
PACKAGES,
EXCEPTIONS AND
THREADS

Content


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 packages.

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.

- Packages 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 named mypackage created by a programmer at example.com

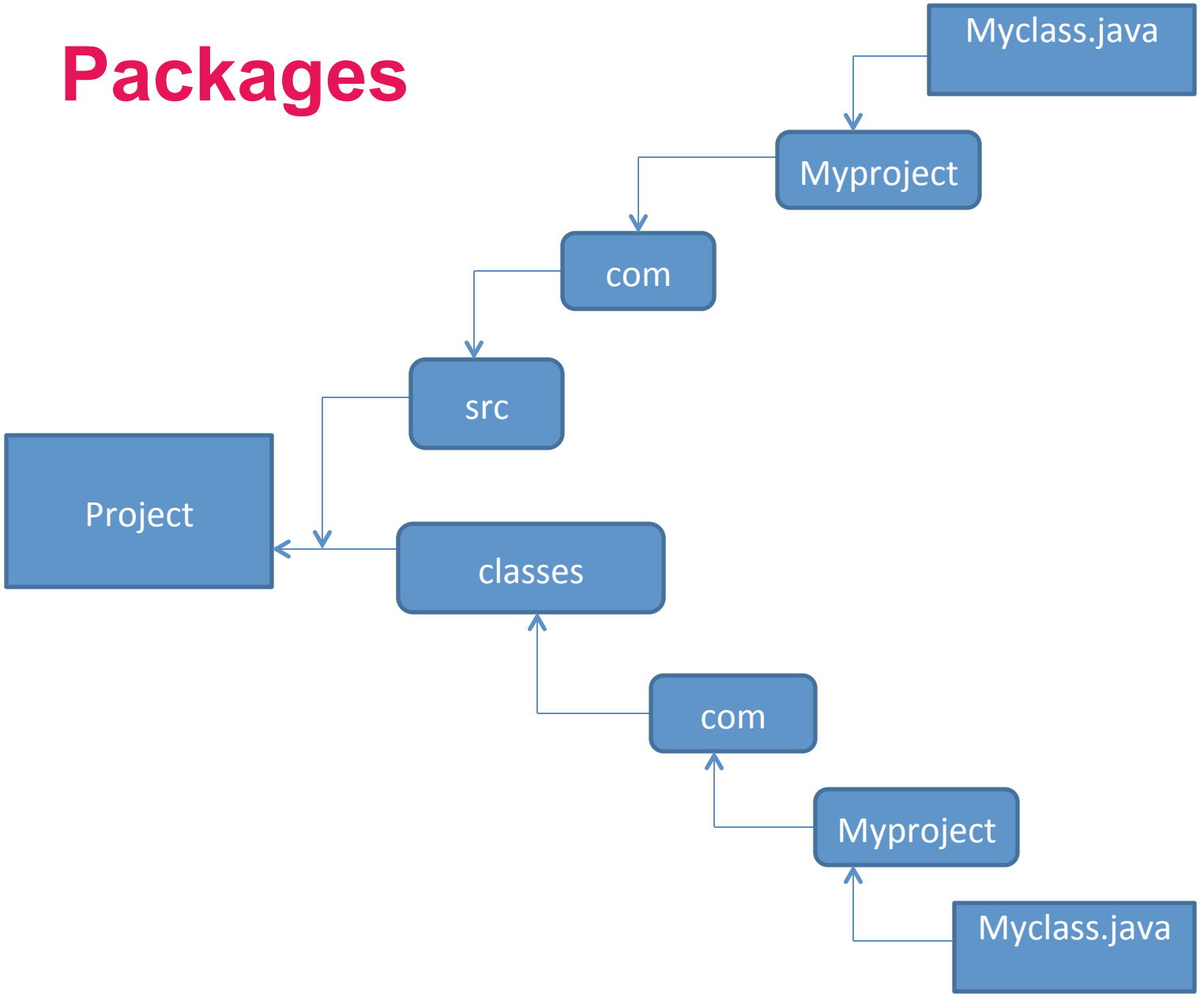
Naming Convention of a Package

If the domain name contains :

- a hypen 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
classes
com
Myproject
Myclass.java

