Inheritance: Access Control, Introduction to Inheritance, Types of Inheritance, Using super, Method Overriding and Dynamic Method Dispatch, Using final, Abstract Classes.

Interfaces: Defining and Implementing Interfaces.

Packages: Creating Packages, Importing Packages, Importance of CLASSPATH.

Exception Handling, Importance of try, catch, throw, throws and finally Block.

Inheritance:

Inheritance is the process of acquiring properties (data members) and functionalities(methods) of one class to another class. It is an important principle of object-oriented programming. Inheritance helps for code reusability.

'extends is the keyword in java to inherit one class functionality to another'

class B extends **A**

here all the properties and functions under class A will be inherited to class B. Along with the inherited, class B can have its own additional properties and functions.

here, class A is called parent class and class B is called child class.

Parent Class:
The class whose properties and functionalities are used(inherited) by another class is known as parent class, super class or Base class.

Child Class:
The class that extends the features of parent class is known as child class, sub class or derived class.

Define a class called calculator, given a two integer values it should perform

i) addition
ii) subtraction
iii) multiplication.

```
import java.util.Scanner;
class Caliculator
{
        int a, b;
        public void add(int i, int j)
        {
                int sum = i+j;
                System.out.println("addition is "+ sum);
        }
        public void substraction(int i, int j)
        {
                int difference = i-j;
                System.out.println("substraction is "+ difference);
        }
        public void multiply(int i, int j)
        {
                int multiple = i*j;
```

```
			System.out.println("mulitplication is "+ multiple);
		}
		public static void main(String args[])
		{
			Caliculator obj = new Caliculator();
			Scanner sc = new Scanner(System.in);
			System.out.print("Enter value for a: ");
			obj.a =sc.nextInt();
			System.out.print("Enter value for b: ");
			obj.b =sc.nextInt();

			// perform caliculator operations in these two values
			obj.add(obj.a, obj.b);
			obj.substraction(obj.a, obj.b);
			obj.multiply(obj.a, obj.b);
		}
}
```

OuPut:

```
C:\Users\UMA SHANKAR\Desktop>javac Caliculator.java

C:\Users\UMA SHANKAR\Desktop>java Caliculator
Enter value for a: 8
Enter value for b: 6
addition is 14
substraction is 2
mulitplication is 48
```

Define a class called ScientificCalculator, given a two integer values it should perform

i) addition
ii) subtraction
iii) multiplication.
iv) power of
v) modulus
vi) division

we have already written first three operations in class Calculator why to write that code again in class ScientificCaliculator ? can’t we reuse that code ?

‘Yes’ extend class Calculator in class ScientificCalculator by which it acquires instance variables a,b and three functionalities add, subtraction and multiply.

```
import java.util.*;
class ScientificCaliculator extends Caliculator
{
		public static void main(String args[])
		{
			ScientificCaliculator ob = new ScientificCaliculator();
			Scanner sc = new Scanner(System.in);
			System.out.print("Enter value for a: ");
```

```
                ob.a = sc.nextInt();
                System.out.print("Enter value for a: ");
                ob.b = sc.nextInt();
        // perform all operations
                ob.add(ob.a, ob.b);
                ob.substraction(ob.a, ob.b);
                ob.multiply(ob.a, ob.b);
                ob.power(ob.a, ob.b);
                ob.modulus(ob.a, ob.b);
                ob.division(ob.a, ob.b);
        }
        public void power(int i, int j)
        {
                double sqr = Math.pow(i,j);
                System.out.println("power is "+ sqr);
        }
        public void modulus(int i, int j)
        {
                int mod = i%j;
                System.out.println("modulus is "+ mod);
        }
        public void division(int i, int j)
        {
                double div = i/j;
                System.out.println("division is "+div);
        }
}
```

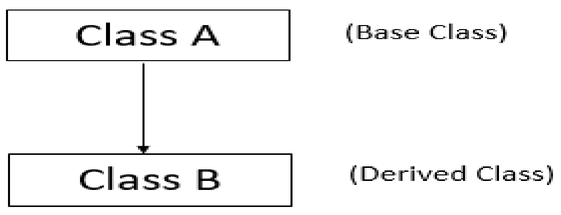
OutPut:

```
C:\Users\UMA SHANKAR\Desktop>java ScientificCalciculator
Enter value for a: 3
Enter value for a: 2
addition is 5
substraction is 1
mulitplication is 6
power is 9.0
modulus is 1
division is 1.0
```

Types of Inheritance:

Single Inheritance:

In Single Inheritance, there will be only two classes, and one class extends another class.



