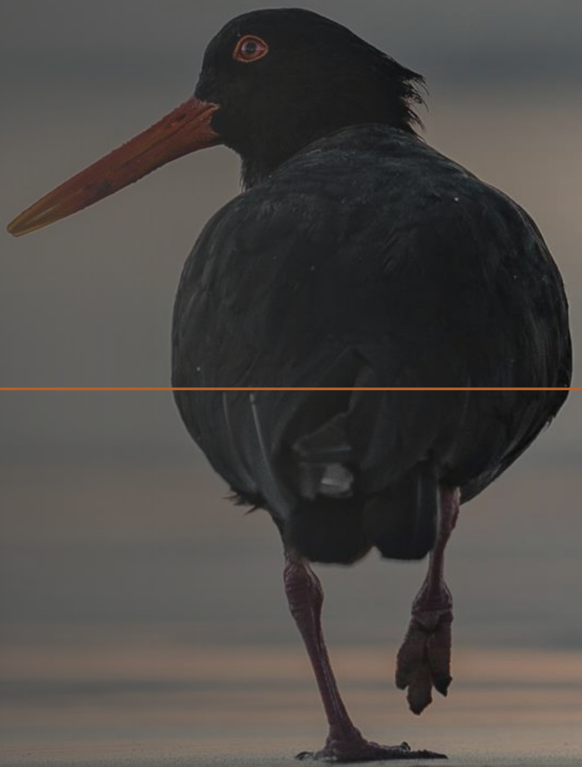
INTRODUCTION TO JAVA

Java 1.0

MULTITHREADING

Lesson # 16



VAR KEYWORD

JAVA VAR KEYWORD

- Java 10 introduced a new way to declare local variables using the var keyword
- Java automatically infers the type of a variable using the initial value assigned to a variable declared with the var keyword

JAVA VAR EXAMPLE

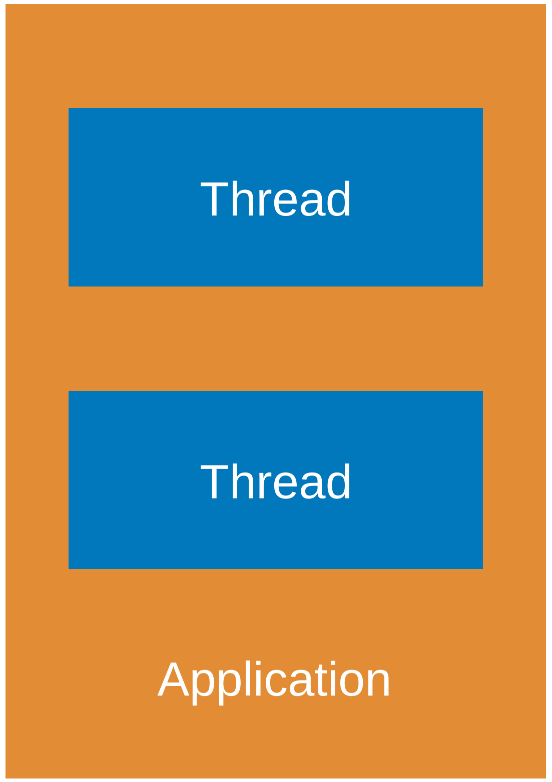
```java
var name = "John Doe";

var numbers = new ArrayList<Integer>();

var amount = 35.0;

var person = new Person();
```

MULITHREADING

WHAT IS MULTITHREADING?

- Multithreading means having multiple threads of execution inside the same application
- A thread is like a separate CPU executing your application
- Thus, a multithreaded application is like an application that has multiple CPUs executing different parts of the code at the same time

