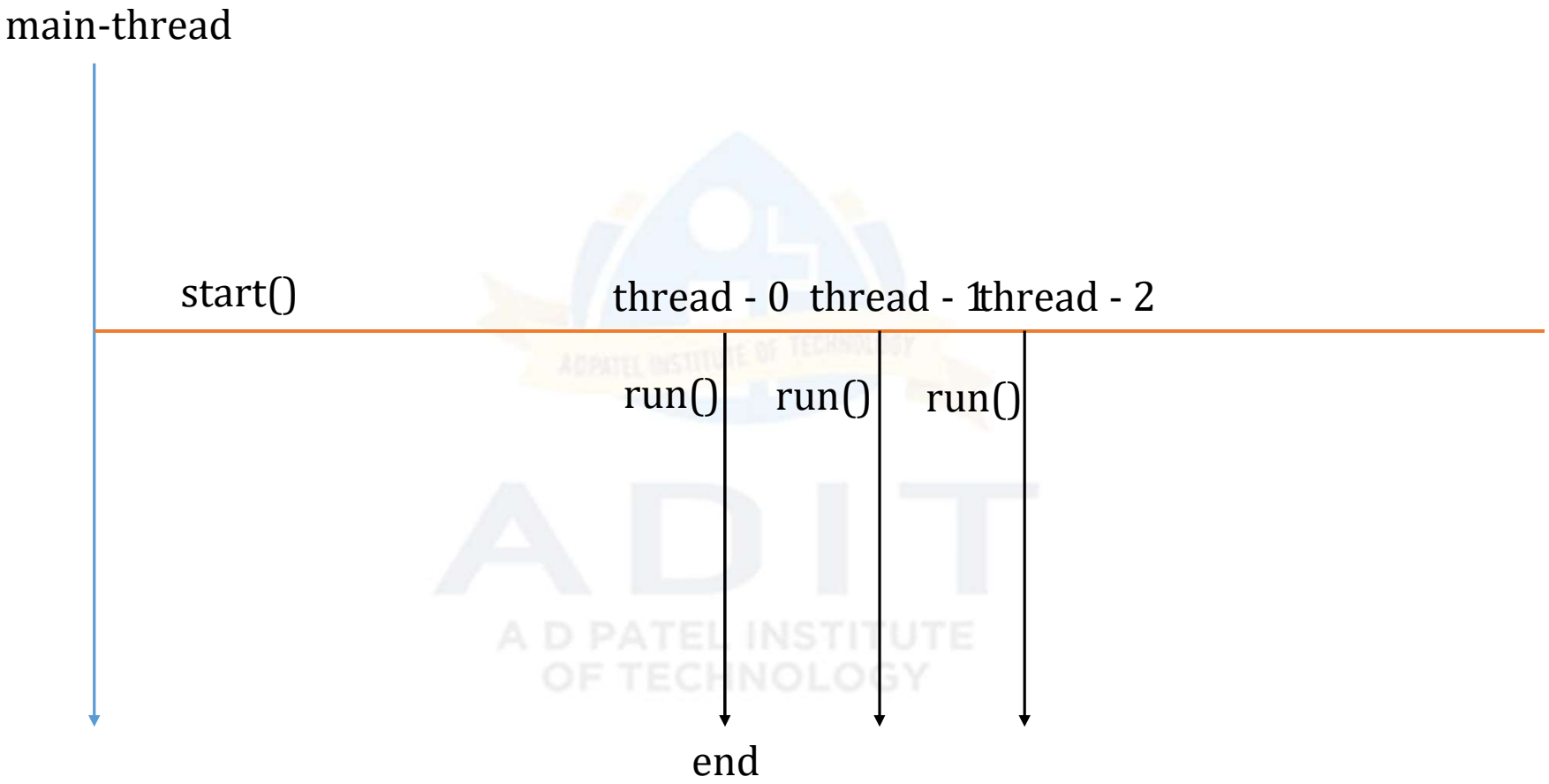
202044502 - Programming with Java

UNIT 4: THREADING

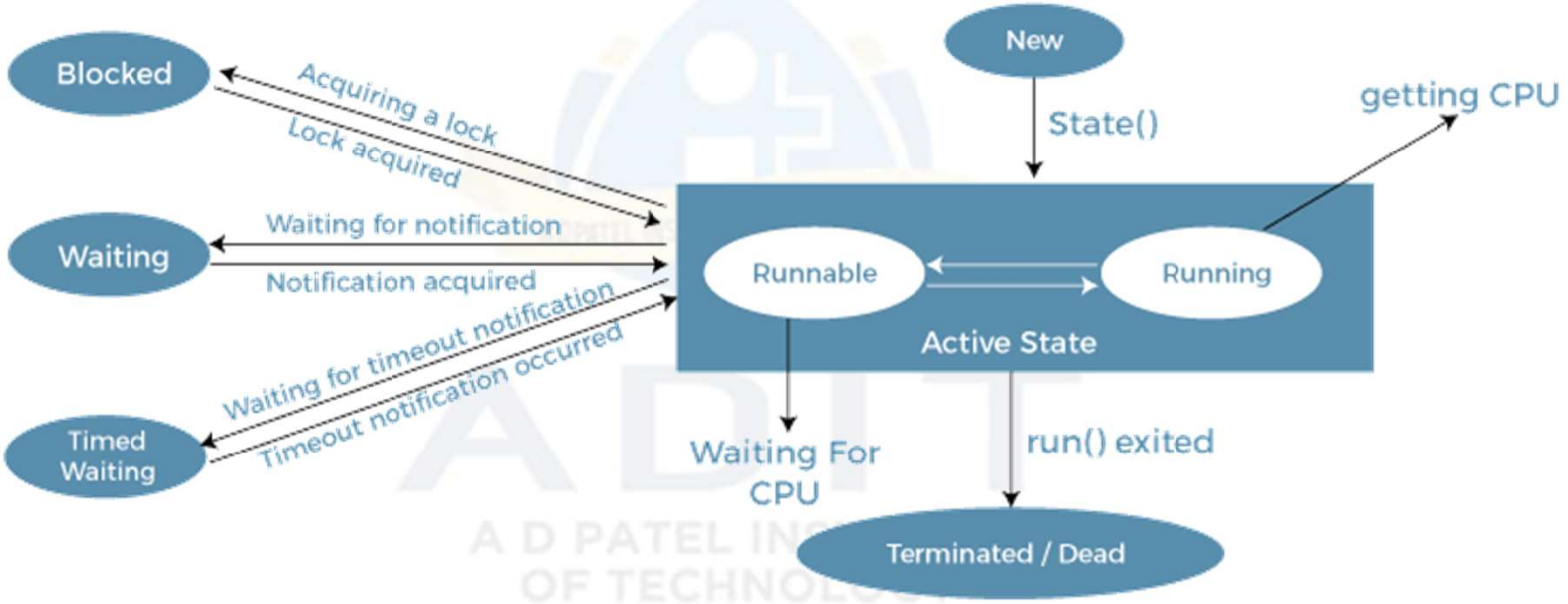
Mali Nayan

Department of Information Technology

A D Patel Institute of Technology

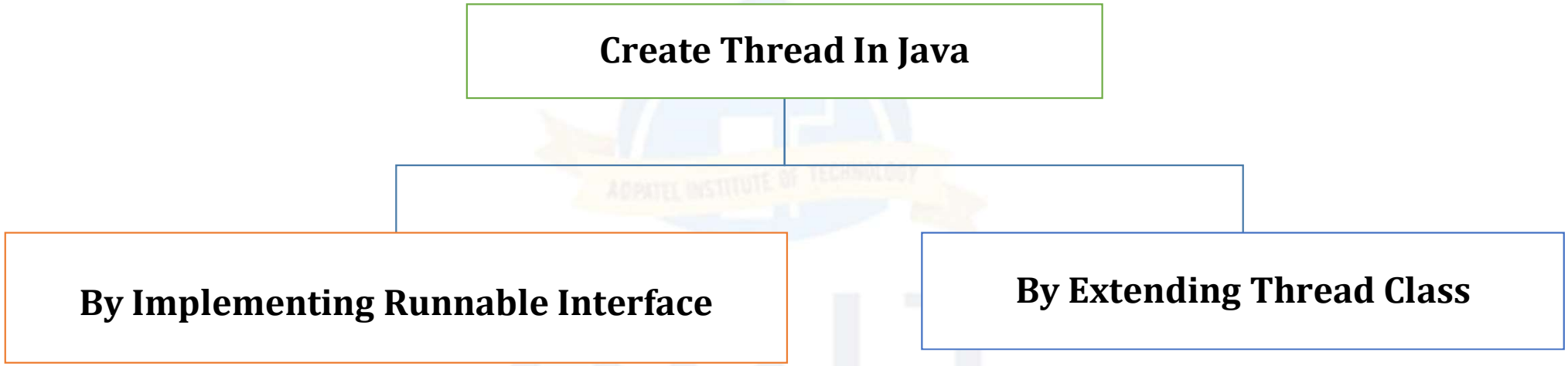

main-thread
start()
thread - 0
thread - 1
thread - 2
run()
run()
run()
end

Thread Life Cycle



Life Cycle of a Thread

[Life cycle of a thread in Java - javatpoint]



Create Thread By Extending Thread Class

```
public class Example1 {
    public static void main(String[] args) {
        System.out.println("Running " + Thread.currentThread().getName());

        System.out.println("Creating New Thread");
        MyThread myThread = new MyThread();
        myThread.start();

        MyThread myThread1 = new MyThread();
        myThread.setName("My Thread 1");
        myThread1.start();
    }
}
class MyThread extends Thread{
    public void run() {
        System.out.println("Running " +
        Thread.currentThread().getName());
    }
}
```

Create Thread By Implementing Runnable Interface

```
package in.ac.adit.it.pwj;

public class Example2 {
   public static void main(String[] args) {
      System.out.println("Running " + Thread.currentThread().getName());

      System.out.println("Creating New Thread");
      DemoThread demoThread = new DemoThread();
      Thread thread = new Thread(demoThread);
      thread.start();
   }
}
class DemoThread implements Runnable{