Example

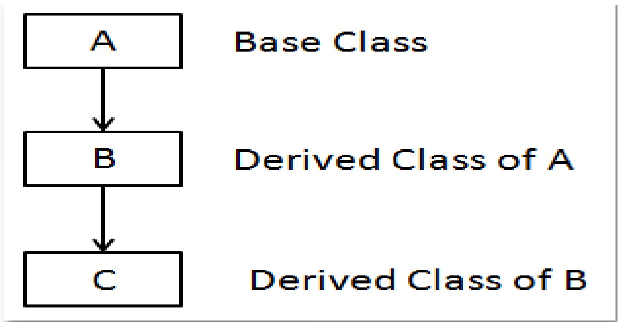
```
Class A
{
  public void methodA()
  {
    System.out.println("methodA of class A");
  }
}

Class B extends A
{
  public void methodB()
  {
    System.out.println("methodB of class B");
  }
  public static void main(String args[])
  {
    B obj = new B();
    obj.methodA();        //calling super class method
    obj.methodB();        //calling local method
 }
}
```

<u>OutPut</u>

```
methodA of class A
methodB of class B
```

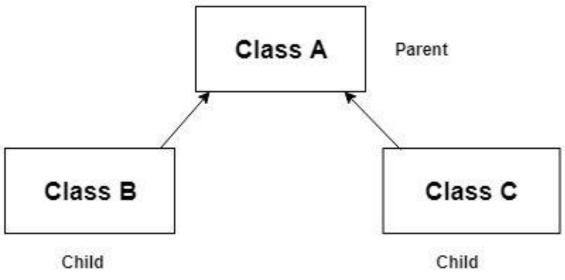
<u>Multilevel inheritance:</u>

In Multilevel Inheritance, a class inherits from a derived class. Hence, the derived class becomes the base class for the new class. refer the diagram.



<u>Hierarchical Inheritance:</u>

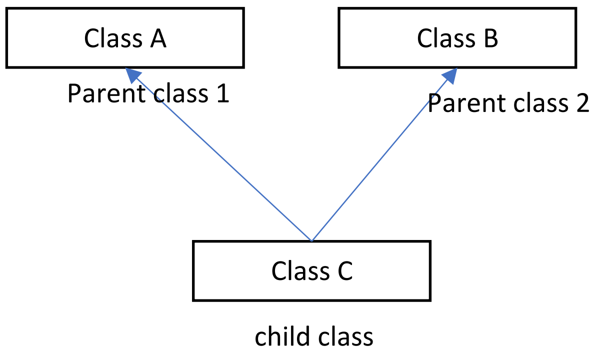
In Hierarchical Inheritance, one class is inherited by many sub classes.



*** <u>Java doesn't support following two types of inheritances</u>**

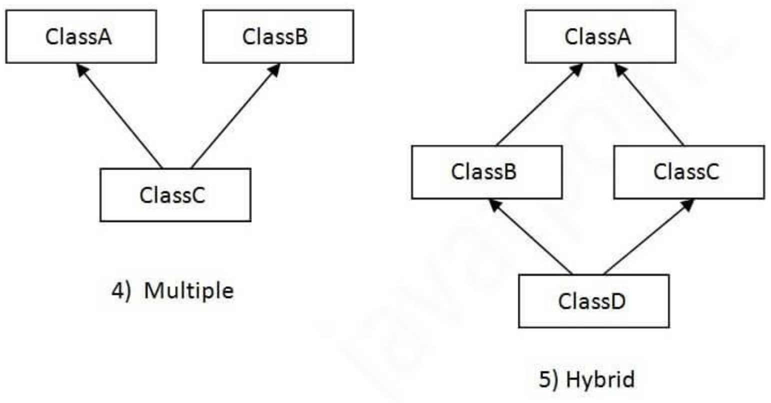
<u>Multiple Inheritance:</u>

In Multiple Inheritance, a class will extend more than one class. Java does not support multiple inheritance.



<u>Hybrid Inheritance:</u>

Hybrid inheritance is a combination of Hierarchical and Multiple inheritance.



To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call which method either of class A or B.

Java raises compile-time error if you try to inherit 2 classes. So, whether you have same method or different, there will be compile time error.

<u>Method Overriding:</u>

If subclass (child class) has the same method as declared in the parent class, then the function is said to be overridden in child class. When we create on object and try to invoke the method with same name, method from child class will be invoked. It is also called as dynamic polymorphism.

write a program to explain how method overriding takes place between parent and child class

```java
class Parent
{
	void display()
	{
		System.out.println("method display of class Parent");
	}
}

public class Child extends Parent
{
	void display()
	{
		System.out.println("method display of class Child");
	}
	public static void main(String args[])
	{
		Child ch = new Child();
		ch.display(); // calls child display method
	}
}
```

OUTPUT:

```
method display of class Child
```

Note: When we want to access parent class overridden functions we can do, using super.

super Keyword:

The **super** keyword in java is a reference variable that is used to refer parent class objects. when a derived class and base class have members (either instance variables or functions) with same name. JVM will always pick child class member to execute. In such a scenario If we want to access parent class member's we can use super reference in child class to refer parent class members.

