

Experimento de Refatoração de Test Smells

Nome: Wilhelm de Sousa Steins - 495961

Projeto: [Repositório](#)

Eu estou atualmente trabalhando na refatoração dos test smells seguintes:

5 Test smells do tipo Ignored Test no arquivo:

[commons-cli\src\test\java\org\apache\commons\cli\GnuParserTest.java](#)

Código original:

```
123     @Override
124     @Test
125     @Ignore("not supported by the GnuParser")
126     public void testStopBursting2() throws Exception {
127     }
128
129     @Override
130     @Test
131     @Ignore("not supported by the GnuParser")
132     public void testUnambiguousPartialLongOption1() throws Exception {
133     }
134
135     @Override
136     @Test
137     @Ignore("not supported by the GnuParser")
138     public void testUnambiguousPartialLongOption2() throws Exception {
139     }
140
141     @Override
142     @Test
143     @Ignore("not supported by the GnuParser")
144     public void testUnambiguousPartialLongOption3() throws Exception {
145     }
146
147     @Override
148     @Test
149     @Ignore("not supported by the GnuParser")
150     public void testUnambiguousPartialLongOption4() throws Exception {
151     }
```

Código refatorado: - Refatorar os Ignored Test é só deletar o método vazio.

5 test smells do tipo Magic Number Test no arquivo:

[commons-pool\src\test\java\org\apache\commons\pool2\imp\TestEvictionTimer.java](#)

Código original:

```
69 assertEquals(2, evictionExecutor.getQueue().size()); // Reaper plus one eviction task
70 assertEquals(1, EvictionTimer.getNumTasks());
```

```
81 assertEquals(3, evictionExecutor.getQueue().size()); // Reaper plus 2 eviction tasks
82 assertEquals(2, EvictionTimer.getNumTasks());
```

```
93 assertEquals(2, evictionExecutorOnStop.getQueue().size());
94 assertEquals(1, EvictionTimer.getNumTasks());
```

Código refatorado:

```
69 private int var1 = 1;
70 private int var2 = 2;
71 assertEquals(var2, evictionExecutor.getQueue().size()); // Reaper plus one eviction task
72 assertEquals(var1, EvictionTimer.getNumTasks());
```

```
83 private int var3 = 3;
84 assertEquals(var3, evictionExecutor.getQueue().size()); // Reaper plus 2 eviction tasks
85 assertEquals(var2, EvictionTimer.getNumTasks());
```

```
96 assertEquals(var2, evictionExecutorOnStop.getQueue().size());
97 assertEquals(var1, EvictionTimer.getNumTasks());
```

5 test smells do tipo Lazy Test no arquivo:

[commons-math-legacy\src\test\java\org\apache\commons\math4\legacy\ode\nonstiff\AdamsBashforthIntegratorTest.java](#)

Código original:

```
108 @Test(expected = MaxCountExceededException.class)
109 public void exceedMaxEvaluations() throws DimensionMismatchException, NumberIsTooSmallException, MaxCountExceededException, NoBracketingException {
110
111     TestProblem1 pb = new TestProblem1();
112     double range = pb.getFinalTime() - pb.getInitialTime();
113
114     AdamsBashforthIntegrator integ = new AdamsBashforthIntegrator(2, 0, range, 1.0e-12, 1.0e-12);
115     TestProblemHandler handler = new TestProblemHandler(pb, integ);
116     integ.addStepHandler(handler);
117     integ.setMaxEvaluations(650);
118     integ.integrate(pb,
119         pb.getInitialTime(), pb.getInitialState(),
120         pb.getFinalTime(), new double[pb.getDimension()]);
121 }
122
123 @Test
124 public void backward() throws DimensionMismatchException, NumberIsTooSmallException, MaxCountExceededException, NoBracketingException {
125
126     TestProblem5 pb = new TestProblem5();
127     double range = JdkMath.abs(pb.getFinalTime() - pb.getInitialTime());
128
129     AdamsBashforthIntegrator integ = new AdamsBashforthIntegrator(4, 0, range, 1.0e-12, 1.0e-12);
130     integ.setStarterIntegrator(new PerfectStarter(pb, (integ.getNSteps() + 5) / 2));
131     TestProblemHandler handler = new TestProblemHandler(pb, integ);
132     integ.addStepHandler(handler);
133     integ.integrate(pb, pb.getInitialTime(), pb.getInitialState(),
134         pb.getFinalTime(), new double[pb.getDimension()]);
135
136     Assert.assertEquals(0.0, handler.getLastError(), 4.3e-8);
137     Assert.assertEquals(0.0, handler.getMaximalValueError(), 4.3e-8);
138     Assert.assertEquals(0, handler.getMaximalTimeError(), 1.0e-16);
139     Assert.assertEquals("Adams-Bashforth", integ.getName());
140
141
142 @Test
143 public void polynomial() throws DimensionMismatchException, NumberIsTooSmallException, MaxCountExceededException, NoBracketingException {
144     TestProblem6 pb = new TestProblem6();
145     double range = JdkMath.abs(pb.getFinalTime() - pb.getInitialTime());
146
147     for (int nSteps = 2; nSteps < 8; ++nSteps) {
148         AdamsBashforthIntegrator integ =
149             new AdamsBashforthIntegrator(nSteps, 1.0e-6 * range, 0.1 * range, 1.0e-4, 1.0e-4);
150         integ.setStarterIntegrator(new PerfectStarter(pb, nSteps));
151         TestProblemHandler handler = new TestProblemHandler(pb, integ);
152         integ.addStepHandler(handler);
153         integ.integrate(pb, pb.getInitialTime(), pb.getInitialState(),
154             pb.getFinalTime(), new double[pb.getDimension()]);
155         if (nSteps < 5) {
156             Assert.assertTrue(handler.getMaximalValueError() > 0.005);
157         } else {
158             Assert.assertTrue(handler.getMaximalValueError() < 5.0e-10);
159         }
160     }
161 }
```

