Test-Driven Development

A Powerful Design and Programming Technique

... or The Bowling Game Kata in C++/QUnit

Olve Maudal , oma@pvv.org

This is a modified version of an internal talk given at TANDBERG TechZone, Lillehammer 2007.
Feel free to use this material for whatever you want.

(19. January 2007)

(from TechZone 2007)

Olve Maudal
Email: oma@pvv.org
Mobile: +47 90093309
Skype: olve.maudal
MSN: oma@pvv.org

Background:

- BEng (Hons) Software Engineering (UMIST, Manchester)
- MSc Intelligent Robotics (DAI, Edinburgh)
- ~4 years with Schlumberger, developing systems for finding oil
- ~4 years with BBS, developing systems for moving money
- ~2 years with TANDBERG, developing systems for audiovisual communication

- Brief introduction to Test-Driven Development
- QUnit - A simple framework for unit testing in C++
- The Bowling Game Kata in C++
- More TDD examples
- Q&A

(~40 minutes + QA)

- **Brief introduction to Test-Driven Development**
- QUnit - A simple framework for unit testing in C++
- The Bowling Game Kata in C++
- More TDD examples
- Q&A

Test-Driven Development (defined)

Test-Driven Development (defined)

Test-Driven Development (TDD) is a computer programming technique that involves repeatedly first writing a test case and then implementing only the code necessary to pass the test.

(source: Wikipedia)

Test-Driven Development (defined)

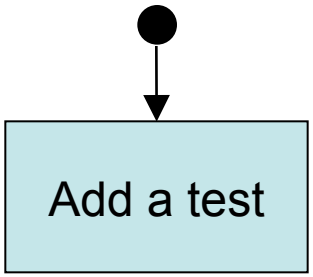
Test-Driven Development (TDD) is a computer programming technique that involves repeatedly first writing a test case and then implementing only the code necessary to pass the test.

... Practitioners emphasize that test-driven development is a method of designing software, not merely a method of testing.

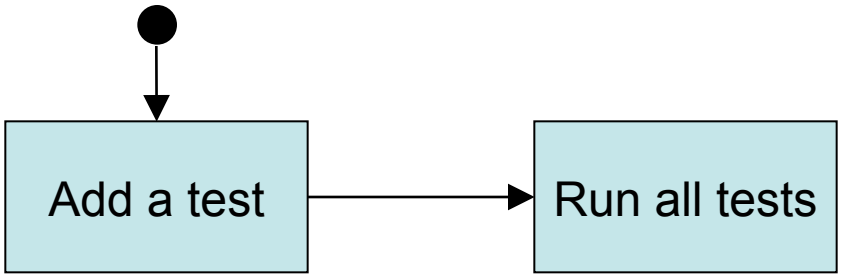
(source: Wikipedia)

Test-Driven Development Cycle

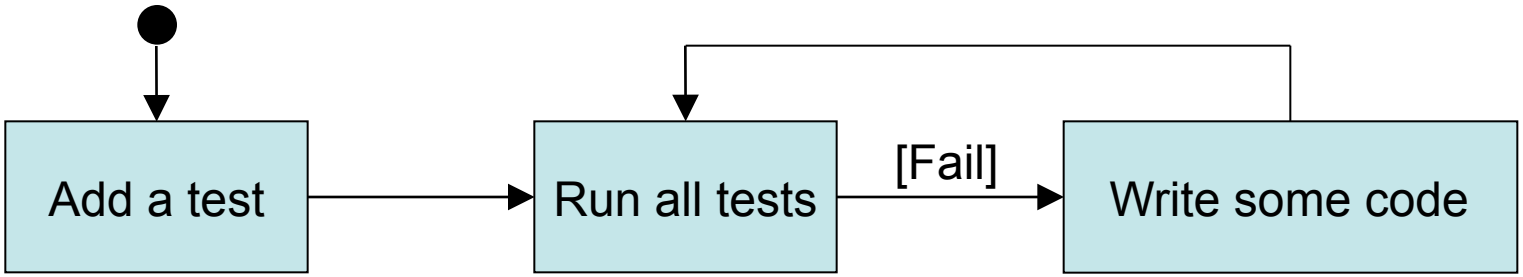
Test-Driven Development Cycle



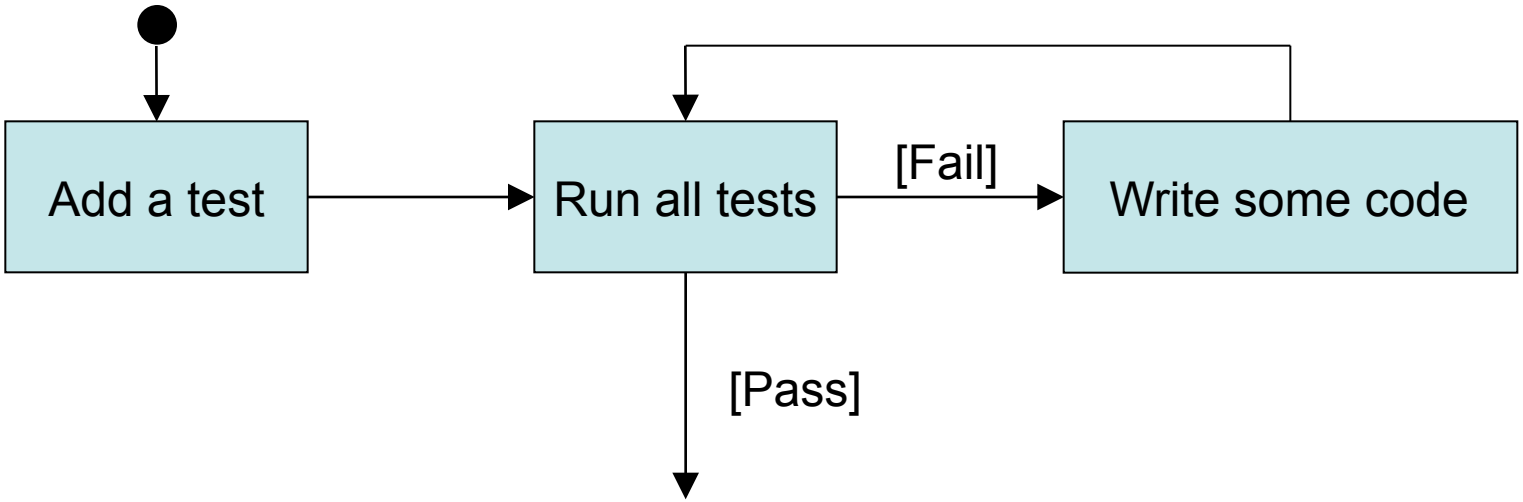
Test-Driven Development Cycle



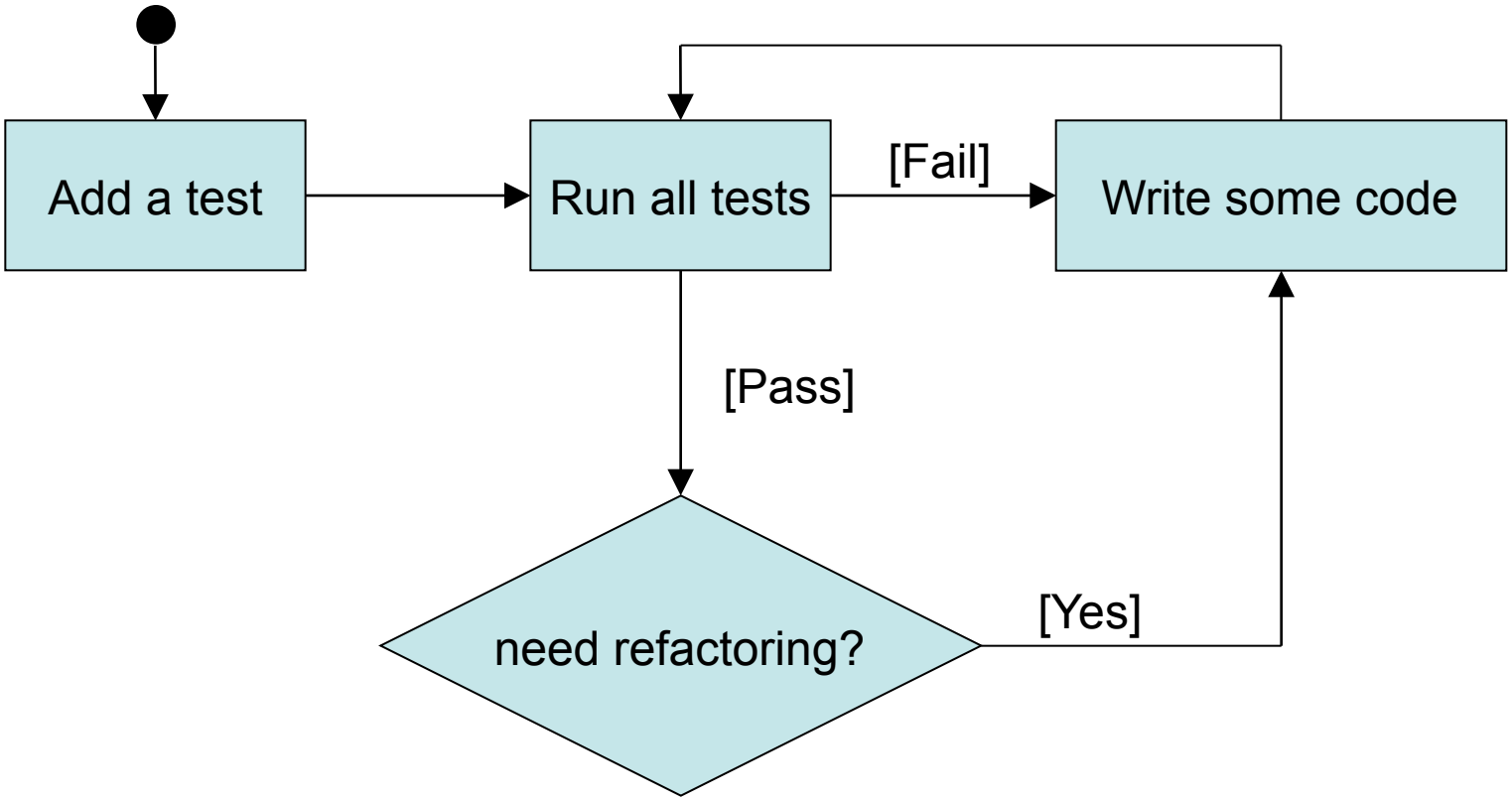
Test-Driven Development Cycle



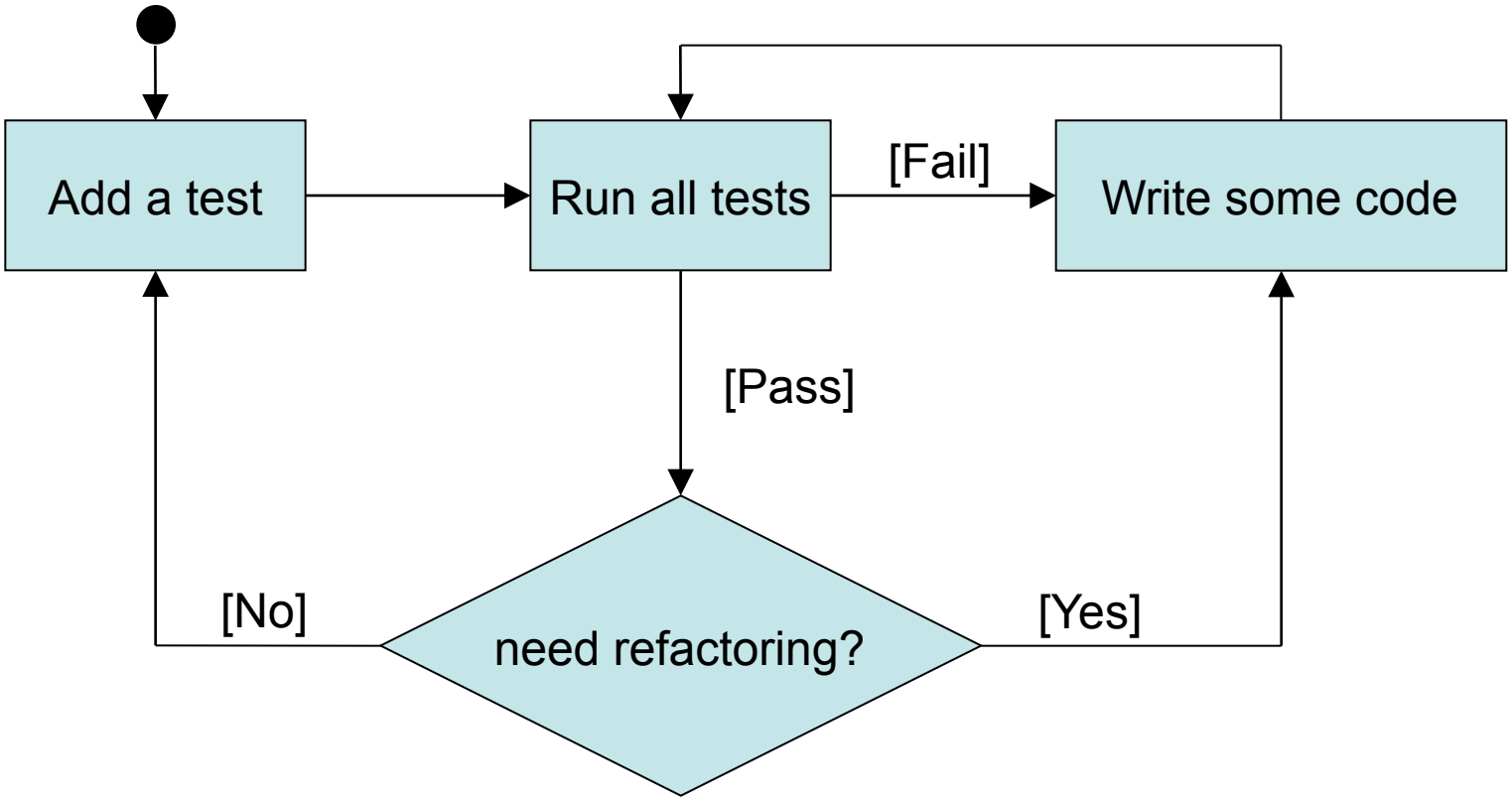
Test-Driven Development Cycle



Test-Driven Development Cycle



Test-Driven Development Cycle



Why do you need TDD?

Why do you need TDD?

“Make everything as simple as possible, but not simpler.”
(Albert Einstein)

Why do you need TDD?

“Make everything as simple as possible, but not simpler.”
(Albert Einstein)

“Anything that does not directly contribute value to the customer is waste. Perhaps the single biggest source of waste in software development is unused functionality.”
(leansoftwareinstitute.com)

Why do you need TDD?

“Make everything as simple as possible, but not simpler.”
(Albert Einstein)

“Anything that does not directly contribute value to the customer is waste. Perhaps the single biggest source of waste in software development is unused functionality.”
(leansoftwareinstitute.com)

... to avoid design paralysis.

