

Multithreading



Liang, Introduction to Java Programming and Data Structures,
Twelfth Edition, (c) 2020 Pearson Education, Inc. All rights reserved.

By: Mamoun Nawahdah (Ph.D.)

2022/2023

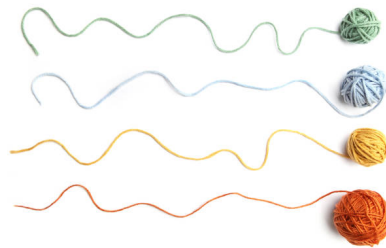
Objectives

- ❖ To get an overview of multithreading
- ❖ To develop task classes by implementing the **Runnable** interface
- ❖ To create threads to run tasks using the **Thread** class
- ❖ To control threads using the methods in the **Thread** class
- ❖ To control animations using threads and use **Platform.runLater** to run the code in application thread
- ❖ To execute tasks in a thread pool



Thread Concepts

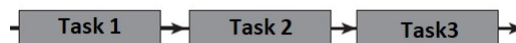
- ❖ A program may consist of many **tasks** that can run concurrently.
- ❖ A **thread** is the flow of execution, from beginning to end, of a task



Thread: a long, thin strand of cotton, nylon, or other fibres used in sewing or weaving.

Threads Concept

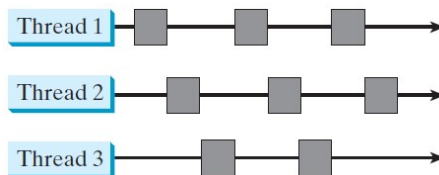
Multiple tasks
sharing a single
CPU –No threading



Multiple
threads on
multiple CPUs



Multiple threads
sharing a single
CPU



Multithreading

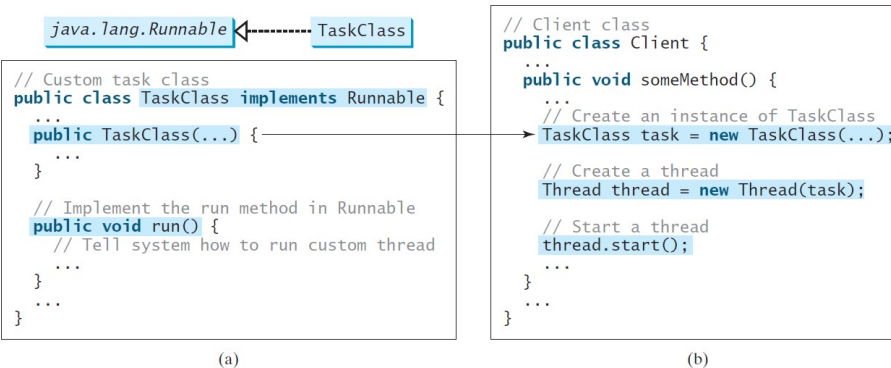
- ❖ In single-processor systems, the multiple threads share CPU time, known as **time sharing**.
- ❖ The operating system is responsible for scheduling and allocating resources to them.
- ❖ This arrangement is practical because most of the time the CPU is **idle**.
- ❖ Multithreading can make your program more **responsive** and **interactive** as well as enhance performance.



Creating Tasks and Threads

You can create additional threads to run concurrent tasks in the program. In Java, each task is an instance of the **Runnable** interface, also called a *runnable object*.

A *thread* is essentially an object that facilitates the execution of a task.



Example:

- ❖ Create and run three threads:
 - The 1st thread prints the **A** 15 times.
 - The 2nd thread prints the **B** 15 times.
 - The 3rd thread prints the **C** 15 times.



7

```
class PrintChar implements Runnable {  
    private char charToPrint;  
    private int times;  
  
    public PrintChar(char c, int t) {  
        charToPrint = c;  
        times = t;  
    }  
  
    @Override  
    public void run() {  
        for (int i = 0; i < times; i++)  
            System.out.print(charToPrint);  
    }  
}
```



```

public class TestThreads {
    public static void main(String[] args) {
        Runnable printA = new PrintChar('A', 15);
        Runnable printB = new PrintChar('B', 15);
        Runnable printC = new PrintChar('C', 15);

        // Create threads
        Thread thread1 = new Thread(printA);
        Thread thread2 = new Thread(printB);
        Thread thread3 = new Thread(printC);

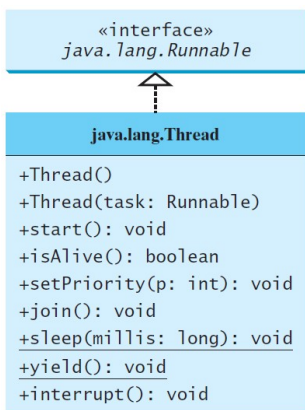
        // Start threads
        thread1.start();
        thread2.start();
        thread3.start();
    }
}

```



AAAAAAAAACCCCCCCCCBBBBBBBBBBBBBBBCCCCCAAAAAA

The Thread Class



Creates an empty thread.