Program on Package

Company URL-> auribises.com

Package name-> com.auribises.db

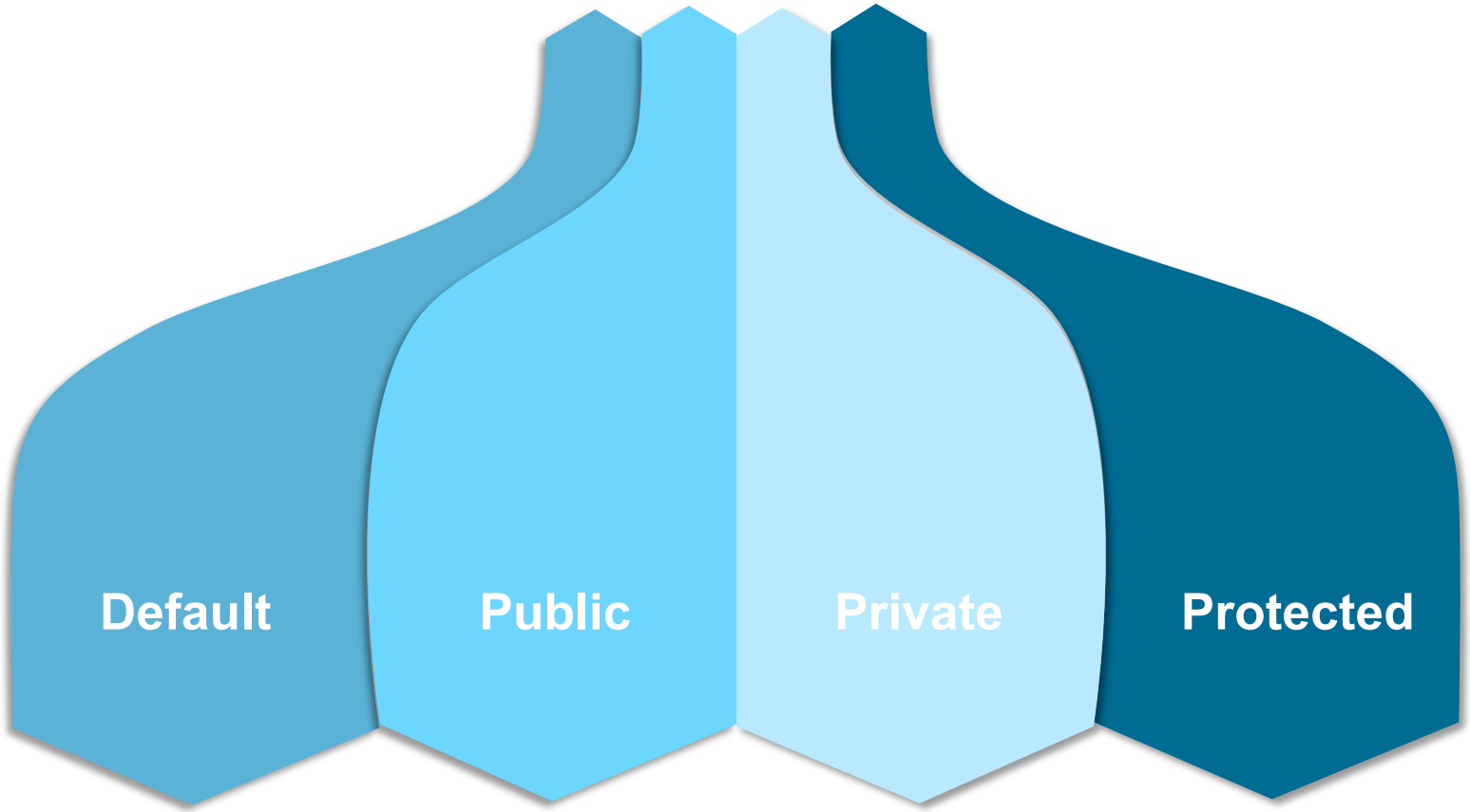
Package and Import

- 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 “keyword”.
- 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.*;
```

Access Modifiers

Access modifiers helps to restrict the scope of a class, constructor , variable , method or data member



Access Modifier

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.

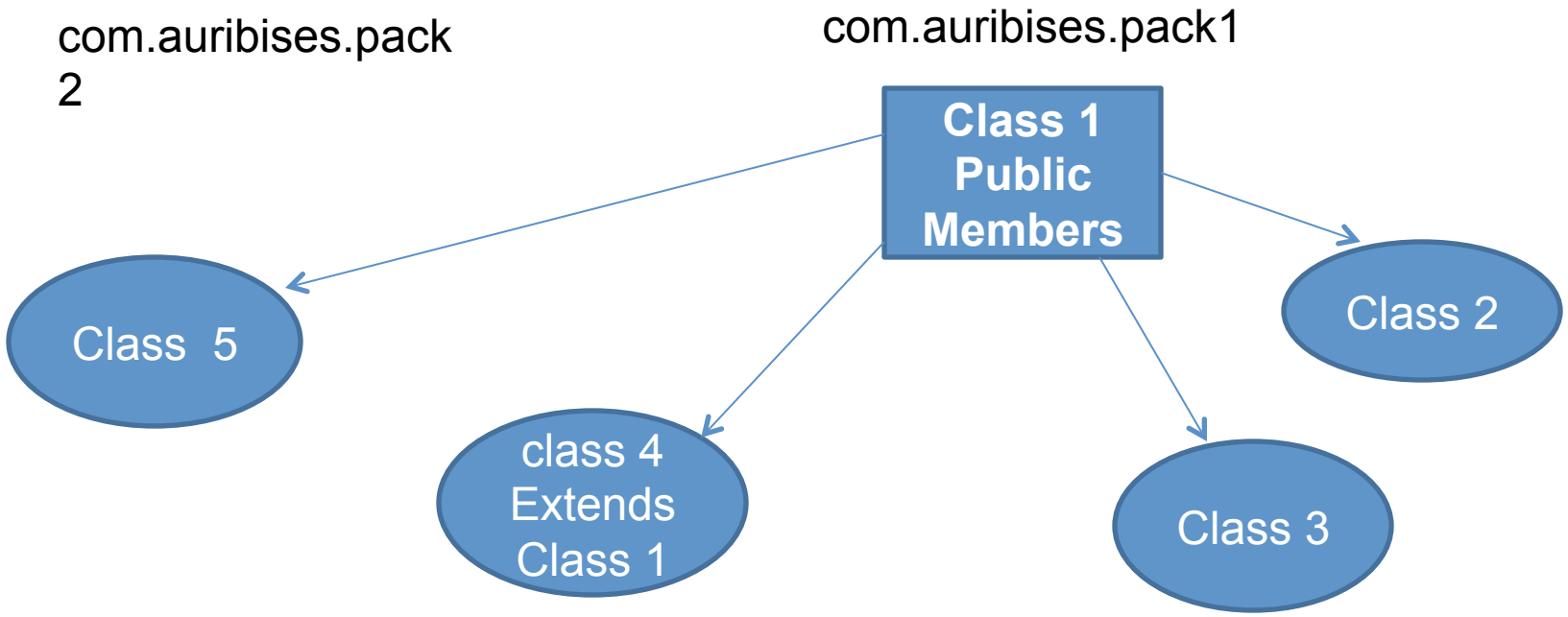
Use of Access Modifier:

- 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.

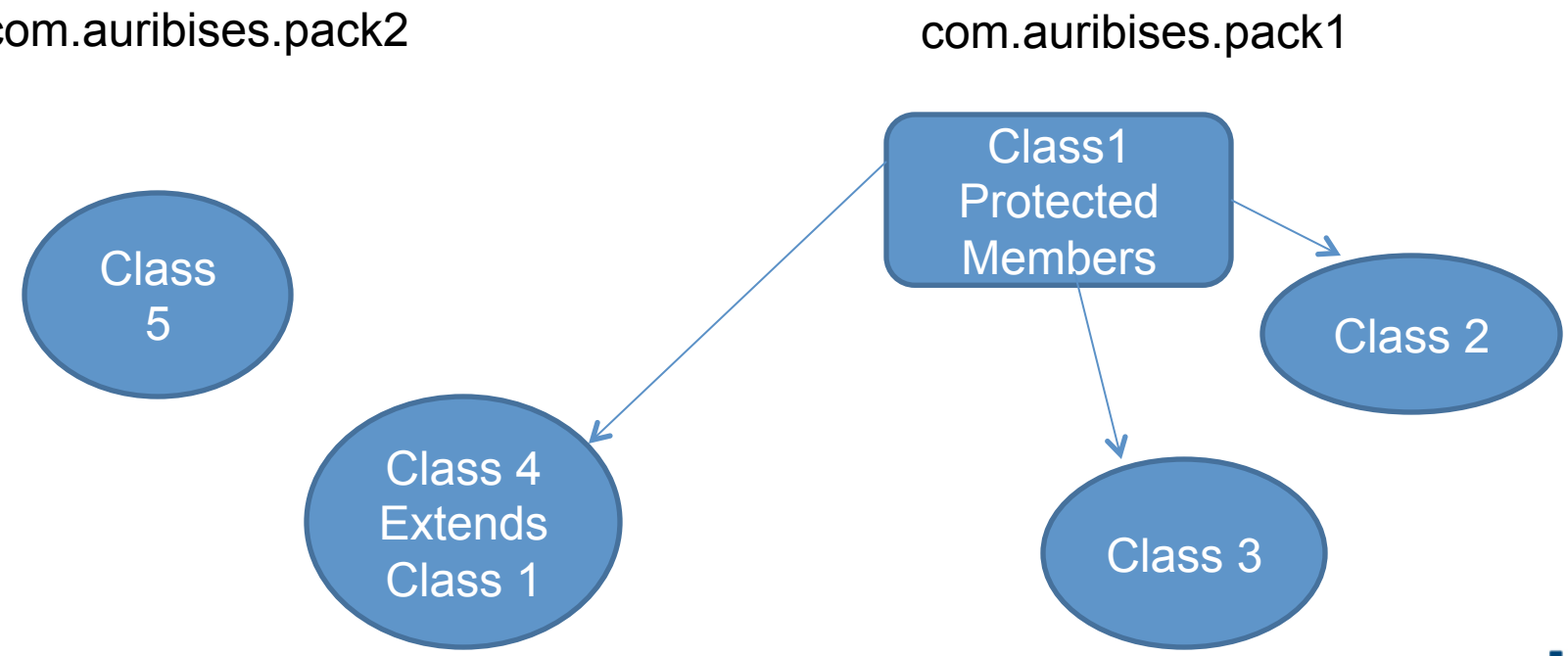
Access Modifier – Public

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



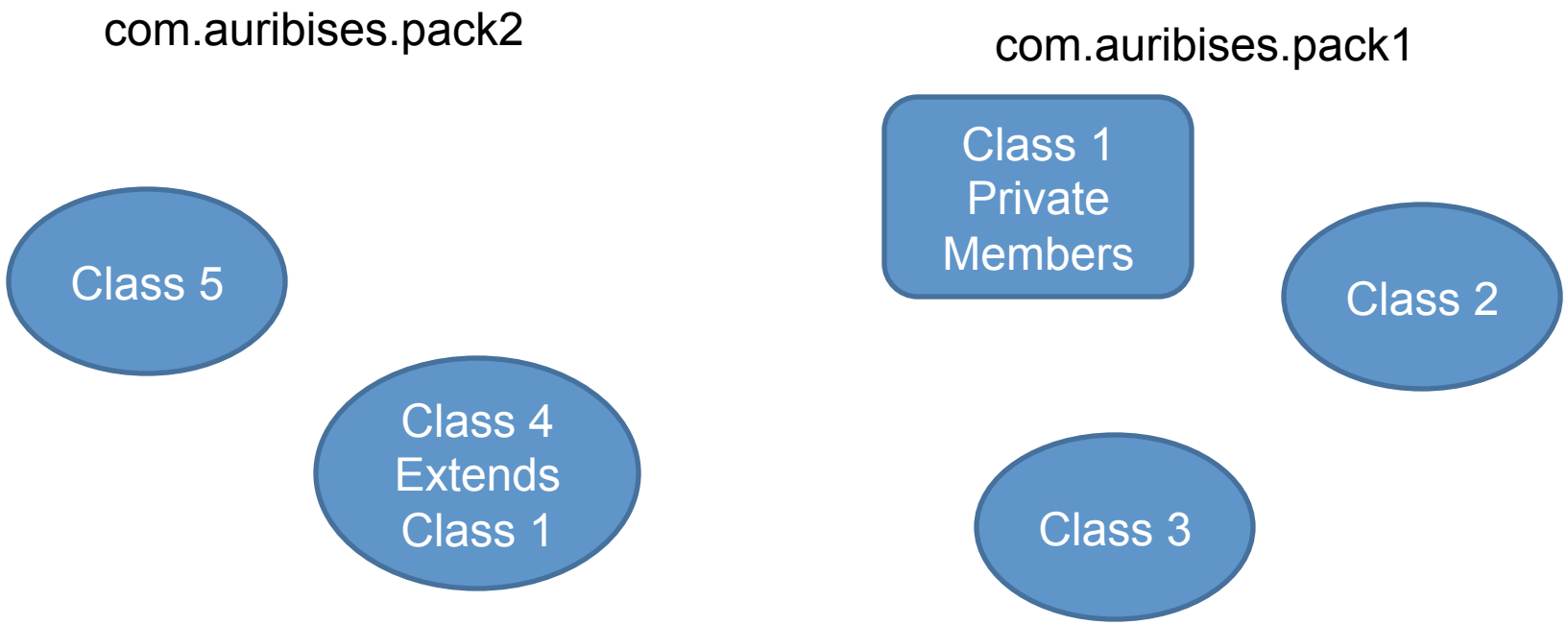
Access Modifier-Protected

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



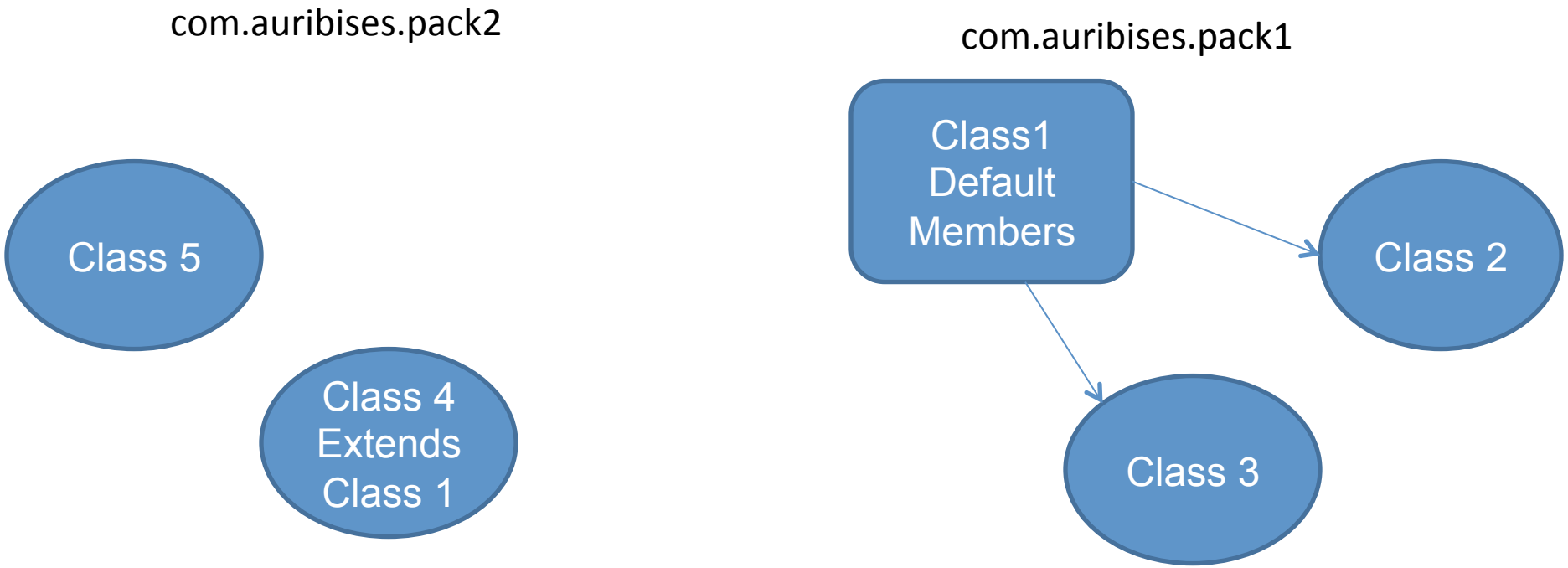
Access Modifier- Private

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 to have default access modifier. This attribute/method is used only in the given package . It is not accessible outside the package.



Access Modifiers

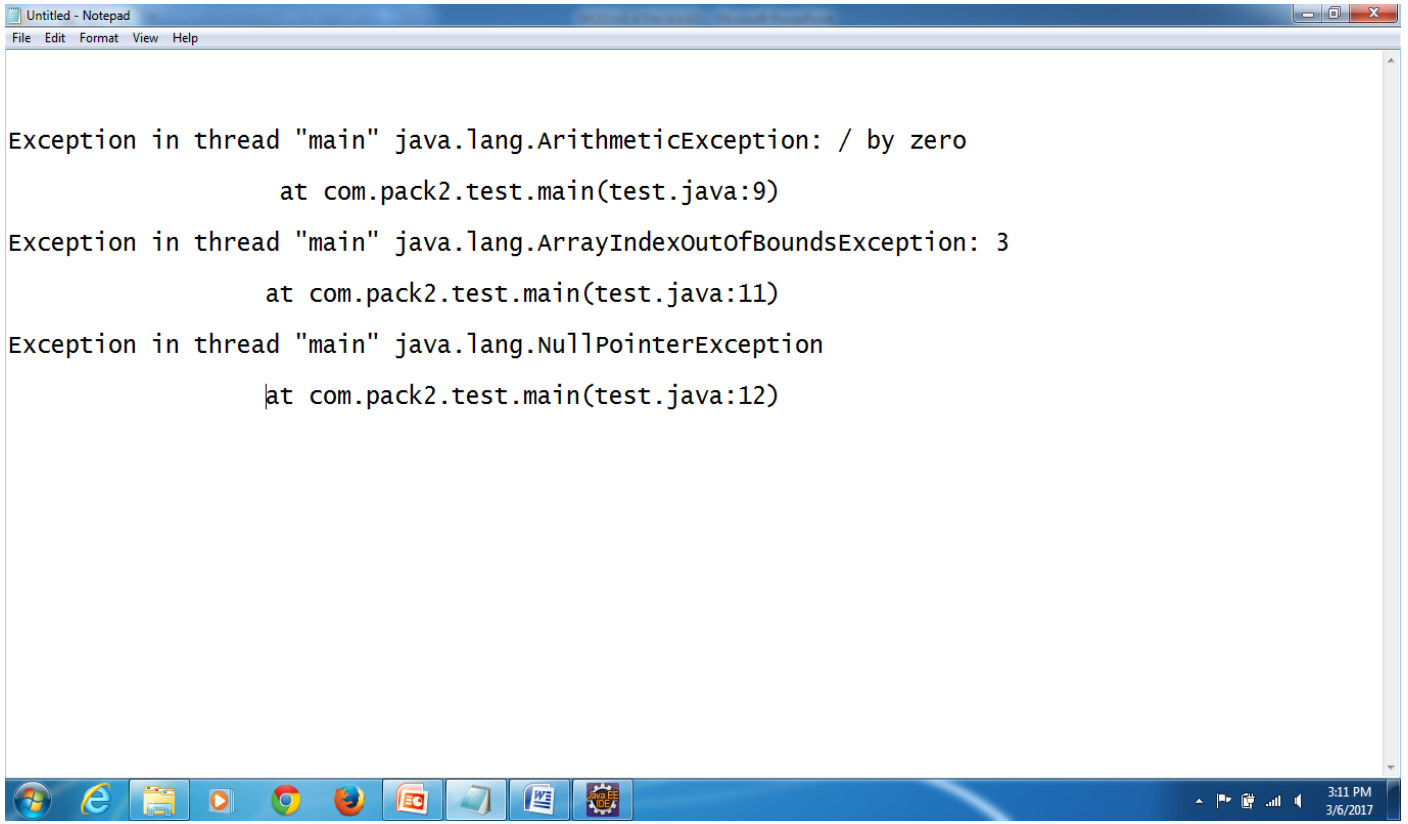
