edureka!

MODULE-4
PACKAGES AND MULTI THREADING

www.edureka.co/java-j2ee-soa-training

Course Topics

edureka!

Objectives

At the end of this module, you will be able to

→ Implement interface and use it

→ Extend interface with other interface

→ Create package and name it

→ Import packages while creating a new class

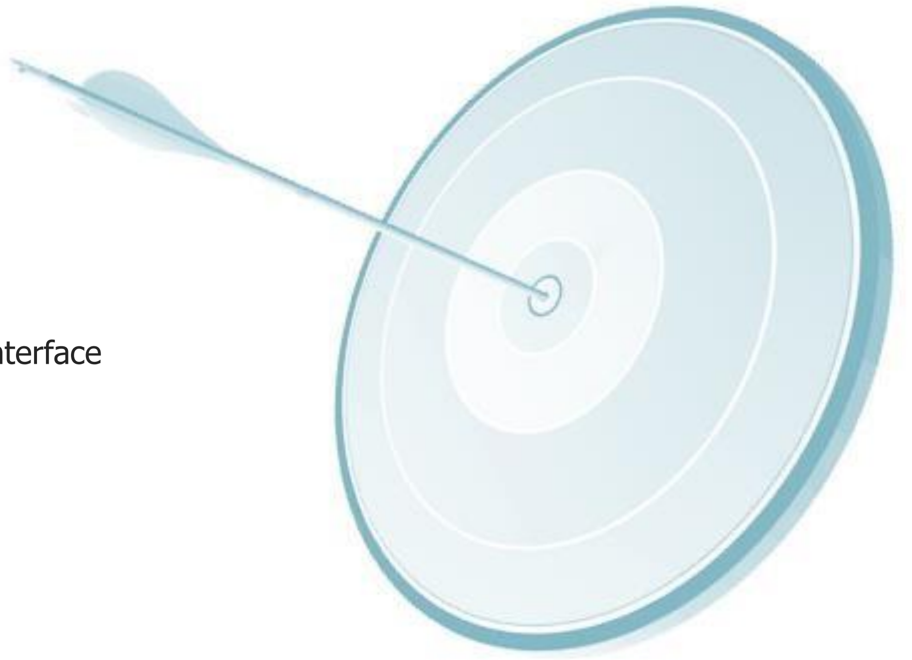
→ Understand various exceptions

→ Handle exception using try catch block

→ Handle exception using throw and throws keyword

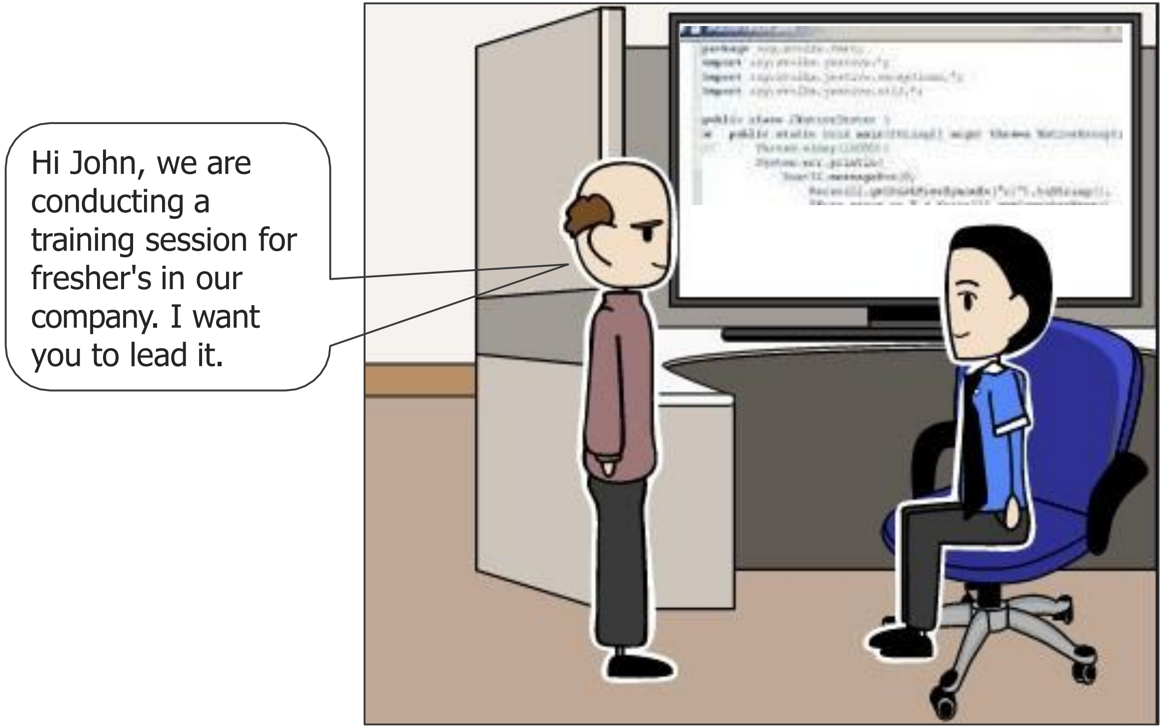
→ Implement threads using thread class and runnable interface

→ Understand and implement multithreading

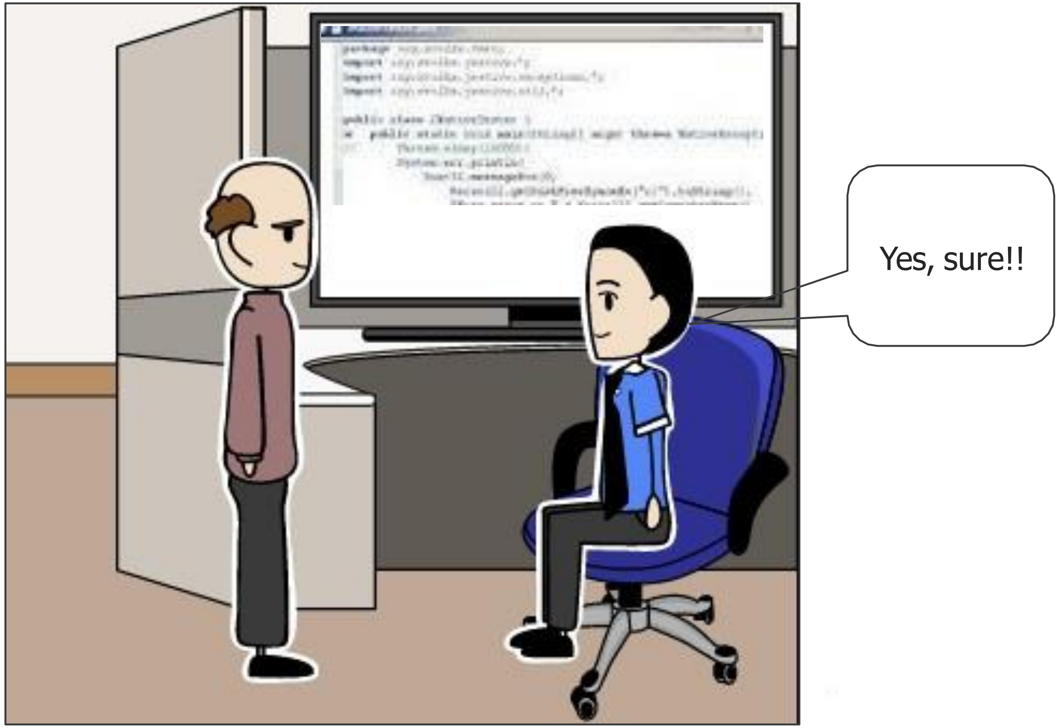


John gets a Task

edureka!

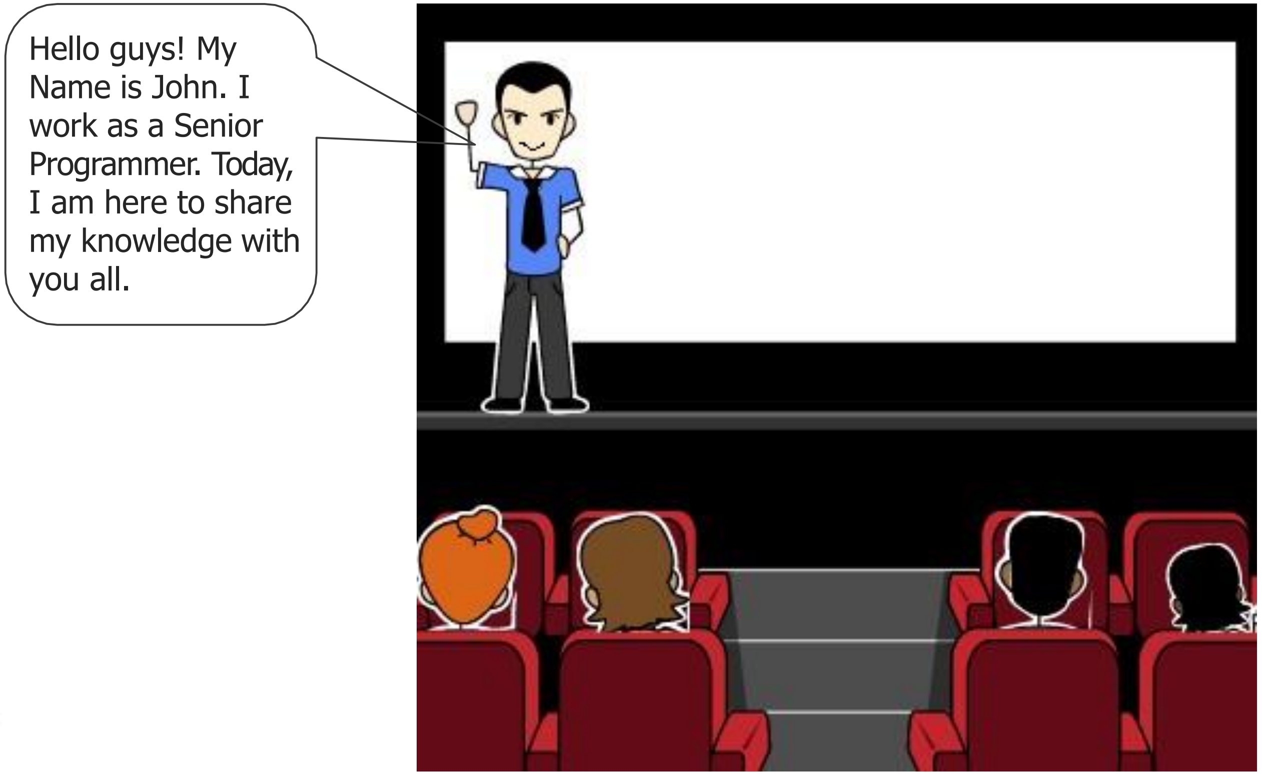

Hi John, we are conducting a training session for fresher's in our company. I want you to lead it.

John to train Fresher's!

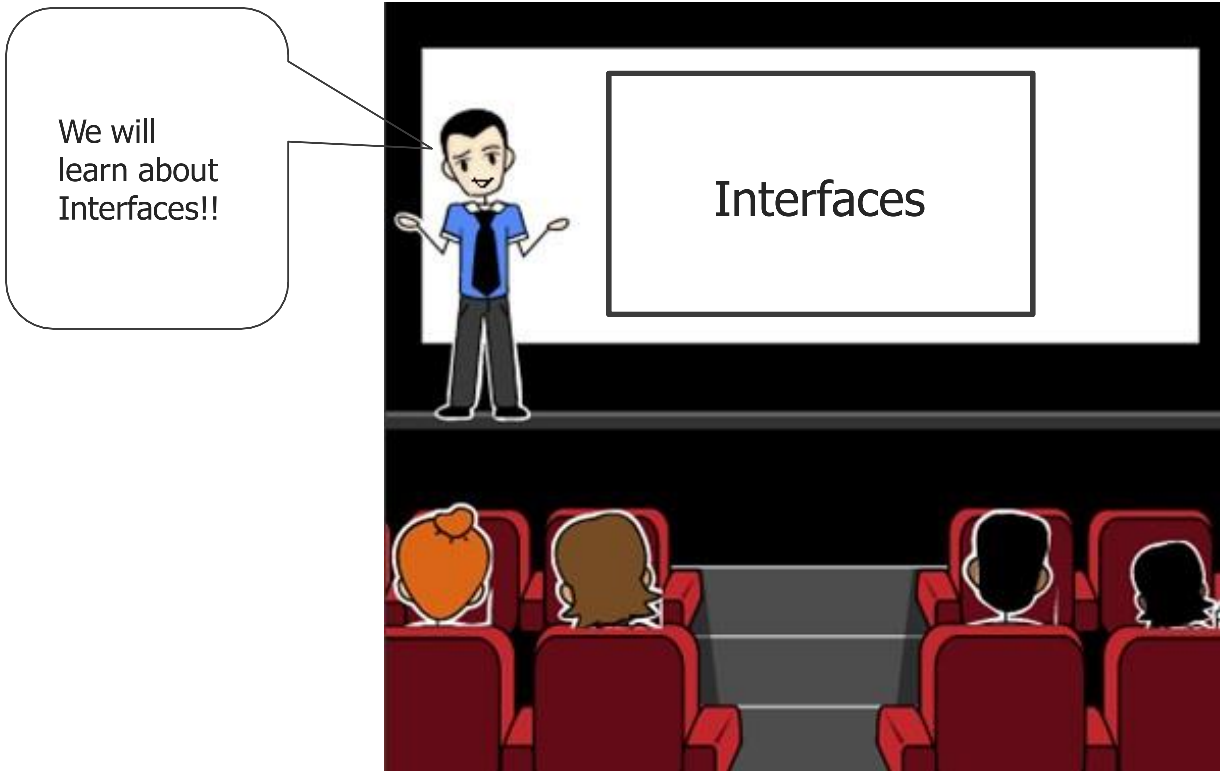


John starts Training

edureka!


Hello guys! My Name is John. I work as a Senior Programmer. Today, I am here to share my knowledge with you all.