Creates a thread for a specified task.

Starts the thread that causes the run() method to be invoked by the JVM.

Tests whether the thread is currently running.

Sets priority p (ranging from 1 to 10) for this thread.

Waits for this thread to finish.

Puts a thread to sleep for a specified time in milliseconds.

Causes a thread to pause temporarily and allow other threads to execute.

Interrupts this thread.



Thread Priority

- ❖ Each thread is assigned a default priority of `Thread.NORM_PRIORITY`.
- ❖ You can reset the priority using `setPriority(int priority)`
- ❖ Some constants for priorities include
`Thread.MIN_PRIORITY`
`Thread.MAX_PRIORITY`
`Thread.NORM_PRIORITY`

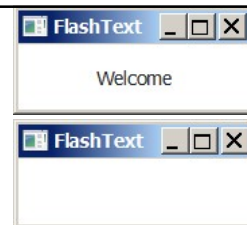


11

Animation Using Threads

Example: Flashing Text

```
new Thread(new Runnable() {
    public void run() {
        try {
            while (true) {
                if (lblText.getText().trim().length() == 0)
                    text = "Welcome";
                else
                    text = "";
                Platform.runLater(new Runnable() { // Run from JavaFX GUI
                    public void run() {
                        lblText.setText(text);
                    }
                });
                Thread.sleep(200);
            }
        } catch (InterruptedException ex) { }
    }
}).start();
```



12

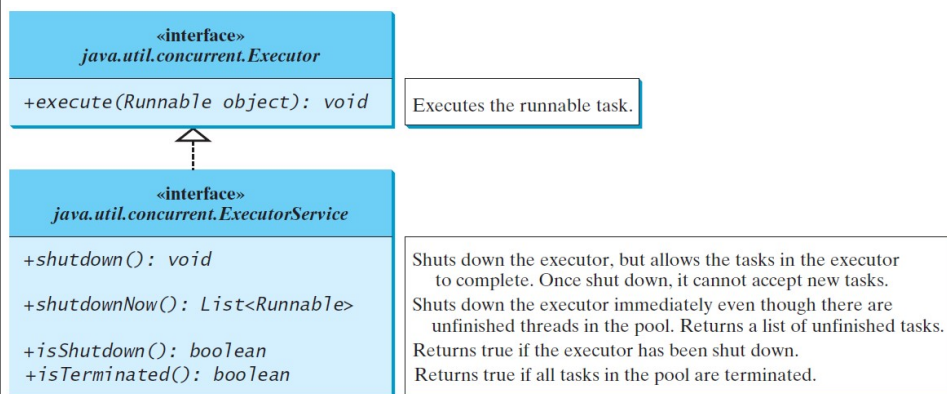
Thread Pools

- ❖ Starting a new thread for each task could limit throughput and cause poor performance.
- ❖ A thread pool is ideal to manage the number of tasks executing concurrently.
- ❖ JDK 1.5 uses the **Executor** interface for executing tasks in a thread pool and the **ExecutorService** interface for managing and controlling tasks.
- ❖ **ExecutorService** is a subinterface of **Executor**.



13

Thread Pools



14

Creating Executors

To create an **Executor** object, use the static methods in the **Executors** class.

`java.util.concurrent.Executors`

`+newFixedThreadPool(numberOfThreads:
int): ExecutorService`

`+newCachedThreadPool():
ExecutorService`

Creates a thread pool with a fixed number of threads executing concurrently. A thread may be reused to execute another task after its current task is finished.

Creates a thread pool that creates new threads as needed, but will reuse previously constructed threads when they are available.

```
public static void main(String[] args) {
    // Create a fixed thread pool with maximum three threads
    ExecutorService executor = Executors.newFixedThreadPool(3);

    // Submit runnable tasks to the executor
    executor.execute(new PrintChar('a', 100));
    executor.execute(new PrintChar('b', 100));
    executor.execute(new PrintNum(100));

    // Shut down the executor
    executor.shutdown();
}
```



15