What do you need for TDD?

What do you need for TDD?

- a framework for unit testing

What do you need for TDD?

- a framework for unit testing
- a powerful development environment

- Brief introduction to Test-Driven Development
- **QUnit - A simple framework for unit testing in C++**
- The Bowling Game Kata in C++
- More TDD examples
- Q&A

Unit Testing (defined)

Unit Testing (defined)

In computer programming, unit testing is a procedure used to validate that individual modules or units of source code are working properly.

(source: Wikipedia)

Unit Testing (defined)

In computer programming, unit testing is a procedure used to validate that individual modules or units of source code are working properly.

... in Object Oriented Design smallest unit is always Class

(source: Wikipedia)

Unit Testing Frameworks

Check

Boost Test Library

Test::Unit

PyUnit

QuickTest

CuTest

csUnit

GNU Autounit

CPPUnit

GUnit

CUnit

Unit++

CppUnitLite

JUnit

TUT

NUnit

TestNG

C++test

Cgreen

UnitTest++

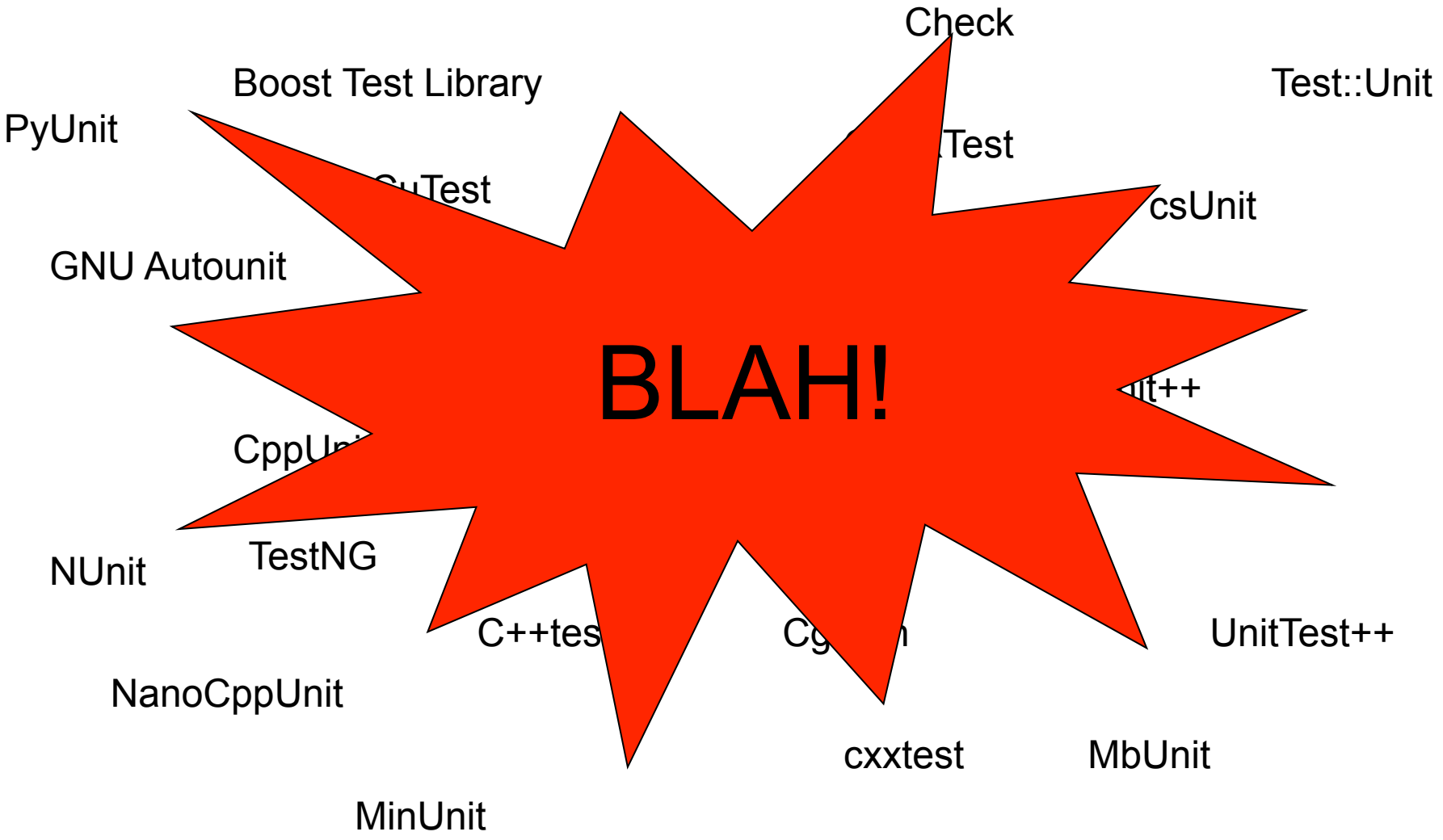
NanoCppUnit

cxxtest

MbUnit

MinUnit

Unit Testing Frameworks



QUnit – A Framework for Unit Testing

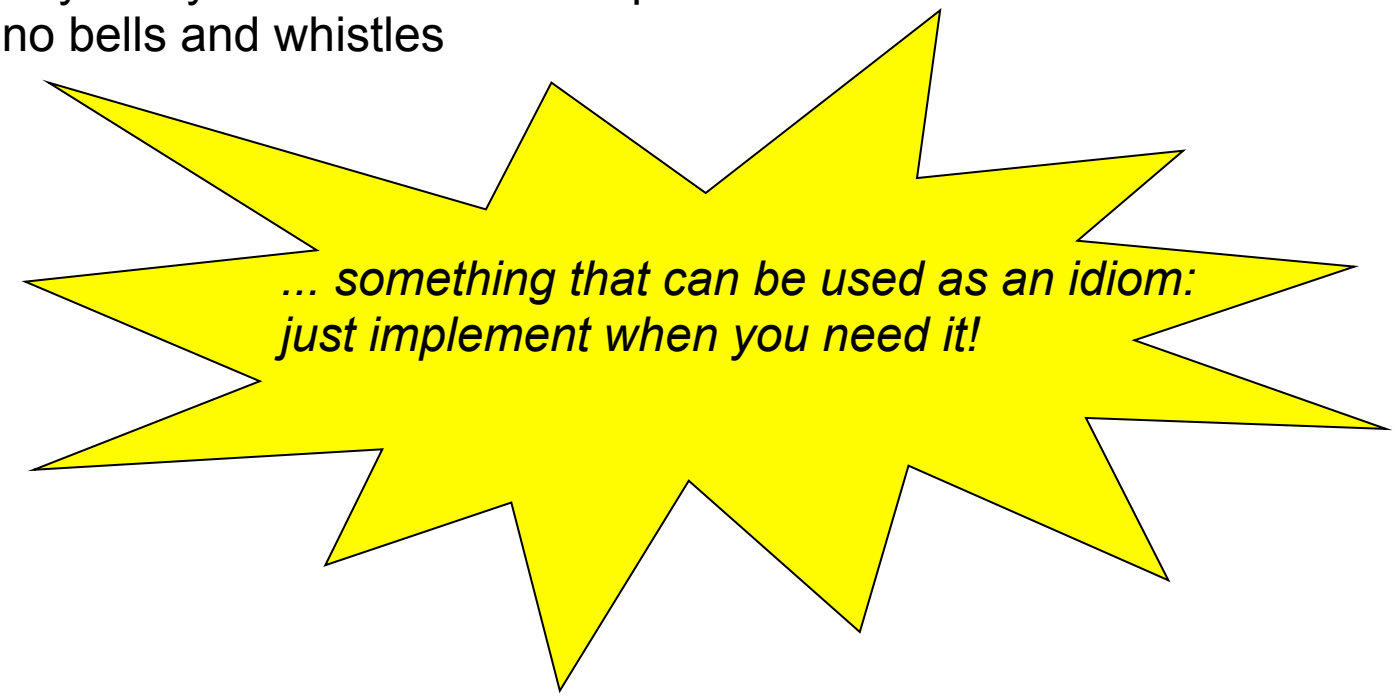
If not already an integrated part of your development environment then you probably want something that:

- feels like ~50 lines of code
- fits nicely into your favourite development environment
- have no bells and whistles

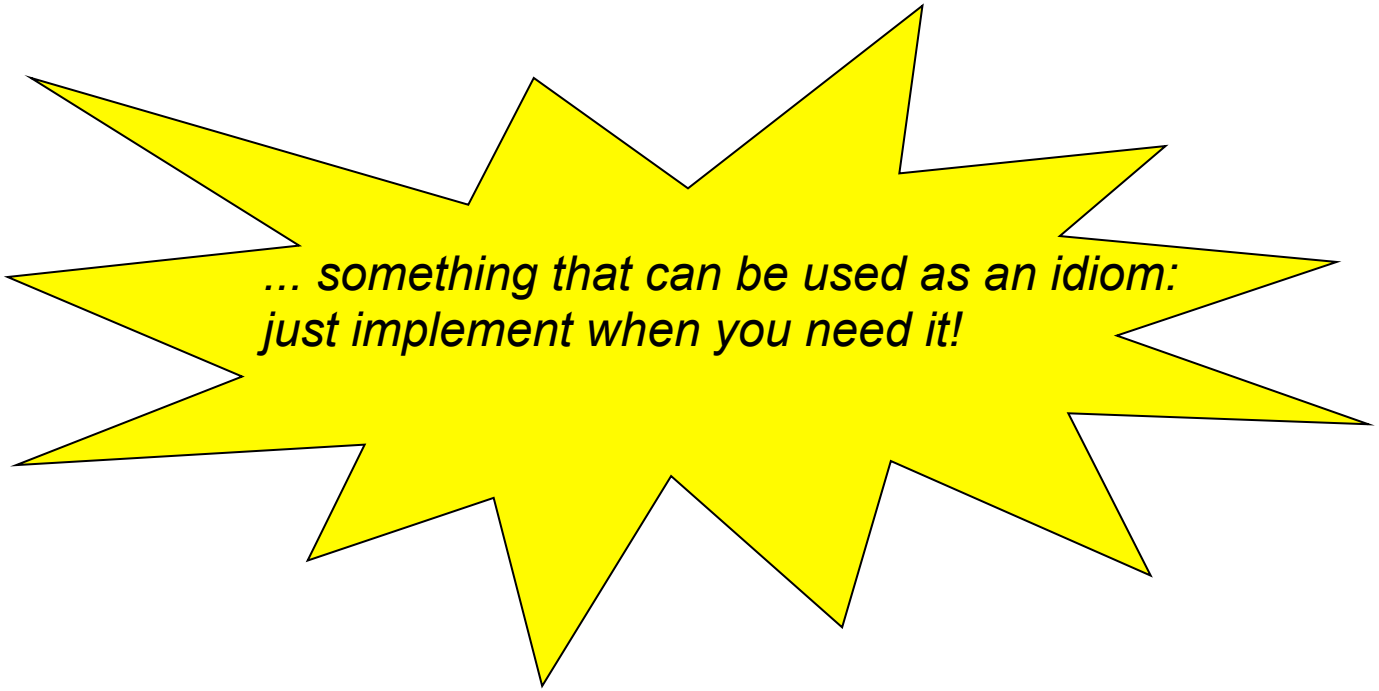
QUnit – A Framework for Unit Testing

If not already an integrated part of your development environment then you probably want something that:

- feels like ~50 lines of code
- fits nicely into your favourite development environment
- have no bells and whistles



QUnit – A Framework for Unit Testing



so here I give it to you...

```
// QUnit.hpp - A Unit Testing Framework - Olve Maudal (2007)

#ifndef QUNIT_HPP
#define QUNIT_HPP

#include <string>

class QUnit {
private:

    std::string _name;
    std::string _context;
    int _steps;
    int _tests;
    int _errors;
    int _verbose;

protected:

    QUnit(std::string name) : _name(name), _context(""),
        _steps(0), _tests(0), _errors(0), _verbose(0) {}
    virtual ~QUnit() {}
    int verbose() { return _verbose; };
    void setContext(std::string context);
    void assertTrue(int condition, const char * str = NULL,
                    const char * file = NULL, int line = -1);
    virtual void run() = 0;

public:

    int execute(int argc, char ** argv);

};

#endif
```

```
// QUnit.cpp - A Unit Testing Framework - Olve Maudal (2007)

#include "QUnit.hpp"
#include <iostream>

using namespace std;

void QUnit::setContext(string context) {
    _context = context;
    _steps = 0;
}

void QUnit::assertTrue(int condition, const char * str,
                           const char * file, int line) {
    _steps++;
    _tests++;
    _errors += (condition != true);
    if( _verbose ) {
        cout << (condition?"OK    ":"ERROR ") << _name
             << "/" << _context << "/" << _steps << endl;
    }
    if( !condition && str && file) {
        cerr << file << ":" << line << ": unittest failed: "
             << str << " (" << _name << "/" << _context
             << "/" << _steps << ")" << endl;
    }
}

int QUnit::execute(int argc, char ** argv) {
    for ( int i=1; i<argc; ++i ) {
        if ( strcmp(argv[i], "-v") == 0 ) {
            _verbose++;
            continue;
        }
        cerr << "usage: " << argv[0] << " -v" << endl;
        return -1;
    }

    run();

    if( verbose() > 0 ) {
        cout << _name << (_errors? " FAILED":" OK")
             << " (" << _tests << " tests, "
             << _errors << " errors)" << endl;
    }

    return _errors;
}
```

QUnitDemo

```
// QUnitDemo.cpp

#include "QUnit.hpp"

class QUnitDemo : public QUnit {
public:
    QUnitDemo() : QUnit("QUnitDemo") {}
    void run() {
    }
};

int main(int argc, char** argv) {
    QUnitDemo t;
    return t.execute(argc, argv);
}
```

QUnitDemo

```
> g++ -c QUnit.cpp
> g++ -o QUnitDemo QUnitDemo.cpp QUnit.o
> QUnitDemo -v
QUnitDemo OK (0 tests, 0 errors)
```

QUnitDemo

```
// QUnitDemo.cpp

#include "QUnit.hpp"

class QUnitDemo : public QUnit {
public:
    QUnitDemo() : QUnit("QUnitDemo") {}
    void run() {
        assertTrue(1 == 1);
    }
};

int main(int argc, char** argv) {
    QUnitDemo t;
    return t.execute(argc, argv);
}
```

QUnitDemo

```
g++ -o QUnitDemo QUnitDemo.cpp QUnit.o
QUnitDemo -v
OK    QUnitDemo//1
QUnitDemo OK (1 tests, 0 errors)
```

QUnitDemo

```
// QUnitDemo.cpp

#include "QUnit.hpp"

class QUnitDemo : public QUnit {
public:
    QUnitDemo() : QUnit("QUnitDemo") {}
    void run() {
        assertTrue(1 == 1);
        assertTrue(2 == 2);
        assertTrue(3 == 3);
        assertTrue(4 == 7);
        assertTrue(5 == 5);
    }
};

int main(int argc, char** argv) {
    QUnitDemo t;
    return t.execute(argc, argv);
}
```