      public void run() {
            System.out.println("Running " +
Thread.currentThread().getName());
      }
}
```

Multi Threading

```
public class MultiThreading {
	public static void main(String[] args) {
		MultiThread multiThread1 = new MultiThread();
		MultiThread multiThread2 = new MultiThread();

		multiThread1.start();
		multiThread2.start();
	}
}
class MultiThread extends Thread{
	public void run() {
		for(int i=0; i<10; i++) {
			System.out.println(Thread.currentThread().getName()+ " i = "+i);
			try {
				Thread.sleep(1000);
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
		}
	}
}
```

Thread Priority

```
public class ThreadPriorityDemo {
	public static void main(String[] args) {
		ThreadPriority threadPriority1 = new ThreadPriority();
		ThreadPriority threadPriority2 = new ThreadPriority();

		threadPriority1.setPriority(Thread.MAX_PRIORITY);
		threadPriority2.setPriority(Thread.MIN_PRIORITY);
		threadPriority1.start();
		threadPriority2.start();
	}
}
class ThreadPriority extends Thread{
	public void run() {
		for(int i=0; i<10; i++) {
			System.out.println(Thread.currentThread().getName()+ " i = "+i);
		}
	}
}
```

Thread Methods

- start()
- run()
- join()
- sleep()
- Yield()
- interrupt()

Join

- Thread does not start running until another thread ends
- If join() is called on a Thread instance, the currently running thread will block until the Thread instance has finished executing. The join() method waits at most this many milliseconds for this thread to die. A timeout of 0 means to wait forever

public final void join() throws InterruptedException

public final void join(long millis) throws InterruptedException

```
package in.ac.adit.it.pwj.methods;
public class JoinDemo {
    public static void main(String[] args) {
        System.out.println("Running " + Thread.currentThread().getName());
        System.out.println("Creating New Thread");

        Example example1 = new Example();
        Example example2 = new Example();

        example1.setName("Example 1");
        example2.setName("Example 2");

        example1.start();
        try {
                example1.join();
        } catch (InterruptedException e) {
                e.printStackTrace();
        }
    example2.start();
    }
}
```

```
class Example extends Thread {
    public void run() {
        System.out.println("Running " + Thread.currentThread().getName());

        for (int i = 0; i < 10; i++) {
            System.out.println(Thread.currentThread().getName() + " - " + i);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Yield()

- The **yield()** basically means that the thread is not doing anything particularly important and if any other threads or processes need to be run, they should run. Otherwise, the current thread will continue to run.
- Yield means currently executing thread gives chance to the threads that have equal priority in the Thread-pool.