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

1. Write program to explain accessing parent class instance overridden(variable with same name in parent and child class) variable from child class.

```
class Parent
{
//parent class instance variable
	int var=54;
}

public class Child extends Parent
{
//child class instance variable
	int var=74;
	public static void main(String[] args)
	{
		Child obj = new Child();
		obj.display();
	}
	void display()
	{
		System.out.println(var); //refers child class var
		System.out.println(super.var); // refers parent class var
	}

}
```

OUTPUT:

```
74
54
```

2. Write program to explain accessing parent class overridden method (method with same name in parent and child class) from child class.

```
class Parent
{
//parent class print method
	void print()
	{
		System.out.println("print method of parent class");
	}
}

public class Child extends Parent
{

	public static void main(String[] args)
	{
		Child obj = new Child();
		obj.display();
	}
	void display()
	{
		print(); // executes child class print method
		super.print(); // executed parent class print method
	}
//child class print method
	void print()
	{
		System.out.println("print method of child class");
	}

}
```

OUTPUT:

```
print method of child class
print method of parent class
```

3. Write a program to invoke parent class no argument constructor from child class using super

```
class Parent
{
//parent class constructor
	Parent()
	{
		System.out.println("no argument constructor of parent
		class");
	}
}

public class Child extends Parent
{
//child class constructor
	Child()
	{
		super(); // calls parent class no argument constructor
			System.out.println("no argument constructor of child
			class");
	}
	public static void main(String[] args)
	{
		Child obj = new Child(); // calls child class constructor
	}
}
```

OUTPUT:

```
no argument constructor of parent class
no argument constructor of child class
```

4. Write a program to invoke parent class constructor with arguments from child class using super

```
class Parent
{
//parent class constructor
      int a,b;
      Parent()
      {
            System.out.println("parent class constructor with out
                                    arguments");
      }
      Parent(int i, int j)
      {
            a = i;
            b = j;
            System.out.println("parent class instance variables are
                                    initialized");
            System.out.println("a = "+a+" b = "+b);
      }
}

public class Child extends Parent
{
//child class constructor
      Child()
      {
            super(5,4); //calls parent class constructor with arguments
            System.out.println("If we use super it should be first
                               statement in constructor");
      }
      public static void main(String[] args)
      {
            Child obj = new Child(); // calls child class constructor
      }
}
```

OUTPUT:

```
parent class instance variables are initialized
a = 5 b = 4
If we use super it should be first statement in constructor.
```

Final keyword:

The **final keyword** in java is used to restrict the access to parent class variables and methods, and even to prevent the class from being inherited.

A Final can be applied for variables, methods, and class.

Java final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

Write a program using final variable, after initializing it with a value. try to update the variable again. mention the compile time error that occurs while modifying final variable.

```
class Parent
{
	final int i =65;
	void display()
	{
		i = 60;
		System.out.println(i);
		System.out.println("method display of class Parent");
	}
	public static void main(String args[])
	{
		Parent ob = new Parent();
		ob.display();
	}

}
```

Error :
java:6: error: cannot assign a value to final variable i
 i = 60;
Write a program, with method declared final and try to override that in child class. mention compile time error while you do that.

Note : we cannot override a final method in child class

```
class Parent
{
	final void display()
	{
		System.out.println("This method cannot be overriden in
				child class");
	}
}

public class Child extends Parent
{
	int i = 80;
	void display()
	{
		System.out.println("this will not executed, as we cannot
		                        override final method");
	}
	public static void main(String args[])
	{
		Child ch = new Child();
		ch.display(); // calls child display method
	}
}
```

Error:

```
Child.java:13: error: display() in Child cannot override display() in Parent
       void display()
            ^
  overridden method is final
1 error
```

Write a program to extend a class that is final, mention the error while you obtain while implementing that

Note: We cannot extend a final class.

```
final class Parent
{
	final void display()
	{
		System.out.println("This method cannot be overriden in
		                        child class");
	}
}

public class Child extends Parent
{
	public static void main(String args[])
	{
		Child ch = new Child();
	}

}
```

```
Child.java:9: error: cannot inherit from final Parent
public class Child extends Parent
                           ^
1 error
```

Abstract classes

Abstarct method: A method which is declared as abstract and does not have implementation is known as an abstract method.

abstract void printStatus(); // no functionality will be defined when declared abstract.

