent functio

In method Overriding signature of the method should not change, only we can change the body.

Method Overloading	Method overriding
No Inheritance required	Inheritance required.
Method Signature must change	Method Signature must be same

```
class Bank{
    int getRateOfInterst()
    {
        return 0;
    }
}


class SBI extends Bank {
    int getRateOfInterst() // Overriding
    {
        return 15;
    }
}

class WellsFargo extends Bank {
    int getRateOfInterst()      // Overriding
    {
        return 11;
    }
}
```

```
Types of inheritance
--------------
1) Single
2) Multi level
3) Multiple Inheritance(cannot achieve using class)
4) Hirarchical
5) Hybrid
-----------------------
```

Final keyword

Interface

Packages

1- Final keyword can be used for variable and method & class.

Final keyword for variable makes it constant.

Final keyword for method: method can not be override in the child class.

Final keyword for class: class cannot be extended.

2- Interface

- It makes it possible to achieve some Inheritance types that it not possible alone with classes.
- Structure of Interface is exactly as class.
- Interface contains only static method and static variables. (By default, they are static). Also, Abstract methods.
- In interface by default methods are public.
- In interface methods are by default abstract
- We cannot instantiate interface (make a object of Interface)

Abstract: A method have only the definition(signature) but not implementation (Body).

We use implements keyword to extends an interface

Difference between abstract class and interface

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

But there are many differences between abstract class and interface that are given below.

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods.
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 have static methods, main method and constructor**.	Interface **can't have static methods, main method or constructor**.
5) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
6) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.

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

Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%).

Multiple Inheritance using Interface

Class to class we use extends

Interface to class we use implements

Interface to Interface we use extends

```
interface A{
    int a = 10;
    void display();
}
interface B{
    int b= 20;
    void show();
}
public class MultipleInheritance implements A,B {

    @Override
    public void show() {
        // TODO Auto-generated method stub
        System.out.println(a);
    }

    @Override
    public void display() {
        // TODO Auto-generated method stub
        System.out.println(b);
    }

}
```

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

    MultipleInheritance v1 =new MultipleInheritance();
    v1.display();
    v1.show();


     }
```

Note: A.display() and B.show() by default are public methods so when implementing in the child class we must change Access modifiers to public.

Access modifiers:

- Public: everywhere
- Default: no access modifier is set. Accessible only within the package. (Classes in pkg)
- Protected: accessible within the package and outside of the package only through inheritance
- Private: Only within the class

Access Levels

	default	public	private	protected
Same Class	YES	YES	YES	YES
Same package subclass	YES	YES	NO	YES
Same package non-subclass	YES	YES	NO	YES
Different package subclass	NO	YES	NO	YES
Different package non-subclass	NO	YES	NO	NO

Access Modifier	within the class	within the package	outside of the package
private	YES	NO	NO
default	YES	YES	NO
protected	YES	YES	YES (through inheritance)
public	YES	YES	YES

Exception handling

Exception is an event which will terminate program unexpectedly.

There are two type exceptions:

1. Checked Exception: Identified by Java Compiler before execution.

   Ex IOException, FileNotFound Exception ……

2. Unchecked Exception: Exceptions which are not identified with java compiler.

   ArithmeticException, nullPointer Exception, ArrayOutofBound Exception, …...

```
ArrayIndexOutOfBoundException


try
{
// specify the statement which causes exception
}

catch(Exception Type)
{
//write the code
}

finally
{
//some code
}

1) Exception occurs, catch block handles, finally block also execute
2) Exception occurs, catch block not handles, finally block execute
3) Exception not occurs, catch block will ignore, finally block execute
```

```java
// handling ----------------------------------------
int [] arr = new int[5];
try {
    arr[100] = 10;

}catch(ArrayIndexOutOfBoundsException e){
    System.out.println(e.getMessage());
}
//--------------------------------------------------

String s = null;
try {
    s.length();
}catch(NullPointerException e) {
    System.out.println(e.getMessage());
}
//--------------------------------------------------
int j = 0;
try {
        System.out.println(j/0);
}catch (ArithmeticException e) {
    System.out.println(e.getMessage());
}

finally {
    System.out.println("problem");
}
```

For checked Exception we have two methods to handle an Exception

```
Checked Exceptions
------------------
InterruptedException
FileNotFoundExeption
IOException

there are two methods to handle exceptions
------------------------------------------
1) try..catch block ( handling checked and un-checked exceptions)
2) throws keyword ( only for checked Exceptions)
```

	Un-checked	Checked	Method Level	Statement
try..catch	YES	YES	no	YES
throws	no	YES	YES	no