```
package in.ac.adit.it.pwj.methods;

public class YieldDemo extends Thread {
    public void run() {
        for (int i = 0; i < 3; i++)
                System.out.println(Thread.currentThread().getName() + " in control");
    }

    public static void main(String[] args) {
        YieldDemo t1 = new YieldDemo();
        YieldDemo t2 = new YieldDemo();

        t1.start();
        t2.start();
        for (int i = 0; i < 10; i++) {
            t1.yield();
            System.out.println(Thread.currentThread().getName() + " in control");
        }
    }
}
```

Sleep()

- Causes the currently executing thread to sleep for the specified number of milliseconds

```
try{
    Thread.sleep(3*1000)
}catch(InterruptedException e){
    e.printStackTrace();
}
```

- public static void sleep(long millis) throws InterruptedException
- public static void sleep(long millis, int nanos) throws InterruptedException

```java
package in.ac.adit.it.pwj.methods;

public class SleepDemo {
    public static void main(String[] args) {
        System.out.println("Running " + Thread.currentThread().getName());
        System.out.println("Creating New Thread");
        MyThread myThread = new MyThread();
        myThread.start();
        MyThread myThread1 = new MyThread();
        myThread.setName("My Thread 1");
        myThread1.start();
    }
}
class MyThread extends Thread{
    public void run() {
        for(int i=0; i<10; i++) {
            try {
                Thread.sleep(3 * 1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Synchronization in Java

- Synchronization in Java is the capability to control the access of multiple threads to any shared resource.
- Java Synchronization is better option where we want to allow only one thread to access the shared resource.

