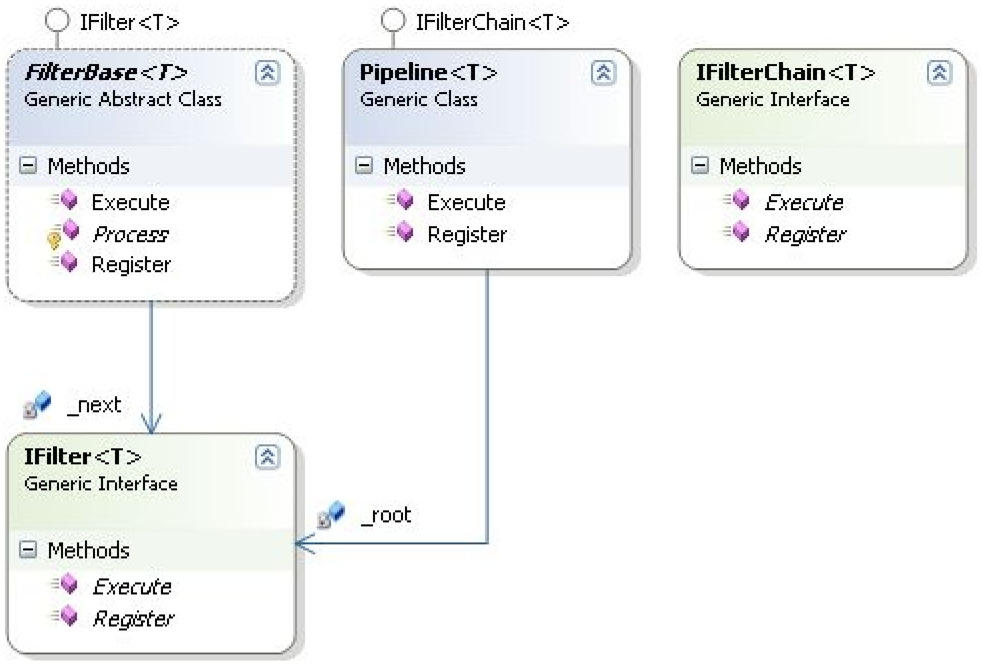
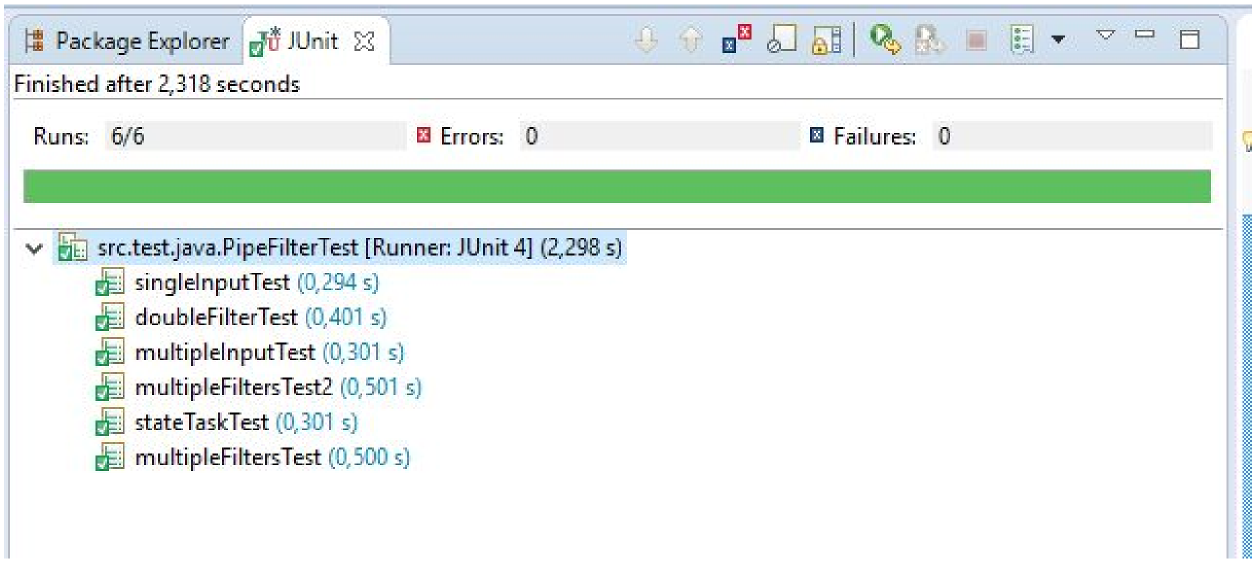
Pipe-filter pattern



We followed the above example for our own implementation of the Pipe-filter design pattern. (http://bl.ocks.org/roryokane/9606238) Was used in order to help us understand the implementation within Java for the Pipe-filter pattern.

Test result:
We tested our implementation using Junit (http://junit.org/) within our Java Gradle build (https://gradle.org/).



