Java Threads

Nimesha Hewawasam
Lecturer
nimesha.h@nsbm.ac.lk

What is a Java Thread?

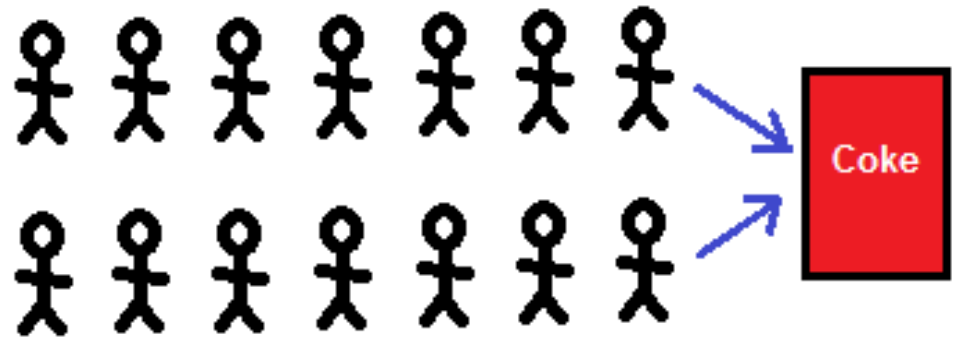
- A **Java thread** is the smallest unit of execution within a program.
- It is often called a **lightweight subprocess** because it runs independently but still shares the same memory space of its parent process.
- By using threads, Java enables **concurrent execution** of multiple tasks, improving efficiency and responsiveness.
- Ex: In **MS Word**:
  - One thread might be **formatting the document** (spell-check, auto-save, background formatting).
  - Another thread simultaneously handles **user input** (keyboard typing, mouse clicks).
  - This way, multiple activities run smoothly without the program freezing.

Multitasking and Multithreading

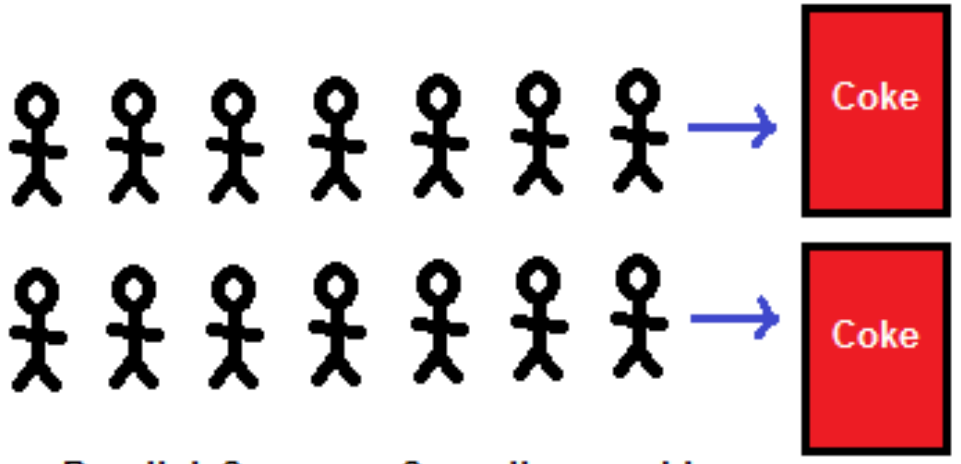
- Multitasking:
  - refers to a computer's ability to perform multiple jobs concurrently
  - more than one program are running concurrently,
    - e.g., UNIX, Linux

- Multithreading:
  - A thread is a single sequence of execution within a program
  - refers to multiple threads of control within a single program
  - each program can run multiple threads of control within it,
    - e.g., Web Browser