	Default	Public	Private	Protected
Same class	Yes	Yes	Yes	Yes
Same Package subclass	Yes	No	Yes	Yes
Same Package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Exception

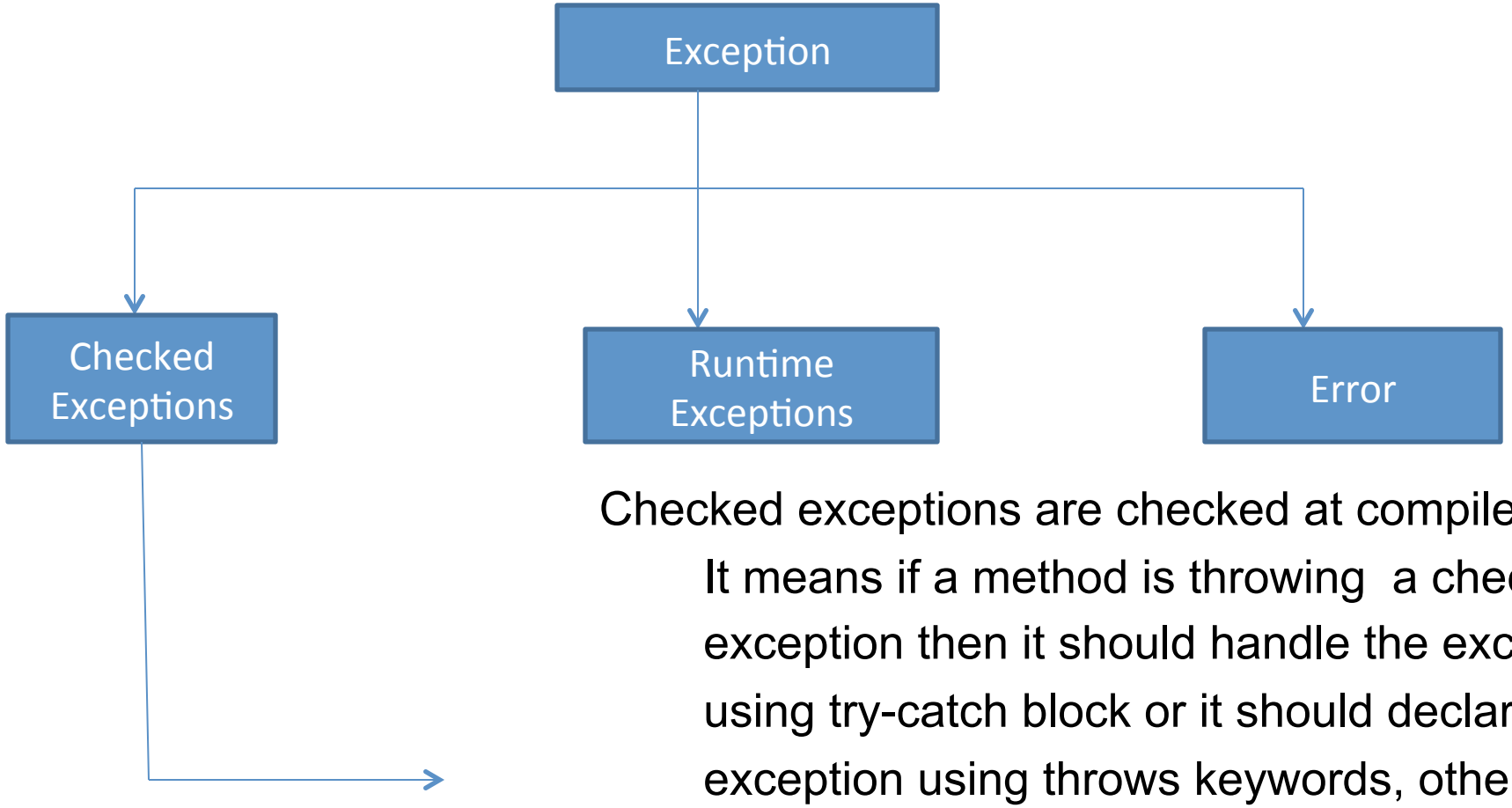
An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions—oracle.

It is often referred to as run-time error.

Below are few of them :



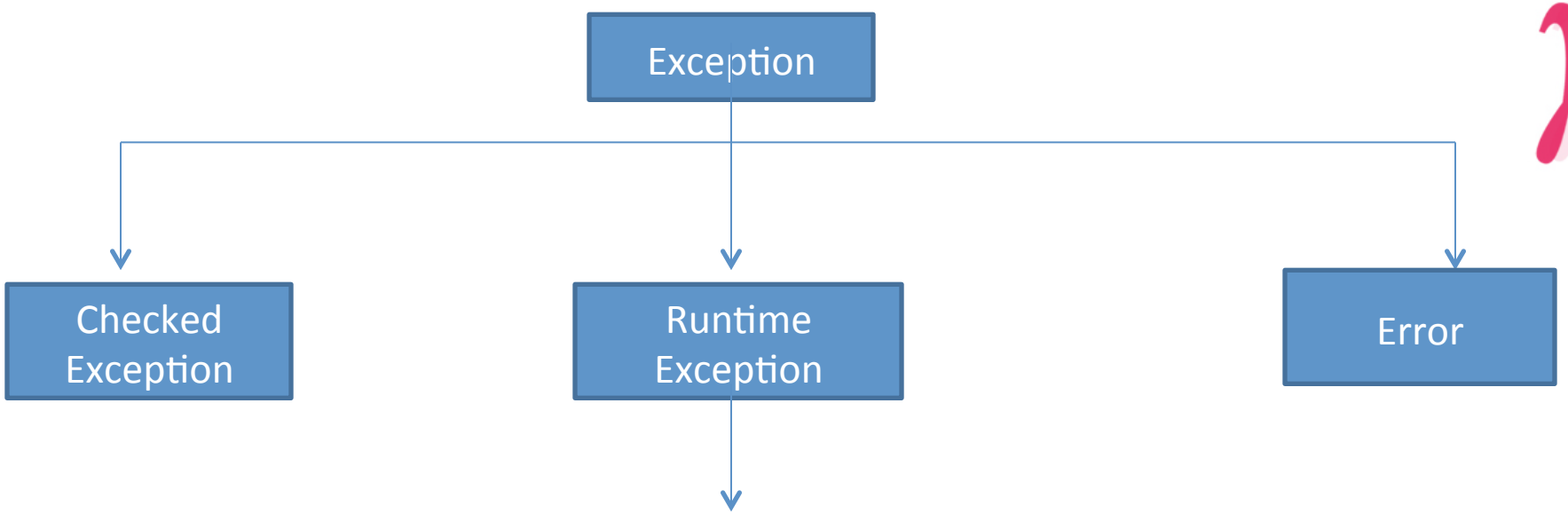
Types of Exception



Checked exceptions are checked at compile-time.

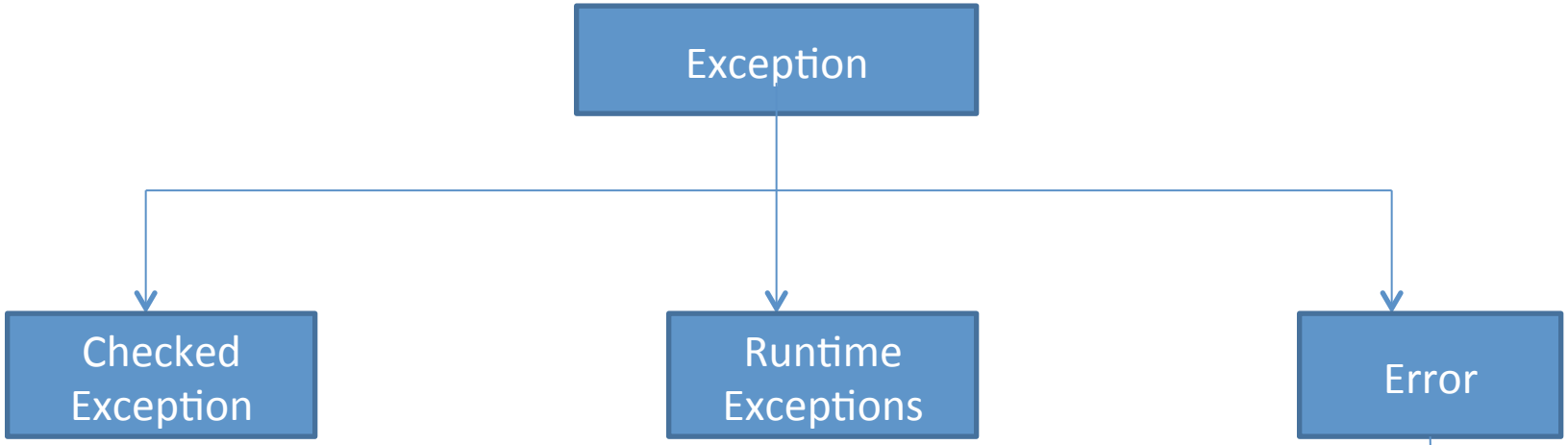
It means if a method is throwing a checked exception then it should handle the exception using try-catch block or it should declare the exception using throws keywords, otherwise the program will give a compilation error.

Types of Exception



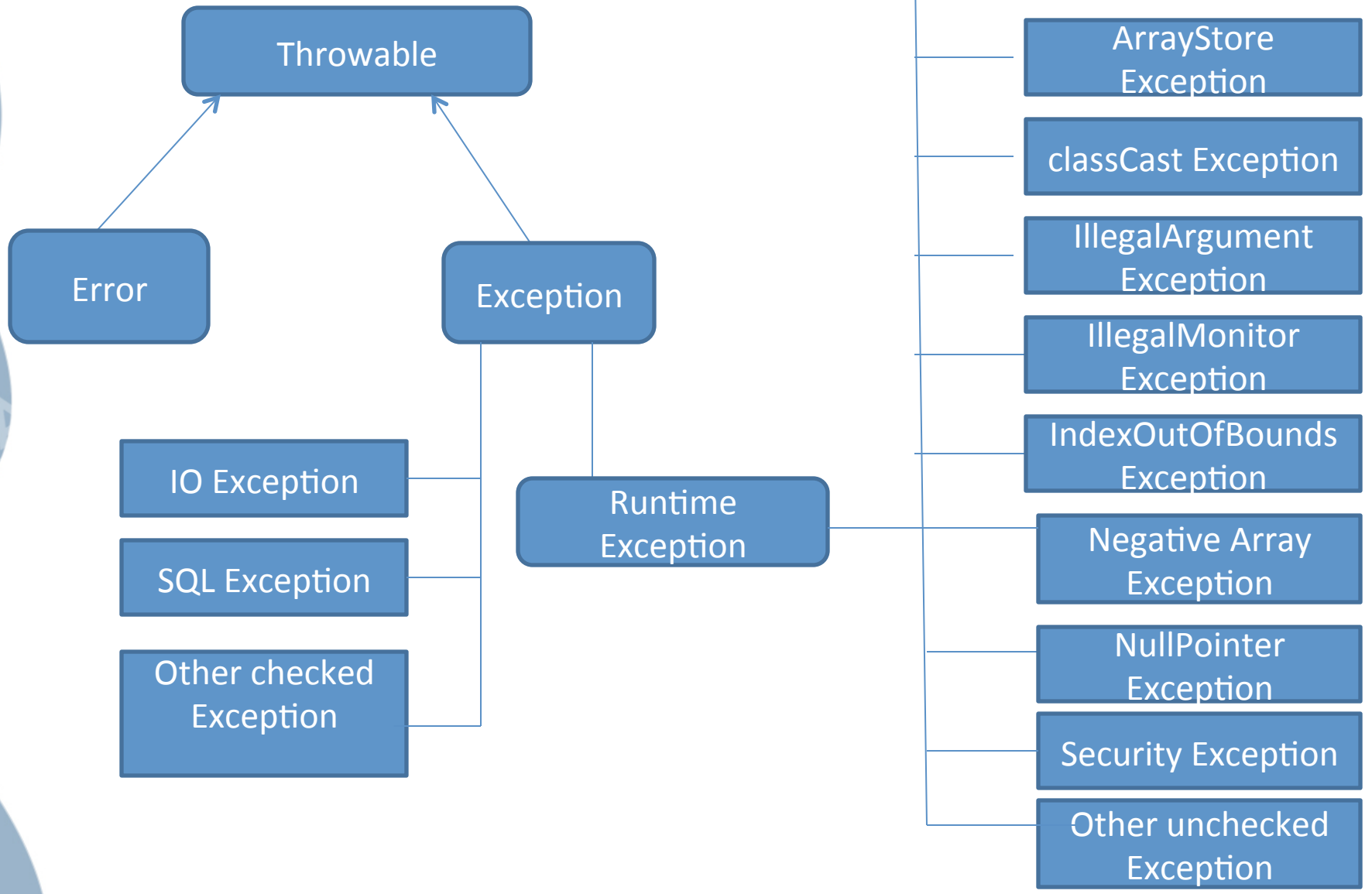
Unchecked exceptions are not checked at compile time . It means if your program is throwing an unchecked exception and even if you didn't handle/declare that exception, the program won't give a compilation error. It is up to the programmer to judge the conditions in advance, that can cause such exceptions 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?