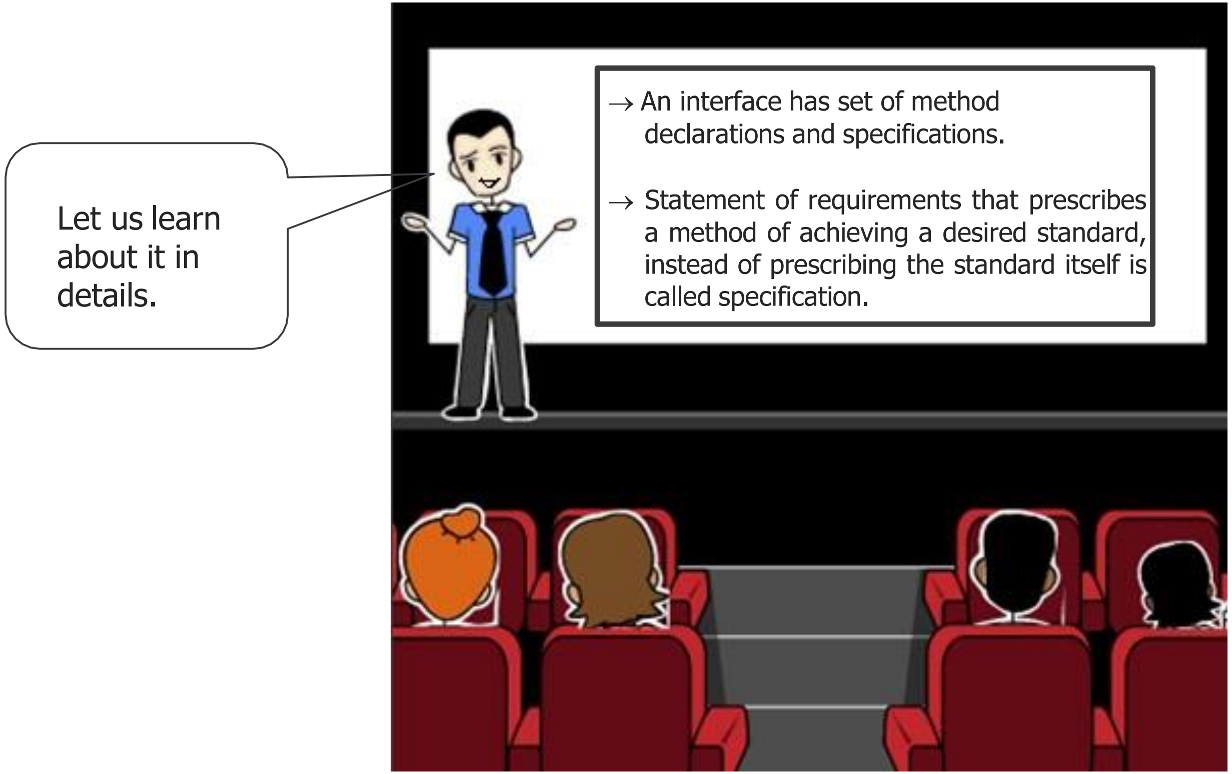
John talks about Interfaces



John explains Interface with an Example

Every remote whether it is a TV remote or gaming remote needs to have certain specifications. For example power button, directional buttons etc. which are common to every remote.
Hence an interface contains all these specification and can be used while creating a new remote.

John explains Interface and Specifications



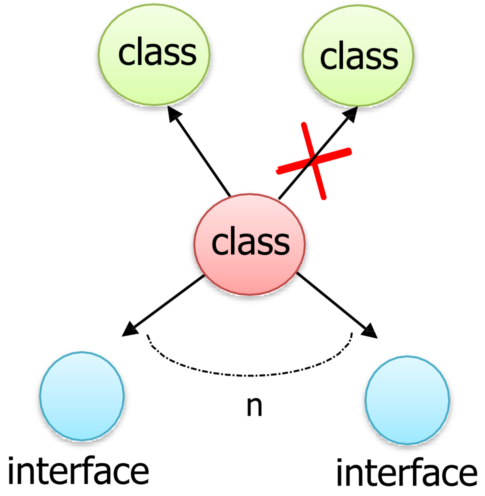
→ An interface has set of method declarations and specifications.

→ Statement of requirements that prescribes a method of achieving a desired standard, instead of prescribing the standard itself is called specification.

Why are "Interfaces" used?

→ Interfaces are used to implement the expected behaviour of a system / data type.

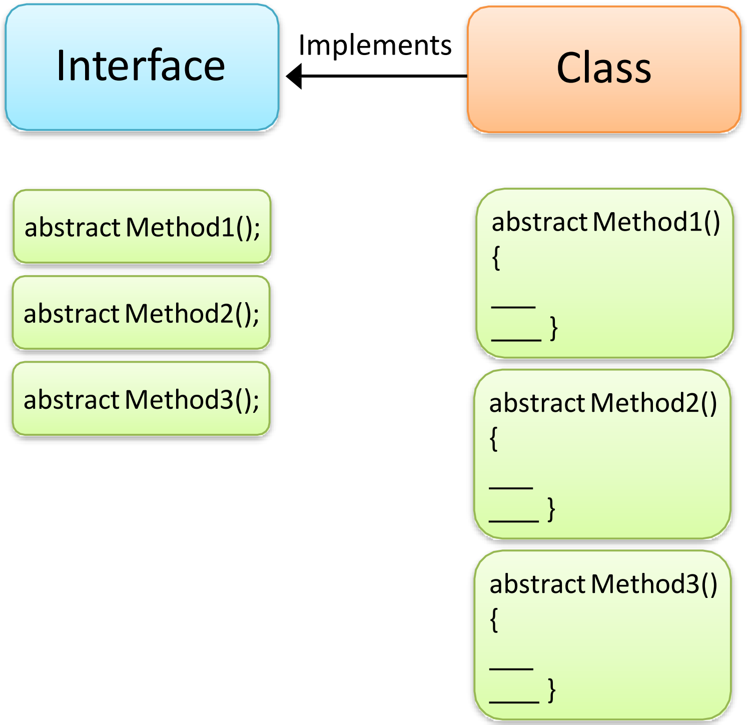
→ Java does not support multiple inheritance, hence interfaces are implemented.



A class can extend 1 class but implement many interfaces.

Interface

edureka!



→ An interface has a set of method declarations.

→ It does not have the method body but only method declaration.

→ To use an interface in a class, "implements" keyword is used.

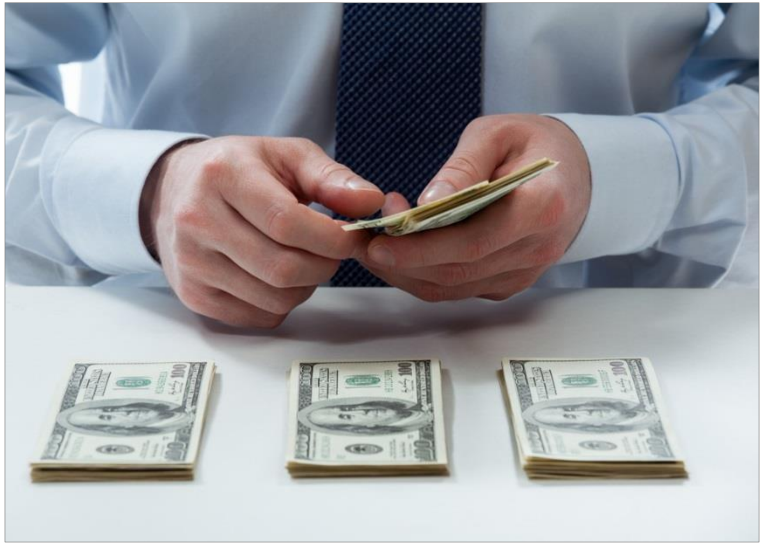
→ In case you don't override all the methods in the class the class has to be defined as abstract.

→ An interface is same as class except class can be instantiated but interface cannot be.

→ An interface can be defined by using interface keyword and the name of the interface.

  » interface interface1{}

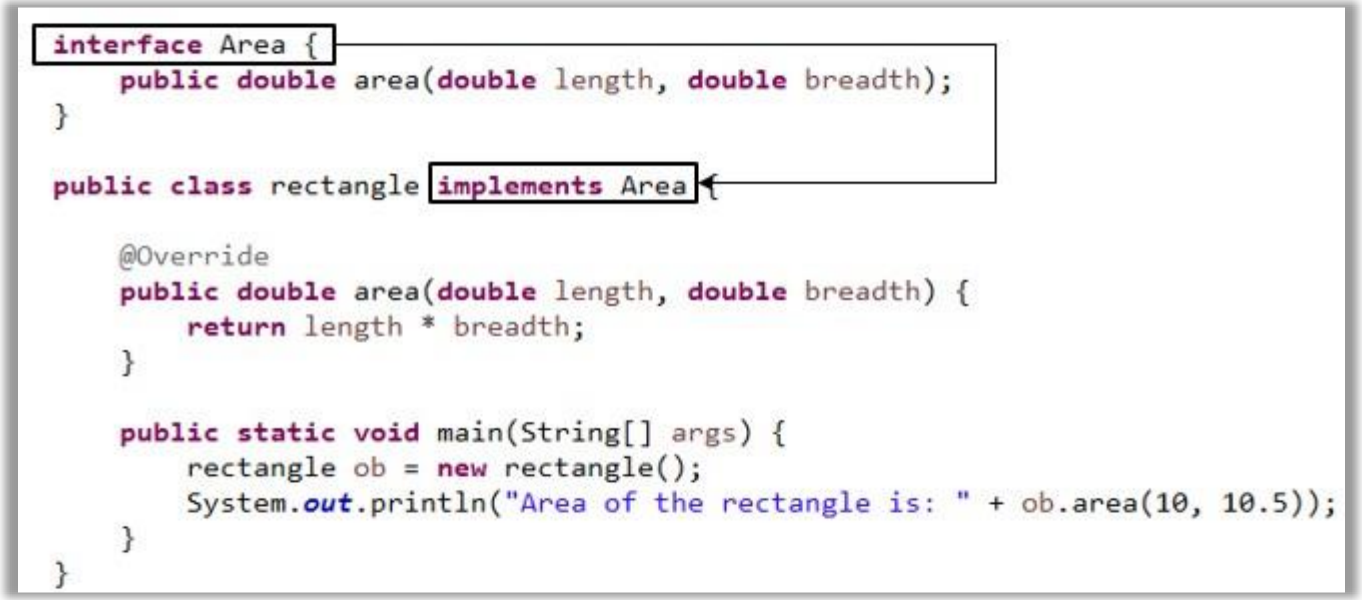
Where do we use interfaces?



Any system which needs the expected functionality, requires interfaces.

For example, 2 basic functions of a bank is deposit and withdraw. These two functions can be part of interfaces for banking application.

A Program on Interface



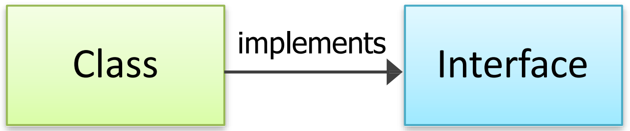
```
interface Area {
    public double area(double length, double breadth);
}

public class rectangle implements Area {

    @Override
    public double area(double length, double breadth) {
        return length * breadth;
    }

    public static void main(String[] args) {
        rectangle ob = new rectangle();
        System.out.println("Area of the rectangle is: " + ob.area(10, 10.5));
    }
}
```

Interfaces

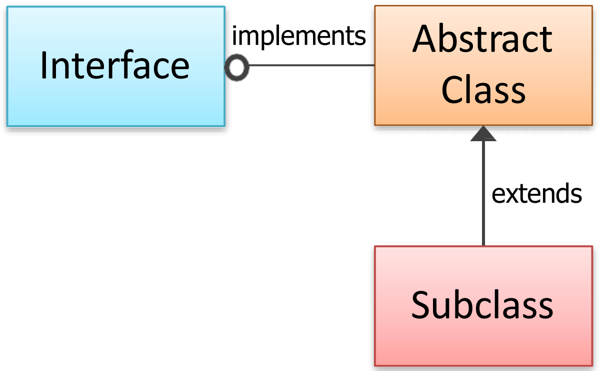
→ Attributes can be defined in interfaces.

→ These attributes can be used in the implemented class.

→ The attribute values can't be changed in the class as it acts like final variables.



Class implementing an interface



Class extending an Abstract class which implements an interface

Annie's Question

What is a final variable?

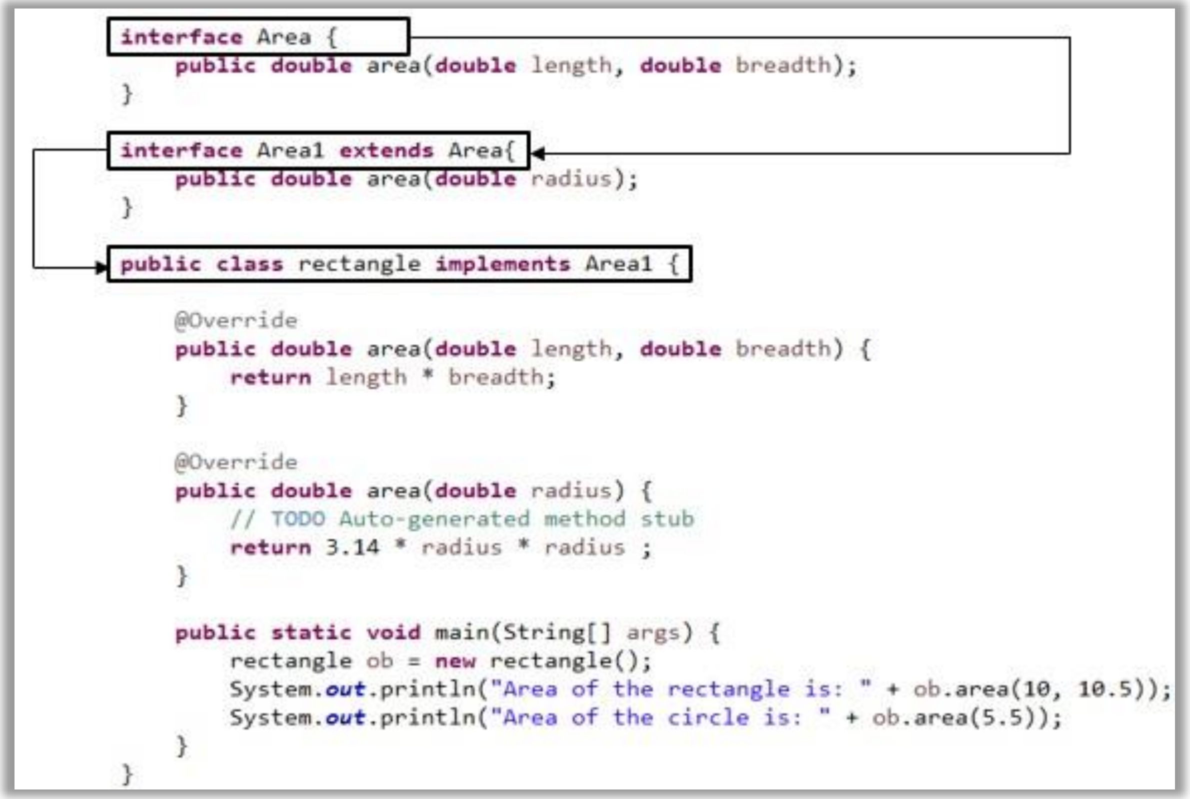
Annie's Answer

edureka!