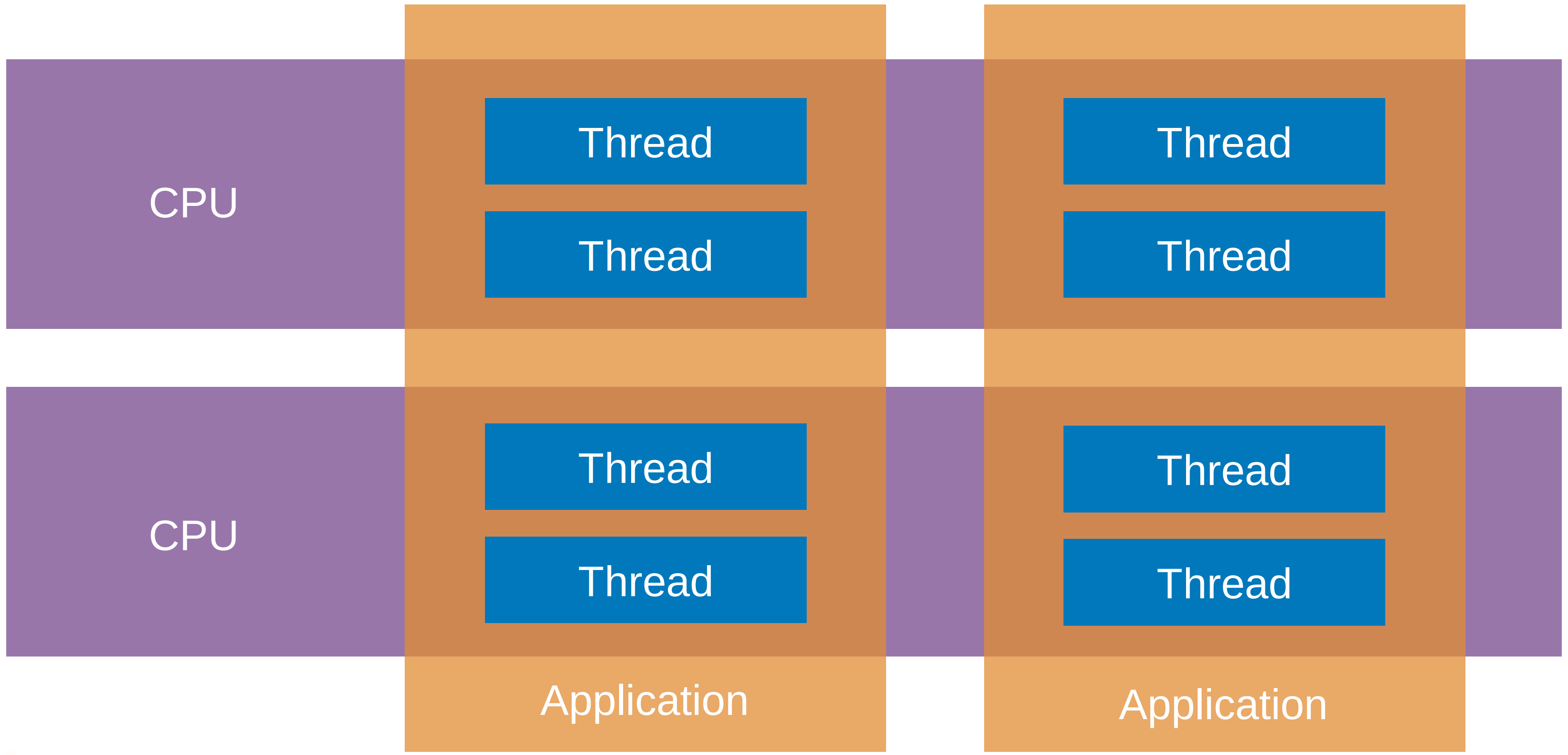
MULTITHREADED APPLICATION



THREAD VS CPU

- A thread is not equal to a CPU
- A single CPU will share its execution time among multiple threads, switching between executing each of the threads for a given amount of time
- It is also possible to have the threads of an application executed by different CPUs.

MULTITHREAD APPLICATION EXECUTION



BENEFITS OF MULTITHREADING

- The most common reasons for multithreading are:
  - Better utilization of a single CPU
  - Better utilization of multiple CPUs or CPU cores
  - Better user experience with regard to responsiveness
  - Better user experience with regard to fairness

MULTITHREADING IN JAVA

- Multithreading is a feature in Java that allows concurrent execution of two or more parts of a program for maximum utilization of CPU
- Each part of such a program is called a thread
- Java was one of the first languages to make multithreading easily available to developers

JAVA THREADS

MULTITHREADING IN JAVA

- Threads can be created by using two mechanisms:
  - Extending the Thread class
  - Implementing the Runnable Interface
- Java support two concurrency models:
  - Shared state
  - Separate state

EXTENDING THREAD CLASS

- Create a class that extends the java.lang.Thread class
- Class should override the run() method of the Thread class
- A thread begins its life inside run() method
- A thread is started by calling start() method

THREAD EXTENSION EXAMPLE

```
public class MyThread extends Thread {

    @Override
    public void run() {
        System.out.println("Tread " +
                Thread.currentThread().getId() + " is running");
    }

}
```

THREAD EXTENSION DEMO

```java
public class ThreadDemo {

    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            Thread thread = new MyThread();
            thread.start();
        }
    }

}
```

```
Tread 31 is running
Tread 24 is running
Tread 28 is running
Tread 30 is running
Tread 25 is running
Tread 33 is running
Tread 27 is running
Tread 29 is running
Tread 26 is running
Tread 32 is running
```

IMPLEMENTING RUNNABLE CLASS

- Create a new class that implements java.lang.Runnable interface
- Implement run() method
- Instantiate a Thread object by passing runnable implementation as an argument
- Call the start method on the Thread object.