```
// 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)
```

Array Index Out of Bound

```
int  arr[] = {1, 2, 3} ;

System.out.println(arr[3]) ;
```

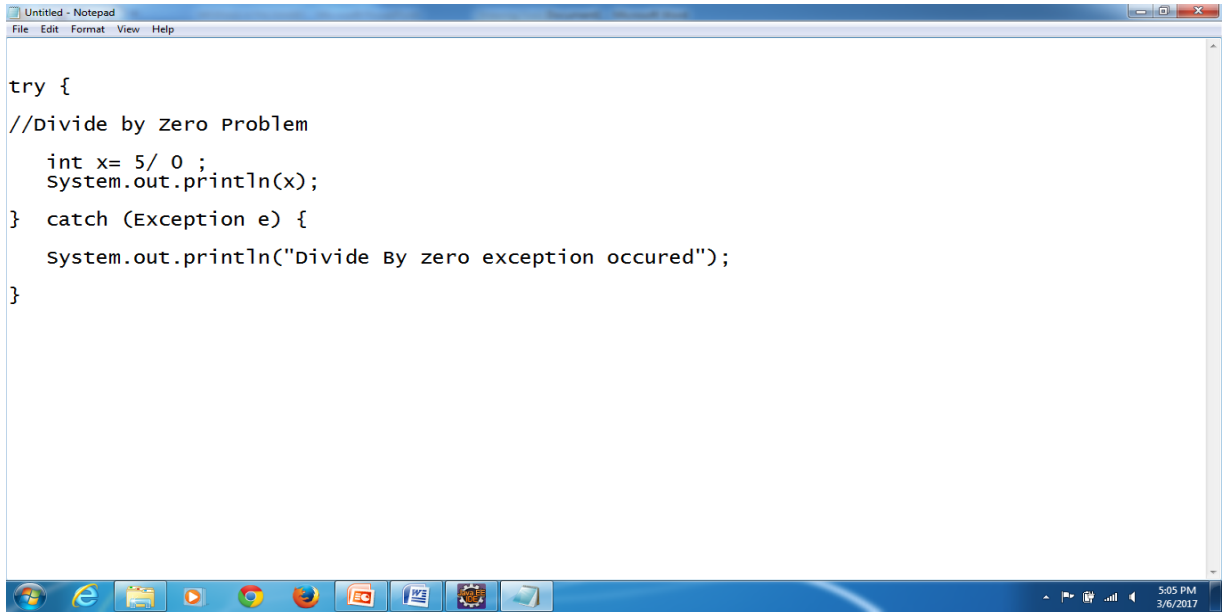
output

error!

program will halt

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

Solution



Output

Divide By Zero exception occurred.

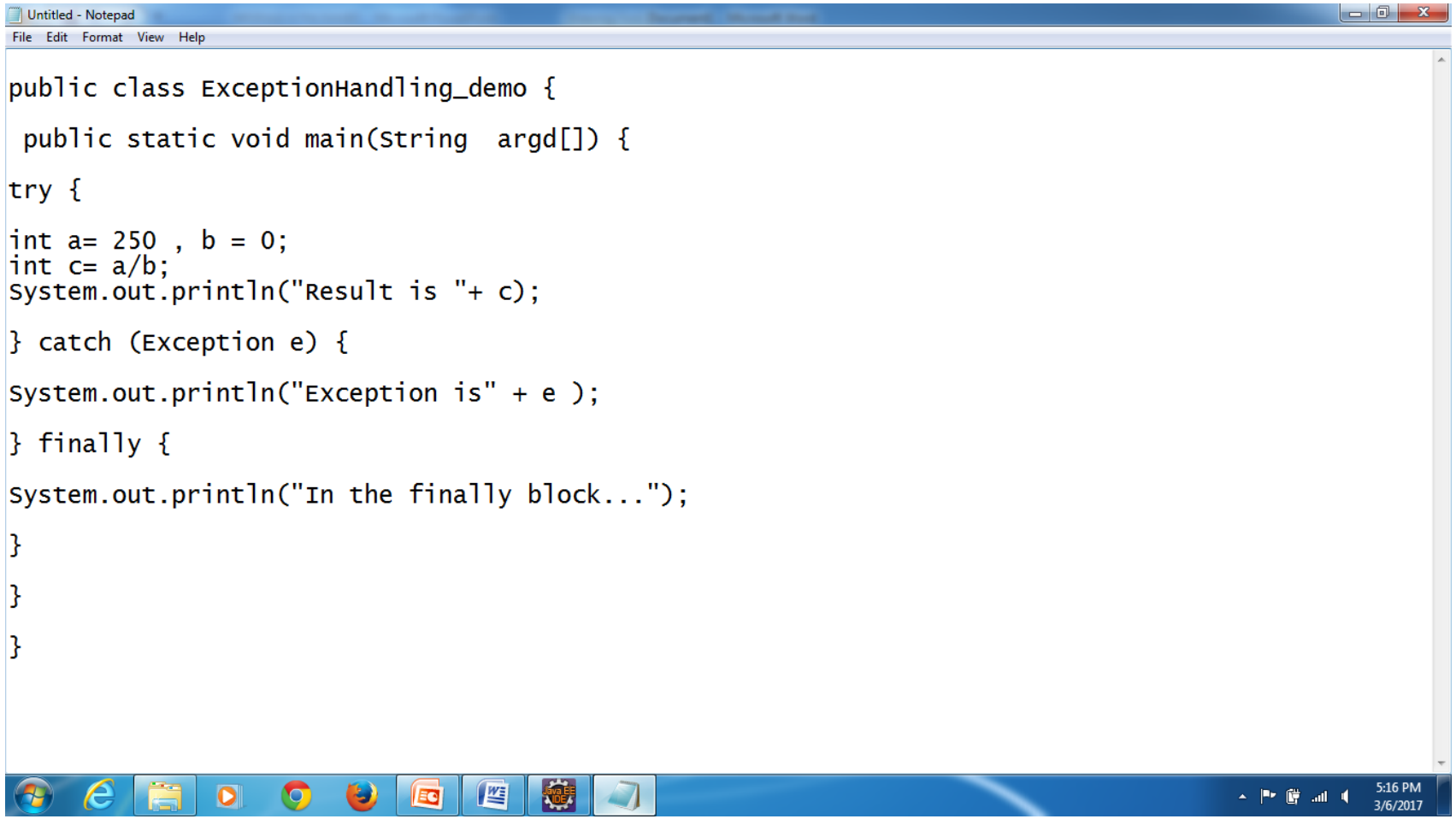
Exception Handling

- If there is a run time error then program is crashed and control 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.)

- 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.

Program on Exception Handling



Exception Handling

- One try can have multiple catch blocks. In this scenarios, depends on the type of exception thrown corresponding catch blocks 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 Multiple Catch Blocks

Untitled - Notepad

File Edit Format View Help

```
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...");

}

}

}
```

5:27 PM
3/6/2017

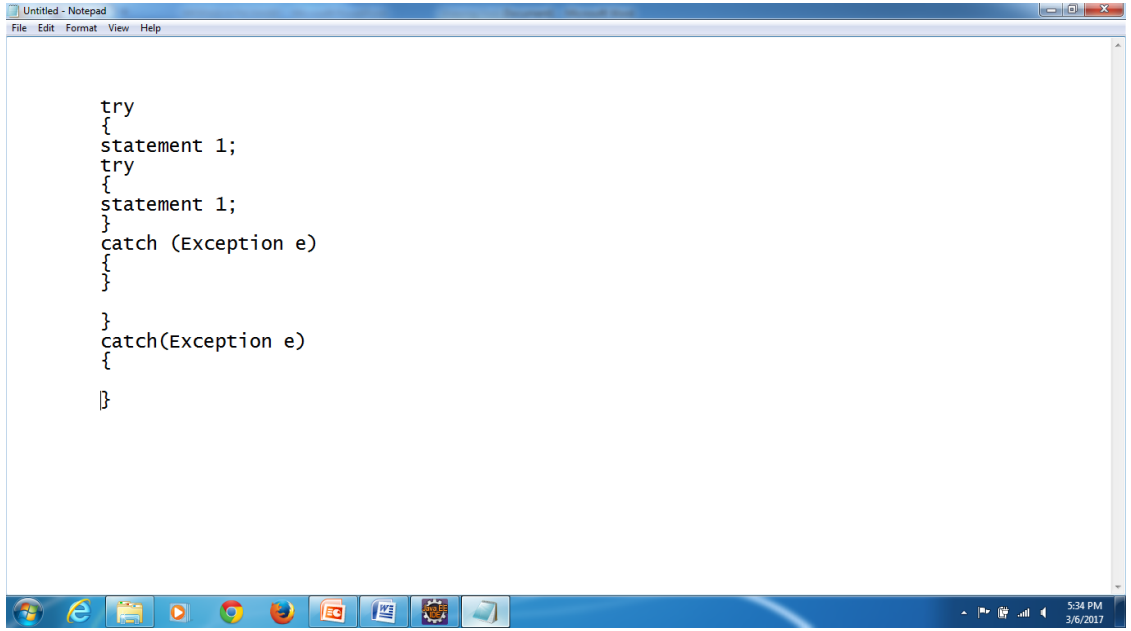
Nested try catch

Why use nested try block?

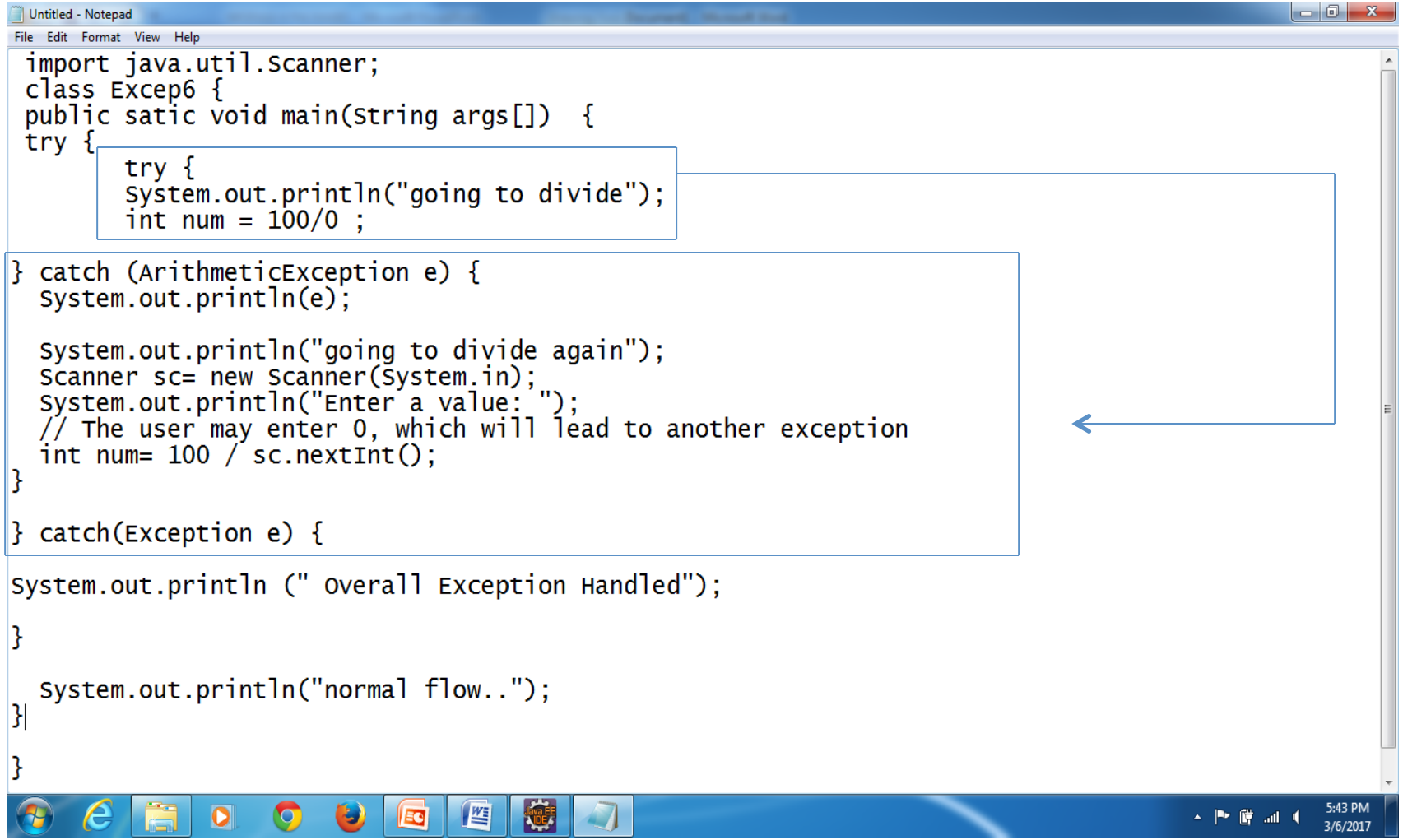
Sometimes a situation may arise where a part of a block may cause one error and the entire block itself ma cause another error.

In such cases, exception handlers have to be nested.

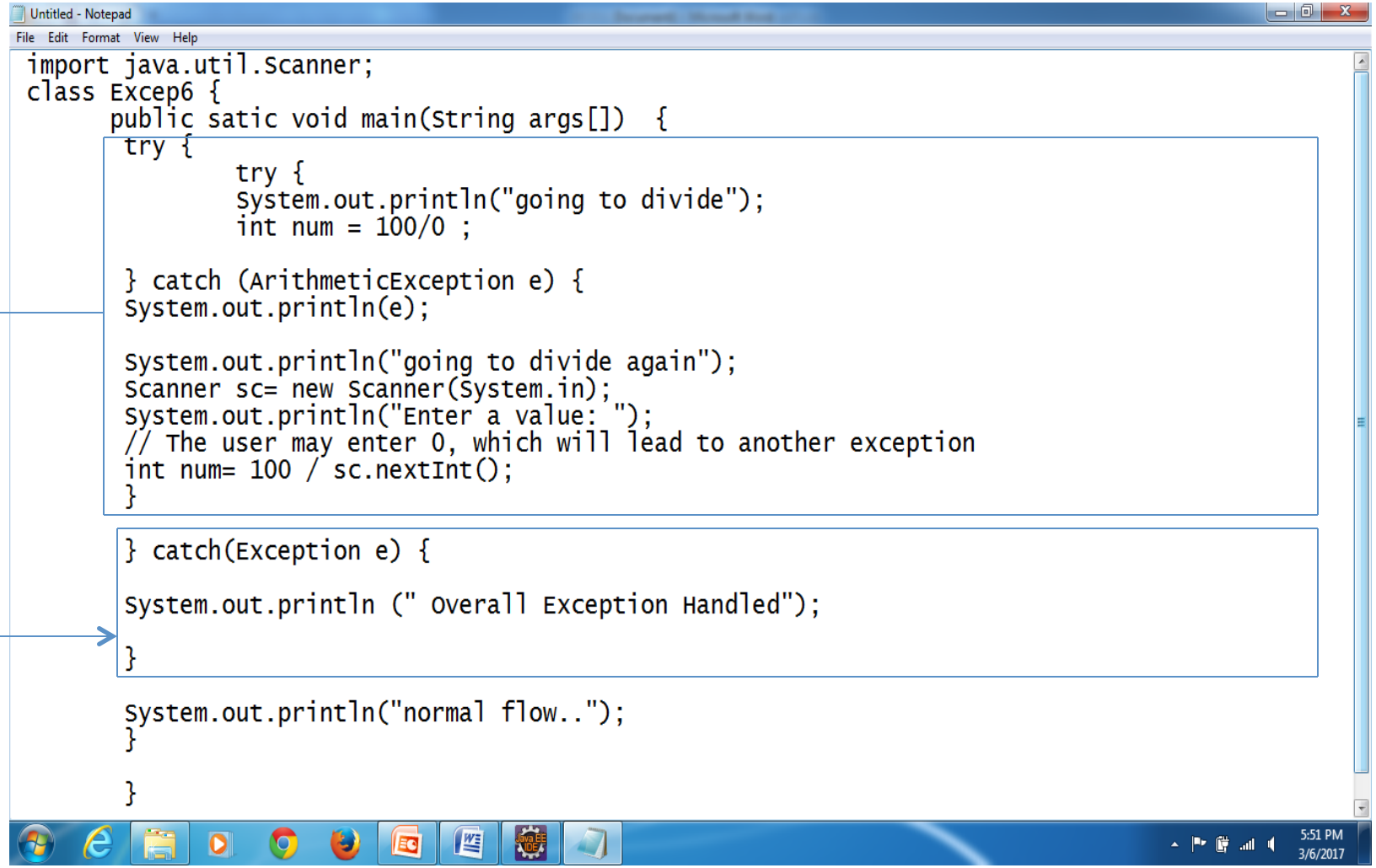
Syntax:



Program on Nested try catch



Program on Nested try catch



Why throw?

Example1 : 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

Untitled - Notepad

File Edit Format View Help