If the variable is declared as final, the value of that variable cannot be modified.

LAB

Interfaces can be Extended



```
interface Area {
    public double area(double length, double breadth);
}

interface Area1 extends Area{
    public double area(double radius);
}

public class rectangle implements Area1 {

    @Override
    public double area(double length, double breadth) {
        return length * breadth;
    }

    @Override
    public double area(double radius) {
        // TODO Auto-generated method stub
        return 3.14 * radius * radius ;
    }

    public static void main(String[] args) {
        rectangle ob = new rectangle();
        System.out.println("Area of the rectangle is: " + ob.area(10, 10.5));
        System.out.println("Area of the circle is: " + ob.area(5.5));
    }
}
```

Interfaces can be Extended (Contd.)

edureka!

→ Like the classes can be extended, even interfaces can be extended.

→ In the above program, there are two interfaces, interface1 and interface2.

→ Interface2 is extended from interface1. If a class is implementing interface2 then all the methods of interface2 and 1 should be implemented, as interface2 is extended from interface1.

Packages

Why Packages?

→ Programmers can easily determine that these classes are related.

→ Programmers know where to find files of similar types.

→ The names won’t conflict.

→ You can have define access of the types within the package.

What is a Package?

→ A Java Package is a mechanism for organizing Java classes into namespaces

→ Programmers use package to organize classes belonging to the same category

→ Classes in the same package can access each other's package-access members

Naming Convention of a Package

Package names are written in all lower case. (It is not mandatory. However, it is standard convention that is followed)

Companies use their reversed internet domain name to begin their package names.

For example: com.example.mypackage for a package named mypackage created by a programmer at example.com

Naming Convention of a Package

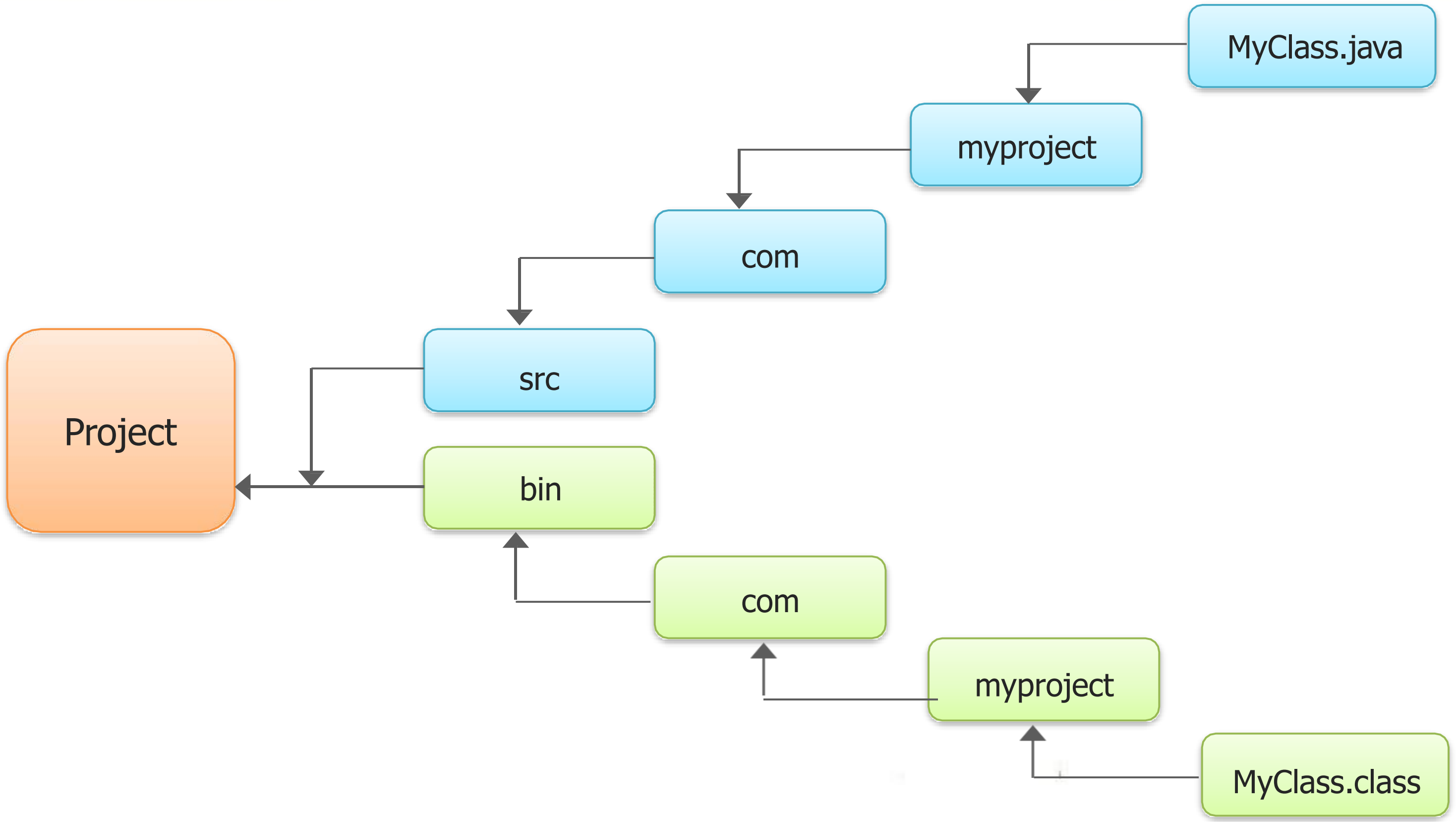
If the domain name contains:

→ a hyphen or a special character
→ If the package name begins with a digit, illegal character reserved Java keyword such as "int"

In this event, the suggested convention is to add an underscore as follows:

Legalizing Package Names	
Domain Name	Package Name Prefix
hyphenated-name.example.org	org.example.hyphenated_name
example.int	int_.example
123name.example.com	com.example_123name

Packages


MyClass.java
myproject
com
src
Project
bin
com
myproject
MyClass.class

Annie's Question

edureka!

What can be the advantage of having all the related classes in one package?

Annie’s Answer

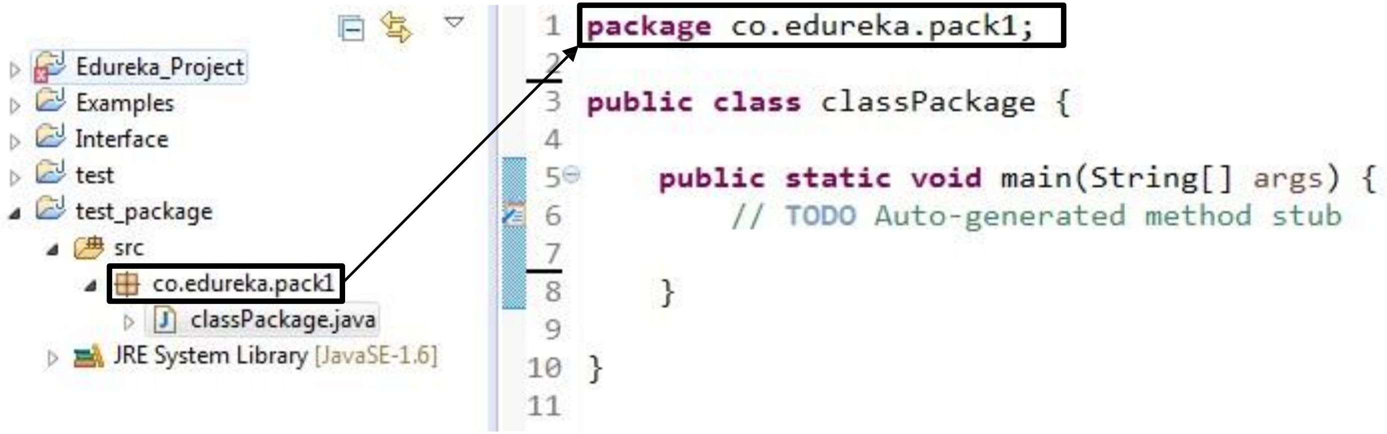
1. It is modularized.
2. Easy to handle. Easy to copy from one location to another.
3. All the classes in the package can be loaded at one time import and * character.

Program on Package

edureka!

Company URL → edureka.co

Package name → co.edureka.pack1



```java
package co.edureka.pack1;

public class classPackage {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }

}
```

Package and Import

edureka!

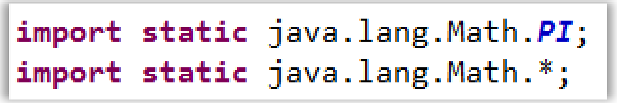
→ A package can have many classes and each class can have many methods.

→ These methods can be used by another class in another package by using the keyword "import".

→ Syntax is: import <package name>. <class name>

→ Or import <package name>.*; → This loads all the classes in the given package.

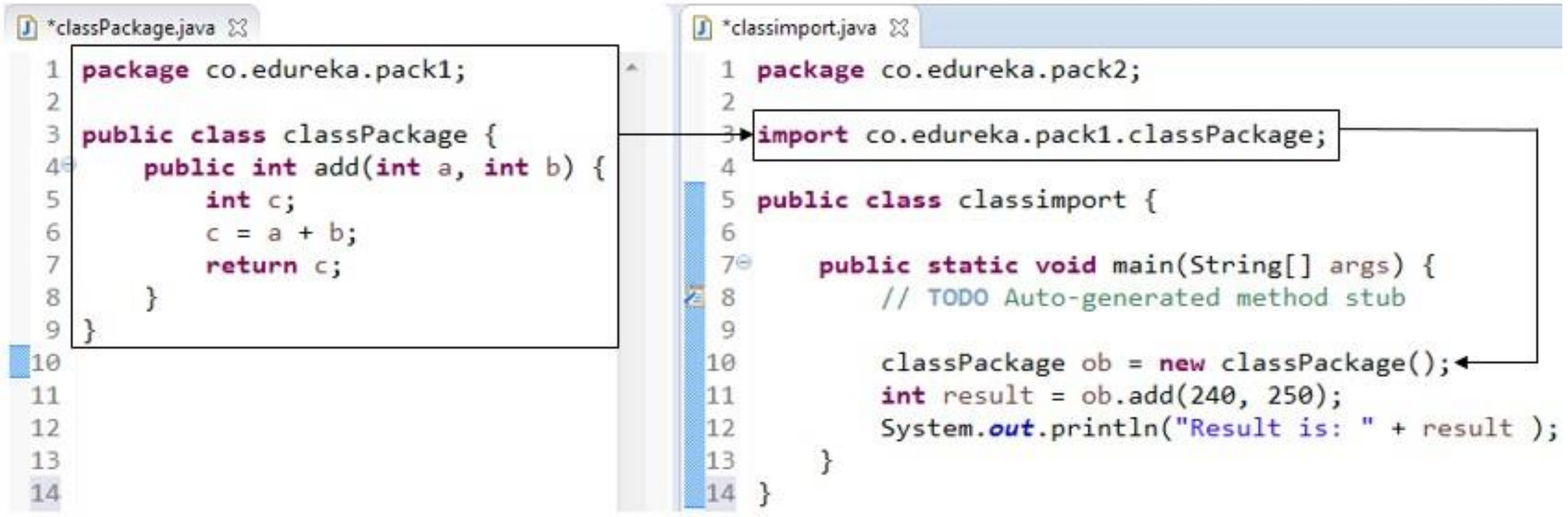
→ We can also import static members. For eg: PI, cos etc.



```
import static java.lang.Math.PI;
import static java.lang.Math.*;
```

Program on Packages

edureka!



*classPackage.java

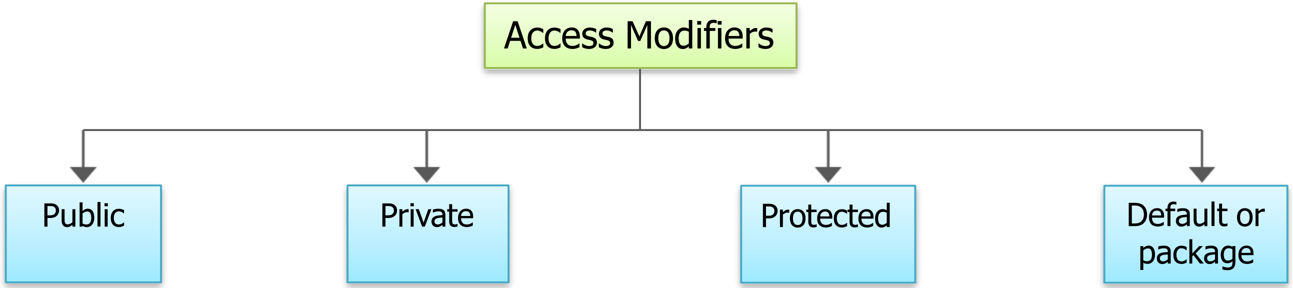
```java
package co.edureka.pack1;

public class classPackage {
    public int add(int a, int b) {
        int c;
        c = a + b;
        return c;
    }
}
```

*classimport.java

```java
package co.edureka.pack2;

import co.edureka.pack1.classPackage;

public class classimport {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        classPackage ob = new classPackage();
        int result = ob.add(240, 250);
        System.out.println("Result is: " + result );
    }
}
```

Access Modifiers

There are 4 Access Modifiers:



These Scope Modifiers can be used for an attribute or for a method.