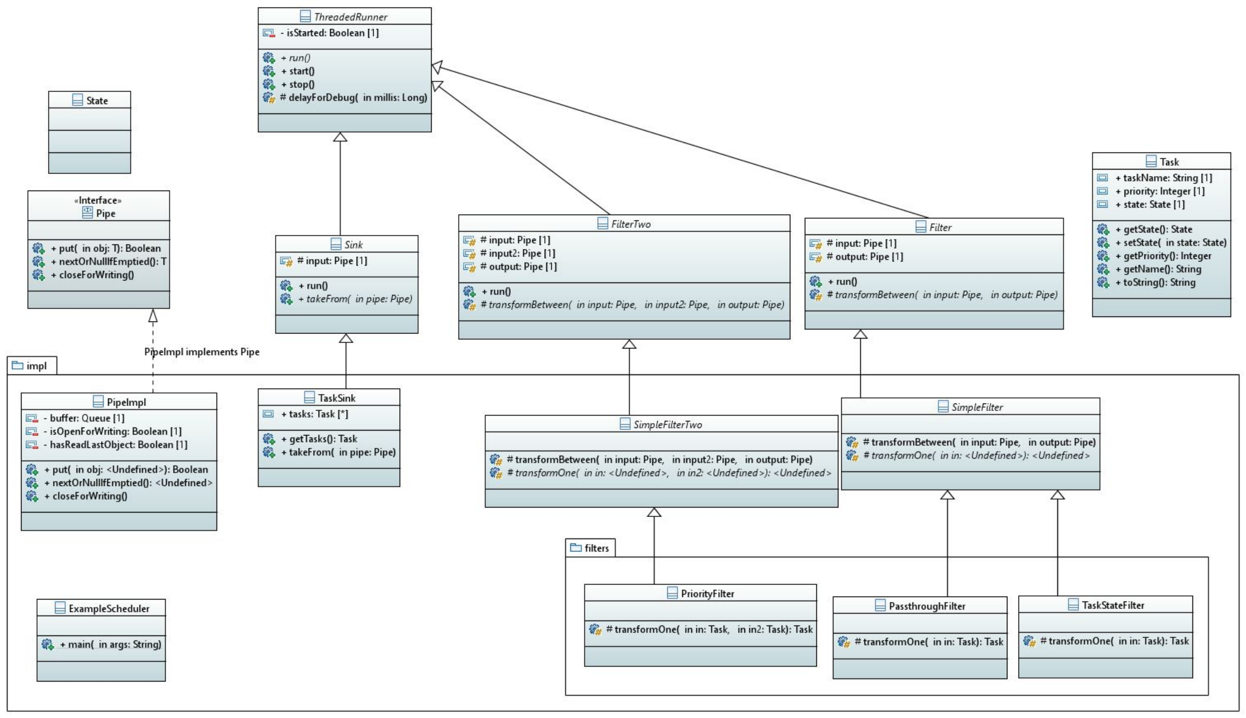
```
Finished after 2,318 seconds
Runs: 6/6    Errors: 0    Failures: 0

src.test.java.PipeFilterTest [Runner: JUnit 4] (2,298 s)
    singleInputTest (0,294 s)
    doubleFilterTest (0,401 s)
    multipleInputTest (0,301 s)
    multipleFiltersTest2 (0,501 s)
    stateTaskTest (0,301 s)
    multipleFiltersTest (0,500 s)
```

We've tested a number of situations with a few example filters. All of our testing was done with Task objects. These task objects contained some fields with their name, priority and state. We chose this for our example because it might resemble a real life robot application should this robot ever be created.

In our implementation we've created three distinct filters. The first filter doesn't modify the data and operates with one input and one output. The second filter modifies the State of a task object with one in- and output pipe. And our last filter receives two input streams and has one output stream. This filter determines which of the two tasks it received to put through. and which to discard based on each task's priority. None of the above mentioned filters would likely be used (except for maybe the priority based filter) in real life scenario's but for our demonstration of the pipe and filter pattern they will suffice.

UML for our implementation


ThreadedRunner
- isStarted: Boolean [1]
+ run()
+ start()
+ stop()
# delayForDebug(in millis: Long)
State
«Interface»
Pipe
+ put(in obj: T): Boolean
+ nextOrNullIfEmptied(): T
+ closeForWriting()
Sink
# input: Pipe [1]
+ run()
+ takeFrom(in pipe: Pipe)
FilterTwo
# input: Pipe [1]
# input2: Pipe [1]
# output: Pipe [1]
+ run()
# transformBetween(in input: Pipe, in input2: Pipe, in output: Pipe)
Filter
# input: Pipe [1]
# output: Pipe [1]
+ run()
# transformBetween(in input: Pipe, in output: Pipe)
Task
+ taskName: String [1]
+ priority: Integer [1]
+ state: State [1]
+ getState(): State
+ setState(in state: State)
+ getPriority(): Integer
+ getName(): String
+ toString(): String
PipeImpl implements Pipe
impl
PipeImpl
- buffer: Queue [1]
- isOpenForWriting: Boolean [1]
- hasReadLastObject: Boolean [1]
+ put(in obj: <Undefined>): Boolean
+ nextOrNullIfEmptied(): <Undefined>
+ closeForWriting()
TaskSink
+ tasks: Task [*]
+ getTasks(): Task
+ takeFrom(in pipe: Pipe)
SimpleFilterTwo
# transformBetween(in input: Pipe, in input2: Pipe, in output: Pipe)
# transformOne(in in: <Undefined>, in in2: <Undefined>): <Undefined>
SimpleFilter
# transformBetween(in input: Pipe, in output: Pipe)
# transformOne(in in: <Undefined>): <Undefined>
filters
PriorityFilter
# transformOne(in in: Task, in in2: Task): Task
PassthroughFilter
# transformOne(in in: Task): Task
TaskStateFilter
# transformOne(in in: Task): Task
ExampleScheduler
+ main(in args: String)