QUnitDemo

```
OK    QUnitDemo//1
OK    QUnitDemo//2
OK    QUnitDemo//3
ERROR QUnitDemo//4
OK    QUnitDemo//5
QUnitDemo FAILED (5 tests, 1 errors)
```

QUnitDemo

```
// QUnitDemo.cpp

#include "QUnit.hpp"

#include <string>
using namespace std;

class QUnitDemo : public QUnit {
public:
    QUnitDemo() : QUnit("QUnitDemo") {}
    void run() {
        assertTrue(1 == 1);
        assertTrue(2 == 2);
        assertTrue(3 == 3);
        assertTrue(4 == 7);
        assertTrue(5 == 5);

        string str1 = "a";
        string str2 = "b";
        string str3 = "ab";

        assertTrue( str1 == str2 );
        assertTrue( str1 + str2 == str3 );
    }
};

int main(int argc, char** argv) {
    QUnitDemo t;
    return t.execute(argc, argv);
}
```

QUnitDemo

```
OK    QUnitDemo//1
OK    QUnitDemo//2
OK    QUnitDemo//3
ERROR QUnitDemo//4
OK    QUnitDemo//5
ERROR QUnitDemo//6
OK    QUnitDemo//7
QUnitDemo FAILED (7 tests, 2 errors)
```

QUnitDemo

```
// QUnitDemo.cpp

#include "QUnit.hpp"

#include <string>
using namespace std;

class QUnitDemo : public QUnit {
public:
    QUnitDemo() : QUnit("QUnitDemo") {}
    void run() {
        setContext("testBasicStuff");
        assertTrue(1 == 1);
        assertTrue(2 == 2);
        assertTrue(3 == 3);
        assertTrue(4 == 7);
        assertTrue(5 == 5);

        setContext("testStringStuff");
        string str1 = "a";
        string str2 = "b";
        string str3 = "ab";

        assertTrue( str1 == str2 );
        assertTrue( str1 + str2 == str3 );
    }
};

int main(int argc, char** argv) {
    QUnitDemo t;
    return t.execute(argc, argv);
}
```

QUnitDemo

```
OK    QUnitDemo/testBasicStuff/1
OK    QUnitDemo/testBasicStuff/2
OK    QUnitDemo/testBasicStuff/3
ERROR QUnitDemo/testBasicStuff/4
OK    QUnitDemo/testBasicStuff/5
ERROR QUnitDemo/testStringStuff/1
OK    QUnitDemo/testStringStuff/2
QUnitDemo FAILED (7 tests, 2 errors)
```

QUnitDemo

```
// QUnitDemo.cpp

#include "QUnit.hpp"

#include <string>
using namespace std;

class QUnitDemo : public QUnit {
public:
    QUnitDemo() : QUnit("QUnitDemo") {}
    void run() {
        setContext("testBasicStuff");
        assertTrue(1 == 1);
        assertTrue(2 == 2);
        assertTrue(3 == 3);
        assertTrue(4 == 7);
        assertTrue(5 == 5);

        setContext("testStringStuff");
        string str1 = "a";
        string str2 = "b";
        string str3 = "ab";

        assertTrue( str1 == str2, "comparing str1 and str2", __FILE__ , __LINE__ );
        assertTrue( str1 + str2 == str3 );
    }
};

int main(int argc, char** argv) {
    QUnitDemo t;
    return t.execute(argc, argv);
}
```

QUnitDemo

```
OK    QUnitDemo/testBasicStuff/1
OK    QUnitDemo/testBasicStuff/2
OK    QUnitDemo/testBasicStuff/3
ERROR QUnitDemo/testBasicStuff/4
OK    QUnitDemo/testBasicStuff/5
ERROR QUnitDemo/testStringStuff/1
QUnitDemo.cpp:24: unittest failed: comparing str1 and str2 (QUnitDemo/testStringStuff/1)
OK    QUnitDemo/testStringStuff/2
QUnitDemo FAILED (7 tests, 1 errors)
```

QUnitDemo

```
// QUnitDemo.cpp

#include "QUnit.hpp"

#include <string>
using namespace std;

#define ASSERT_TRUE(cond) ( assertTrue(cond, #cond, __FILE__, __LINE__) )

class QUnitDemo : public QUnit {
public:
    QUnitDemo() : QUnit("QUnitDemo") {}
    void run() {
        setContext("testBasicStuff");
        ASSERT_TRUE(1 == 1);
        ASSERT_TRUE(2 == 2);
        ASSERT_TRUE(3 == 3);
        ASSERT_TRUE(4 == 7);
        ASSERT_TRUE(5 == 5);

        setContext("testStringStuff");
        string str1 = "a";
        string str2 = "b";
        string str3 = "ab";

        ASSERT_TRUE( str1 == str2 );
        ASSERT_TRUE( str1 + str2 == str3 );
    }
};

int main(int argc, char** argv) {
    QUnitDemo t;
    return t.execute(argc, argv);
}
```

QUnitDemo

```
OK    QUnitDemo/testBasicStuff/1
OK    QUnitDemo/testBasicStuff/2
OK    QUnitDemo/testBasicStuff/3
ERROR QUnitDemo/testBasicStuff/4
QUnitDemo.cpp:18: unittest failed: 4 == 7 (QUnitDemo/testBasicStuff/4)
OK    QUnitDemo/testBasicStuff/5
ERROR QUnitDemo/testStringStuff/1
QUnitDemo.cpp:26: unittest failed: str1 == str2 (QUnitDemo/testStringStuff/1)
OK    QUnitDemo/testStringStuff/2
QUnitDemo FAILED (7 tests, 2 errors)
```

QUnitDemo

```
// QUnitDemo.cpp

#include "QUnit.hpp"

#include <string>
using namespace std;

#define ASSERT_TRUE(cond) ( assertTrue(cond, #cond, __FILE__, __LINE__) )

class QUnitDemo : public QUnit {
public:
    QUnitDemo() : QUnit("QUnitDemo") {}
    void run() {
        setContext("testBasicStuff");
        testBasicStuff();

        setContext("testStringStuff");
        testStringStuff();
    }

    void testBasicStuff() {
        ASSERT_TRUE(1 == 1);
        ASSERT_TRUE(2 == 2);
        ASSERT_TRUE(3 == 3);
        ASSERT_TRUE(4 == 7);
        ASSERT_TRUE(5 == 5);
    }

    void testStringStuff() {
        string str1 = "a";
        string str2 = "b";
        string str3 = "ab";

        ASSERT_TRUE( str1 == str2 );
        ASSERT_TRUE( str1 + str2 == str3 );
    }
};

int main(int argc, char** argv) {
    QUnitDemo t;
    return t.execute(argc, argv);
}
```

QUnitDemo

```
// QUnitDemo.cpp

#include "QUnit.hpp"

#include <string>
using namespace std;

#define ASSERT_TRUE(cond) ( assertTrue(cond, #cond, __FILE__, __LINE__) )
#define RUNTEST(name) {setContext(#name); name();}

class QUnitDemo : public QUnit {
public:
    QUnitDemo() : QUnit("QUnitDemo") {}
    void run() {
        RUNTEST(testBasicStuff);
        RUNTEST(testStringStuff);
    }

    void testBasicStuff() {
        ASSERT_TRUE(1 == 1);
        ASSERT_TRUE(2 == 2);
        ASSERT_TRUE(3 == 3);
        ASSERT_TRUE(4 == 7);
        ASSERT_TRUE(5 == 5);
    }

    void testStringStuff() {
        string str1 = "a";
        string str2 = "b";
        string str3 = "ab";

        ASSERT_TRUE( str1 == str2 );
        ASSERT_TRUE( str1 + str2 == str3 );
    }
};

int main(int argc, char** argv) {
    QUnitDemo t;
    return t.execute(argc, argv);
}
```

QUnitDemo

```
g++ -o QUnitDemo QUnitDemo.cpp QUnit.cpp && QUnitDemo -v
OK    QUnitDemo/testBasicStuff/1
OK    QUnitDemo/testBasicStuff/2
OK    QUnitDemo/testBasicStuff/3
ERROR QUnitDemo/testBasicStuff/4
QUnitDemo.cpp:23: unittest failed: 4 == 7 (QUnitDemo/testBasicStuff/4)
OK    QUnitDemo/testBasicStuff/5
ERROR QUnitDemo/testStringStuff/1
QUnitDemo.cpp:32: unittest failed: str1 == str2 (QUnitDemo/testStringStuff/1)
OK    QUnitDemo/testStringStuff/2
QUnitDemo FAILED (7 tests, 2 errors)
```

QUnitDemo

```
g++ -o QUnitDemo QUnitDemo.cpp QUnit.cpp && QUnitDemo
QUnitDemo.cpp:23: unittest failed: 4 == 7 (QUnitDemo/testBasicStuff/4)
QUnitDemo.cpp:32: unittest failed: str1 == str2 (QUnitDemo/testStringStuff/1)
```

QUnitDemo

```
// QUnitDemo.cpp

#include "QUnit.hpp"

#include <string>
using namespace std;

#define ASSERT_TRUE(cond) ( assertTrue(cond, #cond, __FILE__, __LINE__) )
#define RUNTEST(name) {setContext(#name); name();}

class QUnitDemo : public QUnit {
public:
    QUnitDemo() : QUnit("QUnitDemo") {}
    void run() {
        RUNTEST(testBasicStuff);
        RUNTEST(testStringStuff);
    }

    void testBasicStuff() {
        ASSERT_TRUE(1 == 1);
        ASSERT_TRUE(2 == 2);
        ASSERT_TRUE(3 == 3);
        ASSERT_TRUE(4 != 7);
        ASSERT_TRUE(5 == 5);
    }

    void testStringStuff() {
        string str1 = "a";
        string str2 = "b";
        string str3 = "ab";

        ASSERT_TRUE( str1 != str2 );
        ASSERT_TRUE( str1 + str2 == str3 );
    }
};

int main(int argc, char** argv) {
    QUnitDemo t;
    return t.execute(argc, argv);
}
```

QUnitDemo

```
g++ -o QUnitDemo QUnitDemo.cpp QUnit.cpp && QUnitDemo
```

QUnitDemo

```
g++ -o QUnitDemo QUnitDemo.cpp QUnit.cpp && QUnitDemo && echo Success
Success
```

- Brief introduction to Test-Driven Development
- QUnit - A simple framework for unit testing in C++
- **The Bowling Game Kata in C++**
- More TDD examples
- Q&A


3
Brunswick

Bowling Game Kata in C++

The following is a demonstration of how to do test-driven development using the our new framework for unit testing in C++. We are going to write some code for scoring a game of bowling.

Since the seminal article "Engineer Notebook: An Extreme Programming Episode" published in 2001 by Robert C. Martin and Robert S. Koss:

- http://www.objectmentor.com/resources/articles/xpepisode.htm

calculating the score for a bowling game has gained status as an advanced "Hello World" for programming languages. For any programming language out there you will find a bowling score implementation insipred by the "XP Episode". There is also a lot of derivative work from this article, some of them demonstrating how design evolves through Test-Driven Development.

What you will see now is taken more or less directly out of the excellent “Bowling Game Kata” presentation by Robert C. Martin.

- http://butunclebob.com
- http://butunclebob.com/ArticleS.UncleBob.TheBowlingGameKata
- http://butunclebob.com/files/downloads/Bowling%20Game%20Kata.ppt

Basically the only thing I have done is to translate from Java/JUnit into C++/QUnit.

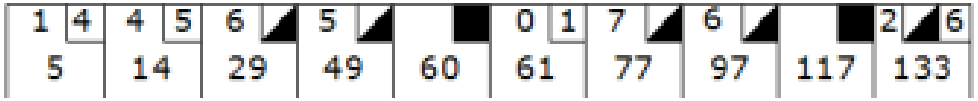
Since Uncle Bob is a nice guy...

... we include this page, because he asked us to do so:

The following slides are not verbatim copies, but they are close enough to deserve a proper copyright notice...

Some of the material is probably Copyright (C) 2005 by Object Mentor. Permission to use was given by Uncle Bob.

Scoring Bowling



The game consists of 10 frames as shown above. In each frame the player has two opportunities to knock down 10 pins. The score for the frame is the total number of pins knocked down, plus bonuses for strikes and spares.

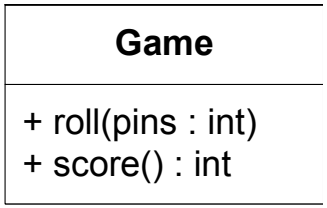
A spare is when the player knocks down all 10 pins in two tries. The bonus for that frame is the number of pins knocked down by the next roll. So in frame 3 above, the score is 10 (the total number knocked down) plus a bonus of 5 (the number of pins knocked down on the next roll.)

A strike is when the player knocks down all 10 pins on his first try. The bonus for that frame is the value of the next two balls rolled.

In the tenth frame a player who rolls a spare or strike is allowed to roll the extra balls to complete the frame. However no more than three balls can be rolled in tenth frame.

[source: Uncle Bob]

The Requirements.

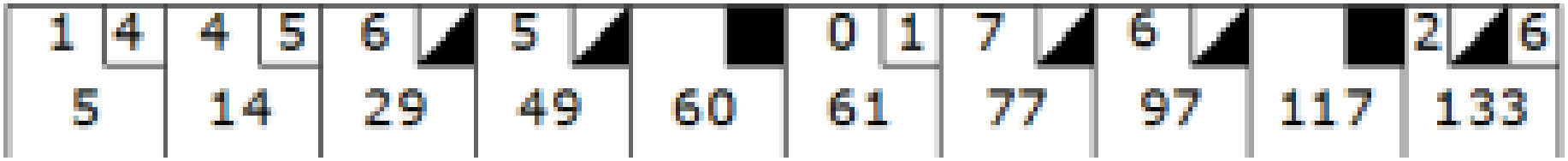


Write a class named “Game” that has two methods:

- roll(pins : int) is called each time the player rolls a ball. The argument is the number of pins knocked down.
- score() : int is called only at the very end of the game. It returns the total score for that game.

[source: Uncle Bob]

Scoring Bowling & The Requirements



The **game** consists of 10 **frames** as shown above. In each **frame** the **player** has two opportunities to knock down 10 **pins**. The **score** for the **frame** is the total number of **pins** knocked down, plus **bonuses** for **strikes** and **spares**.

A **spare** is when the **player** knocks down all 10 **pins** in **two tries**. The **bonus** for that **frame** is the number of **pins** knocked down by the next **roll**. So in **frame** 3 above, the **score** is 10 (the total number knocked down) plus a **bonus** of 5 (the number of pins knocked down on the next roll.)

A **strike** is when the **player** knocks down all 10 **pins** on his first try. The **bonus** for that **frame** is the value of the next two **balls rolled**.