```
        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 IO exception and

the calling function will handle these IOException. Calling function is responsible for IOExceptions. Many exceptions can be added by adding comma opera

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

Program on Throws

Untitled - Notepad

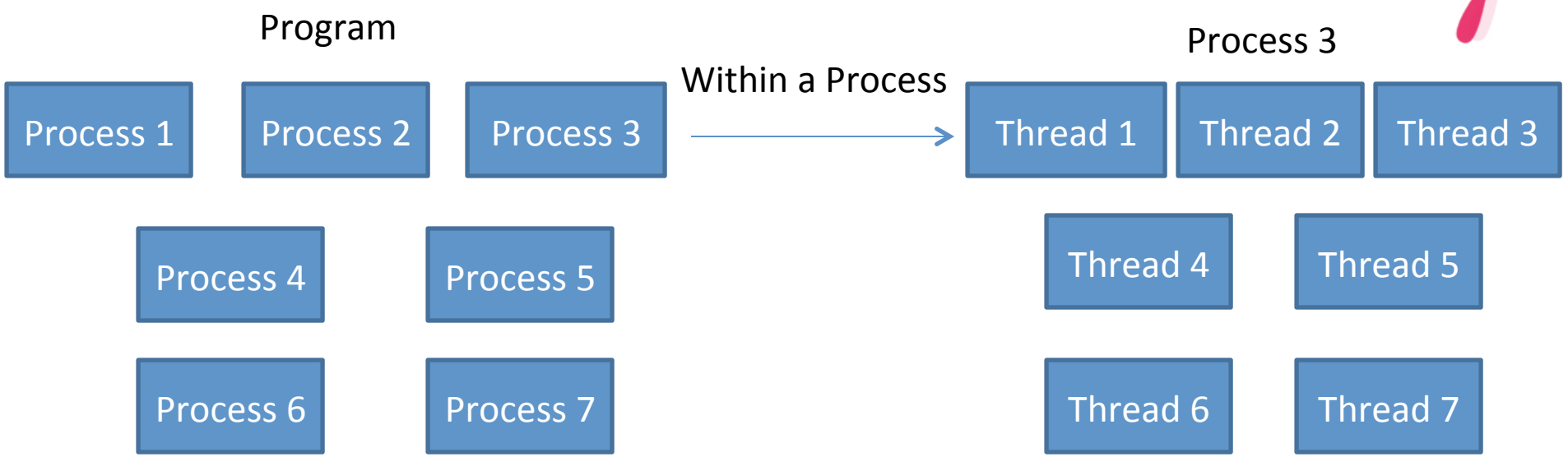
File Edit Format View Help

```
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) ;
        }

      }

  }
```


What is a Java Thread?

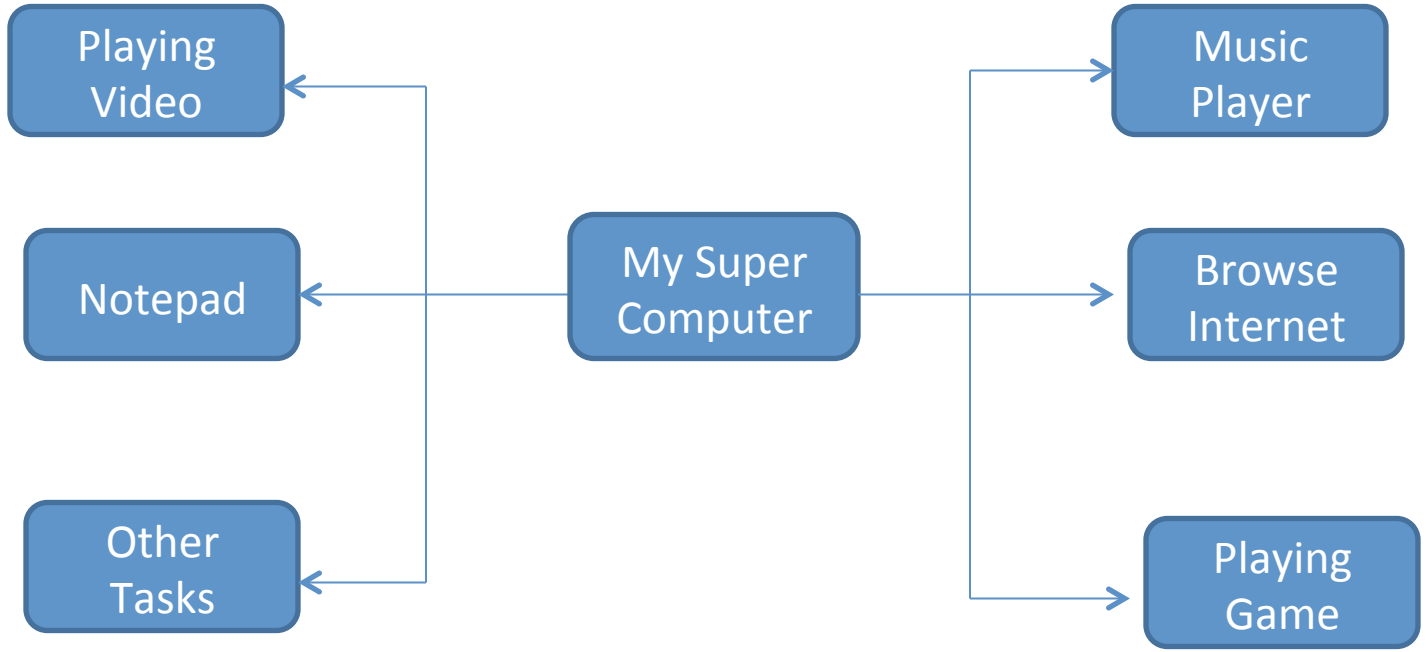
Program Vs Process Vs Thread

- **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.


Program
Process 1
Process 2
Process 3
Process 4
Process 5
Process 6
Process 7
Within a Process
Process 3
Thread 1
Thread 2
Thread 3
Thread 4
Thread 5
Thread 6
Thread 7

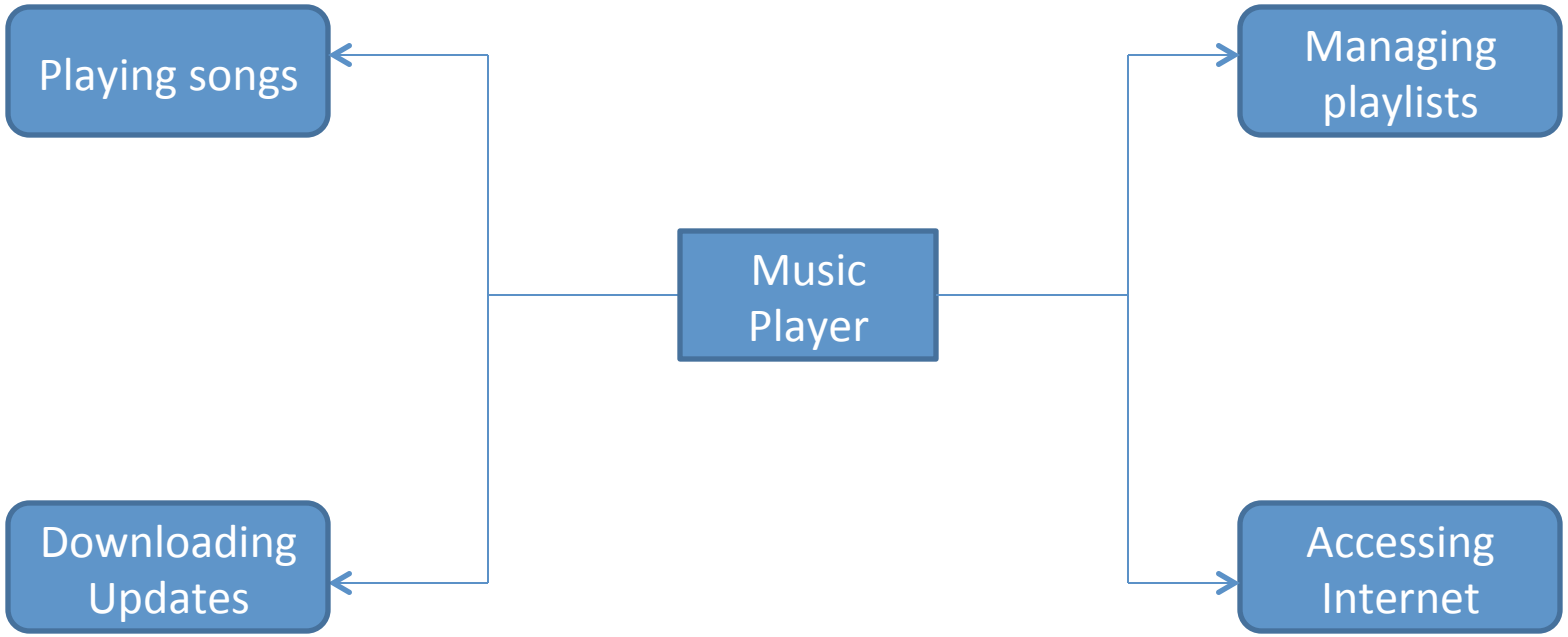
Multi Threading

Every computer has multiple processes running at a time.



Multi Threading (contd.)

Every application has multiple sub-process running at a time.