Abstract class:

A class which is declared abstract, is known as an abstract class. It can have both abstract and non-abstract methods.

1. An abstract class must be declared with an abstract keyword.
2. It can have abstract and non-abstract methods.
3. object cannot be created for abstract classes.
4. It can have constructors and static methods.
5. It can have final methods which will force the subclass not to change the body of the method.

NOTE : To use the abstract class we must **extend** the abstract class, and need to define the functionality for its abstract methods and we can use the inherited non abstract methods of it.

Write a program explaining how to use abstract and non-abstract methods of an abstract class.

```
abstract class Parent
{
	void method1()
	{
		System.out.println("this is a non abstract method class 
				Parent");
	}
// abstract methods will not have any functionality
	abstract int method2(int a, int b);
}
class Child extends Parent
{
	public static void main(String args[])
	{
		Child ob = new Child();
		ob.method1();
		int sum = ob.method2(8,9);
		System.out.println("sum of two values is "+sum);
	}
	int method2(int i, int j)
	{
		return i+j;
	}

}
```

OUTPUT:

```
this is a non abstract method class Parent
sum of two values is 17
```

Note : If we extend an abstract class, we must implement all the abstract methods of the parent abstract class. otherwise we should declare implementing class also as abstract.

Interfaces:

An interface is a class declared by keyword interface, in interface all the methods will be abstract.

1. You cannot instantiate an interface.
2. An interface does not contain any constructors.
3. All of the methods in an interface are abstract.
4. An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
5. An interface is not extended by a class, it is implemented by a class.
6. An interface can extend multiple interfaces.

Interface declaration

```
interface interfaceName
{
        method declaration      // abstract, by default
}
```

Note:

1. An interface is implicitly abstract. You do not need to use the **abstract** keyword while declaring an interface.
2. Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
3. Methods in an interface are implicitly public

Define an interface which specifies there should be functions called sum and multiply that accepts two integer numbers and returns sum and multiplication values respectively. also write a class to implement that interface.

```java
interface InterfaceOne
{
	int sum(int a, int b);
	int multiply(int i, int j);
}

class Impl implements InterfaceOne
{
	public int sum(int a1, int a2)
	{
		int sum;
		sum = a1+a2;
		return sum;
	}
	public int multiply(int b1, int b2)
```

```
        {
                int mul;
                mul = b1 * b2;
                return mul;
        }
        public static void main(String args[])
        {
                Impl obj = new Impl();
                int r1 = obj.sum(2,4);
                System.out.println("sum of values is "+r1);
                int r2 = obj.multiply(3, 4);
                System.out.println("multiplication of values is "+r2);
        }
}
```

OUTPUT:

```
sum of values is 6
multiplication of values is 12
```

NOTE:

1. A class can implement more than one interface at a time.
2. A class can extend only one class, but implement many interfaces.

Extending Interfaces

An interface can extend another interface in the same way that a class can extend another class. The extends keyword is used to extend an interface, and the child interface inherits the methods(abstract) of the parent interface.

Packages:

packages in Java are used to group related classes together. name conflicts can also be resolved using packages and helps in maintaining readability of code. Java uses file system directory for packages.

Types of packages:

1. **Built in packages**
2. **user defined packages**

Built in packages

classes which are a part of Java **API**, are called built in packages.

example: **java.lang, java.io, java.util, etc,.**

User-defined packages -These are the packages defined by the user.

creating a package:

First, we create a folder say, **myPackage** (name should be same as the name of the package we want create). Then create any class say, **MyClass** inside that directory. The first statement should be **package packagename**.

create a class A under package firstpackage, In class A write a print statement in display method as "first program on packages"

First create a directory called firstpackage (create a folder and name it as package name-firstpackage). The first statement in any java file under this package should be

package firstpackage;

```
package firstpackage;
class A
{
	public static void main(String args[])
	{
		display();
	}
	public void display()
	{
		System.out.println("first program on packages");
	}
}

compile
E:\MyApp\src>javac firstpackage/A.java

Note: compile the program from root folder
```

write a program with class B, in package secondpackage. import class A into B and execute display method of A.

```
package secondpackage; //class B belongs to second package

import firstpackage.A;  // importing class A from firstpackage
class B
{
	public static void main(String args[])
	{
		A obj = new A();
		obj.display();
	}

}
E:\MyApp\src>javac secondpackage/B.java
```

There are three ways to access the package from outside the package.

1. import package.*;

   example: import firstpackage.* (all classes inside this package will be imported)
2. import package.classname;

   import firstpackage.A;
3. fully qualified name.

   wherever we access class A, access it through firstpackage.A;

subpackages:

we can create packages inside a package known as sub packages.

```
package firstpackage.p1;
public class inner {

	public void display()
	{
		System.out.println("from method display inner class");
	}
}
```

```
E:\MyApp\src>javac firstpackage/p1/inner.java
```

here the first statement `package firstpackage.p1;`

represents class ‘inner’ belongs to package p1, where p1 is a sub package of firstpackage.