In the **tenth frame** a **player** who **rolls** a **spare** or **strike** is allowed to **roll** the **extra balls** to **complete the frame**. However no more than three **balls** can be rolled in **tenth frame**.

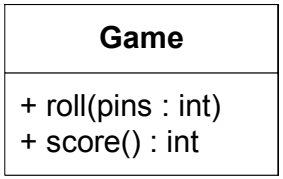
Game
+ roll(pins : int) + score() : int

Write a class named “Game” that has two methods:

- roll(pins : int) is called each time the player rolls a ball. The argument is the number of pins knocked down.
- score() : int is called only at the very end of the game. It returns the total score for that game.

[source: Uncle Bob]

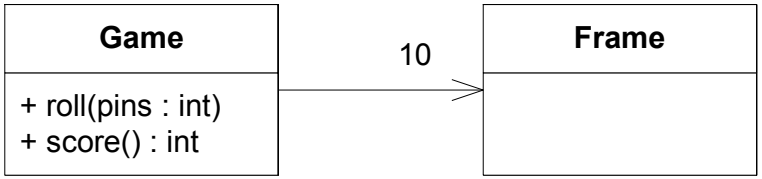
A quick design session



Clearly we need the Game class.

[source: Uncle Bob]

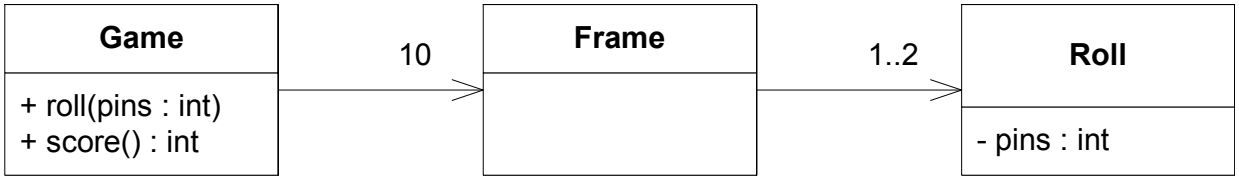
A quick design session



A game has 10 frames.

[source: Uncle Bob]

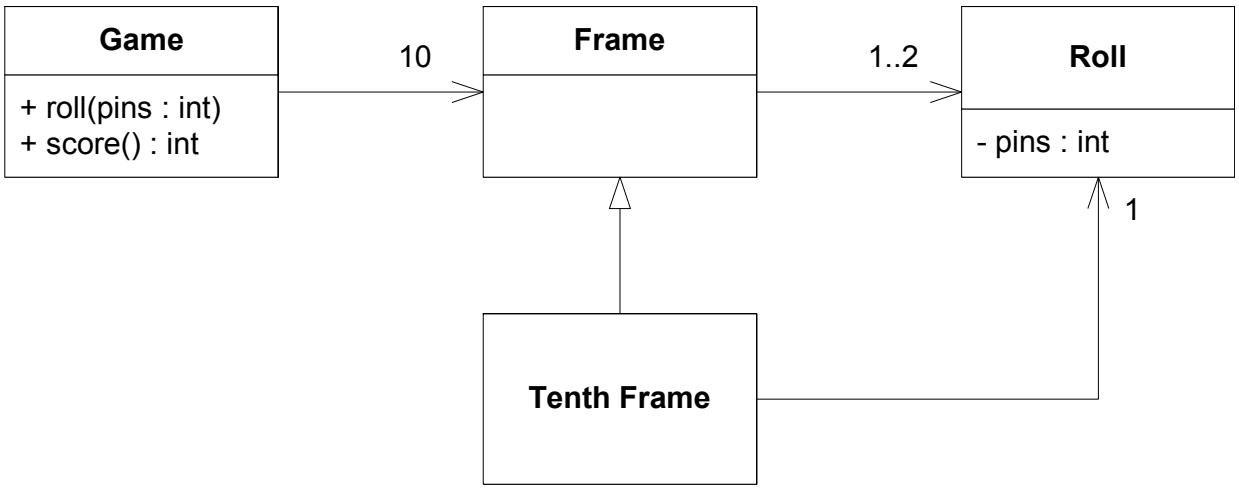
A quick design session



A frame has 1 or two rolls.

[source: Uncle Bob]

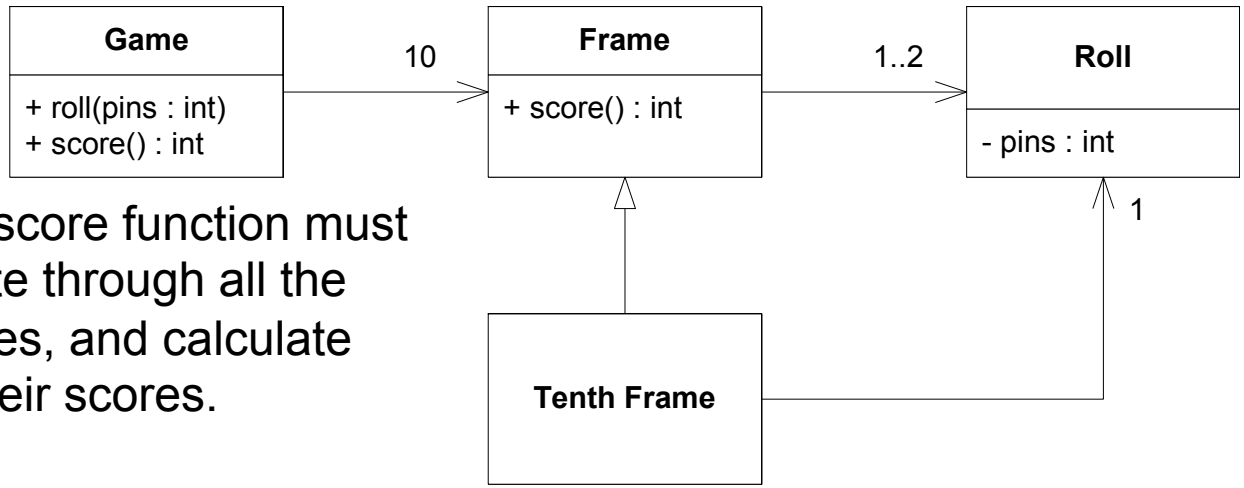
A quick design session



The tenth frame has two or three rolls.
It is different from all the other frames.

[source: Uncle Bob]

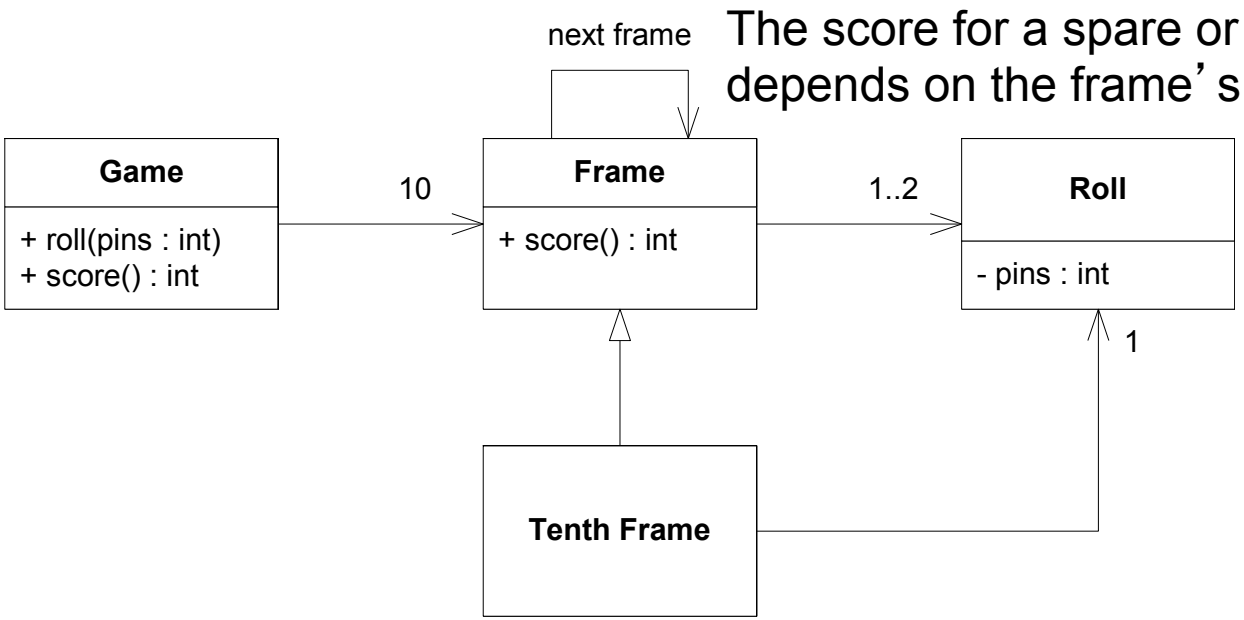
A quick design session



The score function must iterate through all the frames, and calculate all their scores.

[source: Uncle Bob]

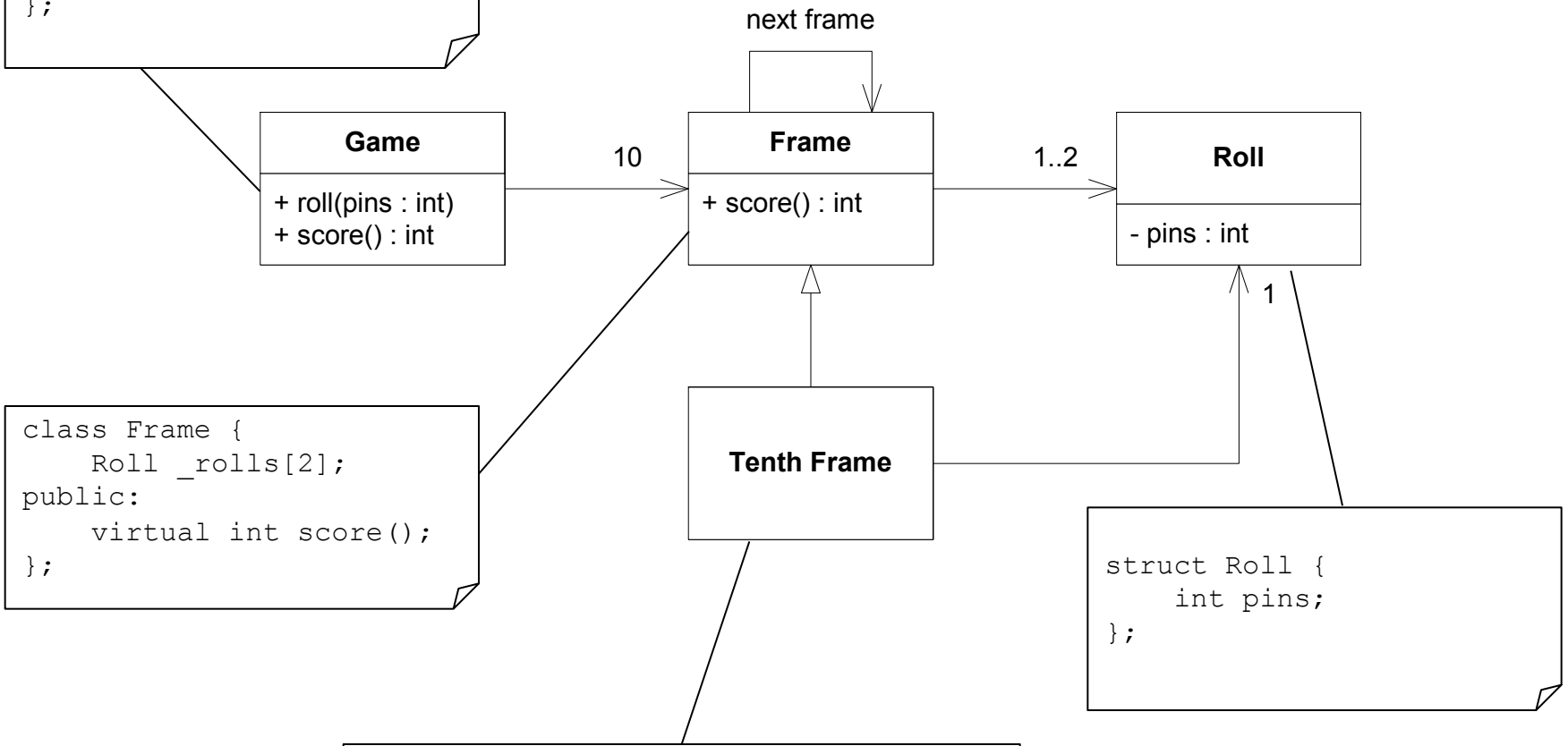
A quick design session



[source: Uncle Bob]

A quick design session

```
class Game {
    Frame _frames[10];
public:
    void roll(int pins);
    int score();
};
```



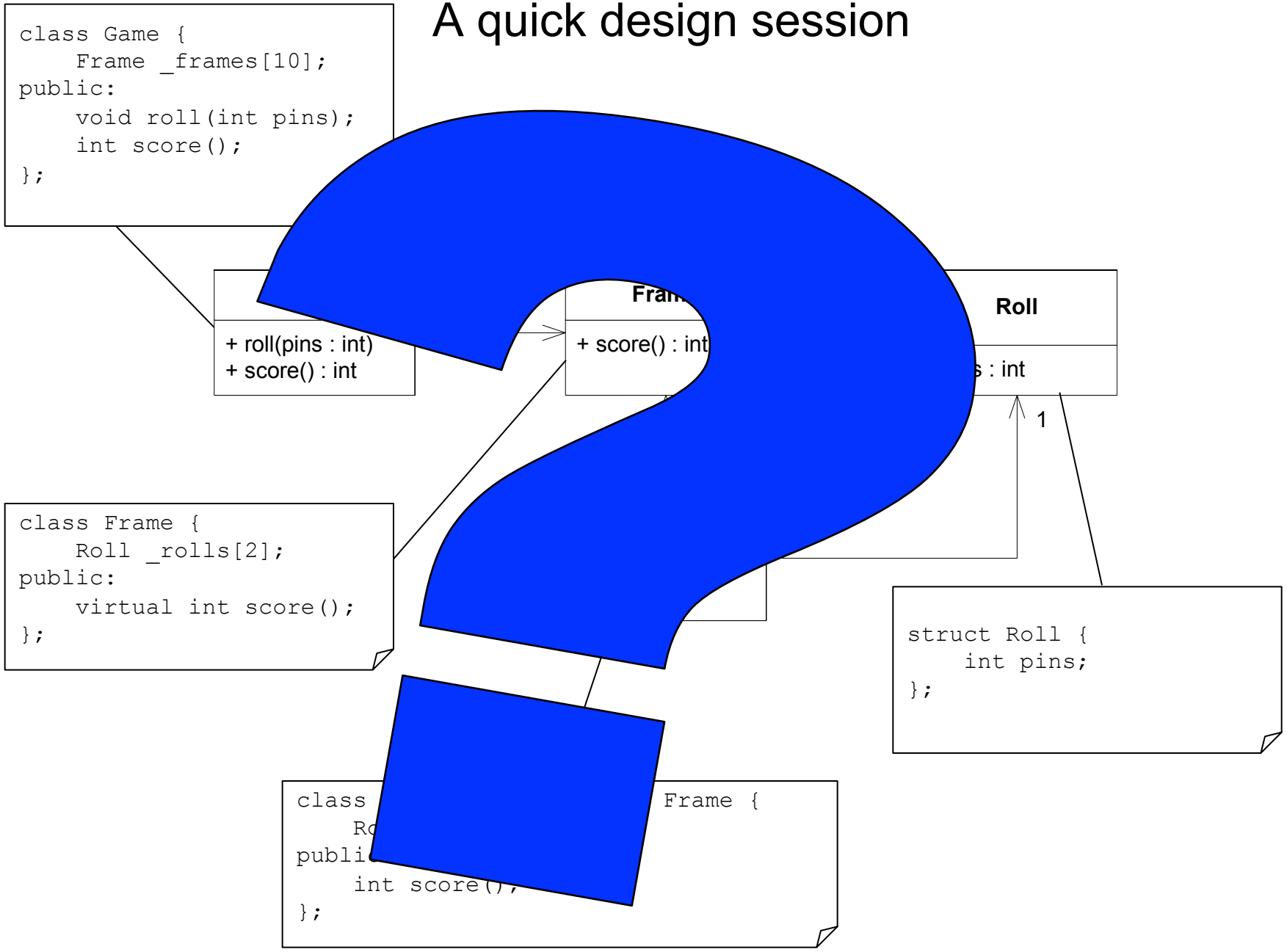

```
class Frame {
    Roll _rolls[2];
public:
    virtual int score();
};
```

```
struct Roll {
    int pins;
};
```

```
class TenthFrame : public Frame {
    Roll _extraroll;
public:
    int score();
};
```