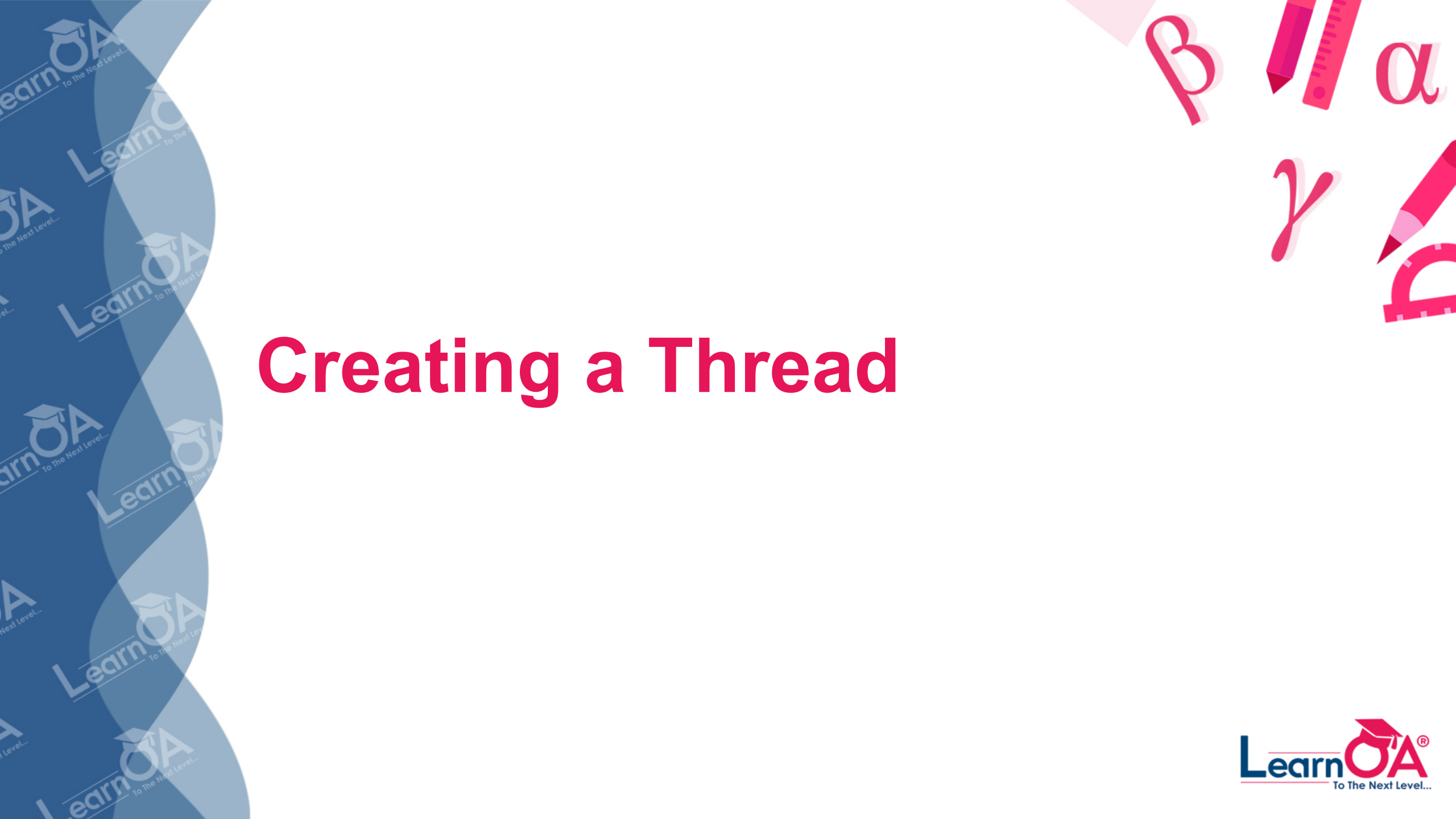


Multi Threading (contd.)

- 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.

What is a Java Thread?

- Thread is a lightweight sub process
- It is the smallest independent unit of a program
- Contains a separate path of execution
- Every Java program contains at least one thread
- A thread is created & controlled by the java.lang.Thread class


Creating a Thread

Creating A Thread

A thread in Java can be created using two ways

Thread Class

```
public class Thread
  extends Object
    implements Runnable
```

Runnable Interface

```
public interface Runnable
```

Thread Class

Runnable Interface

1. Create a Thread class
2. Override run() method
3. Create object of the class
4. Invoke start() method to execute the custom threads run()

```
public class MyThread extends Thread {
   public void run(){
    System.out.println("My Thread…");
  }
   public static void main(String[] args) {
     MyThread obj = new MyThread();
     obj.start();
}
```

Thread Class | Runnable Interface

1. Create a Thread class implementing Runnable interface
2. Override run() method
3. Create object of the class
4. Invoke start() method using the object

```
public class MyThread implements Runnable {
   public void run(){
     System.out.println("My Thread…");
   }
   public static void main(String[] args) {
     Thread t = new Thread(new MyThread());
     t.start();
}
```

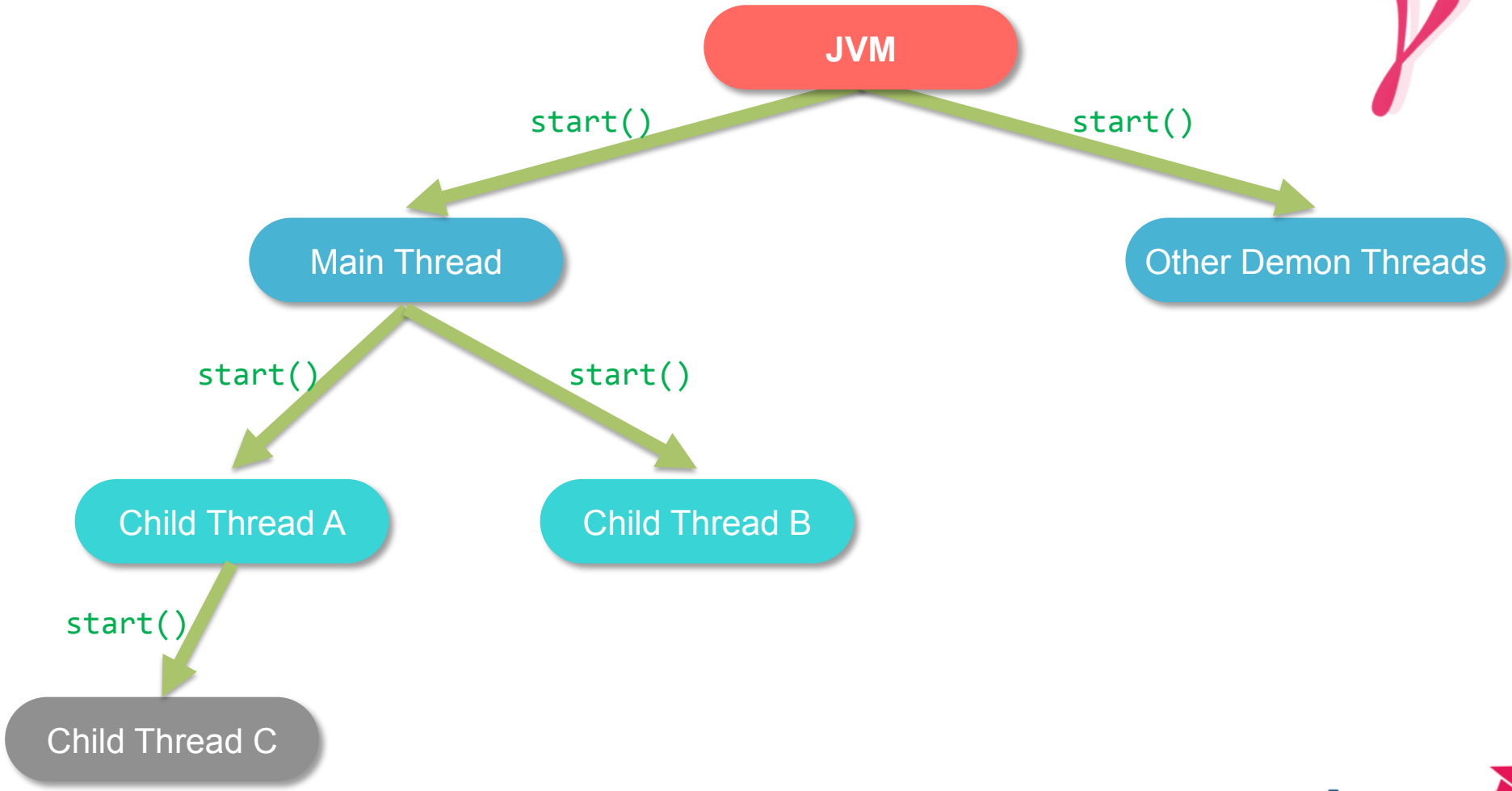
Java Main Thread

Java Main Thread

- Main thread is the most important thread of a Java Program
- It is executed whenever a Java program starts
- Every program must contain this thread for its execution to take place
- Java main Thread is needed because of the following reasons

1. From this other “child” threads are spawned
2. It must be the last thread to finish execution i.e when the main thread stops program terminates

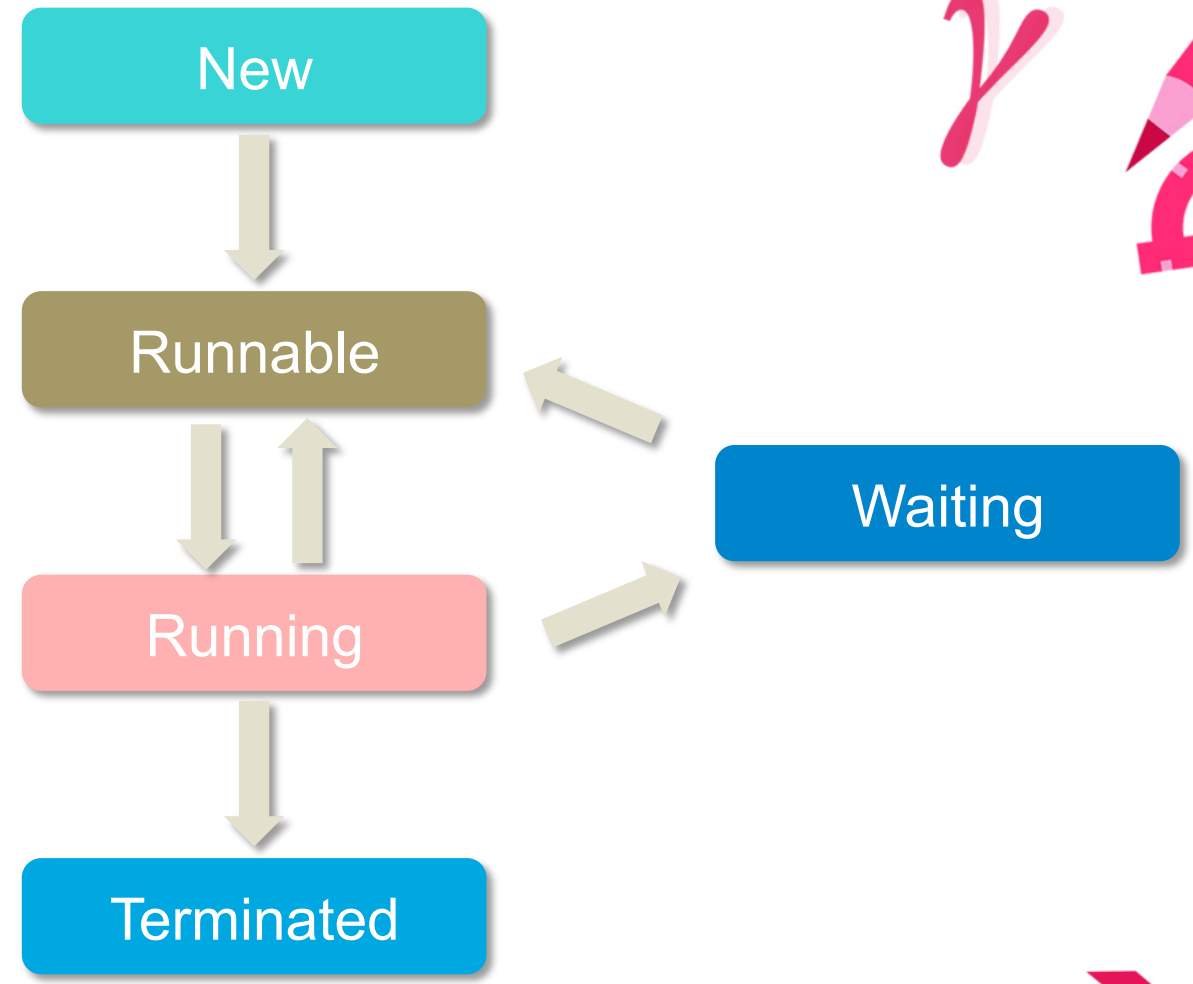
Java Main Thread



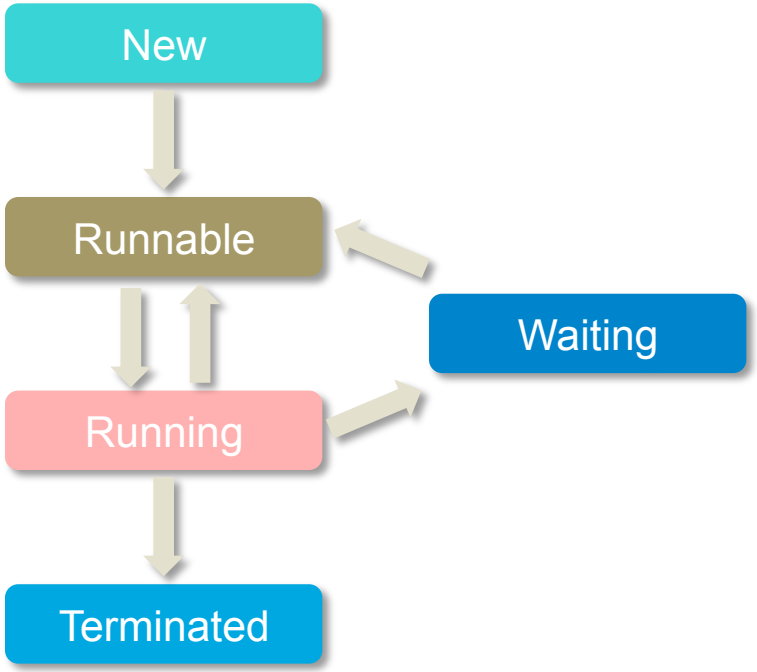
Java Thread Life-Cycle

Thread Lifecycle

A Java thread can lie only in one of the shown states at any point of time



Thread Lifecycle



New

A new thread begins its life cycle in this state & remains here until the program starts the thread. It is also known as a **born thread**.