Access Modifier

edureka!

Access Modifier specifies the scope/accessibility of a variable or a method or a class from the same class or from a different class or from a different package.

Use of Access Modifier:

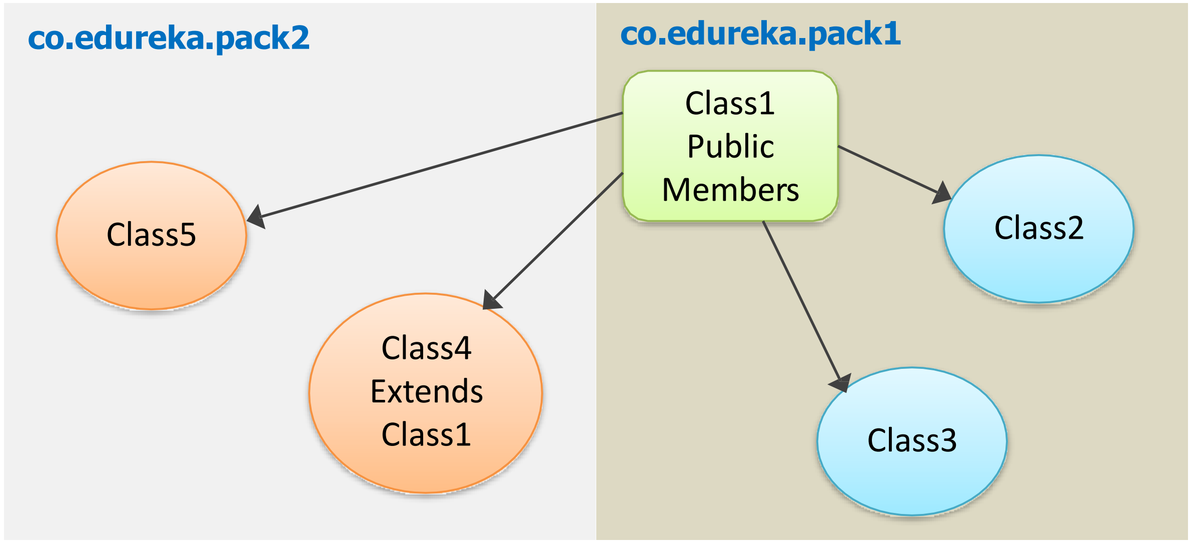
Data abstraction / hiding is one of the concept of Object Oriented Programming. This means, client will not know the implementation details.

This can be achieved through Access Modifier.

For Example: If an attribute is made private then it can be accessed only in the class which defined it.

Access Modifier – Public

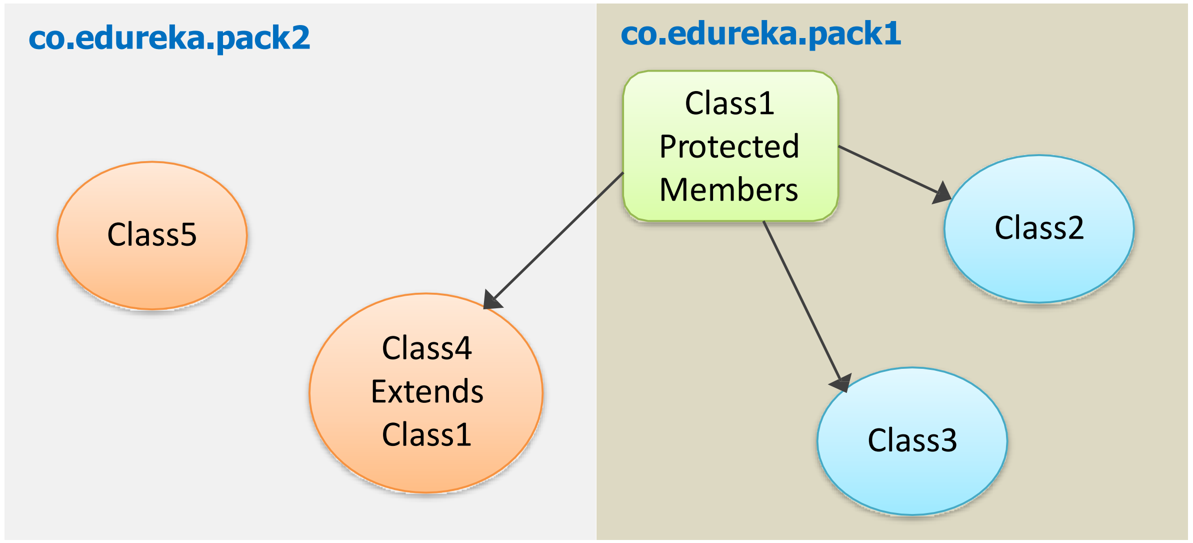
Public: When an attribute or method is declared as public then it can be accessed anywhere. Any package, any class accessibility is available.



Access Modifier – Protected

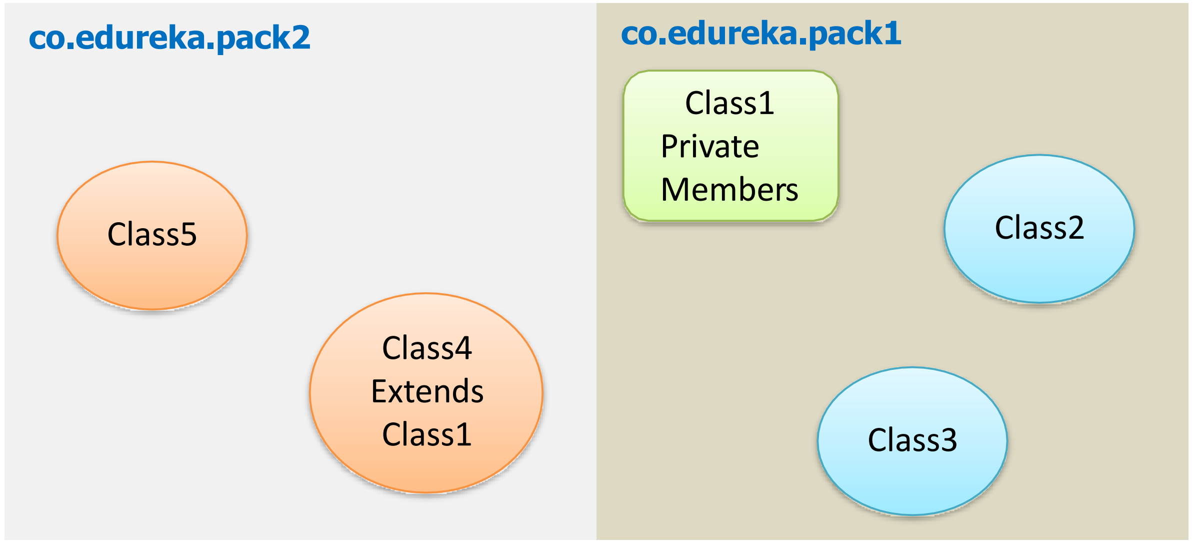
edureka!

Protected: When an attribute or method is declared as protected then it is visible to all the classes in the same package and all subclasses in different package.



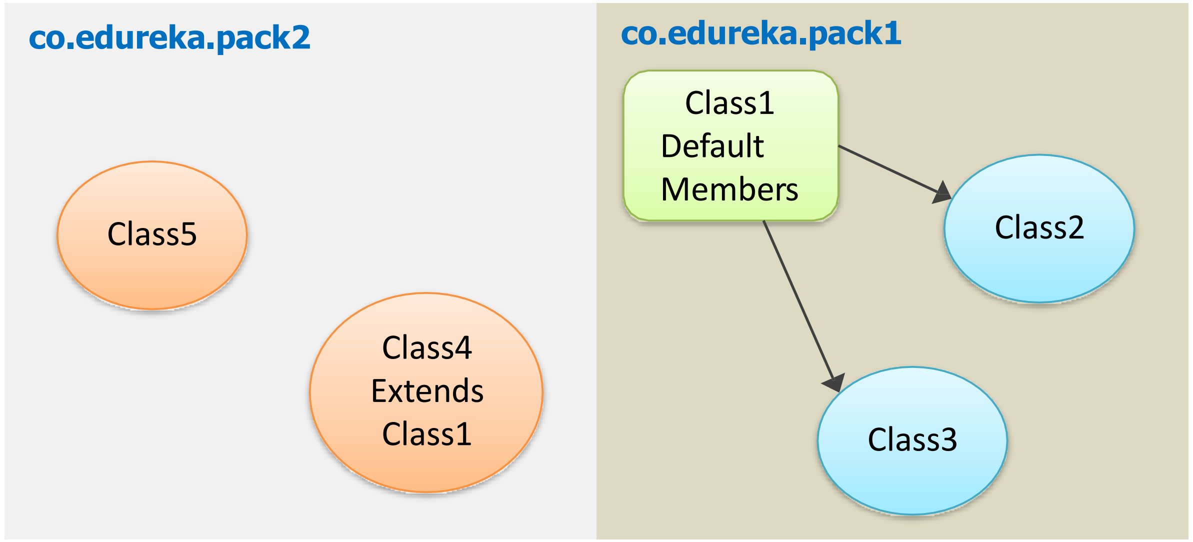
Access Modifier – Private

Private: If a Method, Variable or Constructor is defined as private then it can only be accessed within the declared class itself. Access is not available outside the class.



Access Modifier – Default

Default: When no access modifier is defined then it is said to have default access modifier. This attribute/method is used only in the given package. It is not accessible outside the package.



Access Levels

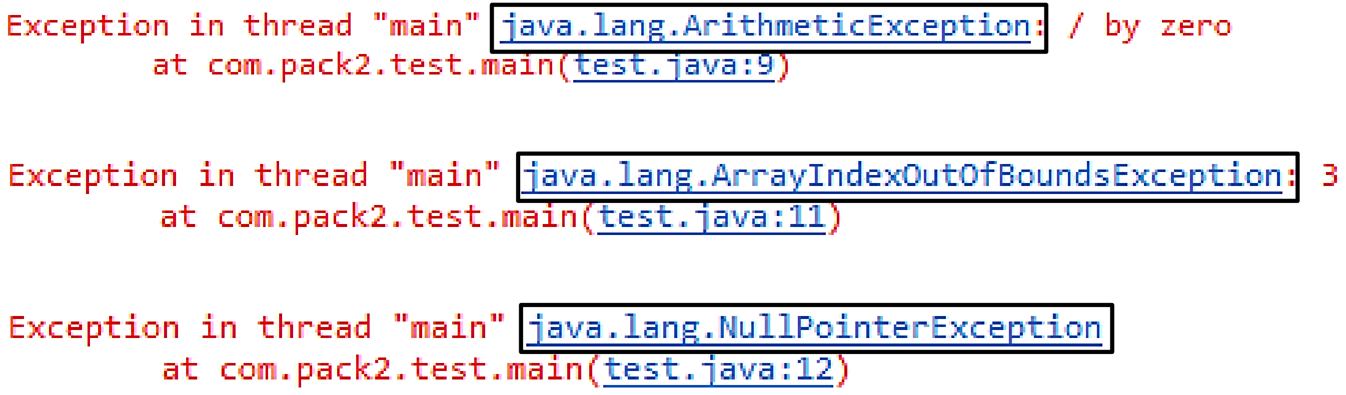
Modifier	Class	Package	Subclass	World
Public	Y	Y	Y	Y
Protected	Y	Y	Y	N
No modifier	Y	Y	N	N
private	Y	N	N	N

LAB

Exception

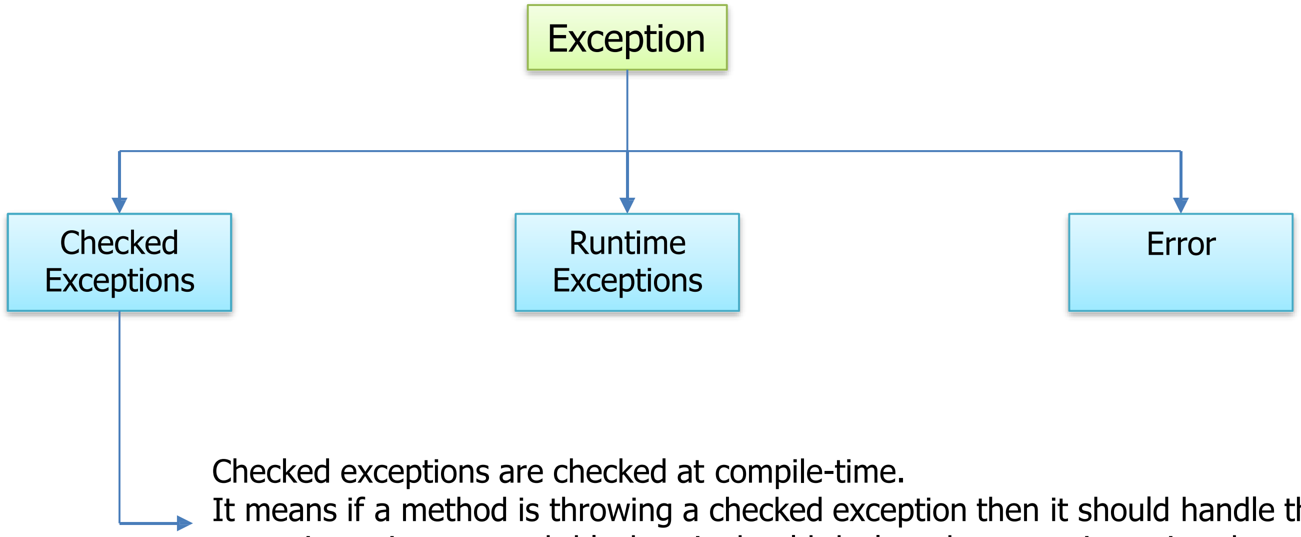
An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions -- Oracle

It is often referred to as run-time error.
Below are few of them:



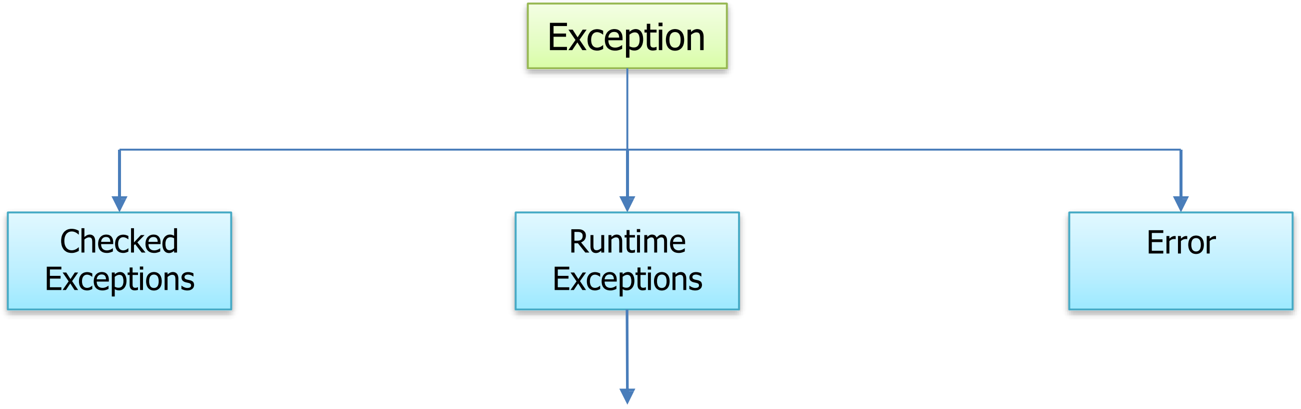
```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at com.pack2.test.main(test.java:9)

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
        at com.pack2.test.main(test.java:11)

Exception in thread "main" java.lang.NullPointerException
        at com.pack2.test.main(test.java:12)
```

Types of Exception



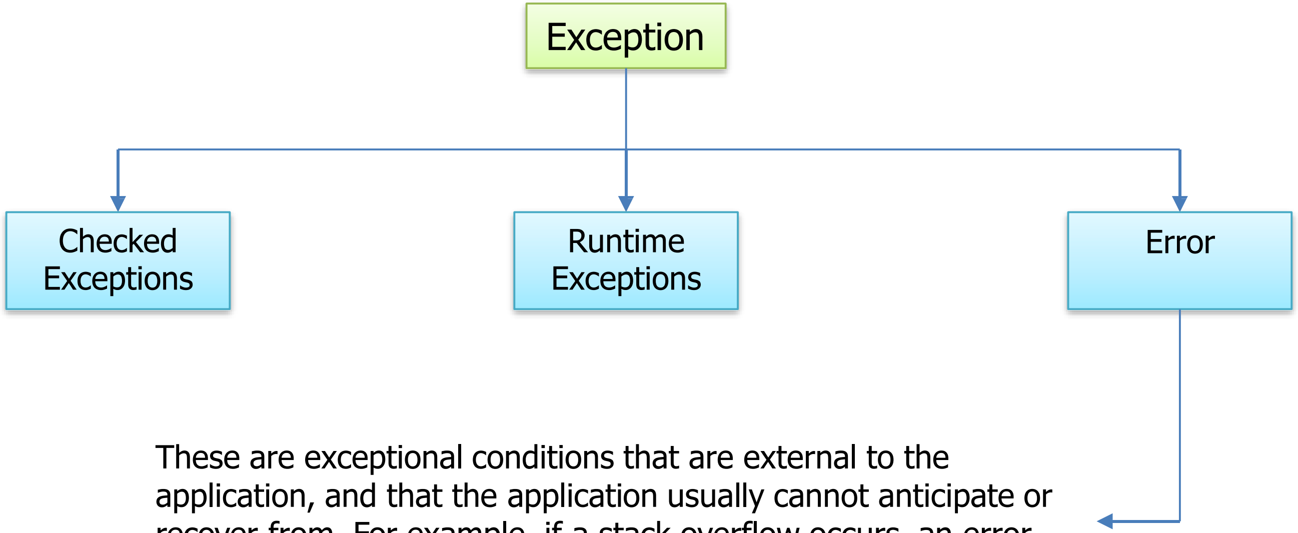
Checked exceptions are checked at compile-time.
It means if a method is throwing a checked exception then it should handle the exception using try-catch block or it should declare the exception using throws keyword, otherwise the program will give a compilation error.

Types of Exception



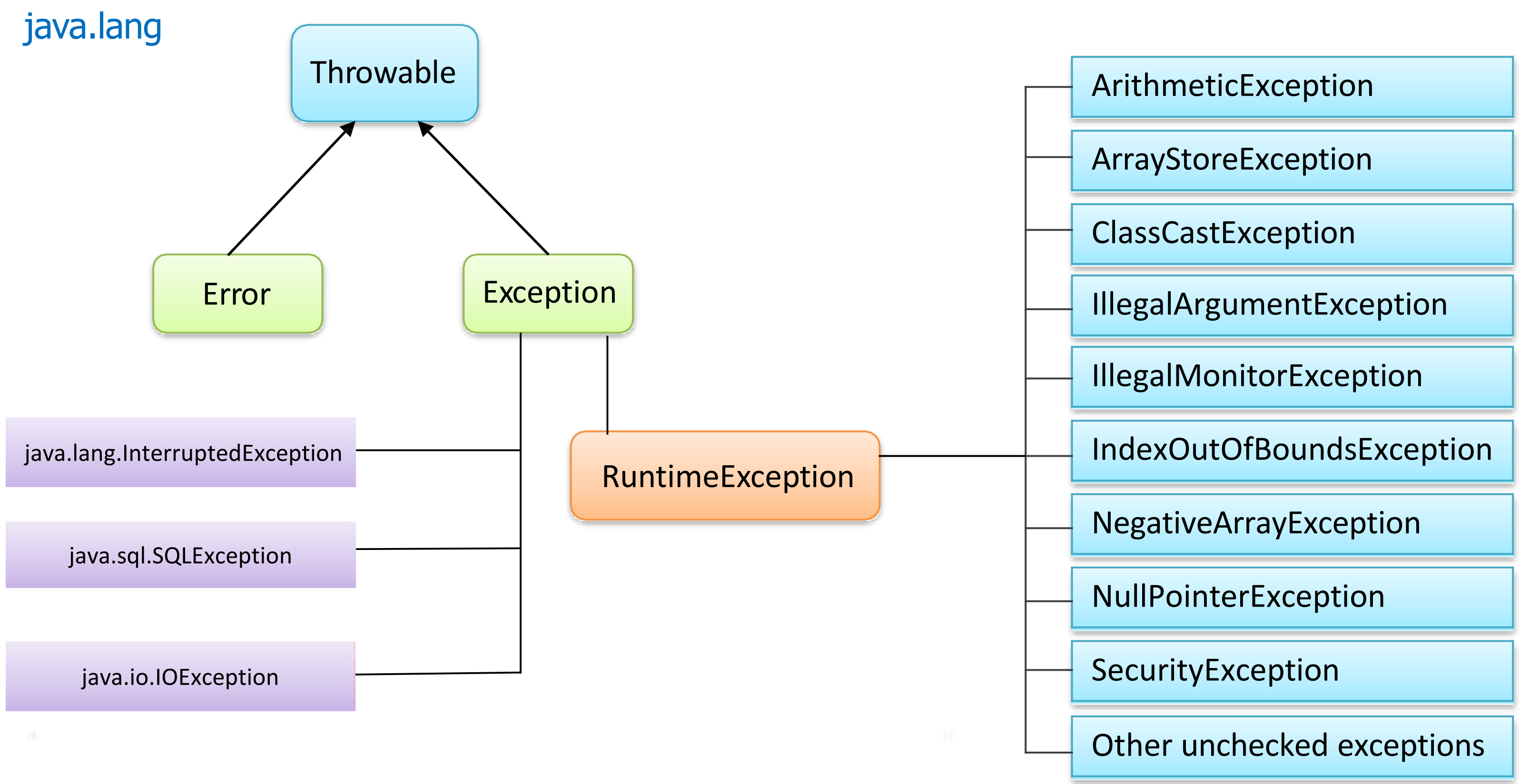
Unchecked exceptions are not checked at compile time. It means if your program is throwing an unchecked exception and even if you didn’t handle/declare that exception, the program won’t give a compilation error. It is up to the programmer to judge the conditions in advance, that can cause such exceptions and handle them appropriately. All unchecked exceptions are direct sub classes of RuntimeException class.

Types of Exception



These are exceptional conditions that are external to the application, and that the application usually cannot anticipate or recover from. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

Exception Class



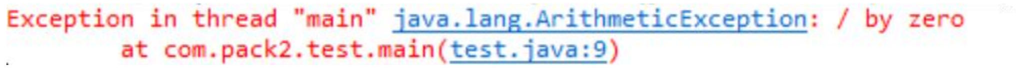
Why Exceptional Handling?

edureka!

```java
// Divide by Zero Problem
int x = 5 / 0;
System.out.println(x);
```

Output

error!
Program will halt.



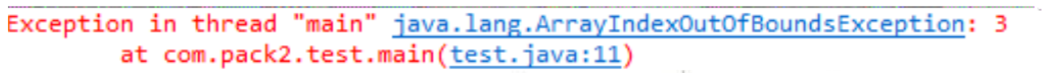
```
Exception in thread "main" java.lang.ArithmeticException: / by zero
	at com.pack2.test.main(test.java:9)
```

Or

```java
// Array Index Out of Bound
int arr[] = {1, 2, 3};
System.out.println(arr[3]);
```

Output

error!
Program will halt.



```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
	at com.pack2.test.main(test.java:11)
```

Solution

edureka!

```
try {
// Divide by Zero Problem
      int x = 5 / 0;
      System.out.println(x);

} catch (Exception e) {
      System.out.println("Divide By Zero
exception occured");
}
```

Output:

```
Divide By Zero exception occured
```

Using Exception handling the execution will continue even if an exception has occurred. The program will not halt because the exception was handled.

Exception Handling

edureka!

→ If there is a run time error then program is crashed and control comes out of the program.

→ This issue can be solved by exception handling.

→ Mainly, try, catch and finally are keywords for exception handling.

Exception Handling (contd.)

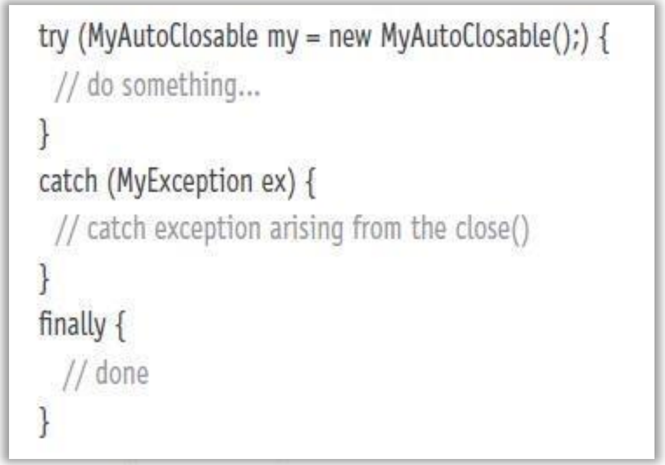
edureka!

→ try: All the statements to be executed should be placed in the try block.

→ catch: If there are any issues or runtime errors, control comes in catch block.

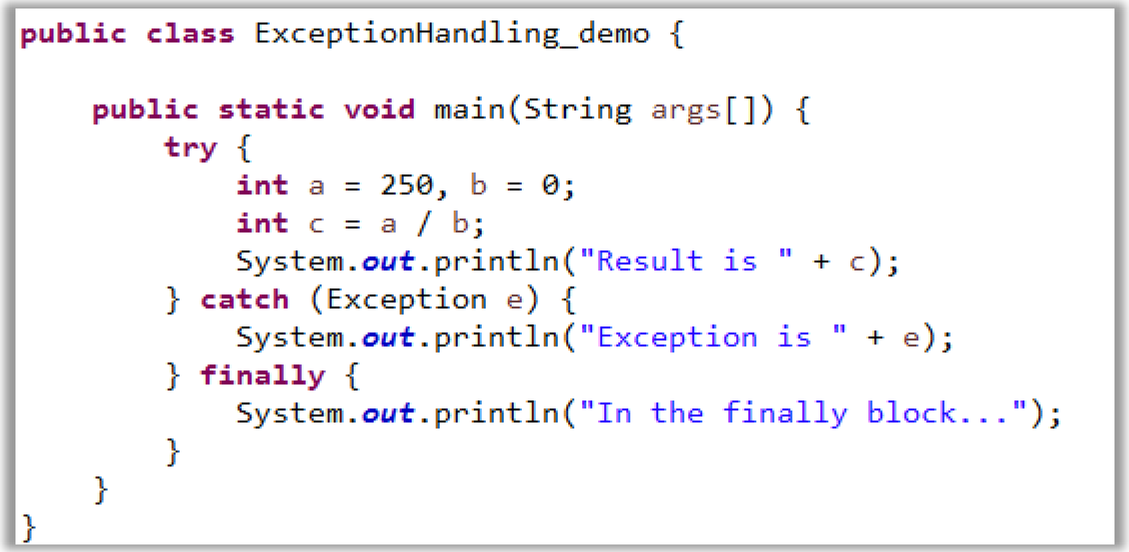
→ finally: Whether successful or unsuccessful execution, statements in the finally block gets executed.

