Jactor for Dummies, by Example

by Bill la Forge, 2014

JActor is a robust and high-performance alternative to threads and locks. JActor for Dummies focuses on a subset of the API that is easy to learn but reasonably comprehensive.

The JActor Model

- Messages are passed between light-weight threads (LWTs), which process one message at a time.
- There are three types of messages: signals, requests and responses.
- Signals are sent immediately while requests and responses are buffered until processing of the current message is complete.
- Requests are isolated from each other and processed only when the previous request returns a result or exception.

The HelloWorld Example

```
package org.agilewiki.jactor2.core.isolation;

import org.agilewiki.jactor2.core.blades.IsolationBladeBase;
import org.agilewiki.jactor2.core.impl.Plant;
import org.agilewiki.jactor2.core.requests.AsyncResponseProcessor;
import org.agilewiki.jactor2.core.requests.impl.AsyncRequestImpl;

public class HelloWorld extends IsolationBladeBase {

    public static void main(final String[] args) throws Exception {
        new Plant();
        new HelloWorld();
        System.out.println("initialized");
    }

    public HelloWorld() throws Exception {
        new AIO("run") {
            @Override
            protected void processAsyncOperation(final AsyncRequestImpl _asyncRequestImpl,
                        final AsyncResponseProcessor<Void> _asyncResponseProcessor)
                    throws Exception {
                System.out.println("Hello world!");
                Plant.close();
                System.out.println("finished");
            }
        }.signal();
    }
}
```

Output:

```
initialized
Hello world!
finished
```

The *HelloWorld* class wraps a LWT, which is created when the default constructor of *IsolationBladeBase* is called.

```
    public static void main(final String[] args) throws Exception {
        new Plant();
        new HelloWorld();
        System.out.println("initialized");
```

```
}
```

The *main* method does three things:

1. An instance of *Plant* is created. This provides the operating environment and configuration for the LWTs.
2. An instance of *HelloWorld* is created. And
3. The line *initialized* is printed.

```
public HelloWorld() throws Exception {
    new AIO("run") {
        @Override
        protected void processAsyncOperation(final AsyncRequestImpl _asyncRequestImpl,
                    final AsyncResponseProcessor<Void> _asyncResponseProcessor)
                throws Exception {
            System.out.println("Hello world!");
            Plant.close();
            System.out.println("finished");
        }
    }.signal();
}
```

The constructor creates a *run* signal which is passed to the *HelloWorld* LWT. On receipt of this signal, the LWT prints the line *Hello world!*, closes the operating environment and then prints the line *finished*.

Notes:

1. The *AIO.signal* method can be called from any thread and within any context. In this case the method was called from the main thread.
2. *AIO* is a nested class, defined in one of the super classes of *HelloWorld*. This is how the *signal* method accesses the LWT of *HelloWorld*.