A quick design session

```
class Game {
    Frame _frames[10];
public:
    void roll(int pins);
    int score();
};
```




```
class Frame {
    Roll _rolls[2];
public:
    virtual int score();
};
```

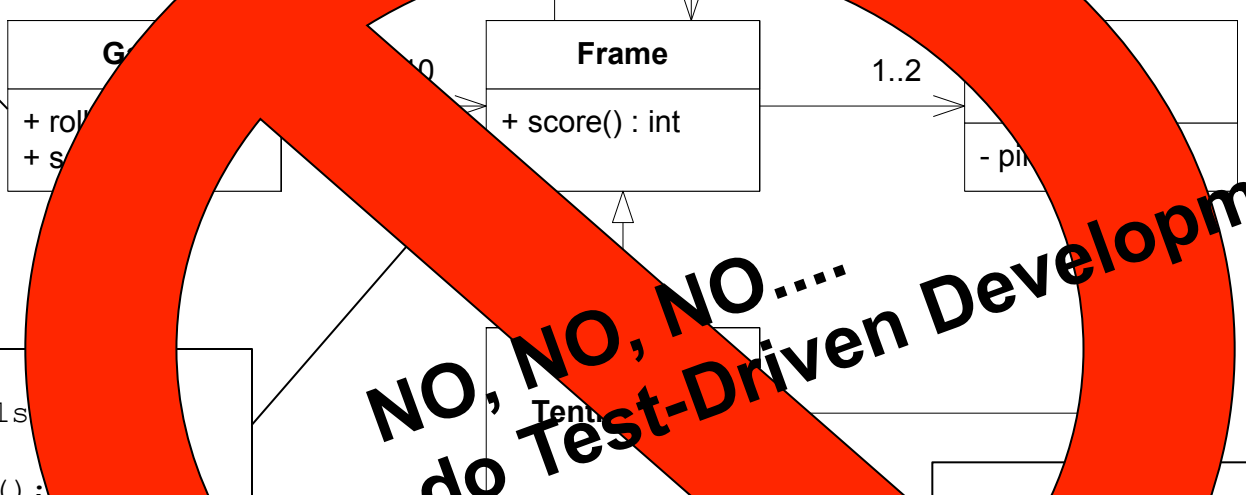
```
struct Roll {
    int pins;
};
```

```
class
    Ro
publi
    int score();
};
```

Frame {

A quick design session

```
class Game {
    Frame _frames[10];
public:
    void roll(int pins);
    int score();
};
```




```
class Frame {
    Roll _rolls
public:
    int score();
};
```

```
class
    Roll _
public:
    int score();
};
```

NO, NO, NO....
we are supposed to do Test-Driven Development now

Traditional OOAD vs Test-Driven Development

Traditional OOAD vs Test-Driven Development

Traditional upfront OOAD often comes up with far too complex solutions, and it is quite common to end up in a state of “design paralysis”.

Traditional OOAD vs Test-Driven Development

Traditional upfront OOAD often comes up with far too complex solutions, and it is quite common to end up in a state of “design paralysis”.

Test-Driven Development is a design technique, perhaps more so than a programming technique. You will often see that TDD leads to a very different design in the end compared to what you often get with traditional OOAD.

Begin with Test-Driven Development

- Create a new project named BowlingGame
- Create an empty BowlingGame class
- Create a unit test named BowlingGameTest
- Set up a unit test environment

```
mkdir BowlingGame
cd BowlingGame
cp ../QUnit.cpp ../QUnit.hpp .
ed BowlingGame.hpp
ed BowlingGameTest
ed Makefile
make
```

The empty class (BowlingGame.hpp)

```
// BowlingGame.hpp - a bowling score calculator

class BowlingGame {
};
```

The empty test (BowlingGameTest.cpp)

```
// BowlingGameTest.cpp - Bowling Game Kata in C++ / QUnit

#include "QUnit.hpp"

#include "BowlingGame.hpp"

#include <iostream>

#define ASSERT_TRUE(cond) ( assertTrue(cond, #cond, __FILE__, __LINE__) )
#define RUNTEST(name) {setContext(#name); name();}

class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        BowlingGame g;
    }
};

int main(int argc, char ** argv) {
    BowlingGameTest t;
    return t.execute(argc, argv);
}
```

Set up a unit test environment (Makefile)

```
# Makefile

BowlingGameTest : BowlingGameTest.cpp BowlingGame.hpp QUnit.o
          g++ -o BowlingGameTest BowlingGameTest.cpp QUnit.o
          BowlingGameTest -v

QUnit.o : QUnit.cpp QUnit.hpp
          g++ -c QUnit.cpp

clean:
          rm BowlingGameTest QUnit.o
```

Verify the set up

```
g++ -c QUnit.cpp
g++ -o BowlingGameTest BowlingGameTest.cpp QUnit.o
BowlingGameTest -v
BowlingGameTest OK  (0 tests, 0 errors)
```

The first test.

The first test.

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest():
        QUnit("BowlingGameTest") {};

    void run() {
        BowlingGame g;
    }
};
```

```
class BowlingGame {
};
```

The first test.

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest():
        QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
    }

    void testGutterGame() {
        BowlingGame g;
    }
};
```

```
class BowlingGame {
};
```

The first test.

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest():
        QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
    }

    void testGutterGame() {
        BowlingGame g;
    }
};
```

```
class BowlingGame {
};
```

```
BowlingGameTest OK  (0 tests, 0 errors)
```

The first test.

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest():
        QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
    }

    void testGutterGame() {
        BowlingGame g;
        for (int i=0; i<20; ++i) {
            g.roll(0);
        }
    }
};
```

```
class BowlingGame {
};
```

The first test.

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest():
        QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
    }

    void testGutterGame() {
        BowlingGame g;
        for (int i=0; i<20; ++i) {
            g.roll(0);
        }
    }
};
```

```
class BowlingGame {
public:
    void roll(int pins) {}
};
```

The first test.

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest():
        QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
    }

    void testGutterGame() {
        BowlingGame g;
        for (int i=0; i<20; ++i) {
            g.roll(0);
        }
    }
};
```

```
class BowlingGame {
public:
    void roll(int pins) {}
};
```

```
BowlingGameTest OK  (0 tests, 0 errors)
```

The first test.

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest():
        QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
    }

    void testGutterGame() {
        BowlingGame g;
        for (int i=0; i<20; ++i) {
            g.roll(0);
        }
        ASSERT_TRUE(g.score() == 0);
    }
};
```

```
class BowlingGame {
public:
    void roll(int pins) {}
};
```

The first test.

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest():
        QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
    }

    void testGutterGame() {
        BowlingGame g;
        for (int i=0; i<20; ++i) {
            g.roll(0);
        }
        ASSERT_TRUE(g.score() == 0);
    }
};
```

```
class BowlingGame {
    int _score;
public:
    void roll(int pins) {}
    int score() { return _score; }
};
```

The first test.

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest():
        QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
    }

    void testGutterGame() {
        BowlingGame g;
        for (int i=0; i<20; ++i) {
            g.roll(0);
        }
        ASSERT_TRUE(g.score() == 0);
    }
};
```

```
class BowlingGame {
    int _score;
public:
    void roll(int pins) {}
    int score() { return _score; }
};
```

```
ERROR BowlingGameTest/testGutterGame/1
BowlingGameTest.cpp:23: unittest failed: g.score() == 0 (BowlingGameTest/testGutterGame/1)
BowlingGameTest FAILED (1 tests, 1 errors)
```

The first test.

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest():
        QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
    }

    void testGutterGame() {
        BowlingGame g;
        for (int i=0; i<20; ++i) {
            g.roll(0);
        }
        ASSERT_TRUE(g.score() == 0);
    }
};
```

```
class BowlingGame {
    int _score;
public:
    BowlingGame() : _score(0) {}
    void roll(int pins) {}
    int score() { return _score; }
};
```

The first test.

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest():
        QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
    }

    void testGutterGame() {
        BowlingGame g;
        for (int i=0; i<20; ++i) {
            g.roll(0);
        }
        ASSERT_TRUE(g.score() == 0);
    }
};
```

```
class BowlingGame {
    int _score;
public:
    BowlingGame() : _score(0) {}
    void roll(int pins) {}
    int score() { return _score; }
};
```

```
BowlingGameTest -v
OK    BowlingGameTest/testGutterGame/1
BowlingGameTest OK  (1 tests, 0 errors)
```

The first test.

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest():
        QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
    }

    void testGutterGame() {
        BowlingGame g;
        for (int i=0; i<20; ++i) {
            g.roll(0);
        }
        ASSERT_TRUE(g.score() == 0);
    }
};
```

```
class BowlingGame {
    int _score;
public:
    BowlingGame() : _score(0) {}
    void roll(int pins) {}
    int score() { return _score; }
};
```

The Second Test

The Second Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
    }

    void testGutterGame() {
        BowlingGame g;
        for (int i=0; i<20; ++i) {
            g.roll(0);
        }
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        for (int i=0; i<20; ++i) {
            g.roll(1);
        }
        ASSERT_TRUE(g.score() == 20);
    }

};
```

```
class BowlingGame {
    int _score;
public:
    BowlingGame() : _score(0) {}
    void roll(int pins) {}
    int score() { return _score; }
};
```

The Second Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
    }

    void testGutterGame() {
        BowlingGame g;
        for (int i=0; i<20; ++i) {
            g.roll(0);
        }
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        for (int i=0; i<20; ++i) {
            g.roll(1);
        }
        ASSERT_TRUE(g.score() == 20);
    }

};
```

```
class BowlingGame {
    int _score;
public:
    BowlingGame() : _score(0) {}
    void roll(int pins) {}
    int score() { return _score; }
};
```

```
OK    BowlingGameTest/testGutterGame/1
ERROR BowlingGameTest/testAllOnes/1
BowlingGameTest.cpp:32: unittest failed: g.score() == 20 (BowlingGameTest/testAllOnes/1)
BowlingGameTest FAILED (2 tests, 1 errors)
```

The Second Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
    }

    void testGutterGame() {
        BowlingGame g;
        for (int i=0; i<20; ++i) {
            g.roll(0);
        }
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        for (int i=0; i<20; ++i) {
            g.roll(1);
        }
        ASSERT_TRUE(g.score() == 20);
    }

};
```

```
class BowlingGame {
    int _score;
public:
    BowlingGame() : _score(0) {}
    void roll(int pins) {
        _score += pins;
    }
    int score() { return _score; }
};
```

The Second Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
    }

    void testGutterGame() {
        BowlingGame g;
        for (int i=0; i<20; ++i) {
            g.roll(0);
        }
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        for (int i=0; i<20; ++i) {
            g.roll(1);
        }
        ASSERT_TRUE(g.score() == 20);
    }

};
```

```
class BowlingGame {
    int _score;
public:
    BowlingGame() : _score(0) {}
    void roll(int pins) {
        _score += pins;
    }
    int score() { return _score; }
};
```

```
BowlingGameTest -v
OK    BowlingGameTest/testGutterGame/1
OK    BowlingGameTest/testAllOnes/1
BowlingGameTest OK (2 tests, 0 errors)
```

The Second Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
    }

    void testGutterGame() {
        BowlingGame g;
        for (int i=0; i<20; ++i) {
            g.roll(0);
        }
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        for (int i=0; i<20; ++i) {
            g.roll(1);
        }
        ASSERT_TRUE(g.score() == 20);
    }

};
```

```
class BowlingGame {
    int _score;
public:
    BowlingGame() : _score(0) {}
    void roll(int pins) {
        _score += pins;
    }
    int score() { return _score; }
};
```

The Second Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
    }

    void testGutterGame() {
        BowlingGame g;
        for (int i=0; i<20; ++i) {
            g.roll(0);
        }
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        for (int i=0; i<20; ++i) {
            g.roll(1);
        }
        ASSERT_TRUE(g.score() == 20);
    }

};
```

```
class BowlingGame {
    int _score;
public:
    BowlingGame() : _score(0) {}
    void roll(int pins) {
        _score += pins;
    }
    int score() { return _score; }
};
```

Perhaps we need refactoring of test code?

