Session 5 : Locks, Executors

sourcemind

Interrupts

In the following example, main thread spins a new thread.

How to terminate that thread?

```java
public static void main(String[] args) throws InterruptedException {
    Runnable job1 = () -> {
        int i = 1;
        while (true) {
            System.out.println("Doing something " + i);
            i++;
        }
    };
    Thread t1 = new Thread(job1);
    t1.start();
    Thread.sleep(3000);
    System.out.println("Rest of main thread...");
}
```

Thread t1 will never end and main thread won't have a way to stop it after 3 seconds

One way is to declare t1 as a daemon thread but that only works when the main thread also stops

Interrupting a running thread

We can interrupt a thread, but that thread also needs to handle the interrupt and treat it in some way

```java
public static void main(String[] args) throws InterruptedException {
    Runnable job1 = () -> {
        int i = 1;
        while (true) {
            if (Thread.interrupted()) {
                // End the loop when interrupted
                break;
            }
            System.out.println("Doing something " + i);
            i++;
        }
    };

    Thread t1 = new Thread(job1);
    t1.start();
    Thread.sleep(3000);
    t1.interrupt();     // Interrupt the thread t1 after 3 seconds
    System.out.println("Rest of main thread...");
}
```

Interrupting is not same as stopping a thread, since the interrupted thread's running code decides what to do with that interrupt signal

Interrupting a sleeping thread

Thread.sleep(long millis) causes the current thread to go to sleeping mode. It simulates a long-running task, however without consuming CPU time.

All monitors are retained by the current thread

While in sleeping mode, the thread can also be interrupted but since the thread is not in running mode, that will raise an InterruptedException and the thread will be running again

This is similar to the previous example but consuming less resources.

Interrupting a sleeping thread

```
public static void main(String[] args) throws InterruptedException {
    Runnable job1 = () -> {
        int i = 1;
        while (true) {
            try {
                System.out.println("Doing something " + i);
                i++;
                Thread.sleep(100);
            } catch (InterruptedException e) {
                System.out.println("Interrupted. Quitting.");
                break;
            }
        }
    };

    Thread t1 = new Thread(job1);
    t1.start();
    Thread.sleep(3000);
    t1.interrupt();
    System.out.println("Rest of main thread...");
}
```

sourcemind

Locks: Wait and Notify

- Problem: An application should start only some operations are finished by an initializer thread.
- Example: Guarding thread stops the main thread from moving forward, while initializer thread performs some actions

```
public static void main(String[] args) {
    Thread initializer = // performs initialization process
    Thread guard = // checks if initialization is complete

    initializer.start();
    guard.start();
    guard.join();
    // application starts here
}
```

Busy waiting

- An initialized flag controls the process

```
public class StartupBusyWait {

    private static boolean initialized = false;

    public static void main(String[] args) throws InterruptedException {
        Thread guard = new Thread(() -> {
            while (!initialized) {
                System.out.println(Thread.currentThread().getName() + ": Waiting to initialize...");
                // Thread.sleep ?
                // Thread.yield ?
            }
            System.out.println(Thread.currentThread().getName() + ": Program is initialized");
        });
        guard.setName("Guarding Thread");

        Thread initializer = new Thread(() -> {
            try {
                Thread.sleep(3000);
                System.out.println(Thread.currentThread().getName() +": Ready to go");
                initialized = true;
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });
        ……..
    }
}
```

Busy waiting

```java
public class StartupBusyWait {

    private static boolean initialized = false;

    public static void main(String[] args) throws InterruptedException {
        ................
        initializer.setName("Initializer Thread");

        // What if initialization is never done?
        initializer.start();
        guard.start();
        guard.join();
        System.out.println(Thread.currentThread().getName() + ": Main program can start now");
    }
}
```

- Issues
  - ✘ Busy waiting consumes resources
  - ✘ volatile issue
  - ✘ Verify using VisualVM

sourcemind

Wait and notify

- Object has a few methods related to locks:
  - ✘ wait(), wait(long millis): causes the current thread to lose the lock and go to waiting mode
  - ✘ notify(), notifyAll(): causes one or all waiting threads to wake up, acquire the lock again, and continue their work
- Waiting mode is good, since it won't consume CPU resources