Runnable

Once a newly born thread starts, the thread comes under runnable state. A thread stays in this state is until it is executing its task.

Running

In this state a thread starts executing by entering run() method and the yield() method can send them to go back to the Runnable state.

Waiting

A thread enters this state when it is temporarily in an inactive state i.e it is still alive but is not eligible to run. It is can be in waiting, sleeping or blocked state.

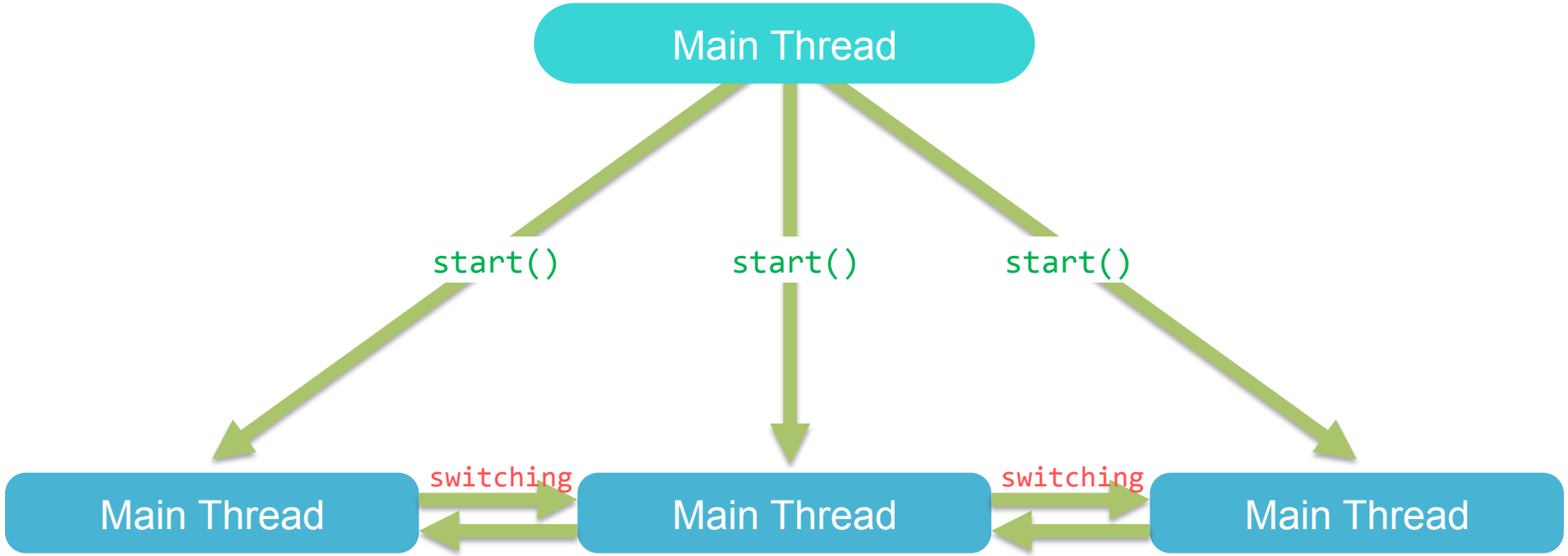
Terminated

A runnable thread enters the terminated state when it completes its task or otherwise terminates.


Multi Threading In Java

Multi – Threading

Multi threading is the ability of a program to run two or more threads concurrently, where each thread can handle a different task at the same time making optimal use of the available resources



Thread Methods

Creating Multiple Threads

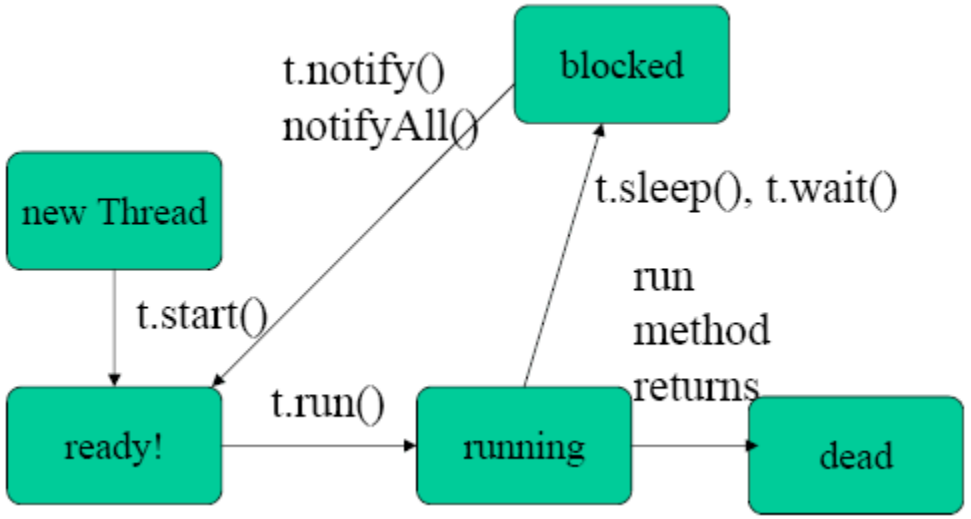
Joining Threads

Thread.sleep()

Inter Thread Communication

Daemon Thread

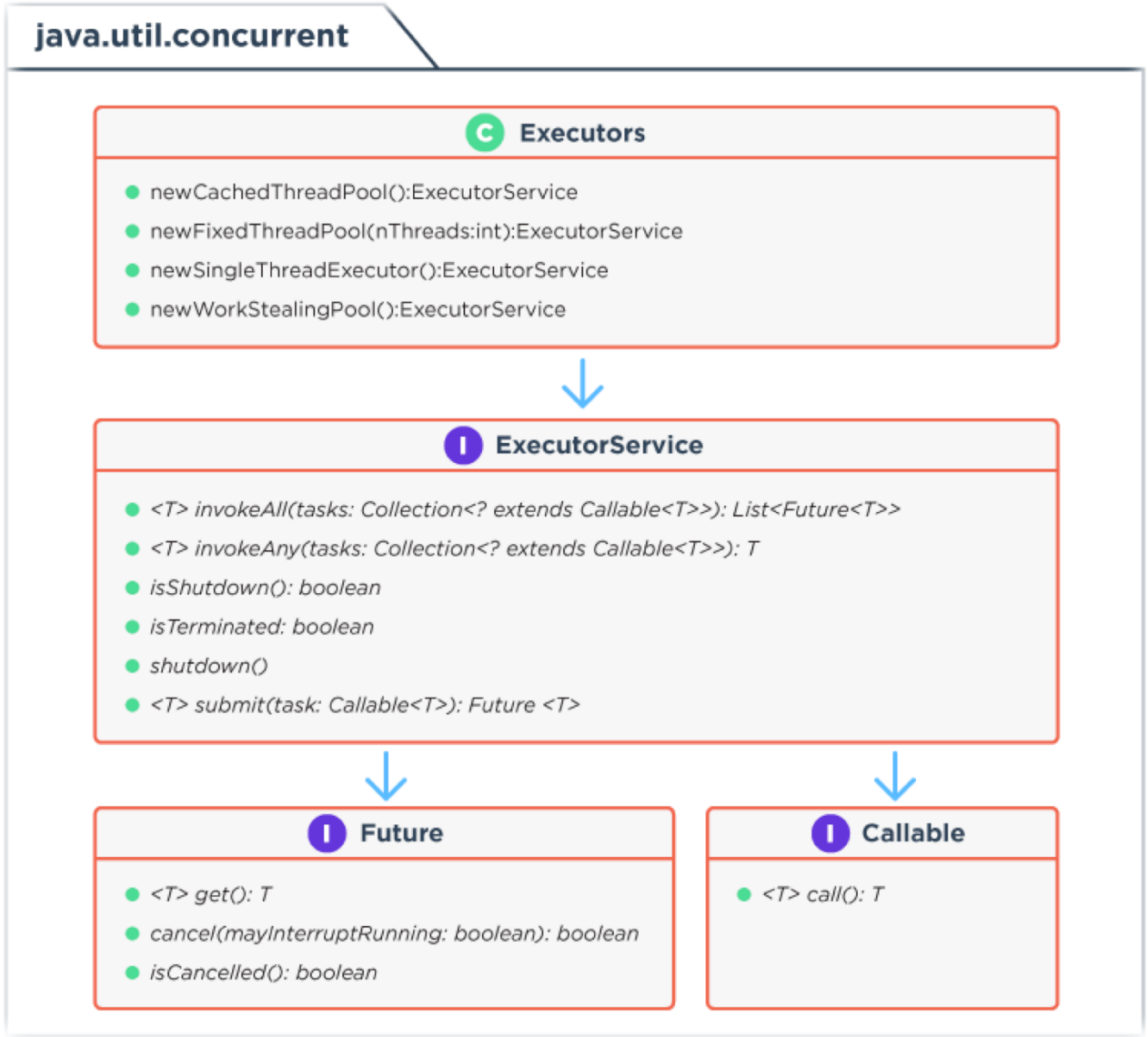
Thread -Methods



- 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.

