Sabancı Üniversitesi

Object Oriented Programming with Java

12 - Threads and Multithreading

Sabancı Üniversitesi

Multitasking

- is divided into two types
  1. Process based: Two or more programs runs concurrently. You can run Spotify and Word at the same time. (Provided by operating systems)
  2. Thread based: A single program can perform two or more tasks simultaneously. For example, text editor can print while formatting is being done. (Called multithreading)

Sabancı Üniversitesi

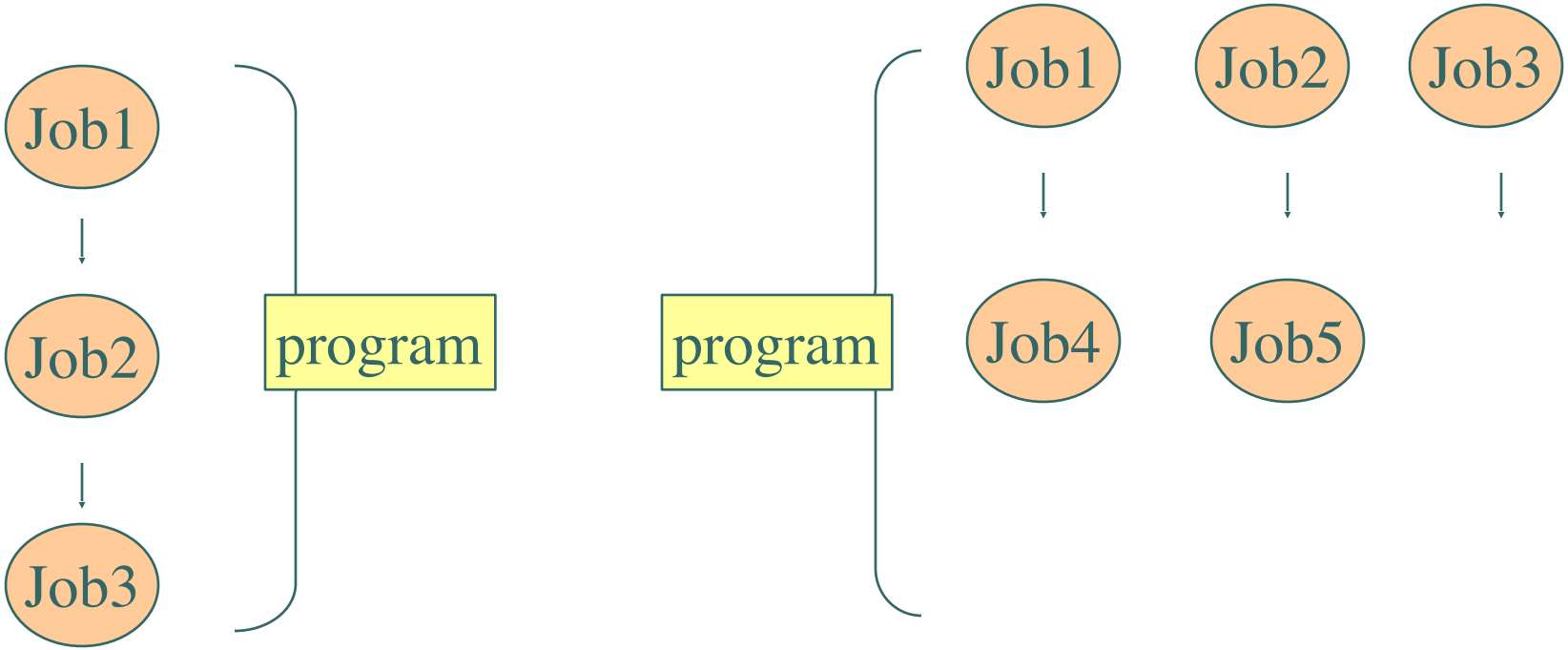
Main Thread

- When a java program starts up, one thread begins running immediately. This is usually called the **main thread** of your program, because it is the one that is executed when your program begins.
- Main thread will be the last thread to finish execution.
- When the main thread stops your program terminates.

Sabancı Üniversitesi

Multithreading

- Multithreading programs *appear* to do more than one thing at a time



Sabancı Üniversitesi

Threads

- A Thread is a single flow of control
  - When you step through a program, you are following a Thread
- Your previous programs all had one Thread (Main Thread)
- A Thread is an Object you can create and control

Sabancı Üniversitesi

Real Life Examples

- *Web servers* starts a new thread whenever a new client makes a request
- Microsoft Outlook can check your inbox for new mail while you are writing another one
- There are bot characters in games acting concurrently, all of them act in a different thread

Sabancı Üniversitesi

Sleeping

- `Thread.sleep(int milliseconds);`
  - A millisecond is 1/1000 of a second
  - sleep() is a static method, can be called from anywhere
- sleep only works for the current Thread

Sabancı Üniversitesi

Two ways of creating Threads

- You can extend the `Thread` class:
  - `class MyThread extends Thread {…}`
  - Limiting, since you can only extend one class
- Or you can implement the `Runnable` interface:
  - `class MyThread implements Runnable {…}`
  - requires public void run( ) overridden