- Synchronized
  - Method
  - Block
  - Static

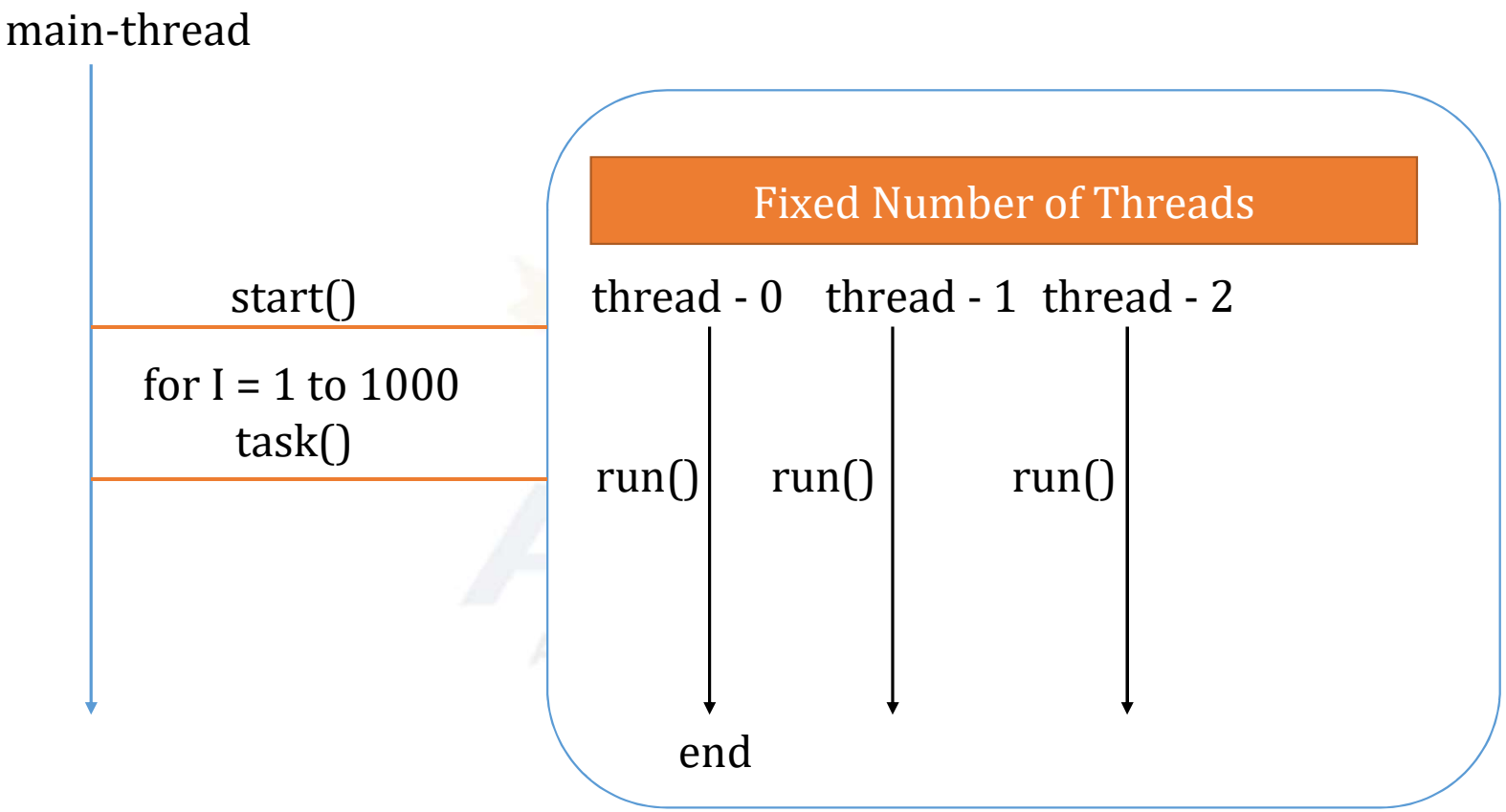
```
class Table {
   public synchronized void printTable(int value) {
      for (int i = 1; i <= 10; i++) {
      System.out.println(Thread.currentThread().getName() + "
      - " + i + " * " + value + " = " + i * value);
         try {
            Thread.sleep(1000);
         } catch (InterruptedException e) {
            e.printStackTrace();
         }
      }
   }
}
```

```
public class SynchronizationDemo {
        public static void main(String[] args) {
                PrintTable printTable = new PrintTable();

                Thread thread1 = new Thread() {
                        public void run() {
                                printTable.display(50);
                        }
                };
                Thread thread2 = new Thread() {
                        public void run() {
                                printTable.display(50);
                        }
                };
                thread1.setName("Thread 1");
                thread2.setName("Thread 2");

                thread1.start();
                thread2.start();
        }
                class PrintTable {
                        public synchronized void display(int value) {
                                for (int i = 0; i < value; i++){
                                        System.out.println(Thread.currentThread() + " Value : " + i);
                                }
                        }
                }
}
```

```
class BankAccount {
    private double balance = 10000;

    public synchronized void withdraw(double amount) throws InterruptedException {
        System.out.println("Withdraw Amout : " + amount);
        if ((balance - amount) < 0) {
            System.out.println("Waiting For Depositing Amount");
            wait();
        }
        Thread.sleep(2000);
        balance = balance - amount;

        System.out.println("Updated Balance: " + balance);
    }

    public synchronized void deposit(double amount) throws InterruptedException {
        System.out.println("Deposit Amout " + amount);
        balance = balance + amount;
        System.out.println("Updated Balance Amout " + balance);

        Thread.sleep(2000);

        notify();
    }
}
```

```
class InnerThreadCommunication {
    public static void main(String args[]) throws InterruptedException {
        BankAccount account = new BankAccount();
        Thread thread = new Thread() {
            public void run() {
                try {
                    account.withdraw(15000);
                } catch (InterruptedException ex) {
                    ex.printStackTrace();
                }
            }
        };
        Thread thread2 = new Thread() {
            public void run() {
                try {
                    account.deposit(7000);
                } catch (InterruptedException ex) {
                    ex.printStackTrace();
                }
            }
        };

        thread.start();
        thread2.start();
    }
}
```

Thread Pool


main-thread
start()
for I = 1 to 1000
task()
Fixed Number of Threads
thread - 0
thread - 1
thread - 2
run()
run()
run()
end

```java
package in.ac.adit.it.pwj.ThreadPool;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
public class ThreadPoolExample1 {
	public static void main(String[] args) {
	    int core = Runtime.getRuntime().availableProcessors();
	    System.out.println("Core : "+core);
	    ExecutorService executors = Executors.newFixedThreadPool(10);
	    for(int i = 0; i<1000; i++) {
	        executors.execute(new ThreadPoolTask());
	        try {
	                Thread.sleep(1000);
	        } catch (InterruptedException e) {
	                e.printStackTrace();
	        }
	    }
	}
}
class ThreadPoolTask implements Runnable {
	public void run() {
	        System.out.println(Thread.currentThread().getName());
	}
}
```