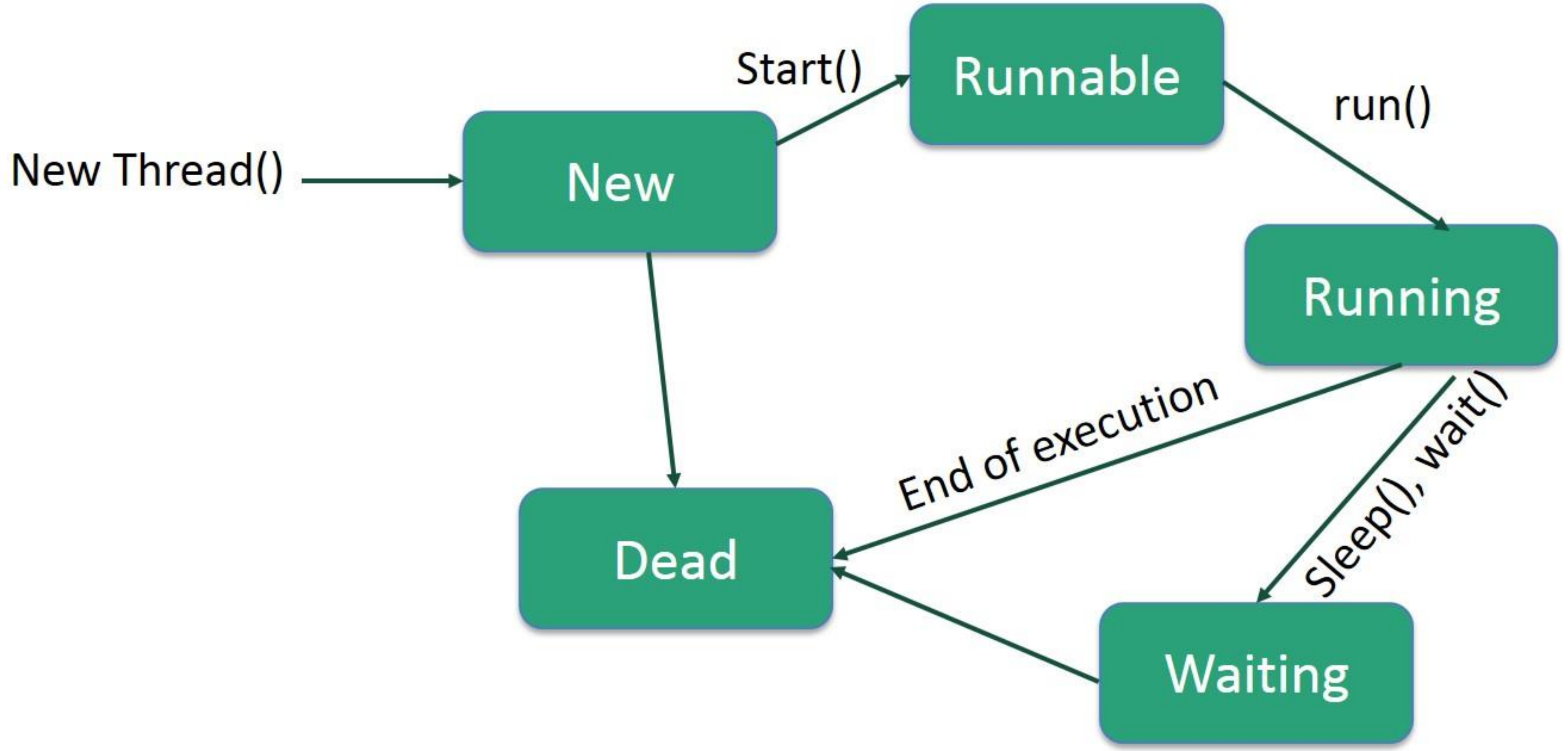
RUNNABLE IMPLEMENTATION EXAMPLE

```
public class MyRunnable implements Runnable {

    @Override
    public void run() {
        System.out.println("Tread " +
                Thread.currentThread().getId() + " is running");
    }

}
```

RUNNABLE IMPLEMENTATION EXAMPLE

```java
public class MyRunnable implements Runnable {

    @Override
    public void run() {
        System.out.println("Tread " +
                Thread.currentThread().getId() + " is running");
    }

}
```

THREAD LIFECYCLE DIAGRAM



THREAD LIFECYCLE STAGES

- New:
  - A new thread begins its life cycle in the new state
  - It remains in this state until the program starts the thread
  - It is also referred to as a born thread
- Runnable:
  - After a newly born thread is started, the thread becomes runnable
  - A thread in this state is considered to be executing its task.