Finally block is required to release the resource and required for clean up purposes.



```
try (MyAutoClosable my = new MyAutoClosable();) {
  // do something...
}
catch (MyException ex) {
  // catch exception arising from the close()
}
finally {
   // done
}
```

Program on Exception Handling



```
public class ExceptionHandling_demo {

    public static void main(String args[]) {
        try {
            int a = 250, b = 0;
            int c = a / b;
            System.out.println("Result is " + c);
        } catch (Exception e) {
            System.out.println("Exception is " + e);
        } finally {
            System.out.println("In the finally block...");
        }
    }
}
```

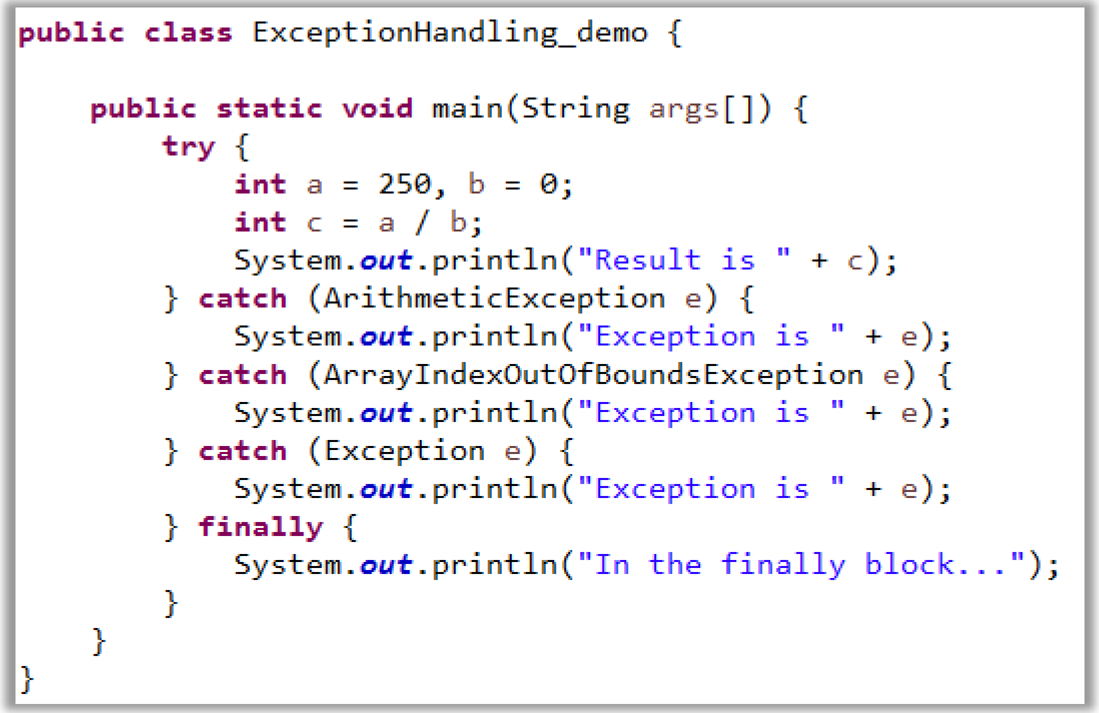
Exception Handling

One try can have multiple catch blocks. In this scenarios, depends on the type of exception thrown corresponding catch block is invoked.

Since all the exceptions are derived from Exception, catch (Exception e) should be placed at last. It can catch all the exceptions.

Program on Mutiple Catch Blocks

edureka!



```
public class ExceptionHandling_demo {

    public static void main(String args[]) {
        try {
            int a = 250, b = 0;
            int c = a / b;
            System.out.println("Result is " + c);
        } catch (ArithmeticException e) {
            System.out.println("Exception is " + e);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Exception is " + e);
        } catch (Exception e) {
            System.out.println("Exception is " + e);
        } finally {
            System.out.println("In the finally block...");
        }
    }
}
```

Nested try catch

edureka!

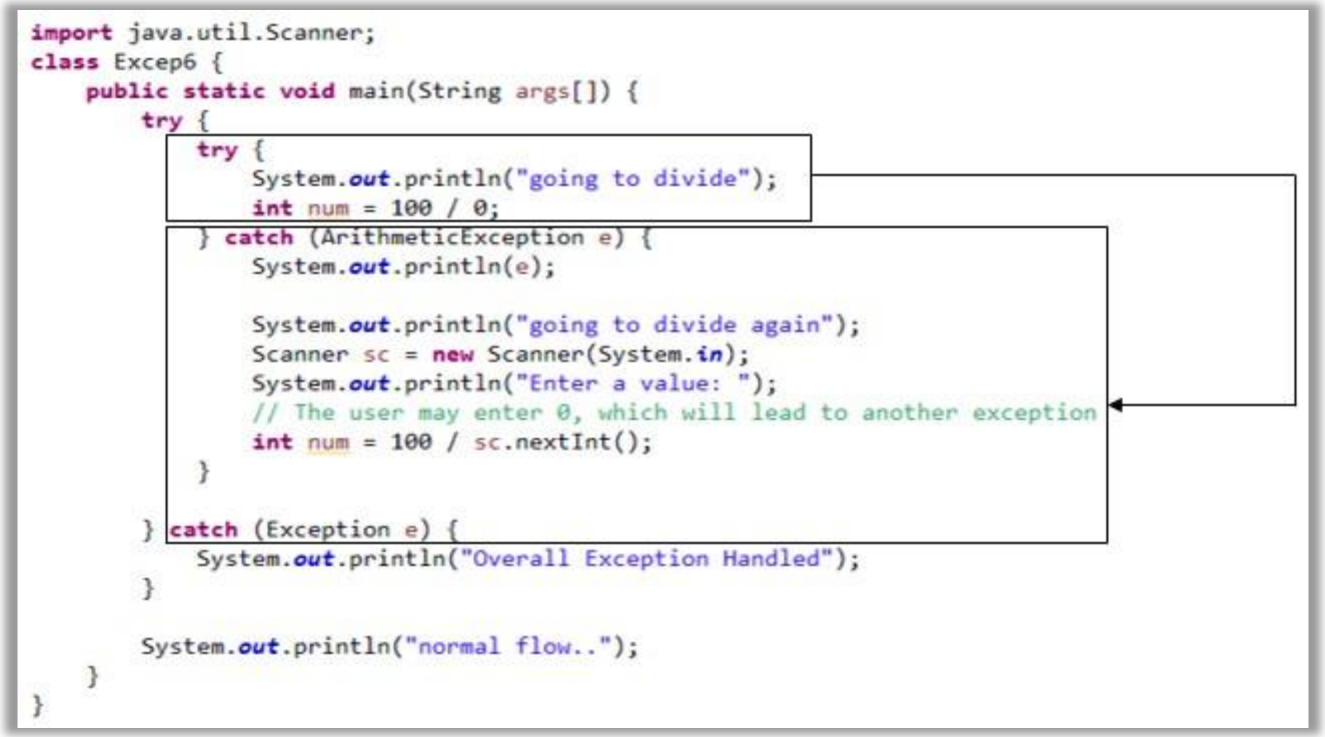
Why use nested try block?

Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.

Syntax:

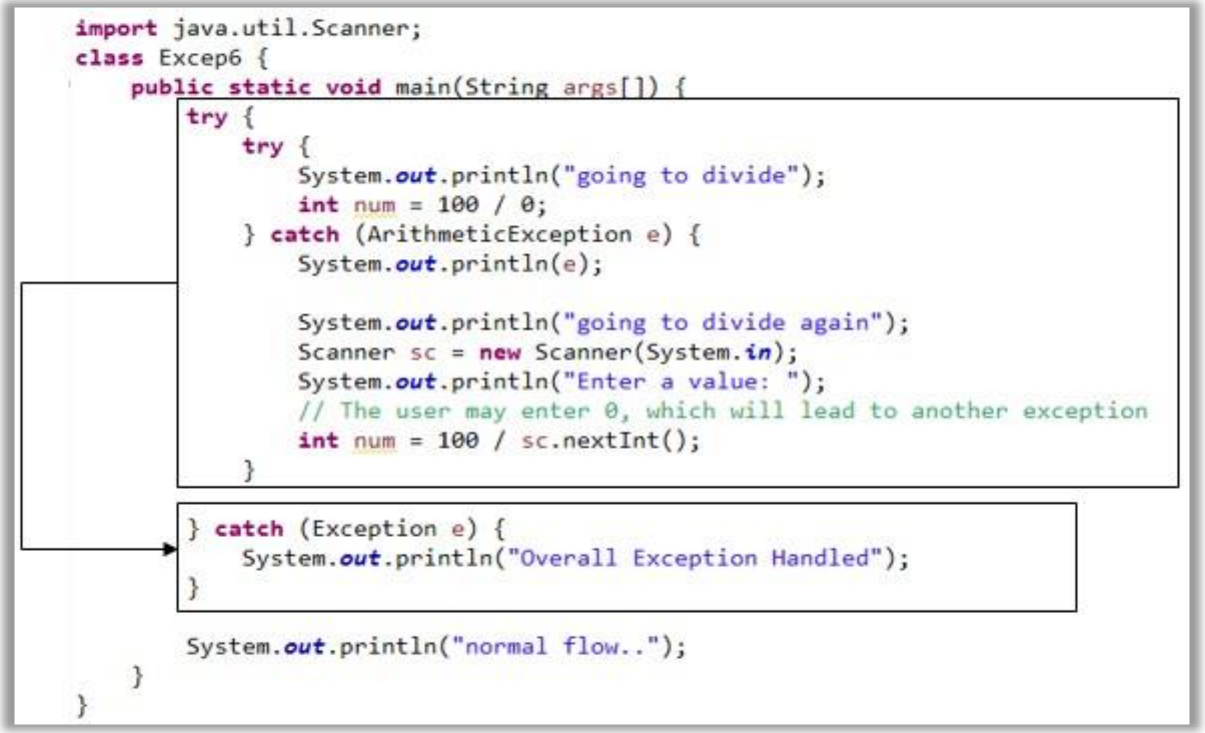
```
try
{
   statement 1;
   try
   {
      statement 1;
   }
   catch(Exception e)
   {
   }
}
catch(Exception e)
{
}
```

Program on Nested try catch



```
import java.util.Scanner;
class Excep6 {
    public static void main(String args[]) {
        try {
            try {
                System.out.println("going to divide");
                int num = 100 / 0;
            } catch (ArithmeticException e) {
                System.out.println(e);

                System.out.println("going to divide again");
                Scanner sc = new Scanner(System.in);
                System.out.println("Enter a value: ");
                // The user may enter 0, which will lead to another exception
                int num = 100 / sc.nextInt();
            }

        } catch (Exception e) {
            System.out.println("Overall Exception Handled");
        }

        System.out.println("normal flow..");
    }
}
```

Program on Nested try catch



```
import java.util.Scanner;
class Excep6 {
    public static void main(String args[]) {
        try {
            try {
                System.out.println("going to divide");
                int num = 100 / 0;
            } catch (ArithmeticException e) {
                System.out.println(e);

                System.out.println("going to divide again");
                Scanner sc = new Scanner(System.in);
                System.out.println("Enter a value: ");
                // The user may enter 0, which will lead to another exception
                int num = 100 / sc.nextInt();
            }
        } catch (Exception e) {
            System.out.println("Overall Exception Handled");
        }

        System.out.println("normal flow..");
    }
}
```

Why throw?

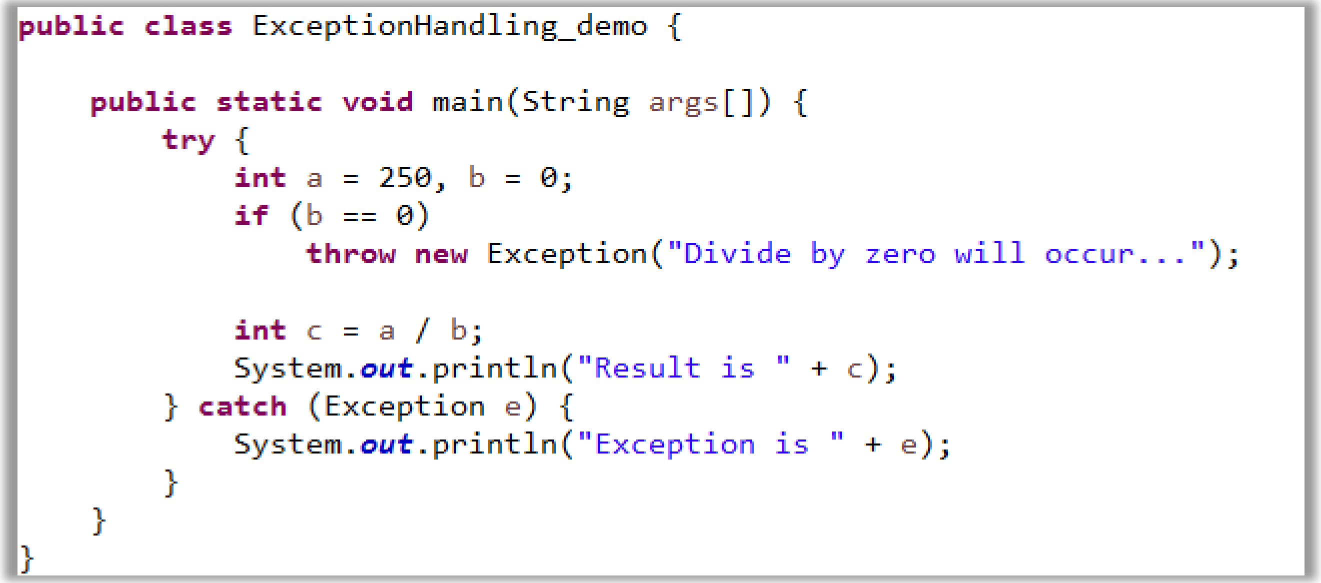
edureka!

Example 1: If there is a chance of a serious logic error or operational error then developer can also throw an exception. For example, if we are developing software for elections. For voting, minimum age required is 18. If the voter's age is below 18 then we can not continue any further, as the basic requirement itself is not met, hence developer can throw an exception.

Example 2: In banking application, one user account is blocked or closed and if the bank gets the cheque to clear the amount from this account then it is not possible to continue any further hence developer can throw an exception.

All the possible scenarios, developer has to use the throw keyword to throw an exception.

Throw


```
public class ExceptionHandling_demo {

    public static void main(String args[]) {
        try {
            int a = 250, b = 0;
            if (b == 0)
                throw new Exception("Divide by zero will occur...");

            int c = a / b;
            System.out.println("Result is " + c);
        } catch (Exception e) {
            System.out.println("Exception is " + e);
        }
    }
}
```

Why throws?

Design requirement: In an organization, employees provide the service. If there are any issues, in some scenarios, it is not possible for the employees to handle and it has to be escalated to the management to handle it. For example, contract signatures, handling legal issues etc.