Concurrent Package

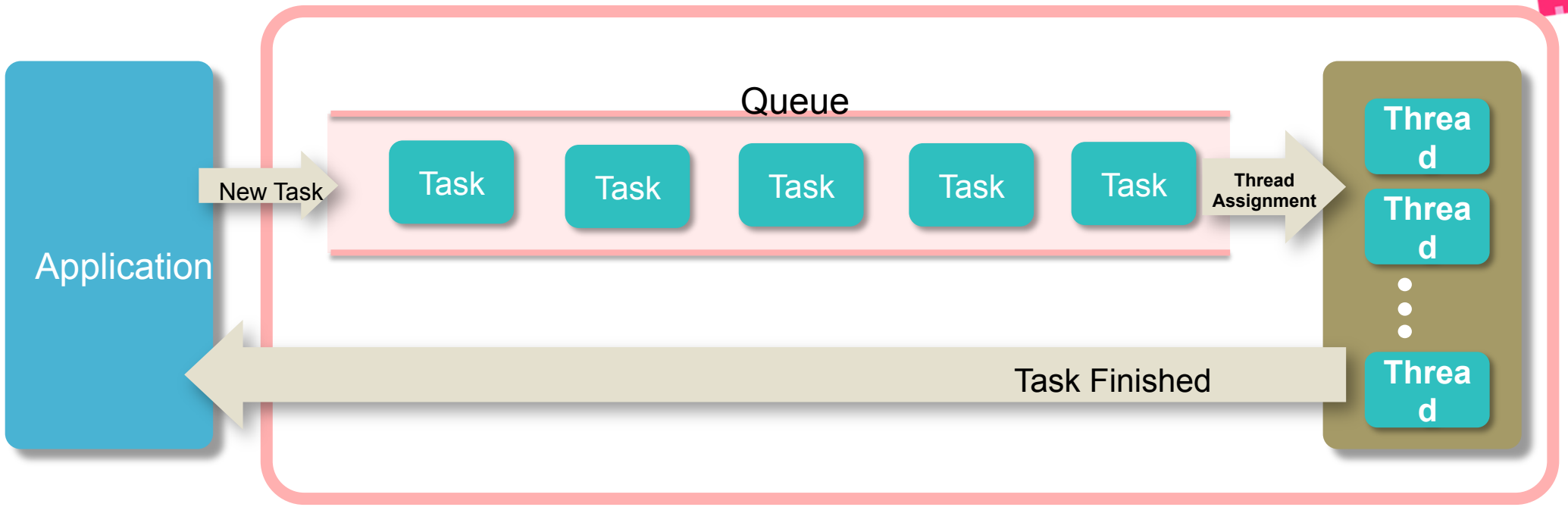
Java Concurrent Package




Thread Pool

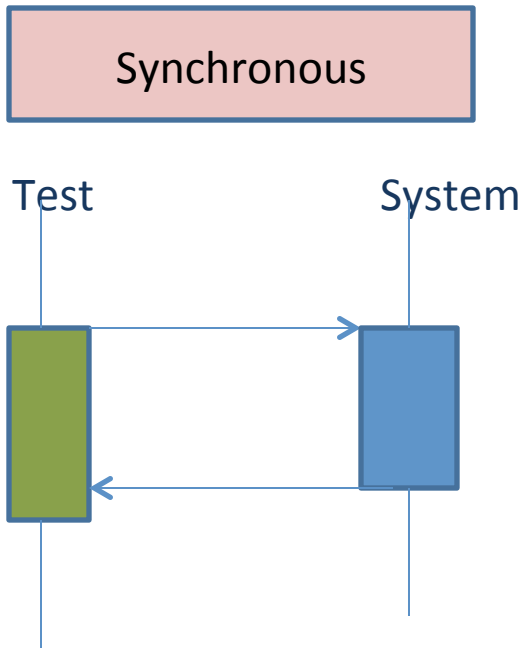
Thread Pool

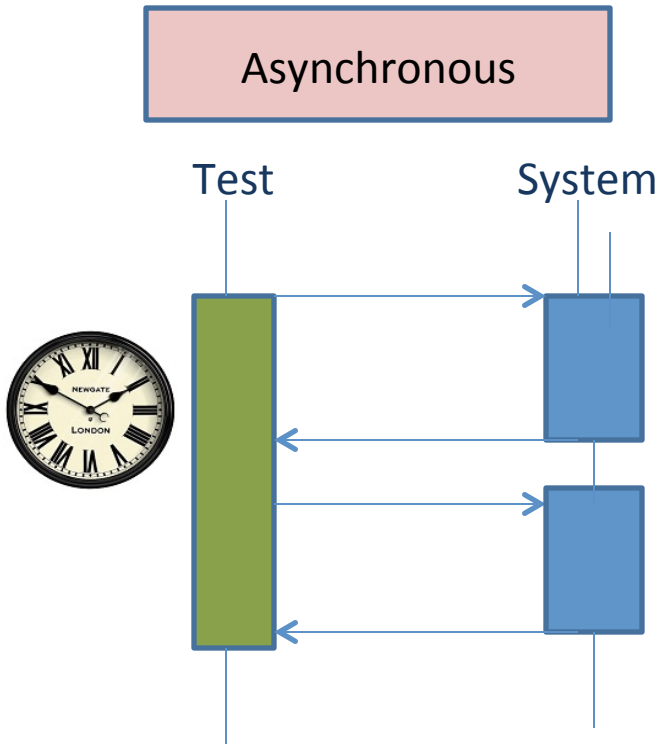
Java thread pool manages the pool of worker threads and contains a queue that keep the tasks waiting to get executed



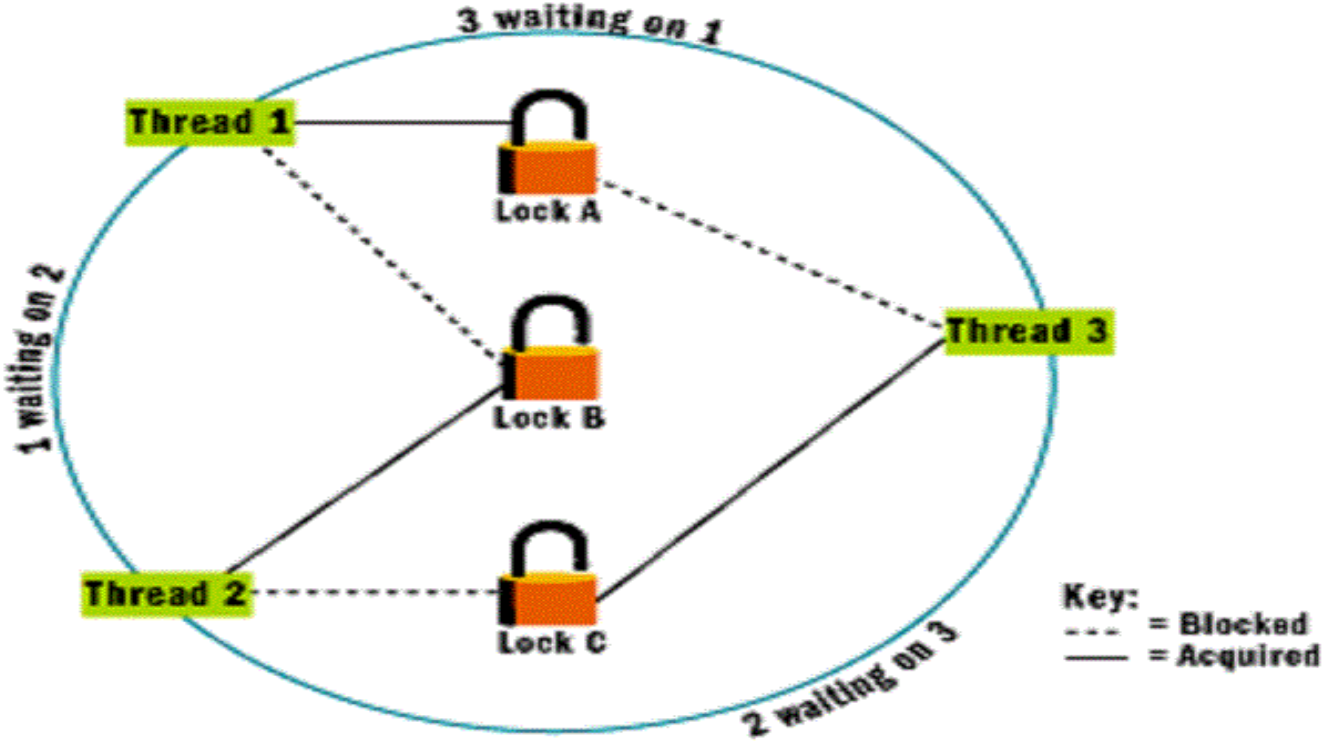
Synchronization

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.

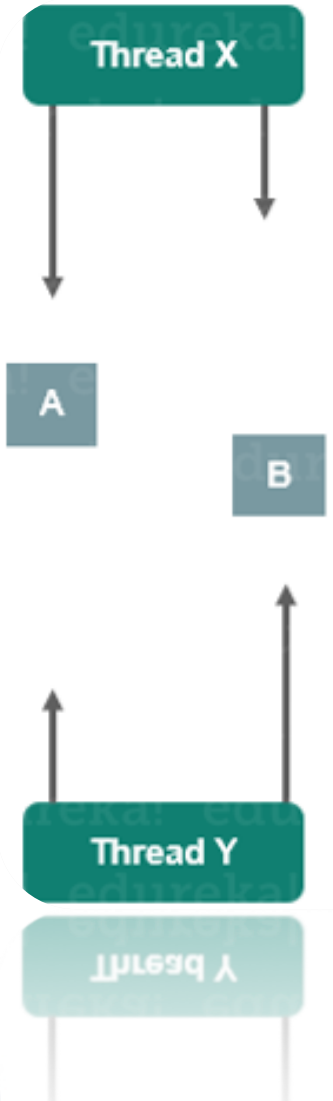
Deadlock

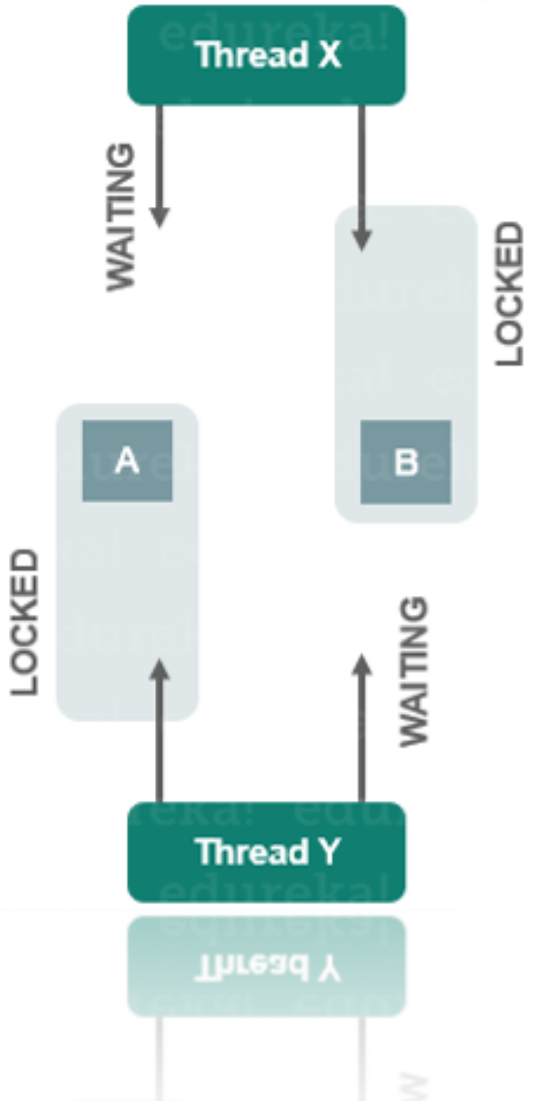
- Deadlock describes a situation where two or more threads are blocked forever, waiting for each other.

or

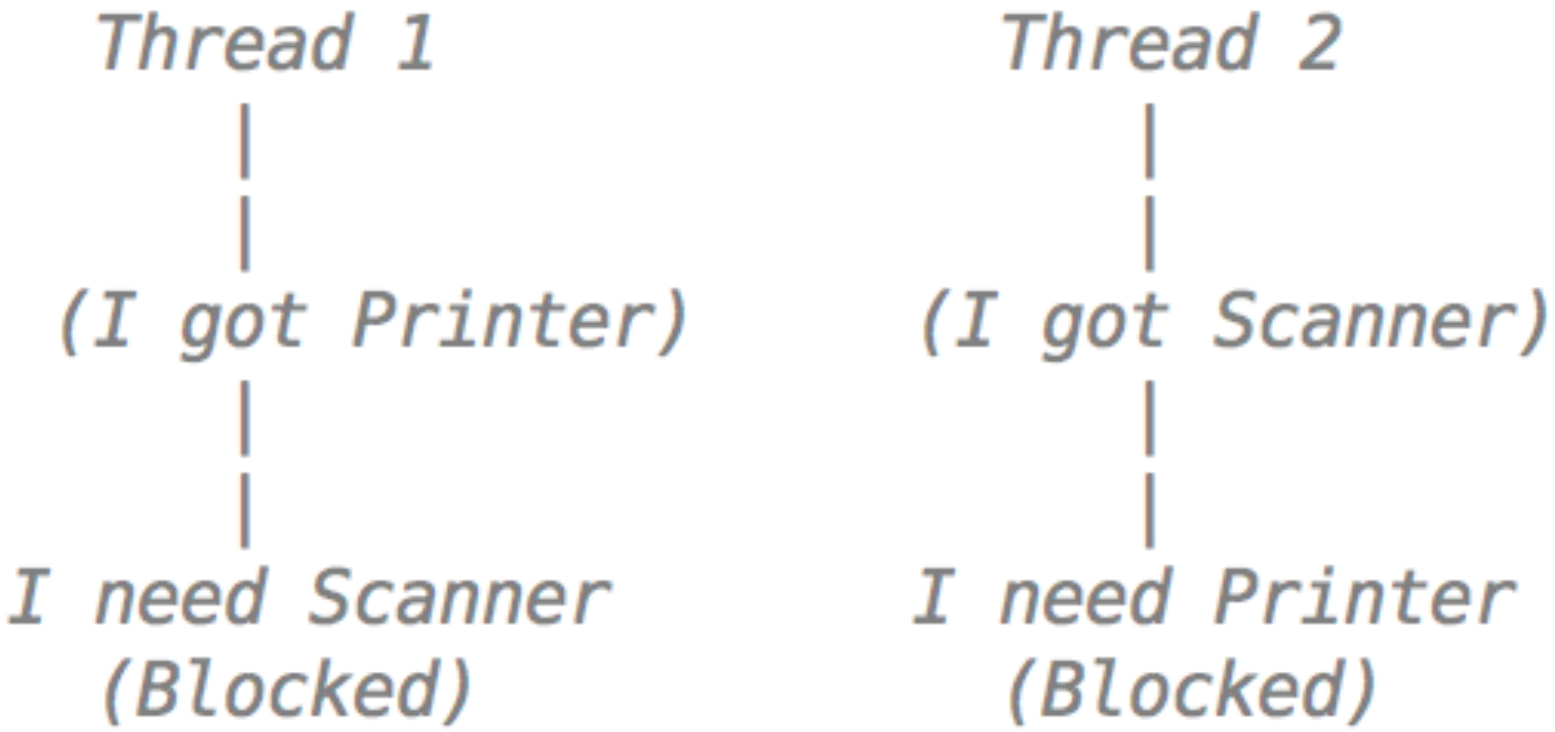
- A deadlock is a situation in which two or more competing actions are each waiting for the other to finish, and thus neither ever does.

Deadlock


Thread X
A
B
Thread Y


Thread X
WAITING
LOCKED
A
B
LOCKED
WAITING
Thread Y

Deadlock - Example



Thank You