THREAD LIFECYCLE STAGES

- Waiting:
    - A thread transitions to the waiting state while the thread waits for another thread to perform a task
    - A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing

THREAD LIFECYCLE STAGES

- Timed Waiting
  - A runnable thread can enter the timed waiting state for a specified time interval
  - A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs
- Terminated (Dead)
  - A runnable thread enters the terminated state when it completes its task or otherwise terminates.

THREAD CLASS CORE METHODS

Method	Purpose
public void start()	Starts the thread in a separate path of execution, then invokes the run() method on this Thread object
public void run()	If this Thread object was instantiated using a separate Runnable target, the run() method is invoked on that Runnable object
public final void setName(String name)	Changes the name of the Thread object. There is also a getName() method for retrieving the name
public final void setPriority(int priority)	Sets the priority of this Thread object. The possible values are between 1 and 10.
public final void join()	The current thread invokes this method on a second thread, causing the current thread to block until the second thread terminates
public static void sleep(long millisec)	This causes the currently running thread to block for at least the specified number of milliseconds

THREAD SLEEP EXAMPLE

```
Thread helloThread = new Thread(() -> {
    try {
        //Pauses thread for 100 milliseconds
        Thread.sleep(100);
        System.out.println("Hello!");
    } catch (InterruptedException e) {
        throw new RuntimeException(e);
    }
});

Thread goodbyeThread = new Thread(() -> System.out.println("Goodbye!"));

helloThread.start();
goodbyeThread.start();
```

```
Goodbye!
Hello!
```

THREAD JOIN EXAMPLE

```
Thread helloThread = new Thread(() -> System.out.println("Hello!"));
Thread goodbyeThread = new Thread(() -> System.out.println("Goodbye!"));

//Hello thread starts
helloThread.start();
//Blocks execution until Hello thread is completed
helloThread.join();

//Goodbye thread starts
goodbyeThread.start();
//Blocks execution until Goodbye thread is completed
goodbyeThread.join();
```

SYNCHRONIZATION

THREAD SYNCHRONIZATION

- A Java synchronized block marks a method or code block as synchronized
- A synchronized block in Java can only be executed in a single thread at a time
- Java synchronized blocks can be used to avoid race conditions

SYNCHRONIZED KEYWORD

- Synchronized blocks in Java are marked with the synchronized keyword
- The synchronized keyword can be used to mark four different types of blocks:
    - Instance methods
    - Static methods
    - Code blocks inside instance methods
    - Code blocks inside static methods

SYNCHRONIZED METHOD SYNTAX

```
modifier synchronized type methodName(type arg1, type agr2, ....) {
    ...
}
```

Synchronized keyword

```
modifier static synchronized type methodName(type arg1, type agr2, ....) {
    ...
}
```

SYNCHRONIZED METHOD EXAMPLE

```
public class Counter {

    private int count = 0;

    public synchronized void add(int value){
        this.count += value;
    }
    public synchronized void subtract(int value){
        this.count -= value;
    }

}
```

SYNCHRONIZED CODE BLOCK SYNTAX

```
synchronized (objectInstance) {
    ...
}
```

SYNCHRONIZED CODE BLOCK SYNTAX

```
synchronized (objectInstance) {
    ...
}
```

SYNCHRONIZED CODE BLOCK EXAMPLE

```java
public class Counter {

    private int count = 0;

    public void add(int value) {
        synchronized (this) {
            this.count += value;
        }
    }

    public synchronized void subtract(int value) {
        synchronized (this) {
           this.count -= value;
        }
    }

}
```

{JG} JavaGuru

REFERENCES

REFERENCES

- https://www.educative.io/answers/what-is-the-var-keyword-in-java
- https://www.tutorialspoint.com/java/java_multithreading.htm
- https://www.javatpoint.com/multithreading-in-java
- https://www.guru99.com/multithreading-java.html
- https://www.w3schools.in/java/multithreading

QUESTIONS?

THANK YOU!