The Second Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

};
```

```
class BowlingGame {
    int _score;
public:
    BowlingGame() : _score(0) {}
    void roll(int pins) {
        _score += pins;
    }
    int score() { return _score; }
};
```

The Second Test

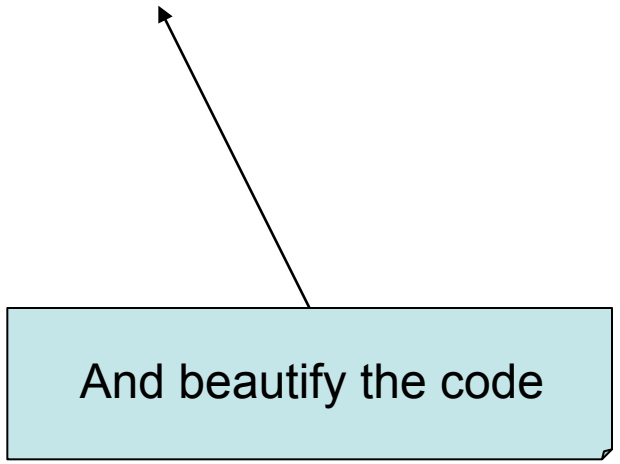
```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

};
```

```
class BowlingGame {
    int _score;
public:
    BowlingGame() : _score(0) {}
    void roll(int pins) {
        _score += pins;
    }
    int score() { return _score; }
};
```

The Second Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

};
```

```
class BowlingGame {
    int _score;
public:
    BowlingGame() : _score(0) {}
    void roll(int pins) {
        _score += pins;
    }
    int score() {
        return _score;
    }
};
```



The Second Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

};
```

```
class BowlingGame {
    int _score;
public:
    BowlingGame() : _score(0) {}
    void roll(int pins) {
        _score += pins;
    }
    int score() {
        return _score;
    }
};
```

The Third Test

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }
};
```

```
class BowlingGame {
    int _score;
public:
    BowlingGame() : _score(0) {}
    void roll(int pins) {
        _score += pins;
    }
    int score() {
        return _score;
    }
};
```

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }
};
```

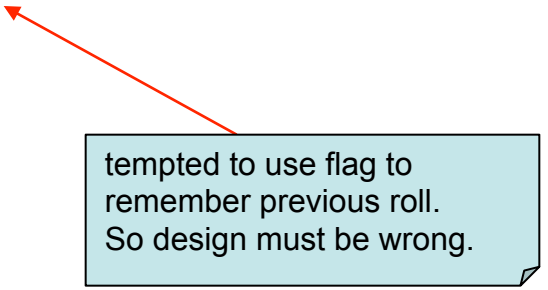
```
class BowlingGame {
    int _score;
public:
    BowlingGame() : _score(0) {}
    void roll(int pins) {
        _score += pins;
    }
    int score() {
        return _score;
    }
};
```

```
BowlingGameTest.cpp:44: unittest failed: g.score() == 16 (BowlingGameTest/testOneSpare/1)
BowlingGameTest FAILED (3 tests, 1 errors)
```

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }
};
```

```
class BowlingGame {
    int _score;
public:
    BowlingGame() : _score(0) {}
    void roll(int pins) {
        _score += pins;
    }
    int score() {
        return _score;
    }
};
```



The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }
};
```

```
class BowlingGame {
    int _score;
public:
    BowlingGame() : _score(0) {}
    void roll(int pins) {
        _score += pins;
    }
    int score() {
        return _score;
    }
};
```

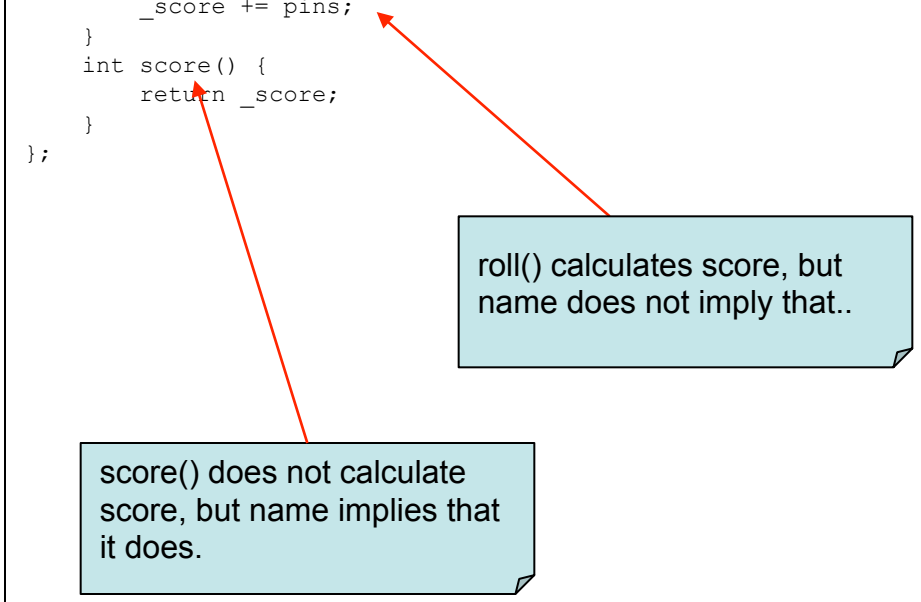
roll() calculates score, but name does not imply that..

score() does not calculate score, but name implies that it does.

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }
};
```

```
class BowlingGame {
    int _score;
public:
    BowlingGame() : _score(0) {}
    void roll(int pins) {
        _score += pins;
    }
    int score() {
        return _score;
    }
};
```



Design is wrong. Responsibilities are misplaced.

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        // RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    // void testOneSpare() {
    //     BowlingGame g;
    //     g.roll(5);
    //     g.roll(5); // spare
    //     g.roll(3);
    //     rollMany(g, 17, 0);
    //     ASSERT_TRUE(g.score() == 16);
    // }
};
```

```
class BowlingGame {
    int _score;
public:
    BowlingGame() : _score(0) {}
    void roll(int pins) {
        _score += pins;
    }
    int score() {
        return _score;
    }
};
```

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        // RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    // void testOneSpare() {
    //     BowlingGame g;
    //     g.roll(5);
    //     g.roll(5); // spare
    //     g.roll(3);
    //     rollMany(g, 17, 0);
    //     ASSERT_TRUE(g.score() == 16);
    // }
};
```

```
class BowlingGame {
    int _score;
public:
    BowlingGame() : _score(0) {}
    void roll(int pins) {
        _score += pins;
    }
    int score() {
        return _score;
    }
};
```

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        // RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    // void testOneSpare() {
    //     BowlingGame g;
    //     g.roll(5);
    //     g.roll(5); // spare
    //     g.roll(3);
    //     rollMany(g, 17, 0);
    //     ASSERT_TRUE(g.score() == 16);
    // }
};
```

```
class BowlingGame {
    int _score;
    int _rolls[21];
    int _currentRoll;
public:
    BowlingGame() : _score(0) , _currentRoll(0) {}
    void roll(int pins) {
        _score += pins;
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        return _score;
    }
};
```

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        // RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    // void testOneSpare() {
    //     BowlingGame g;
    //     g.roll(5);
    //     g.roll(5); // spare
    //     g.roll(3);
    //     rollMany(g, 17, 0);
    //     ASSERT_TRUE(g.score() == 16);
    // }
};
```

```
class BowlingGame {
    int _score;
    int _rolls[21];
    int _currentRoll;
public:
    BowlingGame() : _score(0) , _currentRoll(0) {}
    void roll(int pins) {
        _score += pins;
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        for (int i=0; i<21; ++i) {
            score += _rolls[i];
        }
        return score;
    }
};
```

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        // RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    // void testOneSpare() {
    //     BowlingGame g;
    //     g.roll(5);
    //     g.roll(5); // spare
    //     g.roll(3);
    //     rollMany(g, 17, 0);
    //     ASSERT_TRUE(g.score() == 16);
    // }
};
```

```
class BowlingGame {
    int _score;
    int _rolls[21];
    int _currentRoll;
public:
    BowlingGame() : _score(0) , _currentRoll(0) {}
    void roll(int pins) {
        _score += pins;
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        for (int i=0; i<21; ++i) {
            score += _rolls[i];
        }
        return score;
    }
};
```

will this pass?

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        // RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    // void testOneSpare() {
    //     BowlingGame g;
    //     g.roll(5);
    //     g.roll(5); // spare
    //     g.roll(3);
    //     rollMany(g, 17, 0);
    //     ASSERT_TRUE(g.score() == 16);
    // }
};
```

```
class BowlingGame {
    int _score;
    int _rolls[21];
    int _currentRoll;
public:
    BowlingGame() : _score(0) , _currentRoll(0) {}
    void roll(int pins) {
        _score += pins;
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        for (int i=0; i<21; ++i) {
            score += _rolls[i];
        }
        return score;
    }
};
```

yes

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        // RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    // void testOneSpare() {
    //     BowlingGame g;
    //     g.roll(5);
    //     g.roll(5); // spare
    //     g.roll(3);
    //     rollMany(g, 17, 0);
    //     ASSERT_TRUE(g.score() == 16);
    // }
};
```

```
class BowlingGame {
    int _score;
    int _rolls[21];
    int _currentRoll;
public:
    BowlingGame() : _score(0) , _currentRoll(0) {}
    void roll(int pins) {
        _score += pins;
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        for (int i=0; i<21; ++i) {
            score += _rolls[i];
        }
        return score;
    }
};
```

sometimes :-{

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        // RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    // void testOneSpare() {
    //     BowlingGame g;
    //     g.roll(5);
    //     g.roll(5); // spare
    //     g.roll(3);
    //     rollMany(g, 17, 0);
    //     ASSERT_TRUE(g.score() == 16);
    // }
};
```

```
class BowlingGame {
    const static int MAX_ROLLS = 21;
    int _score;
    int _rolls[MAX_ROLLS];
    int _currentRoll;
public:
    BowlingGame() : _score(0) , _currentRoll(0) {
        for (int i=0; i<MAX_ROLLS; ++i) {
            _rolls[i] = 0;
        }
    }
    void roll(int pins) {
        _score += pins;
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        for (int i=0; i<MAX_ROLLS; ++i) {
            score += _rolls[i];
        }
        return score;
    }
};
```

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        // RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    // void testOneSpare() {
    //     BowlingGame g;
    //     g.roll(5);
    //     g.roll(5); // spare
    //     g.roll(3);
    //     rollMany(g, 17, 0);
    //     ASSERT_TRUE(g.score() == 16);
    // }
};
```

```
class BowlingGame {
    const static int MAX_ROLLS = 21;
    int _score;
    int _rolls[MAX_ROLLS];
    int _currentRoll;
public:
    BowlingGame() : _score(0) , _currentRoll(0) {
        for (int i=0; i<MAX_ROLLS; ++i) {
            _rolls[i] = 0;
        }
    }
    void roll(int pins) {
        _score += pins;
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        for (int i=0; i<MAX_ROLLS; ++i) {
            score += _rolls[i];
        }
        return score;
    }
};
```

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        // RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    // void testOneSpare() {
    //     BowlingGame g;
    //     g.roll(5);
    //     g.roll(5); // spare
    //     g.roll(3);
    //     rollMany(g, 17, 0);
    //     ASSERT_TRUE(g.score() == 16);
    // }
};
```

```
class BowlingGame {
    const static int MAX_ROLLS = 21;
    int _score;
    int _rolls[MAX_ROLLS];
    int _currentRoll;
public:
    BowlingGame() : _score(0) , _currentRoll(0) {
        for (int i=0; i<MAX_ROLLS; ++i) {
            _rolls[i] = 0;
        }
    }
    void roll(int pins) {
        _score += pins;
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        for (int i=0; i<MAX_ROLLS; ++i) {
            score += _rolls[i];
        }
        return score;
    }
};
```

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        // RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    // void testOneSpare() {
    //     BowlingGame g;
    //     g.roll(5);
    //     g.roll(5); // spare
    //     g.roll(3);
    //     rollMany(g, 17, 0);
    //     ASSERT_TRUE(g.score() == 16);
    // }
};
```

```
class BowlingGame {
    const static int MAX_ROLLS = 21;
    int _rolls[MAX_ROLLS];
    int _currentRoll;
public:
    BowlingGame() : _currentRoll(0) {
        for (int i=0; i<MAX_ROLLS; ++i) {
            _rolls[i] = 0;
        }
    }
    void roll(int pins) {
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        for (int i=0; i<MAX_ROLLS; ++i) {
            score += _rolls[i];
        }
        return score;
    }
};
```

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }
};
```

```
class BowlingGame {
    const static int MAX_ROLLS = 21;
    int _rolls[MAX_ROLLS];
    int _currentRoll;
public:
    BowlingGame() : _currentRoll(0) {
        for (int i=0; i<MAX_ROLLS; ++i) {
            _rolls[i] = 0;
        }
    }
    void roll(int pins) {
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        for (int i=0; i<MAX_ROLLS; ++i) {
            score += _rolls[i];
        }
        return score;
    }
};
```

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }
};
```

```
class BowlingGame {
    const static int MAX_ROLLS = 21;
    int _rolls[MAX_ROLLS];
    int _currentRoll;
public:
    BowlingGame() : _currentRoll(0) {
        for (int i=0; i<MAX_ROLLS; ++i) {
            _rolls[i] = 0;
        }
    }
    void roll(int pins) {
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        for (int i=0; i<MAX_ROLLS; ++i) {
            score += _rolls[i];
        }
        return score;
    }
};
```

```
BowlingGameTest.cpp:44: unittest failed: g.score() == 16 (BowlingGameTest/testOneSpare/1)
BowlingGameTest FAILED (3 tests, 1 errors)
```

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }
};
```

```
class BowlingGame {
    const static int MAX_ROLLS = 21;
    int _rolls[MAX_ROLLS];
    int _currentRoll;
public:
    BowlingGame() : _currentRoll(0) {
        for (int i=0; i<MAX_ROLLS; ++i) {
            _rolls[i] = 0;
        }
    }
    void roll(int pins) {
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        for (int i=0; i<MAX_ROLLS; ++i) {
            if (rolls[i] + rolls[i+1] == 10) { // spare
                score += ...
            score += _rolls[i];
        }
        return score;
    }
};
```

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }
};
```

```
class BowlingGame {
    const static int MAX_ROLLS = 21;
    int _rolls[MAX_ROLLS];
    int _currentRoll;
public:
    BowlingGame() : _currentRoll(0) {
        for (int i=0; i<MAX_ROLLS; ++i) {
            _rolls[i] = 0;
        }
    }
    void roll(int pins) {
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        for (int i=0; i<MAX_ROLLS; ++i) {
            if (rolls[i] + rolls[i+1] == 10) { // spare
                score += ...
            score += _rolls[i];
        }
        return score;
    }
};
```

This isn't going to work because i might not refer to the first ball of the frame.

Design is still wrong.

Need to walk through array two balls (one frame) at a time.

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        // RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    // void testOneSpare() {
    //     BowlingGame g;
    //     g.roll(5);
    //     g.roll(5); // spare
    //     g.roll(3);
    //     rollMany(g, 17, 0);
    //     ASSERT_TRUE(g.score() == 16);
    // }
};
```

```
class BowlingGame {
    const static int MAX_ROLLS = 21;
    int _rolls[MAX_ROLLS];
    int _currentRoll;
public:
    BowlingGame() : _currentRoll(0) {
        for (int i=0; i<MAX_ROLLS; ++i) {
            _rolls[i] = 0;
        }
    }
    void roll(int pins) {
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        for (int i=0; i<MAX_ROLLS; ++i) {
            score += _rolls[i];
        }
        return score;
    }
};
```

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        // RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    // void testOneSpare() {
    //     BowlingGame g;
    //     g.roll(5);
    //     g.roll(5); // spare
    //     g.roll(3);
    //     rollMany(g, 17, 0);
    //     ASSERT_TRUE(g.score() == 16);
    // }
};
```