If we want to access this class inner in any other package, we need to import from root package. i.e., import package2.p1.inner observe the following program

```
package package3; //MyClass belongs to package3

import package2.p1.inner; //importing class inner

public class MyClass {

	public void display1()
	{
		System.out.println("display of MyClass in package3");
	}
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		inner obj = new inner();
		obj.display();
	}

}
```

```
E:\MyApp\src>javac package3/MyClass.java
```

Executing MyClass

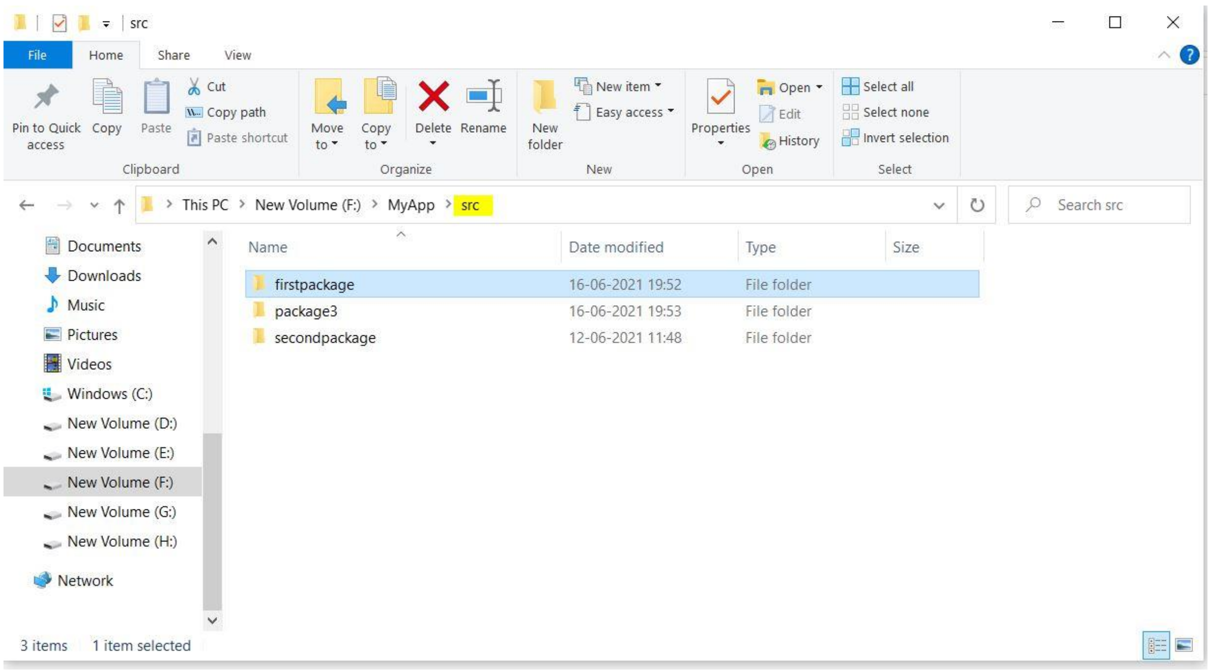
```
E:\MyApp\src>java package3.MyClass
```