Concurrency vs. Parallelism

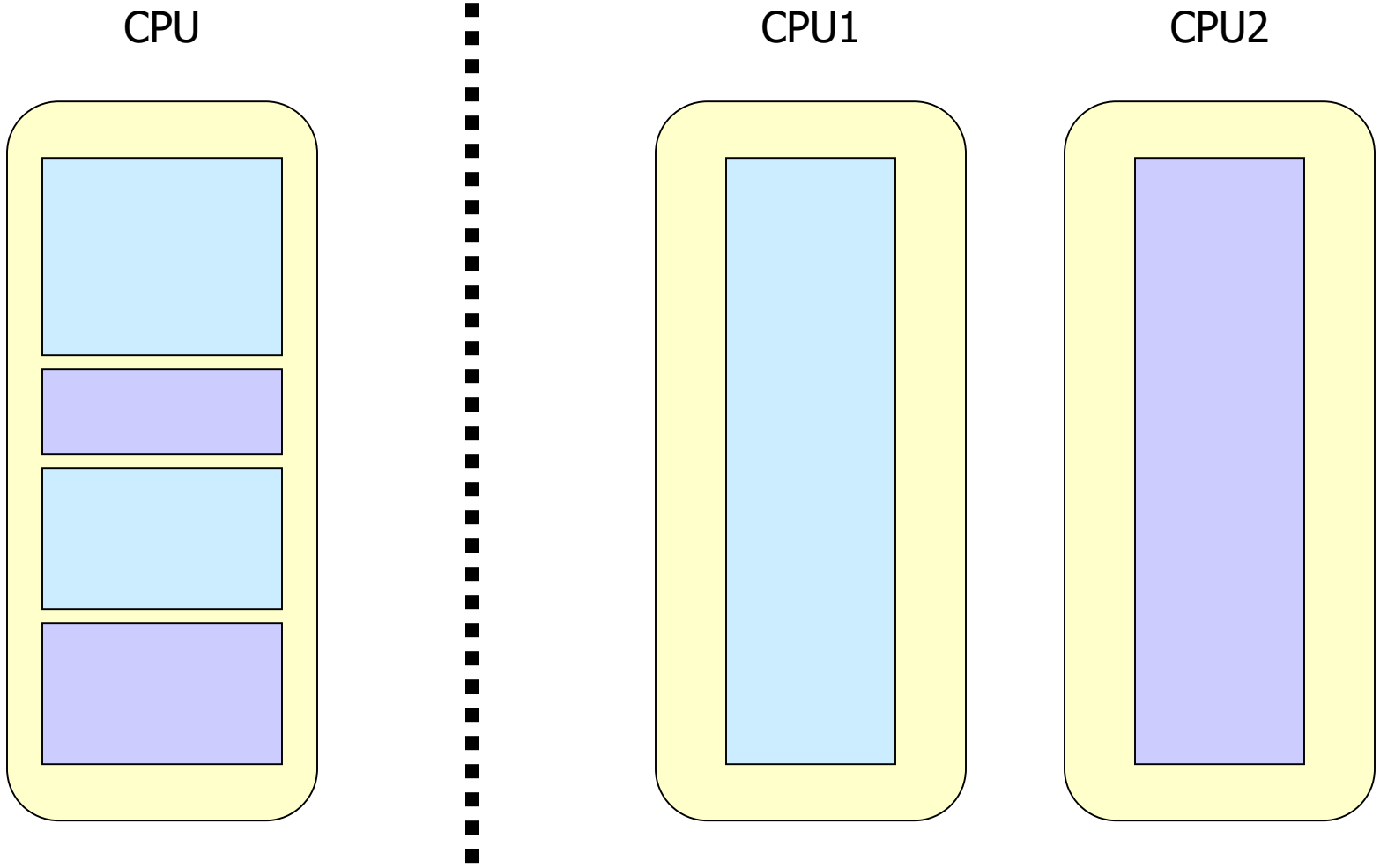


Concurrent: 2 queues, 1 vending machine

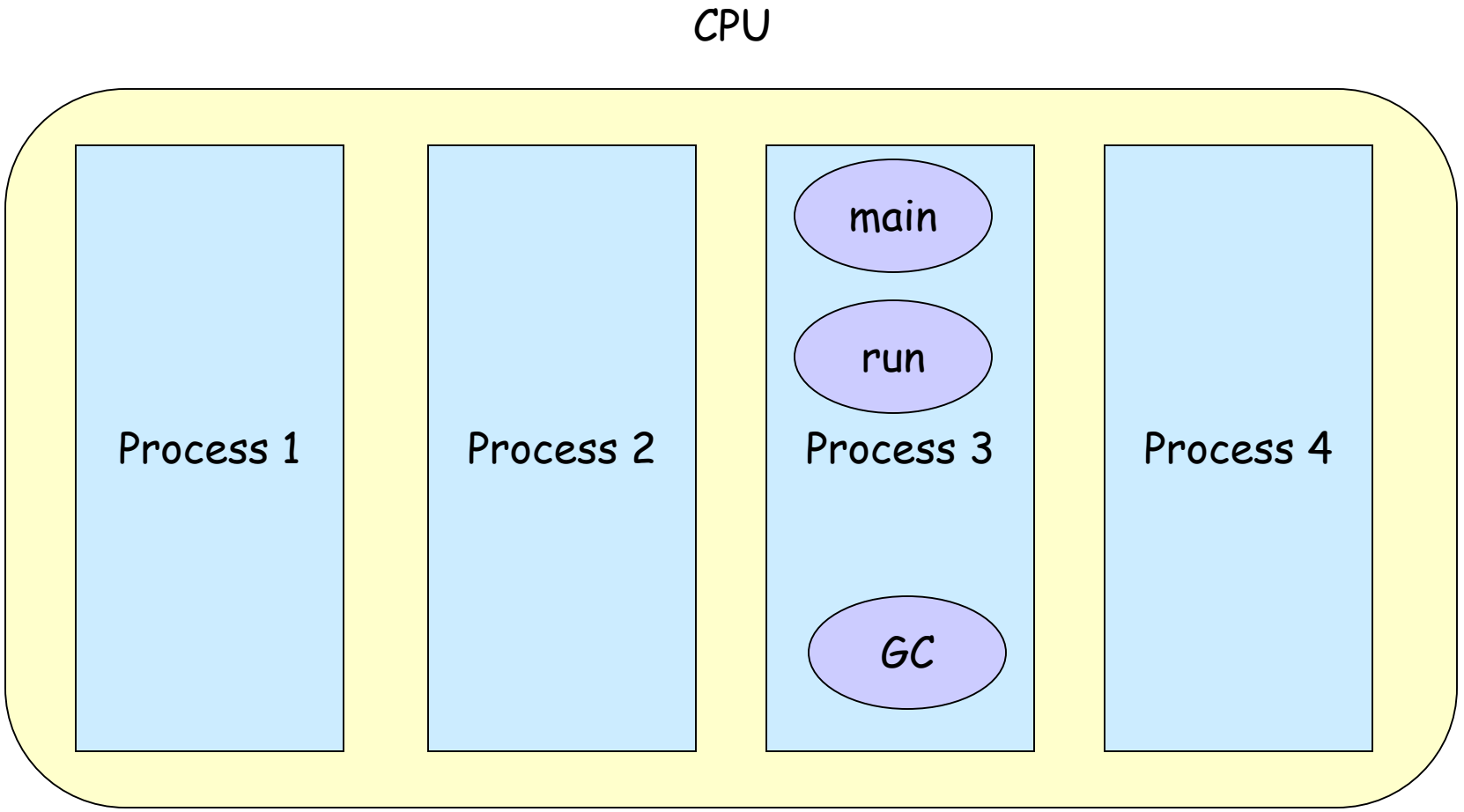


Parallel: 2 queues, 2 vending machines

Concurrency vs. Parallelism



Threads and Processes



What are Threads Good For?

- To maintain **responsiveness** of an application during **a long running task**
- To **enable cancellation** of separable tasks
- To **monitor** status of some resource
- Some **APIs and systems demand it**