Similarly in Java, method which provides the service may not be required to handle certain exceptions and those exceptions should be handled by the calling function.

Throws

Throws will be used next to a function declaration statement as given below:

```
Public void test() throws IOException
```

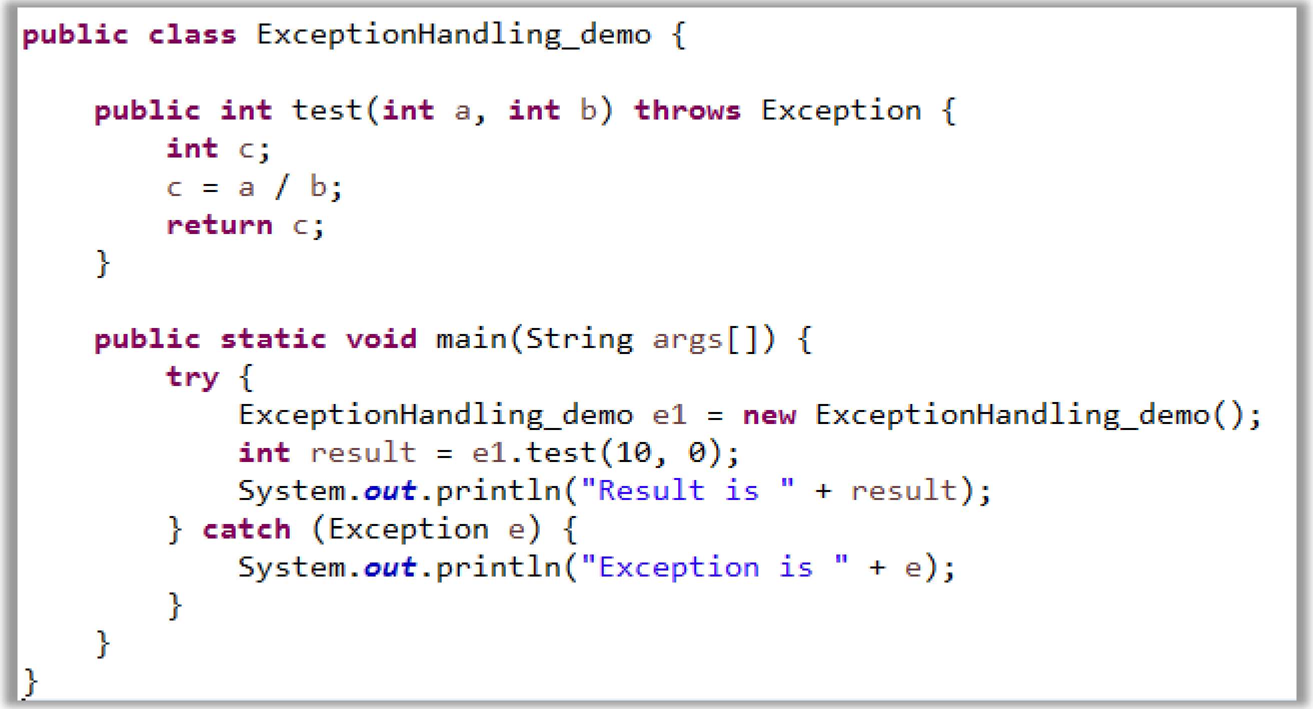
This statement states that the function test() will not handle IOexception and the calling function will handle these IOException. Calling function is responsible for IOExceptions.

Many exceptions can be added by adding comma operator as given below:

```
Public void function() throws IOException, ArrayIndexOutOfBoundsException
```

Program on Throws

edureka!



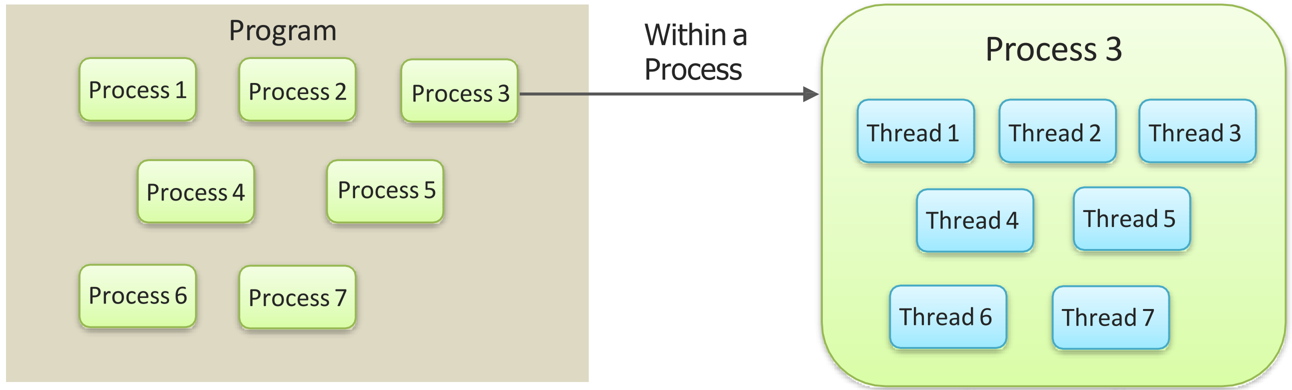
```
public class ExceptionHandling_demo {

    public int test(int a, int b) throws Exception {
        int c;
        c = a / b;
        return c;
    }

    public static void main(String args[]) {
        try {
            ExceptionHandling_demo e1 = new ExceptionHandling_demo();
            int result = e1.test(10, 0);
            System.out.println("Result is " + result);
        } catch (Exception e) {
            System.out.println("Exception is " + e);
        }
    }
}
```

Program Vs Process Vs Thread

Program: A program is a set of instructions stored in the secondary storage device that are intended to carry out a specific job. It is read into the primary memory and executed by the kernel.

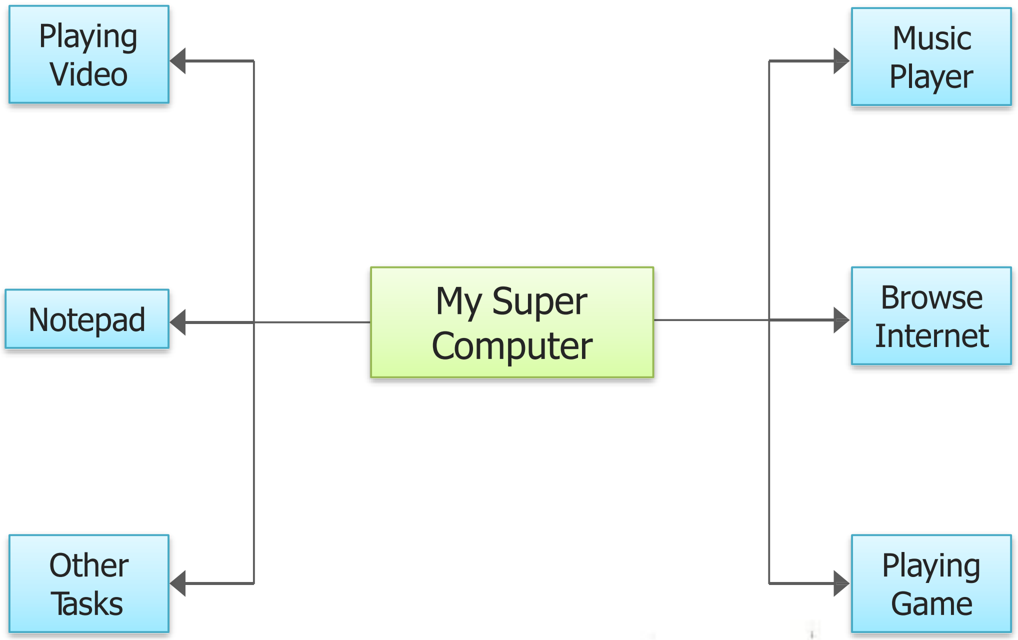
Process: An executing instance of a program is called a process. It is also referred as a task.

Thread: A thread is called a 'lightweight process'. It is similar to a real process but executes within the context of a process and shares the same resources allotted to the process by the kernel.



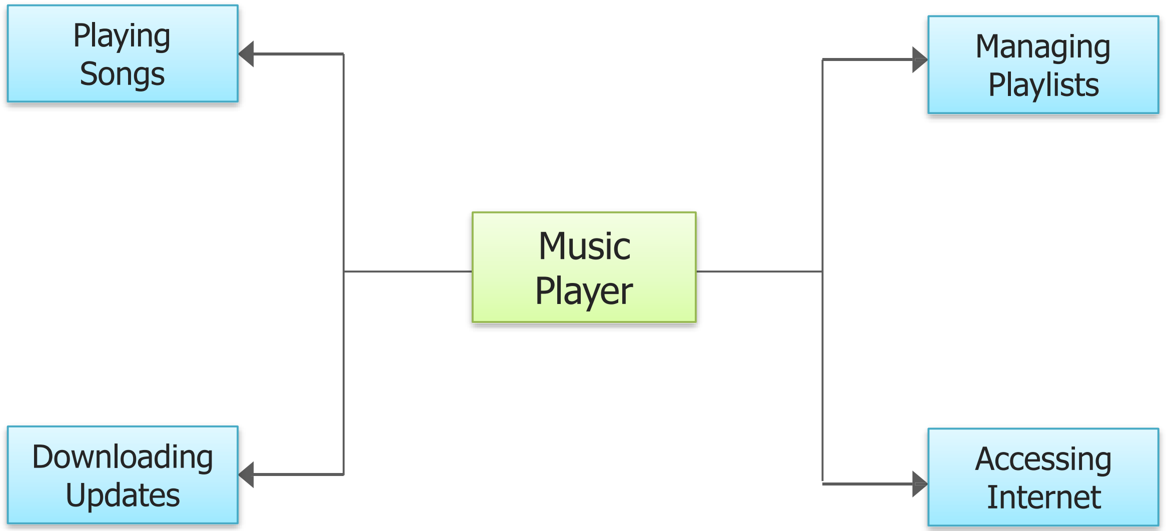
Multi Threading

Every computer has multiple processes running at a time.



Multi Threading (contd.)

Every Application has multiple sub-processes running at a time.



Multi Threading (contd.)

edureka!

Thread is a task to be performed. Multi threading is multiple tasks getting executed at the same time in the same program / process.

Multi threading takes the same memory space for any number of threads.

Annie’s Question

edureka!

What are the advantages of implementing multi threading in a Java program?

Annie's Answer

edureka!

1. Since multiple processes/tasks run at the same time, time is saved, hence speed increases.
2. Instead of running two programs, the same can be achieved by running two threads in the same memory space, hence memory consumption also decreases.

Multi Threading

edureka!

In Java, a thread can be implemented in two ways:

→ extending from a thread class

→ implementing runnable interface

Extending Thread - Example

edureka!

```
class Thread1 extends Thread {
        public void run() {

        }
```

LAB

Runnable Interface - Example

```
class Circle implements Runnable {

        @Override
        public void run() {

        }
}
```

Need of Runnable Interface:

→ In Java Inheritance is limited to one class only.
→ If a class extends non-thread class, then it can't extend thread.
→ So to provide multi-threading into a class that already extends some other class, we use Runnable.

LAB

Constructors of Thread Class

```
Thread()
```

```
Thread(String name)
```

Other Constructors of Thread Class

```
Thread(Runnable)

Thread(Runnable, String)

Thread(ThreadGroup, String)

Thread(ThreadGroup, Runnable)

Thread(ThreadGroup, Runnable, String)

Thread(ThreadGroup, Runnable, String, long)
```

Thread - LifeCycle

New: At this point, the thread is considered not alive.

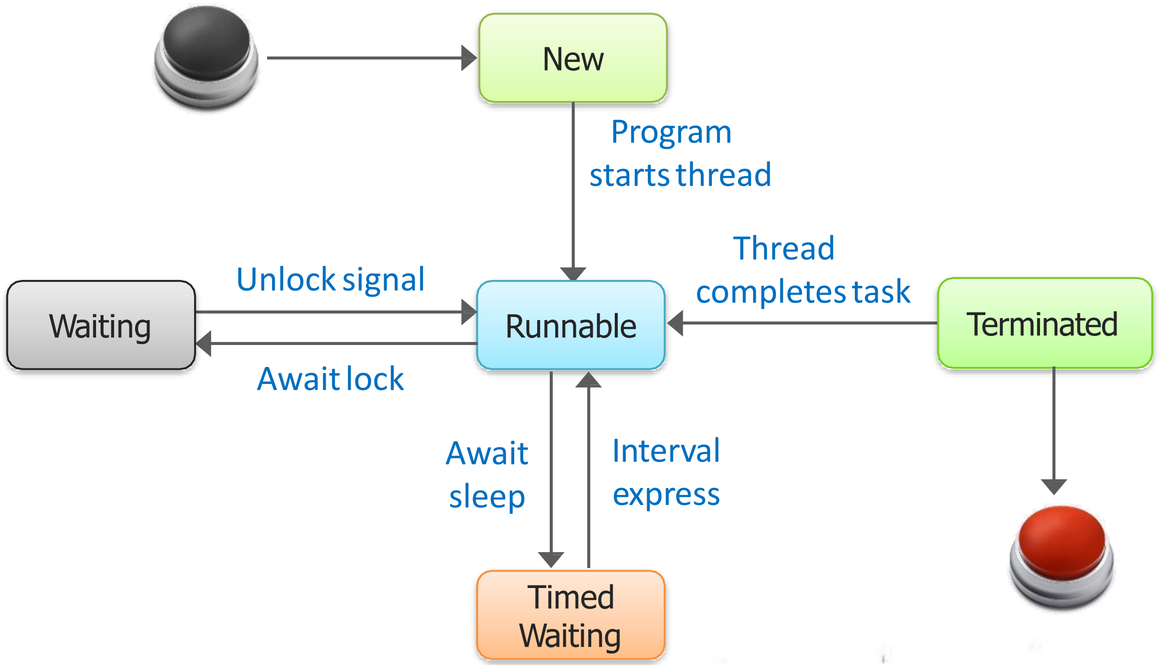
Runnable: A thread first enters runnable state after the invoking of start() method.

Running: A thread is in running state that means the thread is currently executing.