Sabancı Üniversitesi

Extending Thread Class

Write a class that extends `Thread` class:

```
public class MyThread extends Thread{}
```

Override `run` method:

```
public void run(){
        //put your code that would run continously in a loop
}
```

Create an instance of the class and invoke `start()` method:

```
MyThread mThread = new MyThread();
mThread.start();
```

`start()` *method invokes* `run()` *implicitly; if you invoke run() instead, a seperate thread of execution would not be created but the code will compile.*

Sabancı Üniversitesi

Implementing Runnable

Write a class that implements Runnable interface:

```
public class MyThread implements Runnable{}
```

You have to implement run() method in your class:

```
public void run(){
        //Write your continous code here
}
```

Create a Thread instance with an instance of your class:

```
MyThread mThread = new MyThread();
Thread threadToRun = new Thread(mThread);
threadToRun.start();
```

Sabancı Üniversitesi

Thread Methods - `setDaemon()`

- Daemon threads:
  - Two kinds of threads: application and daemon threads
  - JVM will stop an application when all non-daemon threads have finished execution not waiting for the daemon threads to finish

```
MyThread t = new MyThread();
t.setDaemon(true);
t.start();
```

call setDaemon() before starting the thread

Sabancı Üniversitesi

Thread Synchronization

- Two threads may not safely access the same data at the same time
- Two mechanisms are provided to support single threaded access to data:
  - `synchronized` code blocks
  - `synchronized` methods
- Any class that may be accessed by more than one thread must guarantee synchronized access to data

Sabancı Üniversitesi

Synchronized Methods

- Guarantees only one thread can invoke methods on the object instance at a time
- Blocks additional method invocation until the first thread returns

The Object to be reached:

```
class Student {
private String name;
private String lastName;
public Student(String lastName, String name) {
              super();
              this.lastName = lastName;
              this.name = name;
}

public synchronized void showStudentInfo(String threadName)
{
for(int i=0;i<50;i++){
try
{

              {
              Thread.sleep(100);
   System.out.println(threadName+" --- "+this.name +" - "+this.lastName);
}}
catch (Exception e) {
e.printStackTrace();
}

}
```

Thread trying to reach object

```
public class MyThread extends Thread {
private Student stu;
private String threadName;
MyThread(Student stu, String  threadName)
{
this.stu = stu;
this.threadName = threadName;
}
public void run() {
this.stu.showStudentInfo(threadName);
}
}
```

When more than one thread is instanciated and run with the same instance of student, only one will be reaching the method at a time

Sabancı Üniversitesi

Synchronized Blocks

- Allows synchronized access to certain objects

```
public class ThreadSafe extends Thread{

        private String myData;

        public void doWork(){
                synchronized(this){
                myData = …;
                }
        }
}
```

only one thread at a time will reach this instance

- Synchronized methods locks the entire class, whereas synch. blocks lock a specific object

Sabancı Üniversitesi

Thread Methods - `isAlive()` and `isInterrupted()`

- `isAlive()` : Tests if this thread is alive. A thread is alive if it has been started and has not yet died.

- `isInterrupted()`: Tests whether this thread has been interrupted. The *interrupted status* of the thread is unaffected by this method. A thread interruption ignored because a thread was not alive at the time of the interrupt will be reflected by this method returning false.

Sabancı Üniversitesi

Pausing and Suspending Threads

- The proper way to temporarily pause the execution of a thread is to set a variable that the target thread checks occasionally.

- When the target thread detects that the variable is set, it calls `Object.wait()`. The paused thread can then be woken up by calling its `Object.notify()` method.

- Note: `Thread.suspend()` and `Thread.resume()` provide methods for pausing a thread. However, these methods have been **deprecated** because they are very unsafe. Using them often results in deadlocks

Sabancı Üniversitesi

Pausing and Suspending Threads Example

Controlling the thread code:

```
// Create and start the thread
MyThread thread = new MyThread();
thread.start();
while (true) {
// Do work
// Pause the thread
synchronized (thread) {
thread.pleaseWait = true;
}
// Do work
// Resume the thread
synchronized (thread) {
thread.pleaseWait = false;
thread.notify();
}
// Do work }
```

restarts the thread

A thread with a pleaseWait boolean flag

```
class MyThread extends Thread {
boolean pleaseWait = false;
// This method is called when the
//thread runs public void run() {
        while (true) {
        // Do work
        // Check if should wait
        synchronized (this) {
        while (pleaseWait) {
        try {
                wait();
        } catch (Exception e) { }
        }
} // Do work } } }
```

pauses the thread

Sabancı Üniversitesi

Stopping Threads

- The same methodology with pausing threads should be used
- `stop()` method is deprecated

```
class MyThread extends Thread{
        boolean threadCanRun = true;
        public void run(){
        while(threadCanRun)
                //do work
        }
        public void stopThread(){
                this.threadCanRun = false
        }
}
```