Wait and notify

```
public class StartupWaitAndNotify {

    private static final Object lock = new Object();

    private static boolean initialized = false;

    public static void main(String[] args) throws InterruptedException {
        Thread guard = new Thread(() -> {
            synchronized (lock) {
                try {
                    while (!initialized) {  // why not "if"?
                        System.out.println(Thread.currentThread().getName() + ": Waiting to initialize...");
                        lock.wait();
                    }
                    System.out.println(Thread.currentThread().getName() + ": Program is initialized");
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });
        guard.setName("Guarding Thread");

        ............
    }
}
```

sourcemind

Wait and notify

```
public class StartupWaitAndNotify {

    private static final Object lock = new Object();

    private static boolean initialized = false;

    public static void main(String[] args) throws InterruptedException {

       …………...

        Thread initializer = new Thread(() -> {
            try {
                Thread.sleep(3000);
                System.out.println(Thread.currentThread().getName() + ": Ready to go");
                synchronized (lock) {
                    initialized = true;
                    lock.notify();
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });
        initializer.setName("Initializer Thread");

        // What if initialization is never done?
        initializer.start();
        guard.start();

        guard.join();
        System.out.println(Thread.currentThread().getName() + ": Main program can start now");
    }
}
```

sourcemind

Thread Pools

- Constructing a new thread for every little task is a heavy operation.
- Installing a new laundry machine vs. laundry room!
- Thread pool types
  - ✘ Single
  - ✘ Fixed
  - ✘ Cached
  - ✘ etc.

Executors

- Executor Service

```
ExecutorService executor = null;
try {
    executor = Executors.newSingleThreadExecutor();
    executor.execute(() -> System.out.println("Hello"));
    executor.execute(() -> System.out.println("World"));
} finally {
    if (executor != null) executor.shutdown();
}
```

Executors

- Executor states
  - ✘ Active: Accepts new tasks, executes tasks
  - ✘ Shutting down: Rejects new tasks, executes tasks
  - ✘ Shutdown: Rejects new tasks, no tasks running
- ExecutorService isShutdown() vs. isTerminated()
- ExecutorService execute() method works in the **fire and forget** style: It will be completed at some time in the future, but no results are possible to fetch

Future

submit(Runnable): Not useful to return any data, but useful to ask the state of the submitted task

```
ExecutorService executor = Executors.newSingleThreadExecutor();
Future<?> future = executor.submit(() -> {
    try {
        Thread.sleep(3000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
});

while (!future.isDone()) {
    // do nothing
}
System.out.println("Done");
executor.shutdown();
```

Future<T> submit(Runnable, T): Wraps the result of type T in the Future object: `future.get()`

sourcemind

Callable

What if the thead must call a method that returns the result?

Callable<T> represents an interface that will be called asynchronously and will return a result of type T

```
Callable<Integer> factorial = () -> {
    return 0;   // implement
};

Future<Integer> future = executor.submit(factorial);
System.out.println("Result: " + future.get());
```

sourcemind

Thread Pools

Using larger thread pools

```java
Callable<Integer> delay = () -> {
    Random random = new Random();
    int anInt = random.nextInt(100);
    Thread.sleep(anInt);
    return anInt;
};

// How many threads to use?
ExecutorService executor = Executors.newFixedThreadPool(1);

// Submit all tasks (callables)
List<Future<Integer>> futures = new ArrayList<>();
for (int i = 0; i < 100; i++) {
    Future<Integer> future = executor.submit(delay);
    futures.add(future);
}
```

sourcemind

Thread Pools

```java
// Fetch results
int completed = 0;
int totalWait = 0;
long start = System.currentTimeMillis();

for (int i = 0; i < 100; i++) {
    totalWait += futures.get(i).get();
    completed++;
}

System.out.println("Completed tasks: " + completed);
System.out.println("Total wait: " + totalWait);
System.out.println("Duration: " + (System.currentTimeMillis() - start));
```

Future.get(long, TimeUnit) waits until a threshold and throws an exception on timeout

sourcemind

Homework 5: Registeration website

Try to implement homework 4 with the features of this session whenever possible.

sourcemind