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:

```java
    @Override
    @Test
    @Ignore("not supported by the GnuParser")
    public void testStopBursting2() throws Exception {
    }

    @Override
    @Test
    @Ignore("not supported by the GnuParser")
    public void testUnambiguousPartialLongOption1() throws Exception {
    }

    @Override
    @Test
    @Ignore("not supported by the GnuParser")
    public void testUnambiguousPartialLongOption2() throws Exception {
    }

    @Override
    @Test
    @Ignore("not supported by the GnuParser")
    public void testUnambiguousPartialLongOption3() throws Exception {
    }

    @Override
    @Test
    @Ignore("not supported by the GnuParser")
    public void testUnambiguousPartialLongOption4() throws Exception {
    }
```

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\impl\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:

```java
@Test(expected = MaxCountExceededException.class)
public void exceedMaxEvaluations() throws DimensionMismatchException, NumberIsTooSmallException, MaxCountExceededException, NoBracketingException {

    TestProblem1 pb  = new TestProblem1();
    double range = pb.getFinalTime() - pb.getInitialTime();

    AdamsBashforthIntegrator integ = new AdamsBashforthIntegrator(2, 0, range, 1.0e-12, 1.0e-12);
    TestProblemHandler handler = new TestProblemHandler(pb, integ);
    integ.addStepHandler(handler);
    integ.setMaxEvaluations(650);
    integ.integrate(pb,
                    pb.getInitialTime(), pb.getInitialState(),
                    pb.getFinalTime(), new double[pb.getDimension()]);
}

@Test
public void backward() throws DimensionMismatchException, NumberIsTooSmallException, MaxCountExceededException, NoBracketingException {

    TestProblem5 pb = new TestProblem5();
    double range = JdkMath.abs(pb.getFinalTime() - pb.getInitialTime());

    AdamsBashforthIntegrator integ = new AdamsBashforthIntegrator(4, 0, range, 1.0e-12, 1.0e-12);
    integ.setStarterIntegrator(new PerfectStarter(pb, (integ.getNSteps() + 5) / 2));
    TestProblemHandler handler = new TestProblemHandler(pb, integ);
    integ.addStepHandler(handler);
    integ.integrate(pb, pb.getInitialTime(), pb.getInitialState(),
                    pb.getFinalTime(), new double[pb.getDimension()]);

    Assert.assertEquals(0.0, handler.getLastError(), 4.3e-8);
    Assert.assertEquals(0.0, handler.getMaximalValueError(), 4.3e-8);
    Assert.assertEquals(0, handler.getMaximalTimeError(), 1.0e-16);
    Assert.assertEquals("Adams-Bashforth", integ.getName());
}

@Test
public void polynomial() throws DimensionMismatchException, NumberIsTooSmallException, MaxCountExceededException, NoBracketingException {
    TestProblem6 pb = new TestProblem6();
    double range = JdkMath.abs(pb.getFinalTime() - pb.getInitialTime());

    for (int nSteps = 2; nSteps < 8; ++nSteps) {
        AdamsBashforthIntegrator integ =
            new AdamsBashforthIntegrator(nSteps, 1.0e-6 * range, 0.1 * range, 1.0e-4, 1.0e-4);
        integ.setStarterIntegrator(new PerfectStarter(pb, nSteps));
        TestProblemHandler handler = new TestProblemHandler(pb, integ);
        integ.addStepHandler(handler);
        integ.integrate(pb, pb.getInitialTime(), pb.getInitialState(),
                        pb.getFinalTime(), new double[pb.getDimension()]);
        if (nSteps < 5) {
            Assert.assertTrue(handler.getMaximalValueError() > 0.005);
        } else {
            Assert.assertTrue(handler.getMaximalValueError() < 5.0e-10);
        }
    }
}
```

Código refatorado:

```java
@Test(expected = MaxCountExceededException.class)
public void polynomial() throws DimensionMismatchException, NumberIsTooSmallException, MaxCountExceededException, NoBracketingException {
    TestProblem6 pb = new TestProblem6();
    double range = JdkMath.abs(pb.getFinalTime() - pb.getInitialTime());

    for (int nSteps = 2; nSteps < 8; ++nSteps) {
        AdamsBashforthIntegrator integ =
            new AdamsBashforthIntegrator(nSteps, 1.0e-6 * range, 0.1 * range, 1.0e-4, 1.0e-4);
        integ.setStarterIntegrator(new PerfectStarter(pb, nSteps));
        TestProblemHandler handler = new TestProblemHandler(pb, integ);
        integ.addStepHandler(handler);
        integ.integrate(pb, pb.getInitialTime(), pb.getInitialState(),
                        pb.getFinalTime(), new double[pb.getDimension()]);
        if (nSteps < 5) {
            Assert.assertTrue(handler.getMaximalValueError() > 0.005);
        } else {
            Assert.assertTrue(handler.getMaximalValueError() < 5.0e-10);
        }
    }


    TestProblem5 pb2 = new TestProblem5();
    double range2 = JdkMath.abs(pb2.getFinalTime() - pb2.getInitialTime());

    AdamsBashforthIntegrator integ2 = new AdamsBashforthIntegrator(4, 0, range2, 1.0e-12, 1.0e-12);
    integ2.setStarterIntegrator(new PerfectStarter(pb2, (integ2.getNSteps() + 5) / 2));
    TestProblemHandler handler2 = new TestProblemHandler(pb2, integ2);
    integ2.addStepHandler(handler2);
    integ2.integrate(pb2, pb2.getInitialTime(), pb2.getInitialState(),
                     pb2.getFinalTime(), new double[pb2.getDimension()]);

    Assert.assertEquals(0.0, handler2.getLastError(), 4.3e-8);
    Assert.assertEquals(0.0, handler2.getMaximalValueError(), 4.3e-8);
    Assert.assertEquals(0, handler2.getMaximalTimeError(), 1.0e-16);
    Assert.assertEquals("Adams-Bashforth", integ2.getName());


    TestProblem1 pb3  = new TestProblem1();
    double range3 = pb3.getFinalTime() - pb3.getInitialTime();

    AdamsBashforthIntegrator integ3 = new AdamsBashforthIntegrator(2, 0, range3, 1.0e-12, 1.0e-12);
    TestProblemHandler handler3 = new TestProblemHandler(pb3, integ3);
    integ3.addStepHandler(handler3);
    integ3.setMaxEvaluations(650);
    integ3.integrate(pb3,
                     pb3.getInitialTime(), pb3.getInitialState(),
                     pb3.getFinalTime(), new double[pb3.getDimension()]);
}
```

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.