```
class BowlingGame {
    const static int MAX_ROLLS = 21;
    int _rolls[MAX_ROLLS];
    int _currentRoll;
public:
    BowlingGame() : _currentRoll(0) {
        for (int i=0; i<MAX_ROLLS; ++i) {
            _rolls[i] = 0;
        }
    }
    void roll(int pins) {
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        int i = 0;
        for (int frame=0; frame<10; ++frame) {
            score += _rolls[i] + _rolls[i+1];
            i += 2;
        }
        return score;
    }
};
```

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        // RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    // void testOneSpare() {
    //     BowlingGame g;
    //     g.roll(5);
    //     g.roll(5); // spare
    //     g.roll(3);
    //     rollMany(g, 17, 0);
    //     ASSERT_TRUE(g.score() == 16);
    // }
};
```

```
class BowlingGame {
    const static int MAX_ROLLS = 21;
    int _rolls[MAX_ROLLS];
    int _currentRoll;
public:
    BowlingGame() : _currentRoll(0) {
        for (int i=0; i<MAX_ROLLS; ++i) {
            _rolls[i] = 0;
        }
    }
    void roll(int pins) {
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        int i = 0;
        for (int frame=0; frame<10; ++frame) {
            score += _rolls[i] + _rolls[i+1];
            i += 2;
        }
        return score;
    }
};
```

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }
};
```

```
class BowlingGame {
    const static int MAX_ROLLS = 21;
    int _rolls[MAX_ROLLS];
    int _currentRoll;
public:
    BowlingGame() : _currentRoll(0) {
        for (int i=0; i<MAX_ROLLS; ++i) {
            _rolls[i] = 0;
        }
    }
    void roll(int pins) {
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        int i = 0;
        for (int frame=0; frame<10; ++frame) {
            score += _rolls[i] + _rolls[i+1];
            i += 2;
        }
        return score;
    }
};
```

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }
};
```

```
class BowlingGame {
    const static int MAX_ROLLS = 21;
    int _rolls[MAX_ROLLS];
    int _currentRoll;
public:
    BowlingGame() : _currentRoll(0) {
        for (int i=0; i<MAX_ROLLS; ++i) {
            _rolls[i] = 0;
        }
    }
    void roll(int pins) {
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        int i = 0;
        for (int frame=0; frame<10; ++frame) {
            score += _rolls[i] + _rolls[i+1];
            i += 2;
        }
        return score;
    }
};
```

```
BowlingGameTest.cpp:44: unittest failed: g.score() == 16 (BowlingGameTest/testOneSpare/1)
BowlingGameTest FAILED (3 tests, 1 errors)
```

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }
};
```

```
class BowlingGame {
    const static int MAX_ROLLS = 21;
    int _rolls[MAX_ROLLS];
    int _currentRoll;
public:
    BowlingGame() : _currentRoll(0) {
        for (int i=0; i<MAX_ROLLS; ++i) {
            _rolls[i] = 0;
        }
    }
    void roll(int pins) {
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        int i = 0;
        for (int frame=0; frame<10; ++frame) {
            if ( _rolls[i] + _rolls[i+1] == 10 ) { // spare
                score += 10 + _rolls[i+2];
                i += 2;
            } else {
                score += _rolls[i] + _rolls[i+1];
                i += 2;
            }
        }
        return score;
    }
};
```

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }
};
```

```
class BowlingGame {
    const static int MAX_ROLLS = 21;
    int _rolls[MAX_ROLLS];
    int _currentRoll;
public:
    BowlingGame() : _currentRoll(0) {
        for (int i=0; i<MAX_ROLLS; ++i) {
            _rolls[i] = 0;
        }
    }
    void roll(int pins) {
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        int i = 0;
        for (int frame=0; frame<10; ++frame) {
            if ( _rolls[i] + _rolls[i+1] == 10 ) { // spare
                score += 10 + _rolls[i+2];
                i += 2;
            } else {
                score += _rolls[i] + _rolls[i+1];
                i += 2;
            }
        }
        return score;
    }
};
```

```
OK    BowlingGameTest/testGutterGame/1
OK    BowlingGameTest/testAllOnes/1
OK    BowlingGameTest/testOneSpare/1
BowlingGameTest OK (3 tests, 0 errors)
```

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }
};
```

```
class BowlingGame {
    const static int MAX_ROLLS = 21;
    int _rolls[MAX_ROLLS];
    int _currentRoll;
public:
    BowlingGame() : _currentRoll(0) {
        for (int i=0; i<MAX_ROLLS; ++i) {
            _rolls[i] = 0;
        }
    }
    void roll(int pins) {
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        int i = 0;
        for (int frame=0; frame<10; ++frame) {
            if ( _rolls[i] + _rolls[i+1] == 10 ) { // spare
                score += 10 + _rolls[i+2];
                i += 2;
            } else {
                score += _rolls[i] + _rolls[i+1];
                i += 2;
            }
        }
        return score;
    }
};
```

bad name for variable

ugly comment in conditional

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }
};
```

```
class BowlingGame {
    const static int MAX_ROLLS = 21;
    int _rolls[MAX_ROLLS];
    int _currentRoll;
public:
    BowlingGame() : _currentRoll(0) {
        for (int i=0; i<MAX_ROLLS; ++i) {
            _rolls[i] = 0;
        }
    }
    void roll(int pins) {
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        int i = 0;
        for (int frame=0; frame<10; ++frame) {
            if ( _rolls[i] + _rolls[i+1] == 10 ) { // spare
                score += 10 + _rolls[i+2];
                i += 2;
            } else {
                score += _rolls[i] + _rolls[i+1];
                i += 2;
            }
        }
        return score;
    }
};
```

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }
};
```

```
class BowlingGame {
    const static int MAX_ROLLS = 21;
    int _rolls[MAX_ROLLS];
    int _currentRoll;
public:
    BowlingGame() : _currentRoll(0) {
        for (int i=0; i<MAX_ROLLS; ++i) {
            _rolls[i] = 0;
        }
    }
    void roll(int pins) {
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        int frameIndex = 0;
        for (int frame=0; frame<10; ++frame) {
            if ( _rolls[frameIndex] +
                 _rolls[frameIndex+1] == 10 ) { // spare
                score += 10 + _rolls[frameIndex+2];
                frameIndex += 2;
            } else {
                score += _rolls[frameIndex] +
                    _rolls[frameIndex+1];
                frameIndex += 2;
            }
        }
        return score;
    }
};
```

The Third Test

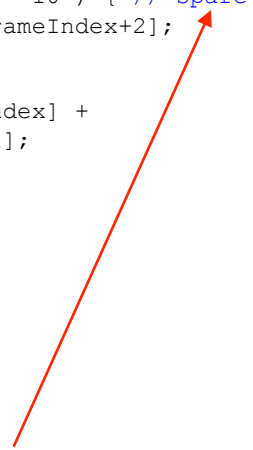
```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }
};
```

```
class BowlingGame {
    const static int MAX_ROLLS = 21;
    int _rolls[MAX_ROLLS];
    int _currentRoll;
public:
    BowlingGame() : _currentRoll(0) {
        for (int i=0; i<MAX_ROLLS; ++i) {
            _rolls[i] = 0;
        }
    }
    void roll(int pins) {
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        int frameIndex = 0;
        for (int frame=0; frame<10; ++frame) {
            if ( _rolls[frameIndex] +
                 _rolls[frameIndex+1] == 10 ) { // spare
                score += 10 + _rolls[frameIndex+2];
                frameIndex += 2;
            } else {
                score += _rolls[frameIndex] +
                    _rolls[frameIndex+1];
                frameIndex += 2;
            }
        }
        return score;
    }
};
```

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }
};
```

```
class BowlingGame {
    const static int MAX_ROLLS = 21;
    int _rolls[MAX_ROLLS];
    int _currentRoll;
public:
    BowlingGame() : _currentRoll(0) {
        for (int i=0; i<MAX_ROLLS; ++i) {
            _rolls[i] = 0;
        }
    }
    void roll(int pins) {
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        int frameIndex = 0;
        for (int frame=0; frame<10; ++frame) {
            if ( _rolls[frameIndex] +
                 _rolls[frameIndex+1] == 10 ) { // spare
                score += 10 + _rolls[frameIndex+2];
                frameIndex += 2;
            } else {
                score += _rolls[frameIndex] +
                    _rolls[frameIndex+1];
                frameIndex += 2;
            }
        }
        return score;
    }
};
```



ugly comment in conditional

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }
};
```

```
class BowlingGame {
    const static int MAX_ROLLS = 21;
    int _rolls[MAX_ROLLS];
    int _currentRoll;

    bool isSpare(int frameIndex) {
        return _rolls[frameIndex] + _rolls[frameIndex+1] == 10;
    }

public:
    BowlingGame() : _currentRoll(0) {
        for (int i=0; i<MAX_ROLLS; ++i) {
            _rolls[i] = 0;
        }
    }
    void roll(int pins) {
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        int frameIndex = 0;
        for (int frame=0; frame<10; ++frame) {
            if ( isSpare(frameIndex) ) {
                score += 10 + _rolls[frameIndex+2];
                frameIndex += 2;
            } else {
                score += _rolls[frameIndex] +
                    _rolls[frameIndex+1];
                frameIndex += 2;
            }
        }
        return score;
    }
};
```

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }
};
```

```
class BowlingGame {
    const static int MAX_ROLLS = 21;
    int _rolls[MAX_ROLLS];
    int _currentRoll;

    bool isSpare(int frameIndex) {
        return _rolls[frameIndex] + _rolls[frameIndex+1] == 10;
    }

public:
    BowlingGame() : _currentRoll(0) {
        for (int i=0; i<MAX_ROLLS; ++i) {
            _rolls[i] = 0;
        }
    }
    void roll(int pins) {
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        int frameIndex = 0;
        for (int frame=0; frame<10; ++frame) {
            if ( isSpare(frameIndex) ) {
                score += 10 + _rolls[frameIndex+2];
                frameIndex += 2;
            } else {
                score += _rolls[frameIndex] +
                    _rolls[frameIndex+1];
                frameIndex += 2;
            }
        }
        return score;
    }
};
```

The Third Test

```
class BowlingGameTest : public QUnit {
public:
    BowlingGameTest() : QUnit("BowlingGameTest") {};

    void run() {
        RUNTEST(testGutterGame);
        RUNTEST(testAllOnes);
        RUNTEST(testOneSpare);
    }

    void rollMany(BowlingGame& g, int n, int pins) {
        for (int i=0; i<n; ++i) {
            g.roll(pins);
        }
    }

    void testGutterGame() {
        BowlingGame g;
        rollMany(g,20,0);
        ASSERT_TRUE(g.score() == 0);
    }

    void testAllOnes() {
        BowlingGame g;
        rollMany(g,20,1);
        ASSERT_TRUE(g.score() == 20);
    }

    void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }
};
```

```
class BowlingGame {
    const static int MAX_ROLLS = 21;
    int _rolls[MAX_ROLLS];
    int _currentRoll;

    bool isSpare(int frameIndex) {
        return _rolls[frameIndex] + _rolls[frameIndex+1] == 10;
    }

public:
    BowlingGame() : _currentRoll(0) {
        for (int i=0; i<MAX_ROLLS; ++i) {
            _rolls[i] = 0;
        }
    }
    void roll(int pins) {
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        int frameIndex = 0;
        for (int frame=0; frame<10; ++frame) {
            if ( isSpare(frameIndex) ) {
                score += 10 + _rolls[frameIndex+2];
                frameIndex += 2;
            } else {
                score += _rolls[frameIndex] +
                    _rolls[frameIndex+1];
                frameIndex += 2;
            }
        }
        return score;
    }
};
```

The Fourth Test

The Fourth Test

```
// ...

    void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }

    void testOneStrike() {
        BowlingGame g;
        g.roll(10); // strike
        g.roll(3);
        g.roll(4);
        rollMany(g, 16, 0);
        ASSERT_TRUE(g.score() == 24);
    }

// ...
```

```
class BowlingGame {
    const static int MAX_ROLLS = 21;
    int _rolls[MAX_ROLLS];
    int _currentRoll;

    bool isSpare(int frameIndex) {
        return _rolls[frameIndex] + _rolls[frameIndex+1] == 10;
    }

public:
    BowlingGame() : _currentRoll(0) {
        for (int i=0; i<MAX_ROLLS; ++i) {
            _rolls[i] = 0;
        }
    }
    void roll(int pins) {
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        int frameIndex = 0;
        for (int frame=0; frame<10; ++frame) {
            if ( isSpare(frameIndex) ) {
                score += 10 + _rolls[frameIndex+2];
                frameIndex += 2;
            } else {
                score += _rolls[frameIndex] +
                    _rolls[frameIndex+1];
                frameIndex += 2;
            }
        }
        return score;
    }
};
```

The Fourth Test

```
// ...

    void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }

    void testOneStrike() {
        BowlingGame g;
        g.roll(10); // strike
        g.roll(3);
        g.roll(4);
        rollMany(g, 16, 0);
        ASSERT_TRUE(g.score() == 24);
    }

// ...
```

```
class BowlingGame {
    const static int MAX_ROLLS = 21;
    int _rolls[MAX_ROLLS];
    int _currentRoll;

    bool isSpare(int frameIndex) {
        return _rolls[frameIndex] + _rolls[frameIndex+1] == 10;
    }

public:
    BowlingGame() : _currentRoll(0) {
        for (int i=0; i<MAX_ROLLS; ++i) {
            _rolls[i] = 0;
        }
    }
    void roll(int pins) {
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        int frameIndex = 0;
        for (int frame=0; frame<10; ++frame) {
            if ( isSpare(frameIndex) ) {
                score += 10 + _rolls[frameIndex+2];
                frameIndex += 2;
            } else {
                score += _rolls[frameIndex] +
                    _rolls[frameIndex+1];
                frameIndex += 2;
            }
        }
        return score;
    }
};
```

```
OK    BowlingGameTest/testGutterGame/1
OK    BowlingGameTest/testAllOnes/1
OK    BowlingGameTest/testOneSpare/1
ERROR BowlingGameTest/testOneStrike/1
BowlingGameTest.cpp:54: unittest failed: g.score() == 24 (BowlingGameTest/testOneStrike/1)
```

The Fourth Test

```
// ...

    void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }

    void testOneStrike() {
        BowlingGame g;
        g.roll(10); // strike
        g.roll(3);
        g.roll(4);
        rollMany(g, 16, 0);
        ASSERT_TRUE(g.score() == 24);
    }

// ...
```

```
class BowlingGame {
    const static int MAX_ROLLS = 21;
    int _rolls[MAX_ROLLS];
    int _currentRoll;

    bool isSpare(int frameIndex) {
        return _rolls[frameIndex] + _rolls[frameIndex+1] == 10;
    }

    bool isStrike(int frameIndex) {
        return _rolls[frameIndex] == 10;
    }

public:
    BowlingGame() : _currentRoll(0) {
        for (int i=0; i<MAX_ROLLS; ++i) {
            _rolls[i] = 0;
        }
    }
    void roll(int pins) {
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        int frameIndex = 0;
        for (int frame=0; frame<10; ++frame) {
            if ( isStrike(frameIndex) ) {
                score += 10 + _rolls[frameIndex+1] +
                    _rolls[frameIndex+2];
                frameIndex += 1;
            } else if ( isSpare(frameIndex) ) {
                score += 10 + _rolls[frameIndex+2];
                frameIndex += 2;
            } else {
                score += _rolls[frameIndex] +
                    _rolls[frameIndex+1];
                frameIndex += 2;
            }
        }
        return score;
    }
};
```