OUTPUT:

```
from method display inner class
```

Here package package3; statement represents MyClass belongs to package3

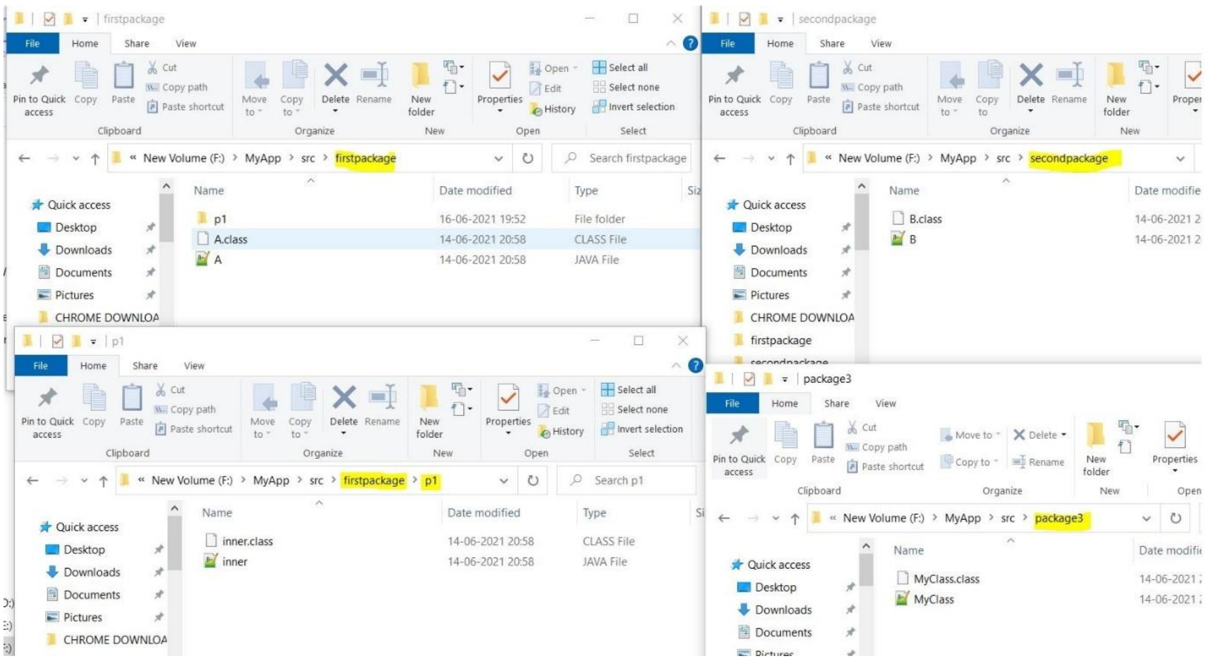
second statement import package2.p1.inner represents we import the class inner.



This PC > New Volume (F:) > MyApp > src

Name	Date modified	Type	Size
firstpackage	16-06-2021 19:52	File folder	
package3	16-06-2021 19:53	File folder	
secondpackage	12-06-2021 11:48	File folder	

3 items 1 item selected



« New Volume (F:) > MyApp > src > firstpackage

Name	Date modified	Type
p1	16-06-2021 19:52	File folder
A.class	14-06-2021 20:58	CLASS File
A	14-06-2021 20:58	JAVA File

« New Volume (F:) > MyApp > src > secondpackage

Name	Date modifie
B.class	14-06-2021 2
B	14-06-2021 2

« New Volume (F:) > MyApp > src > firstpackage > p1

Name	Date modified	Type
inner.class	14-06-2021 20:58	CLASS File
inner	14-06-2021 20:58	JAVA File

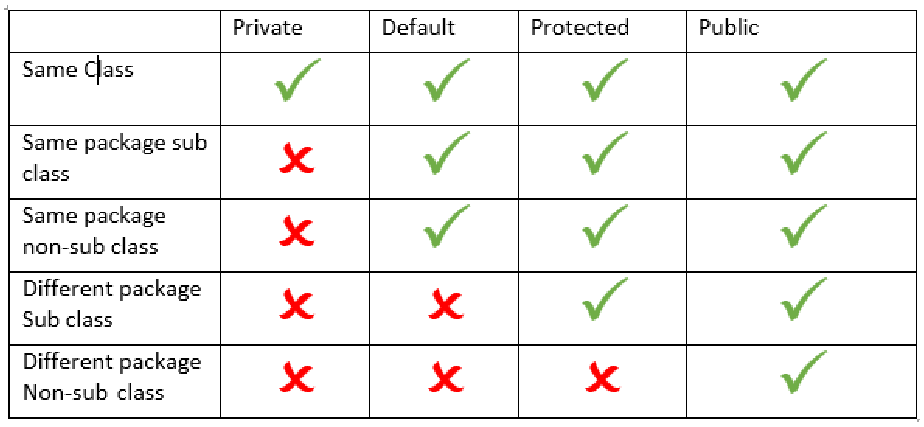
« New Volume (F:) > MyApp > src > package3

Name	Date modifie
MyClass.class	14-06-2021
MyClass	14-06-2021

Access Control / Access modifiers

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.



	Private	Default	Protected	Public
Same Class	✓	✓	✓	✓
Same package sub class	✗	✓	✓	✓
Same package non-sub class	✗	✓	✓	✓
Different package Sub class	✗	✗	✓	✓
Different package Non-sub class	✗	✗	✗	✓

CLASSPATH:

CLASSPATH is an environment variable in Java, and tells Java applications and the Java Virtual Machine (JVM) where to find the classes that we use in the program.

setting class path from command prompt

Example:

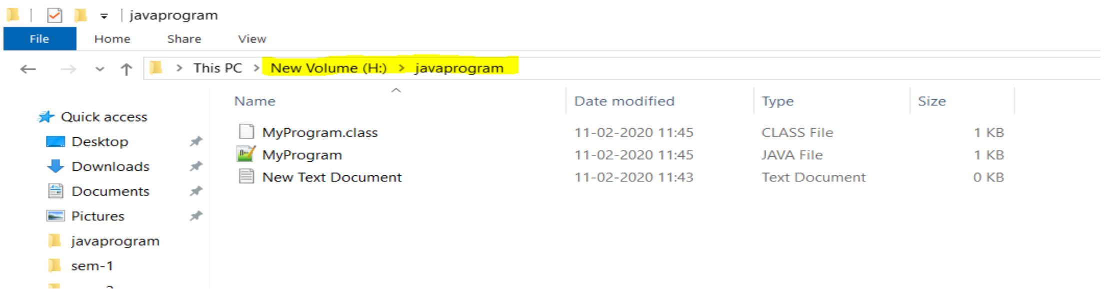
SET CLASSPATH=%CLASSPATH%;H:\javaprogram

Write a program to explain how classes existing in classpath location can be used directly without importing.

```
class MyProgram
{
	public void display()
	{
		System.out.println("from display method of class 
					myprogram");
	}
}
```

This java file exists in the following path

H:\javaprogram



Name	Date modified	Type	Size
MyProgram.class	11-02-2020 11:45	CLASS File	1 KB
MyProgram	11-02-2020 11:45	JAVA File	1 KB
New Text Document	11-02-2020 11:43	Text Document	0 KB

If we access this class MyProgram from another directory, generally we need to import. but if the path of class MyProgram is given in classpath variable JVM can load the class by checking locations specified in classpath and can able to load and execute.

```
public class ClassPathExample {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		MyProgram obj = new MyProgram();
		obj.display();
	}

}
```

the above ClassPathExample, uses MyProgram class and calling it's method display without importing. If classpath is not set with the location of MyProgram, JVM will not be able to fetch and load this MyProgram class and results in following error.

```
F:\example>javac ClassPathExample.java
ClassPathExample.java:6: error: cannot find symbol
                MyProgram obj = new MyProgram();
                ^
  symbol:   class MyProgram
  location: class ClassPathExample
```

after setting classpath using command prompt, as
F:\example>SET CLASSPATH=%CLASSPATH%;H:\javaprogram
(or)
setting classpath variable, with values as location of MyProgram class

JVM will not raise any error, now JVM can load the MyProgram class file by checking out the location specified in classpath variable

F:\example>javac ClassPathExample.java

F:\example>java ClassPathExample

OUTPUT:

```
from display method of class myprogram
```

Exceptions

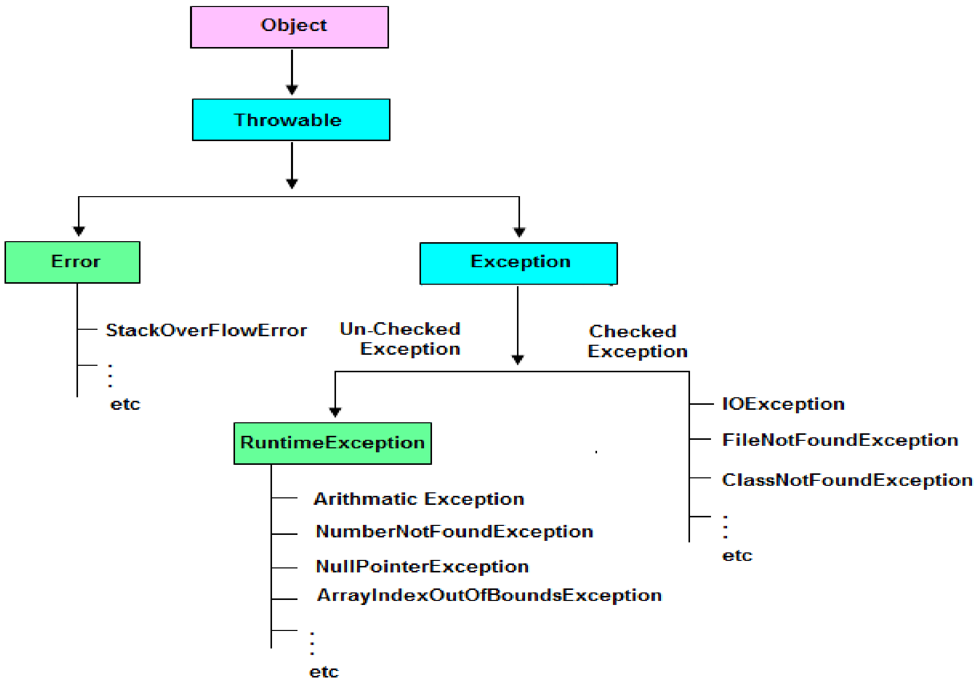
Exception in Java is an event that interrupts the execution of program-instructions and terminates the execution abnormally.

why we need exception handling

Exception handling ensures that the flow of the program doesn't break when an exception occurs. For example, if a program has large no of statements and an exception occurs in middle after executing certain statements then the statements after the exception will not be executed and the program will terminate abruptly. By handling we make sure that all the statements execute and the flow of program doesn't break.