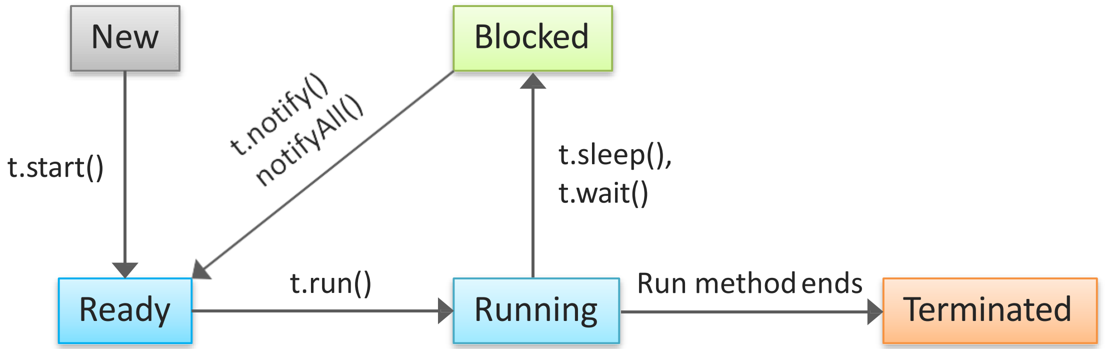
Blocked: A thread can enter in this state because of waiting for the resources or sleeping.

Terminated: A thread can enter in this state when run() method completes. After this method thread cannot ever run again.

Thread - LifeCycle



Thread - Methods

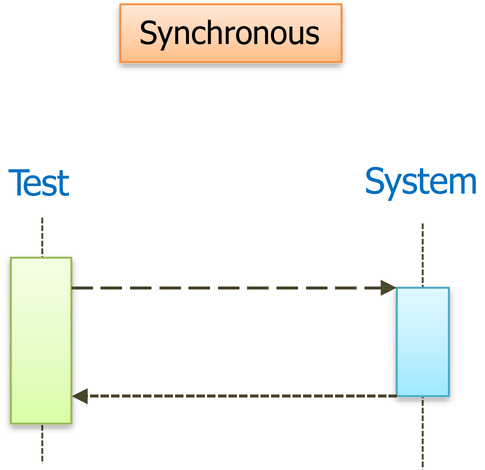


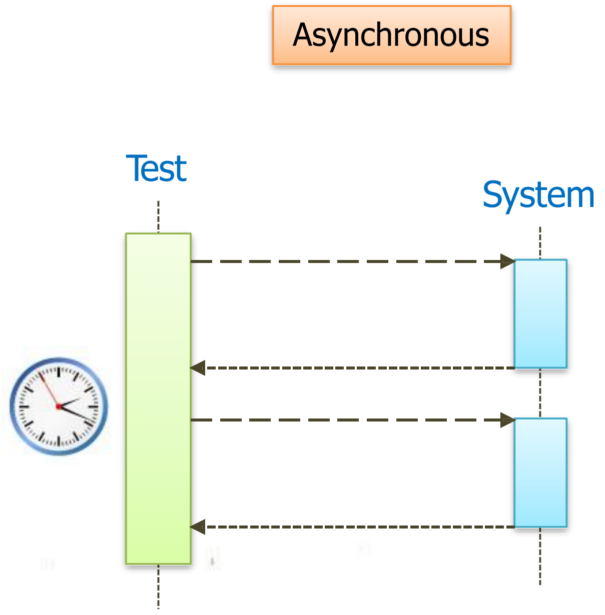
→ Start() method should be called to start a thread.
→ Start() will call the run() method.
→ Run() will have the actual task to be performed by the thread.

Synchronization

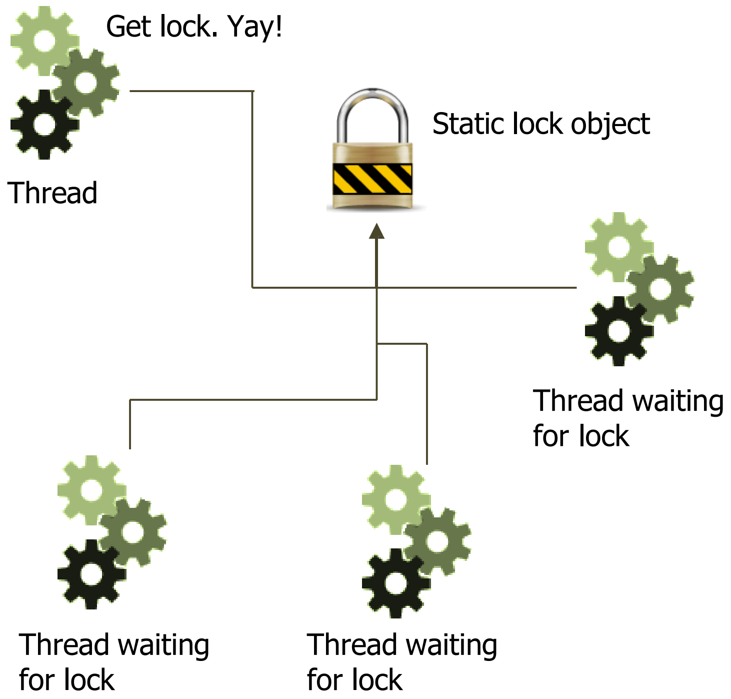
Synchronization is the coordination of events to operate a system in unison.

Synchronization is achieved by the use of locks in Java.





Locks in Java



A lock is a tool for controlling access to a shared resource by multiple threads.

There are four different kinds of locks:

Fat locks: A fat lock is a lock with a history of contention (several threads trying to take the lock simultaneously), or a lock that has been waited on (for notification).

Thin locks: A thin lock is a lock that does not have any contention.

Recursive locks: A recursive lock is a lock that has been taken by a thread several times without having been released.

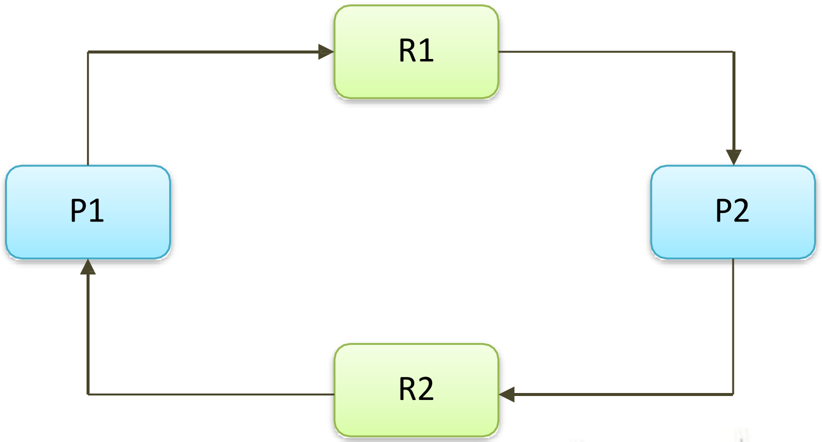
Lazy locks: A lazy lock is a lock that is not released when a critical section is exited. Once a lazy lock is acquired by a thread, other threads that try to acquire the lock have to ensure that the lock is, or can be, released.

Deadlock

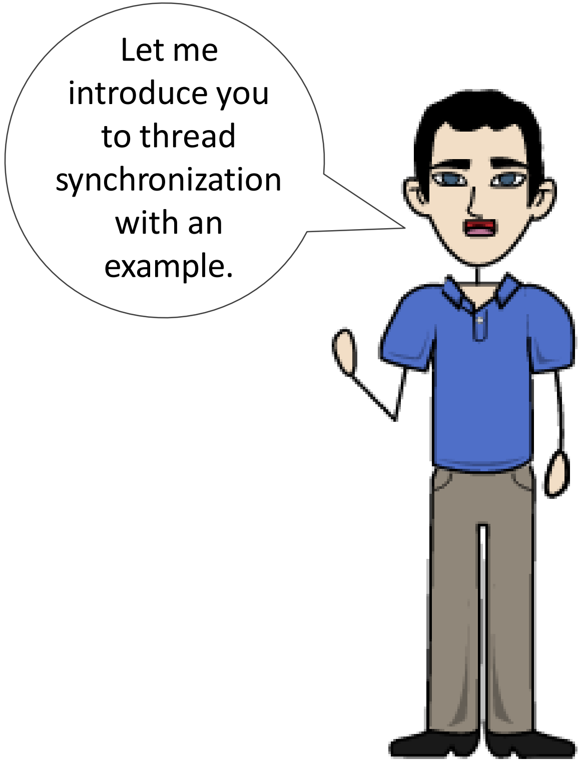
Deadlock describes a situation where two or more threads are blocked forever, waiting for each other.

Or

A deadlock is a situation in which two or more competing actions are each waiting for the other to finish, and thus neither ever does.

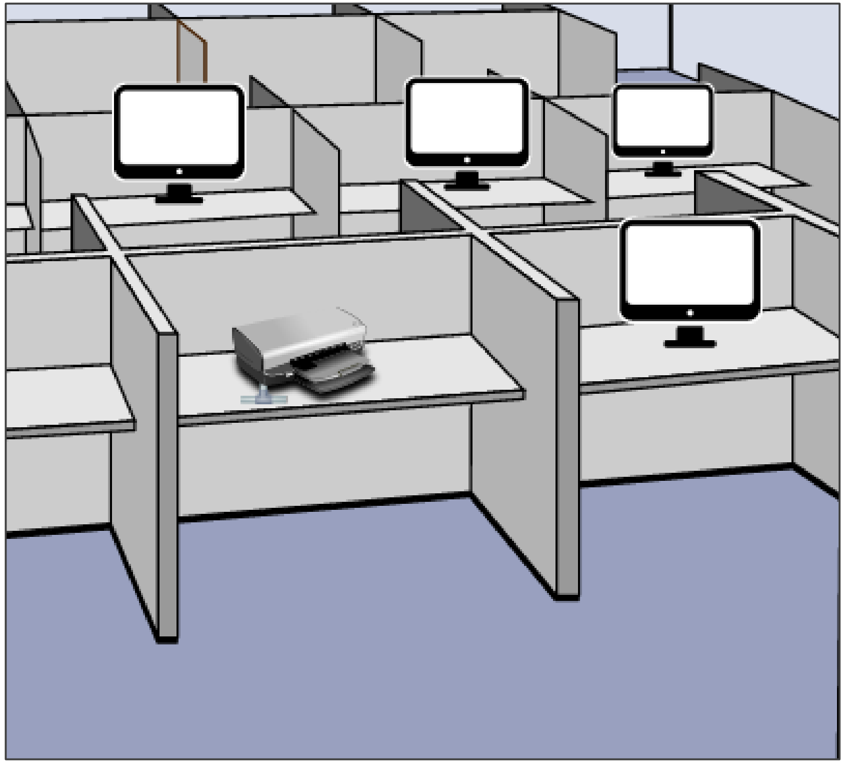


Carl explains Thread Synchronization


Let me introduce you to thread synchronization with an example.

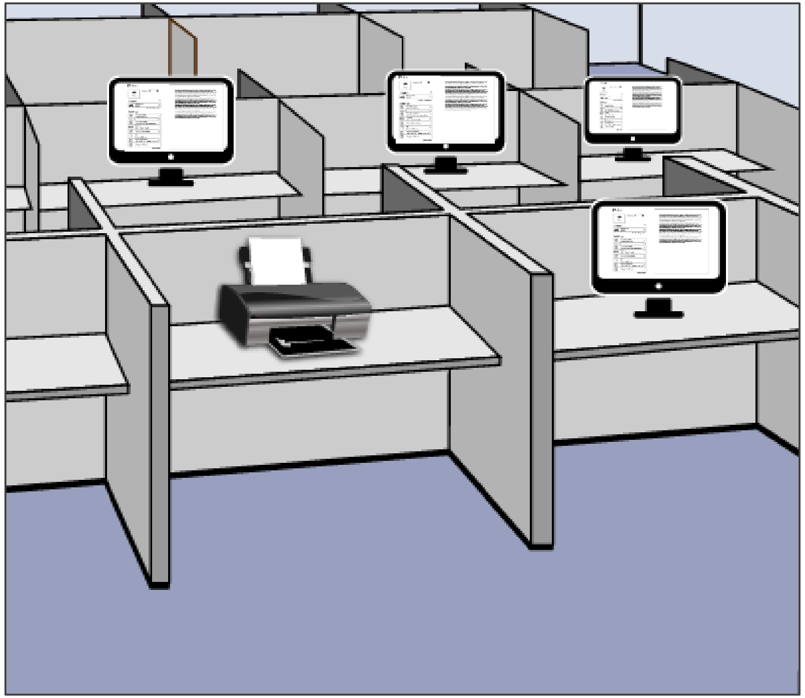
Example

Assume a printer is connected to 4 computers.



Example

All the 4 computers requests to print a document. Printer starts to print first computer document. The second and the other computer documents should be waiting till the first computer document gets printed.



This happens in Java - Thread Synchronization

Similarly, when two or more threads try to access the same resource simultaneously it causes the Java runtime to execute one or more threads more slowly, or even suspend their execution.

The concept of one thread to execute at a time and rest of the threads in waiting state is called thread synchronization.

LAB

edureka!

QUESTIONS

Assignment - Interfaces

→ Write a program to define a queue interface and have insert and delete methods in the interface. Implement these methods in a class.

Assignment - Packages

→ Write a program to define functions for subtract, multiply, divide, factorial and reversing the digits of a number in a package, import this class in another package and use all the methods defined in the primary package.

Assignment - Exception

→ Write a program to demonstrate ArrayIndexOutOfBoundsException.

Assignment - Thread

→ Write a program to print tables of 5 by creating a new thread and display 20 even numbers as a task of main thread.

Agenda for the Next Class

In the next class, you will be able to

→ Identify and use important Inbuilt Java Packages like java.lang, java.io, java.util etc

→ Use wrapper classes

→ Understand collections framework

→ Implement logic using arraylist and vector and queue

→ Use set, hashset and treeset

→ Implement logic using Map HashMap and HashTable

Pre-work

- Brush up on Databases, records.

Survey

Your feedback is important to us, be it a compliment, a suggestion or a complaint. It helps us to make the course better!

Please spare few minutes to take the survey after the webinar.

edureka!