Application Thread

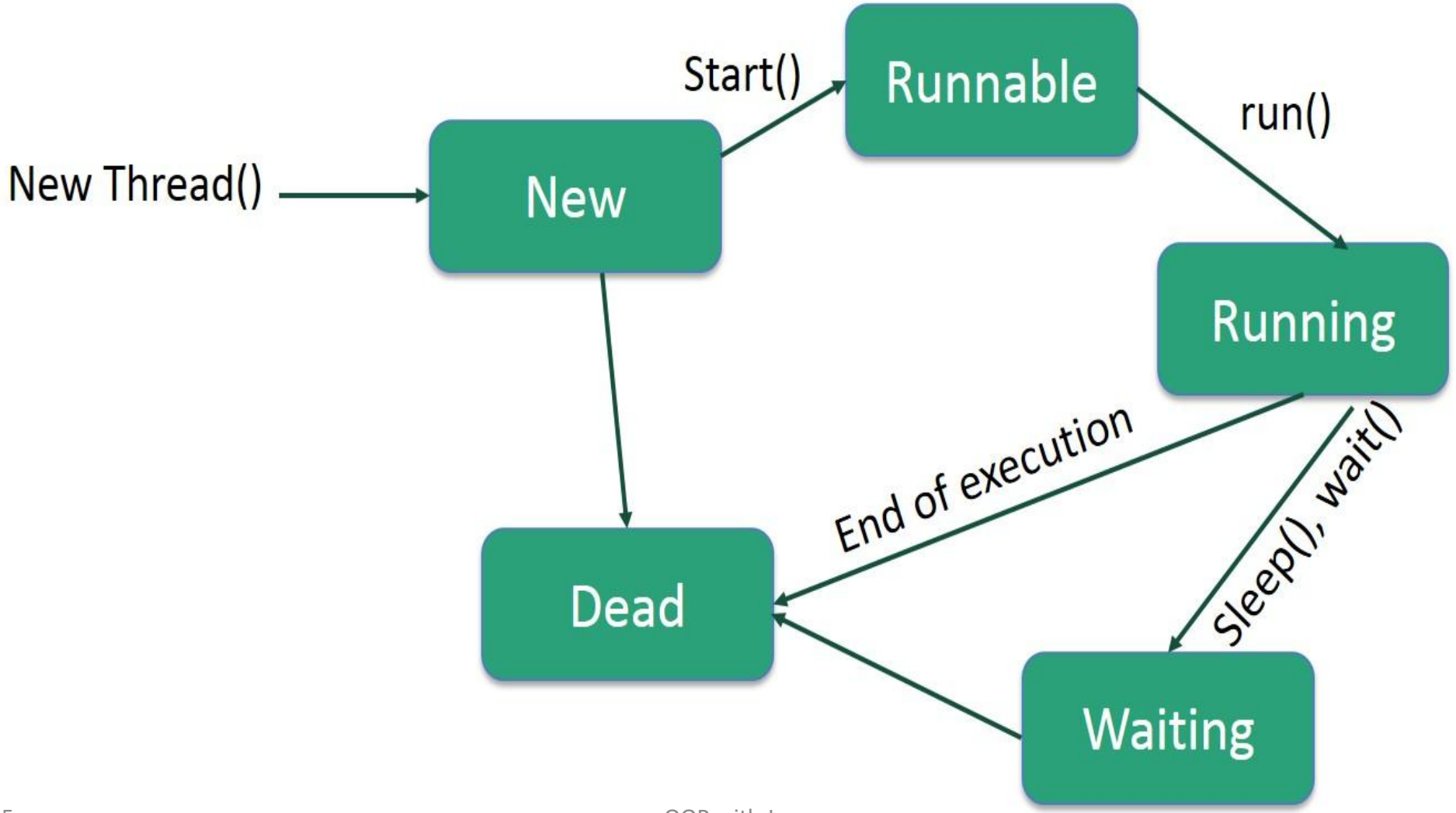
When we execute an application:

1. The JVM creates a Thread object whose task is defined by the **`main()`** method
2. The JVM starts the thread
3. The thread executes the statements of the program one by one
4. After executing all the statements, the method returns and the thread dies

Multiple Threads in an Application

- Each thread has its **private run-time stack**
- **If two threads execute the same method, each will have its own copy of the local variables the methods uses**
- **Two different threads can act on the same object and same static fields concurrently**

Life Cycle of a Thread



Creating Threads

- There are two ways to create our own **Thread** object
  1. Subclassing the **Thread** class and instantiating a new object of that class
  2. Implementing the **Runnable** interface

- In both cases the **run()** method should be implemented

Extending Thread

```
public class ThreadExample extends Thread {
    public void run () {
        for (int i = 1; i <= 100; i++) {
            System.out.println("---");
        }
    }
}
```

Thread Methods

void start()