Exceptions are mainly categorized into

- Checked Exceptions
- unchecked Exceptions



Checked exceptions/Compile time exceptions – A checked exception is an compile time exception that will be checked by the compiler. These exceptions cannot simply be ignored, the programmer should take care of (handle) these exceptions.

Example - FileNotFoundException

Unchecked exceptions/Runtime Exceptions – An unchecked exception is an exception that occurs at the time of execution. Runtime exceptions will not be checked by the compiler. It is the programmer responsibility to handle these exceptions.

Example – ArithmeticException, ArrayIndexOutOfBoundsException

Note : Errors doesn’t come under the category of Exceptions

Errors can be categorized to

1. Compile time errors – Any syntax or semantic error in a program
   example –
   flt a; -> instead of float a;
   a = b+c , missing semicolon

2. Runtime errors

Erros / Runtime errors – Errors are the conditions from which application cannot get recovered by any handling techniques. It will cause termination of the program abnormally. Errors occur at runtime.
example:

StackOverflowError
UnSupportedClassVersionError
VirtualMachineError

Java provides exception handling with five keywords:
try, catch, throw, throws, and finally.

try block

Program statements that can raise exceptions should be written within a try block. If an exception occurs within the try block, it is thrown to corresponding catch block.

catch block

We can catch the exception raised in try and handle the flow of control with out letting it to terminate abruptly because of exception. We can provide any useful information to user regarding the exception.

A try block can be followed by multiple catch blocks.

throw

We can throw the exceptions manually either predefined or user defined, by using the keyword throw.

throws

When an exception causing method wants to return the Exception to its calling method it can be thrown by a throws clause and calling method should handle this exception.

finally

The finally block follows a try block or a catch block. A finally block of code always executes, irrespective of occurrence of an Exception. Using a finally block allows you to run any clean up statements that you want to execute like closing files, session or network connections

Example :
consider the following statement which leads to – ArithmeticException -divide by zero Exception

```
int res = a/b;          // There is chance of getting Exception
```

Handling the exception

```
try {
        int res = a/b ;
}
catch(ArithmeticException e)
{
        System.out.println("Exception caught : "+e);
}
finally
{
        System.out.println("statements under finally will always be executed even when
        exception is not raised");
}
```

user defined Exceptions/ custom exceptions

User Defined Exception or custom exception is creating your own exception class and raising that exception using 'throw' keyword. custom exception class can be created by extending the class Exception.

In the below example problem when amount required to with drawl exceeds the balance we are going to raise a user define exception called OutOfBalanceException

```
class Main
{
      public static void main(String args[])
      {
            Operations obj = new Operations();
            try {
            obj.withdrawl(20000);
            }
            catch(OutOfBalanceException e)
            {
                  System.out.println(e);
            }
      }
}
class OutOfBalanceException extends Exception
{
      public String toString()
      {
            return "insufficient funds can not perform withdrawl";
      }
}
class Operations
{
        static int balance = 10000;
        void withdrawl(int req)throws OutOfBalanceException
        {
                if(balance < req )
                {
                        throw new OutOfBalanceException();
                }
                else
                {
                        balance = balance - req;
                }
        }
}
```

To raise the user defined exception we an can use throw keyword

```
throw new OutOfBalanceException();
```

here the raised exception is transferred to the caller method using throws keyword, the exception should be handled there

```
try {
        obj.withdrawl(20000);
}


void withdrawl(int req)throws OutOfBalanceException
{
        throw new OutOfBalanceException();
}
```

Using parent Exception class reference for child class Exception

In this example, we generate ArithmeticException, but the catch block with corresponding exception type is not defined. In such case, the catch block containing the parent exception class **Exception** will executed.

```
public class CatchParentExceptionClass {

    public static void main(String[] args) {
            try{
                int res = 24/0;
               System.out.println(res);
              }
              catch(NullPointerException e)
                {
                 System.out.println("Arithmetic Exception
                                      occurs");
                }
              catch(ArrayIndexOutOfBoundsException e)
                {
                       System.out.println("ArrayIndexOutOfBounds
                                           Exception occurs");
                }
              catch(Exception e)
                {
                 System.out.println("Parent Exception class
                                      matched");
                }
              System.out.println("rest of the code");
        }
    }

output : Parent Exception class matched
```