Thread Basic

Doing things simultaneously, in parallel.

Outline

- Objectives
- Thread low level
- Executor, ExecutorService, ScheduledExecutorService
- Thread Problems

Objectives

- Create worker threads using Runnable, Callable and use an ExecutorService to concurrently execute tasks.
- Identify potential threading problems among deadlock, starvation, livelock, and race conditions.

The old ways …

Create Thread with a Runnable object then call start from Thread

```
class RunnableImpl implements Runnable {

    @Override
    public void run() {
        System.out.println("Run na ja...");
    }

}

public class ThreadBasic {

    public static void main(String[] args) {
        Thread t = new Thread(new RunnableImpl());
        t.start();
    }

}
```

The old ways …

Override method run in Thread subclass

```
class ThreadSubClass extends Thread {

    @Override
    public void run() {
        System.out.println("Run ran run ...");
    }

}


public class ThreadBasic {

    public static void main(String[] args) {
        Thread t = new ThreadSubClass();
        t.start();
    }

}
```

Executor

interface @since 1.5

```
Runnable r = ...

Thread t = new Thread(r);
t.start();

Thread t2 = new Thread(r);
t2.start();

Thread t3 = new Thread(r);
t3.start();
```

```
Runnable r = …

Executor e = Executors.newSingleThreadExecutor();
e.execute(r);
e.execute(r);
e.execute(r);
```

ExecutorService

interface @since 1.5; extends Executor

Available Methods

- **execute (inherited)**
- **submit**
- **invokeAll**
- **invokeAny**
- shutdown
- shutdownNow
- awaitTermination
- isShutdown
- isTerminated

Executors

- newSingleThreadExecutor
- newCachedThreadPool
- newFixedThreadPool
- newWorkStealingPool

ScheduledExecutorService

interface @since 1.5; extends ExecutorService

Available Methods

- schedule
- scheduleAtFixedRate (no result)
- scheduleAtFixedDelay (no result)

Executors

- newScheduledThreadPool
- newSingleThreadScheduledExecutor

Shutdown Thread Pool

shutdown()

Tells the executor to stop accepting new tasks, but the previous tasks are allowed to continue until the finish

shutdownNow()

Tell the executor to stop accepting new tasks but it will **TRY** to stop all executing tasks immediately

return a list of the tasks that were never started.

Thread Problems

- Deadlock
- Starvation
- Livelock
- Race condition

Deadlock

Two or more threads are blocked forever, waiting for each other to acquire/release some resource.

Starvation

Thread is constantly waiting for a lock, never able to take it because other threads with higher priority are continually acquiring it.

Livelock

Two (or more) threads are blocking each other, but in a livelock, each thread tries to resolve the problem on its own (live) instead of just waiting (dead). They are not blocked, but they are unable to make further progress.

Race condition

Two threads compete to access or modify the same resource at the same time in a way that causes unexpected results (generally, invalid data)

Key Points

- At a low level, we can create a thread in two ways, either by implementing Runnable or by subclassing Thread and overriding the run() method.
- At a high-level, we use Executors, which use thread pools, which in turn use worker threads.
- One type of thread pool is the fixed thread pool, which has a fixed number of threads running. We can also use single-thread pools.

Keypoint

- ExecutorService has methods to execute thread pools that either take a Runnable or Callable task. A Callable returns a result and throws a checked exception.
- The submit() method returns a Future object that represents the result of the task (if the task is a Runnable, null is returned).
- An executor has to be shutdown to close the pool thread with either shutdown() (gracefully) or shutdownNow() (forcefully).
- Deadlock, Starvation, Livelock, Race condition

Last but not least!

```
public interface Runnable {

        public abstract void run();

}
```

```
public interface Callable<V> {

        V call() throws Exception;

}
```