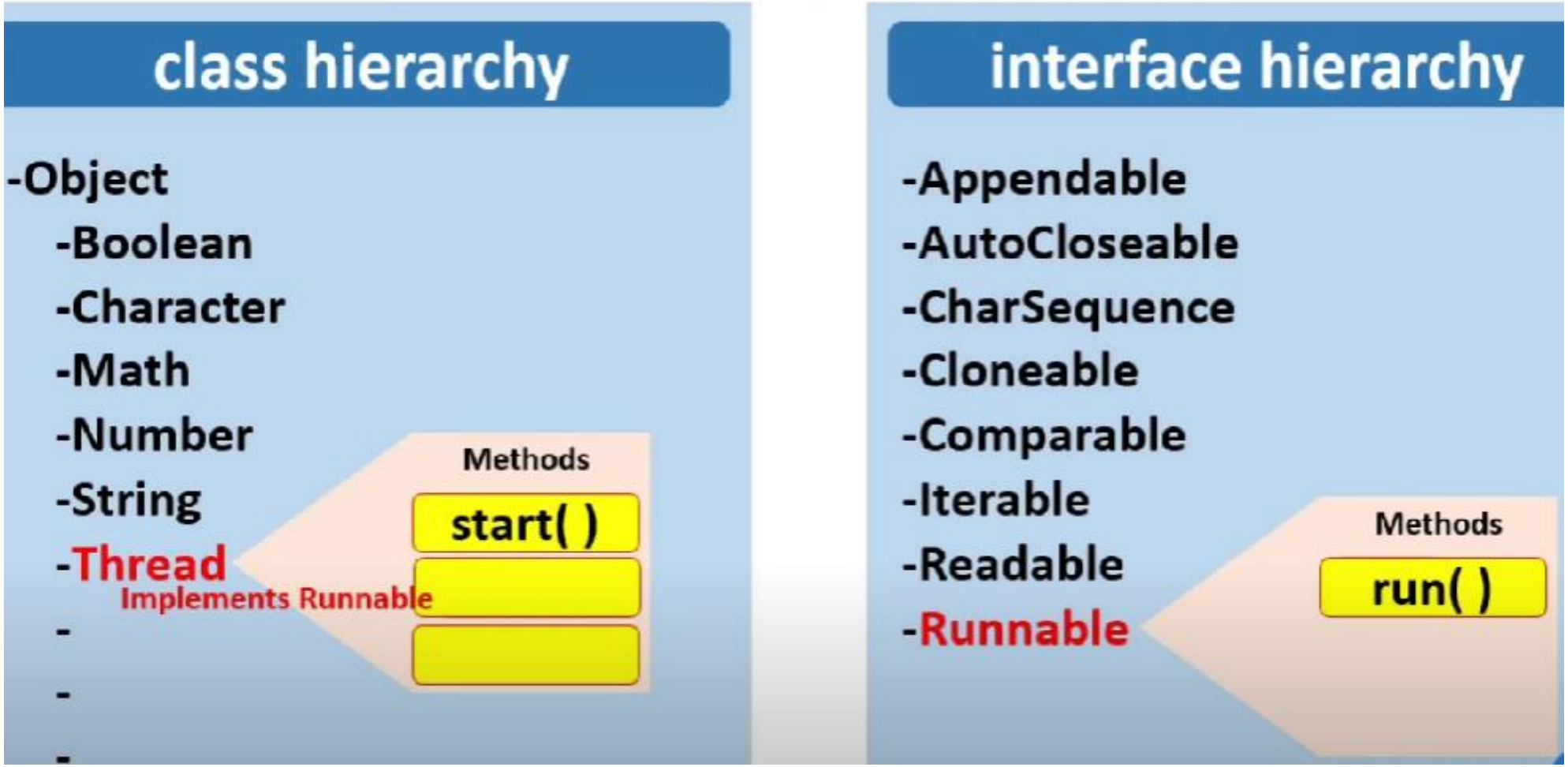
- Creates a new thread and makes it runnable
- This method can be called only once

void run()

- The new thread begins its life inside this method

void stop() (deprecated)

- The thread is being terminated


class hierarchy
-Object
-Boolean
-Character
-Math
-Number
-String
-Thread
Implements Runnable
-
-
-
Methods
start( )
interface hierarchy
-Appendable
-AutoCloseable
-CharSequence
-Cloneable
-Comparable
-Iterable
-Readable
-Runnable
Methods
run( )

Thread Methods

void yield()

- Causes the currently executing thread object to temporarily pause and allow other threads to execute
- Allow only threads of the same priority to run

void sleep(int *m*) or sleep(int *m*, int *n*)

- The thread sleeps for *m* milliseconds, plus *n* nanoseconds

Implementing Runnable

```
public class RunnableExample implements Runnable {
  public void run () {
       for (int i = 1; i <= 100; i++) {
                  System.out.println (“***”);
       }
    }
}
```

A Runnable Object

- When running the Runnable object, a Thread object is created from the Runnable object
- The Thread object′s **run()** method calls the Runnable object′s **run()** method
- Allows threads to run inside any object, regardless of inheritance

Starting the Threads

```
public class ThreadsStartExample {
        public static void main (String argv[]) {
          new ThreadExample ().start ();
          new Thread(new RunnableExample ()).start ();
        }
}
```

What will we see when running ThreadsStartExample?

Thank You