Sabancı Üniversitesi

The `java.util.concurrent` Package

- Java 5 introduced the `java.util.concurrent` package, which contains classes that are useful in concurrent programming. Features include:
  - Concurrent collections
  - Synchronization and locking alternatives
  - Thread pools
    - Fixed and dynamic thread count pools available
    - Parallel divide and conquer (Fork-Join) new in Java 7

Sabancı Üniversitesi

Recommended Threading Classes

- Traditional `Thread` related APIs are difficult to code properly. Recommended concurrency classes include:
  - `java.util.concurrent.ExecutorService`, a higher level mechanism used to execute tasks
    - It may create and reuse `Thread` objects for you.
    - It allows you to submit work and check on the results in the future.
  - The Fork-Join framework, a specialized work-stealing `ExecutorService` new in Java 7

Sabancı Üniversitesi

ExecutorService

- An `ExecutorService` is used to execute tasks.
  - It eliminates the need to manually create and manage threads.
  - Tasks **might** be executed in parallel depending on the `ExecutorService` implementation.
  - Tasks can be:
    - `java.lang.Runnable`
    - `java.util.concurrent.Callable`
  - Implementing instances can be obtained with `Executors`.

```
ExecutorService es = Executors.newCachedThreadPool();
```

```
ExecutorService es = Executors.newFixedThreadPool([number of Threads]);
```

Sabancı Üniversitesi

Example `ExecutorService`

- This example illustrates using an `ExecutorService` to execute `Runnable` tasks:

```
package com.example;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class ExecutorExample {
    public static void main(String[] args) {
        ExecutorService es = Executors.newCachedThreadPool();
        es.execute(new ExampleRunnable("one"));
        es.execute(new ExampleRunnable("two"));
        es.shutdown();
    }
}
```

Execute this Runnable task sometime in the future

Shut down the executor

Sabancı Üniversitesi

Shutting Down an `ExecutorService`

- Shutting down an `ExecutorService` is important because its threads are non-daemon threads and will keep your JVM from shutting down.

```
es.shutdown();
```

Stop accepting new `Callables`, but threads continue running

```
try {
        es.awaitTermination(4000, TimeUnit.SECONDS);
} catch (InterruptedException e) {
        e.printStackTrace();
}
es.shutdownNow();
```

If you want to wait for the `Callable`s to finish

shutdownNow() tries to interrupt running threads, in order to stop them check interrupted status

```
if(Thread.currentThread().isInterrupted()) break;
```

Sabancı Üniversitesi

java.util.concurrent.Callable

- The `Callable` interface:
  - Defines a task submitted to an `ExecutorService`
  - Is similar in nature to `Runnable`, but can:
    - Return a result using generics
    - Throw a checked exception

```
package java.util.concurrent;
public interface Callable<V> {
    V call() throws Exception;
}
```

Sabancı Üniversitesi

Example `Callable` Task

```
public class ExampleCallable implements Callable {

  private final String name;
  private final int len;
  private int sum = 0;

  public ExampleCallable(String name, int len) {
    this.name = name;
    this.len = len;
  }

  @Override
  public String call() throws Exception {
    for (int i = 0; i < len; i++) {
      System.out.println(name + ":" + i);
      sum += i;
    }
    return "sum: " + sum;
  }
}
```

Return a String from this task: the sum of the series

Sabancı Üniversitesi

Future

- The `Future` interface is used to obtain the results from a `Callable`'s `V call()` method.

```
Future<V> future = es.submit(callable);
//submit many callables
try {
    V result = future.get();
} catch (ExecutionException|InterruptedException ex) {

}
```

`ExecutorService` controls when the work is done.

Gets the result of the `Callable`'s `call` method (blocks if needed).

If the `Callable` threw an `Exception`

Sabancı Üniversitesi

Example

```
public static void main(String[] args) {

  ExecutorService es = Executors.newFixedThreadPool(4);
  Future<String> f1 = es.submit(new
    ExampleCallable("one",10));
  Future<String> f2 = es.submit(new
    ExampleCallable("two",20));

  try {
    es.shutdown();
    es.awaitTermination(5, TimeUnit.SECONDS);
    String result1 = f1.get();
    System.out.println("Result of one: " + result1);
    String result2 = f2.get();
    System.out.println("Result of two: " + result2);
  } catch (ExecutionException | InterruptedException ex) {
    System.out.println("Exception: " + ex);
  }

}
```

Wait 5 seconds for the tasks to complete

Get the results of tasks f1 and f2

Sabancı Üniversitesi

Lab – 01 Alarm Clock

- Create a UI which displays a clock
- Users will be able to set an alarm
- When the time of alarm comes, display user a message

Sabancı Üniversitesi

Lab -2 Bouncing Ball

- Create a UI that displays a circle ball.
- The ball will bounce the boundries of the frame

Sabancı Üniversitesi

Lab- 03 Progress Bars

- Display progress bars on a JFrame
- Users will be able to start, stop, pause and resume the progress