Código refatorado:

```
108 | @Test(expected = MaxCountExceededException.class)
109 | public void polynomial() throws DimensionMismatchException, NumberIsTooSmallException, MaxCountExceededException, NoBracketingException {
110 |     TestProblem6 pb = new TestProblem6();
111 |     double range = JdkMath.abs(pb.getFinalTime() - pb.getInitialTime());
112 |
113 |     for (int nSteps = 2; nSteps < 8; ++nSteps) {
114 |         AdamsBashforthIntegrator integ =
115 |             new AdamsBashforthIntegrator(nSteps, 1.0e-6 * range, 0.1 * range, 1.0e-4, 1.0e-4);
116 |         integ.setStarterIntegrator(new PerfectStarter(pb, nSteps));
117 |         TestProblemHandler handler = new TestProblemHandler(pb, integ);
118 |         integ.addStepHandler(handler);
119 |         integ.integrate(pb, pb.getInitialTime(), pb.getInitialState(),
120 |             pb.getFinalTime(), new double[pb.getDimension()]);
121 |         if (nSteps < 5) {
122 |             Assert.assertTrue(handler.getMaximalValueError() > 0.005);
123 |         } else {
124 |             Assert.assertTrue(handler.getMaximalValueError() < 5.0e-10);
125 |         }
126 |     }
127 |
128 |
129 |     TestProblem5 pb2 = new TestProblem5();
130 |     double range2 = JdkMath.abs(pb2.getFinalTime() - pb2.getInitialTime());
131 |
132 |     AdamsBashforthIntegrator integ2 = new AdamsBashforthIntegrator(4, 0, range2, 1.0e-12, 1.0e-12);
133 |     integ2.setStarterIntegrator(new PerfectStarter(pb2, (integ2.getNSteps() + 5) / 2));
134 |     TestProblemHandler handler2 = new TestProblemHandler(pb2, integ2);
135 |     integ2.addStepHandler(handler2);
136 |     integ2.integrate(pb2, pb2.getInitialTime(), pb2.getInitialState(),
137 |         pb2.getFinalTime(), new double[pb2.getDimension()]);
138 |
139 |     Assert.assertEquals(0.0, handler2.getLastError(), 4.3e-8);
140 |     Assert.assertEquals(0.0, handler2.getMaximalValueError(), 4.3e-8);
141 |     Assert.assertEquals(0, handler2.getMaximalTimeError(), 1.0e-16);
142 |     Assert.assertEquals("Adams-Bashforth", integ2.getName());
143 |
144 |
145 |     TestProblem1 pb3 = new TestProblem1();
146 |     double range3 = pb3.getFinalTime() - pb3.getInitialTime();
147 |
148 |     AdamsBashforthIntegrator integ3 = new AdamsBashforthIntegrator(2, 0, range3, 1.0e-12, 1.0e-12);
149 |     TestProblemHandler handler3 = new TestProblemHandler(pb3, integ3);
150 |     integ3.addStepHandler(handler3);
151 |     integ3.setMaxEvaluations(650);
152 |     integ3.integrate(pb3,
153 |         pb3.getInitialTime(), pb3.getInitialState(),
154 |         pb3.getFinalTime(), new double[pb3.getDimension()]);
155 | }
```

Minhas principais dificuldades ao remover essas anormalidades foram:
No Ignored Test, não teve muito, pois era só deletar os métodos.

No Magic Number, era necessário criar uma variável para referenciar números que existiam de forma ser um link.

No Lazy Test, era necessário juntar métodos que estavam separados.

Eu estou usando as seguintes técnicas de refatoração para remover test smells:

Remover os métodos, juntar os métodos e criar variáveis para números sem links.

De 0 a 10, quão prejudicial é esse test smell para o sistema? Por que?

Ignored Test - 3, Contador de testes vai entender os Ignored Test

Magic Number - 2, Número randômico que não é identificado por uma variável, que facilite o entendimento do código.

Lazy Test - 2, Juntando métodos que fazem sentido ficarem juntos.