The Fourth Test

```
// ...

    void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }

    void testOneStrike() {
        BowlingGame g;
        g.roll(10); // strike
        g.roll(3);
        g.roll(4);
        rollMany(g, 16, 0);
        ASSERT_TRUE(g.score() == 24);
    }

// ...
```

```
class BowlingGame {
    const static int MAX_ROLLS = 21;
    int _rolls[MAX_ROLLS];
    int _currentRoll;

    bool isSpare(int frameIndex) {
        return _rolls[frameIndex] + _rolls[frameIndex+1] == 10;
    }

    bool isStrike(int frameIndex) {
        return _rolls[frameIndex] == 10;
    }

public:
    BowlingGame() : _currentRoll(0) {
        for (int i=0; i<MAX_ROLLS; ++i) {
            _rolls[i] = 0;
        }
    }
    void roll(int pins) {
        _rolls[_currentRoll++] = pins;
    }
    int score() {
        int score = 0;
        int frameIndex = 0;
        for (int frame=0; frame<10; ++frame) {
            if ( isStrike(frameIndex) ) {
                score += 10 + _rolls[frameIndex+1] +
                    _rolls[frameIndex+2];
                frameIndex += 1;
            } else if ( isSpare(frameIndex) ) {
                score += 10 + _rolls[frameIndex+2];
                frameIndex += 2;
            } else {
                score += _rolls[frameIndex] +
                    _rolls[frameIndex+1];
                frameIndex += 2;
            }
        }
        return score;
    }
};
```

The Fourth Test

```
// ...

    void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }

    void testOneStrike() {
        BowlingGame g;
        g.roll(10); // strike
        g.roll(3);
        g.roll(4);
        rollMany(g, 16, 0);
        ASSERT_TRUE(g.score() == 24);
    }

// ...
```

```
// ...
private:
    bool isSpare(int frameIndex) {
        return _rolls[frameIndex] + _rolls[frameIndex+1] == 10;
    }

    bool isStrike(int frameIndex) {
        return _rolls[frameIndex] == 10;
    }

    int strikeBonus(int frameIndex) {
        return _rolls[frameIndex+1] + _rolls[frameIndex+2];
    }

    int spareBonus(int frameIndex) {
        return _rolls[frameIndex+2];
    }

    int sumOfRollsInFrame(int frameIndex) {
        return _rolls[frameIndex] + _rolls[frameIndex+1];
    }

public:
    // ...
    int score() {
        int score = 0;
        int frameIndex = 0;
        for (int frame=0; frame<10; ++frame) {
            if ( isStrike(frameIndex) ) {
                score += 10 + strikeBonus(frameIndex);
                frameIndex += 1;
            } else if ( isSpare(frameIndex) ) {
                score += 10 + spareBonus(frameIndex);
                frameIndex += 2;
            } else {
                score += sumOfRollsInFrame(frameIndex);
                frameIndex += 2;
            }
        }
        return score;
    }
```

The Fourth Test

```
// ...

    void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }

    void testOneStrike() {
        BowlingGame g;
        g.roll(10); // strike
        g.roll(3);
        g.roll(4);
        rollMany(g, 16, 0);
        ASSERT_TRUE(g.score() == 24);
    }

// ...
```

```
// ...
private:
    bool isSpare(int frameIndex) {
        return _rolls[frameIndex] + _rolls[frameIndex+1] == 10;
    }

    bool isStrike(int frameIndex) {
        return _rolls[frameIndex] == 10;
    }

    int strikeBonus(int frameIndex) {
        return _rolls[frameIndex+1] + _rolls[frameIndex+2];
    }

    int spareBonus(int frameIndex) {
        return _rolls[frameIndex+2];
    }

    int sumOfRollsInFrame(int frameIndex) {
        return _rolls[frameIndex] + _rolls[frameIndex+1];
    }

public:
    // ...
    int score() {
        int score = 0;
        int frameIndex = 0;
        for (int frame=0; frame<10; ++frame) {
            if ( isStrike(frameIndex) ) {
                score += 10 + strikeBonus(frameIndex);
                frameIndex += 1;
            } else if ( isSpare(frameIndex) ) {
                score += 10 + spareBonus(frameIndex);
                frameIndex += 2;
            } else {
                score += sumOfRollsInFrame(frameIndex);
                frameIndex += 2;
            }
        }
        return score;
    }
```

The Fourth Test

```
// ...

    void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }

    void testOneStrike() {
        BowlingGame g;
        g.roll(10); // strike
        g.roll(3);
        g.roll(4);
        rollMany(g, 16, 0);
        ASSERT_TRUE(g.score() == 24);
    }

// ...
```

```
// ...
private:
    bool isSpare(int frameIndex) {
        return _rolls[frameIndex] + _rolls[frameIndex+1] == 10;
    }

    bool isStrike(int frameIndex) {
        return _rolls[frameIndex] == 10;
    }

    int strikeBonus(int frameIndex) {
        return _rolls[frameIndex+1] + _rolls[frameIndex+2];
    }

    int spareBonus(int frameIndex) {
        return _rolls[frameIndex+2];
    }

    int sumOfRollsInFrame(int frameIndex) {
        return _rolls[frameIndex] + _rolls[frameIndex+1];
    }

public:
    // ...
    int score() {
        int score = 0;
        int frameIndex = 0;
        for (int frame=0; frame<10; ++frame) {
            if ( isStrike(frameIndex) ) {
                score += 10 + strikeBonus(frameIndex);
                frameIndex += 1;
            } else if ( isSpare(frameIndex) ) {
                score += 10 + spareBonus(frameIndex);
                frameIndex += 2;
            } else {
                score += sumOfRollsInFrame(frameIndex);
                frameIndex += 2;
            }
        }
        return score;
    }
```

The Fifth Test

The Fifth Test

```
// ...

 void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }

    void testOneStrike() {
        BowlingGame g;
        g.roll(10); // strike
        g.roll(3);
        g.roll(4);
        rollMany(g, 16, 0);
        ASSERT_TRUE(g.score() == 24);
    }

    void testPerfectGame() {
        BowlingGame g;
        rollMany(g, 12, 10);
        ASSERT_TRUE(g.score() == 300);
    }

// ...
```

```
// ...

    int score() {
        int score = 0;
        int frameIndex = 0;
        for (int frame=0; frame<10; ++frame) {
            if ( isStrike(frameIndex) ) {
                score += 10 + strikeBonus(frameIndex);
                frameIndex += 1;
            } else if ( isSpare(frameIndex) ) {
                score += 10 + spareBonus(frameIndex);
                frameIndex += 2;
            } else {
                score += sumOfRollsInFrame(frameIndex);
                frameIndex += 2;
            }
        }
        return score;
    }
```

The Fifth Test

```
// ...

 void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }

    void testOneStrike() {
        BowlingGame g;
        g.roll(10); // strike
        g.roll(3);
        g.roll(4);
        rollMany(g, 16, 0);
        ASSERT_TRUE(g.score() == 24);
    }

    void testPerfectGame() {
        BowlingGame g;
        rollMany(g, 12, 10);
        ASSERT_TRUE(g.score() == 300);
    }

// ...
```

```
// ...

    int score() {
        int score = 0;
        int frameIndex = 0;
        for (int frame=0; frame<10; ++frame) {
            if ( isStrike(frameIndex) ) {
                score += 10 + strikeBonus(frameIndex);
                frameIndex += 1;
            } else if ( isSpare(frameIndex) ) {
                score += 10 + spareBonus(frameIndex);
                frameIndex += 2;
            } else {
                score += sumOfRollsInFrame(frameIndex);
                frameIndex += 2;
            }
        }
        return score;
    }
```

```
OK    BowlingGameTest/testGutterGame/1
OK    BowlingGameTest/testAllOnes/1
OK    BowlingGameTest/testOneSpare/1
OK    BowlingGameTest/testOneStrike/1
OK    BowlingGameTest/testPerfectGame/1
BowlingGameTest OK (5 tests, 0 errors)
```

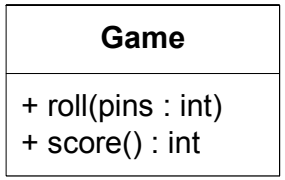
The Fifth Test

```
// ...

 void testOneSpare() {
        BowlingGame g;
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(g, 17, 0);
        ASSERT_TRUE(g.score() == 16);
    }

    void testOneStrike() {
        BowlingGame g;
        g.roll(10); // strike
        g.roll(3);
        g.roll(4);
        rollMany(g, 16, 0);
        ASSERT_TRUE(g.score() == 24);
    }

    void testPerfectGame() {
        BowlingGame g;
        rollMany(g, 12, 10);
        ASSERT_TRUE(g.score() == 300);
    }

// ...
```

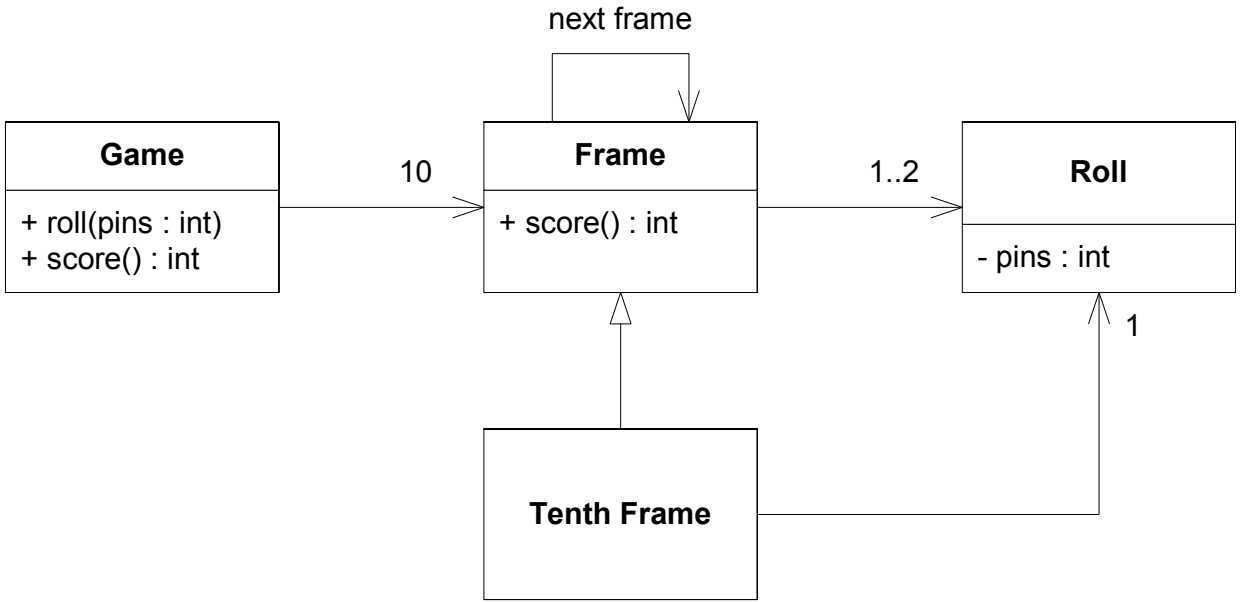
```
// ...

    int score() {
        int score = 0;
        int frameIndex = 0;
        for (int frame=0; frame<10; ++frame) {
            if ( isStrike(frameIndex) ) {
                score += 10 + strikeBonus(frameIndex);
                frameIndex += 1;
            } else if ( isSpare(frameIndex) ) {
                score += 10 + spareBonus(frameIndex);
                frameIndex += 2;
            } else {
                score += sumOfRollsInFrame(frameIndex);
                frameIndex += 2;
            }
        }
        return score;
    }
```

Comparing TDD and OOAD

Design by TDD

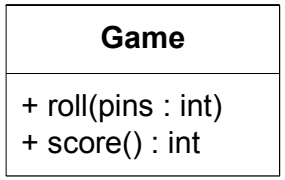


Design by OOAD

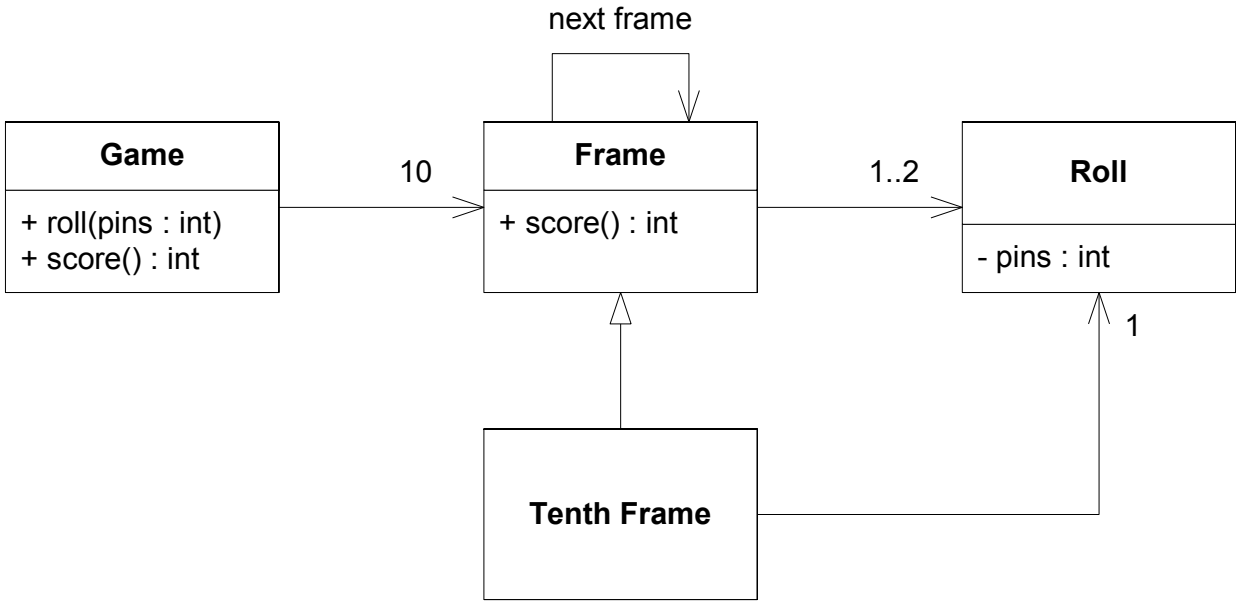


Comparing TDD and OOAD

Design by TDD



Design by OOAD



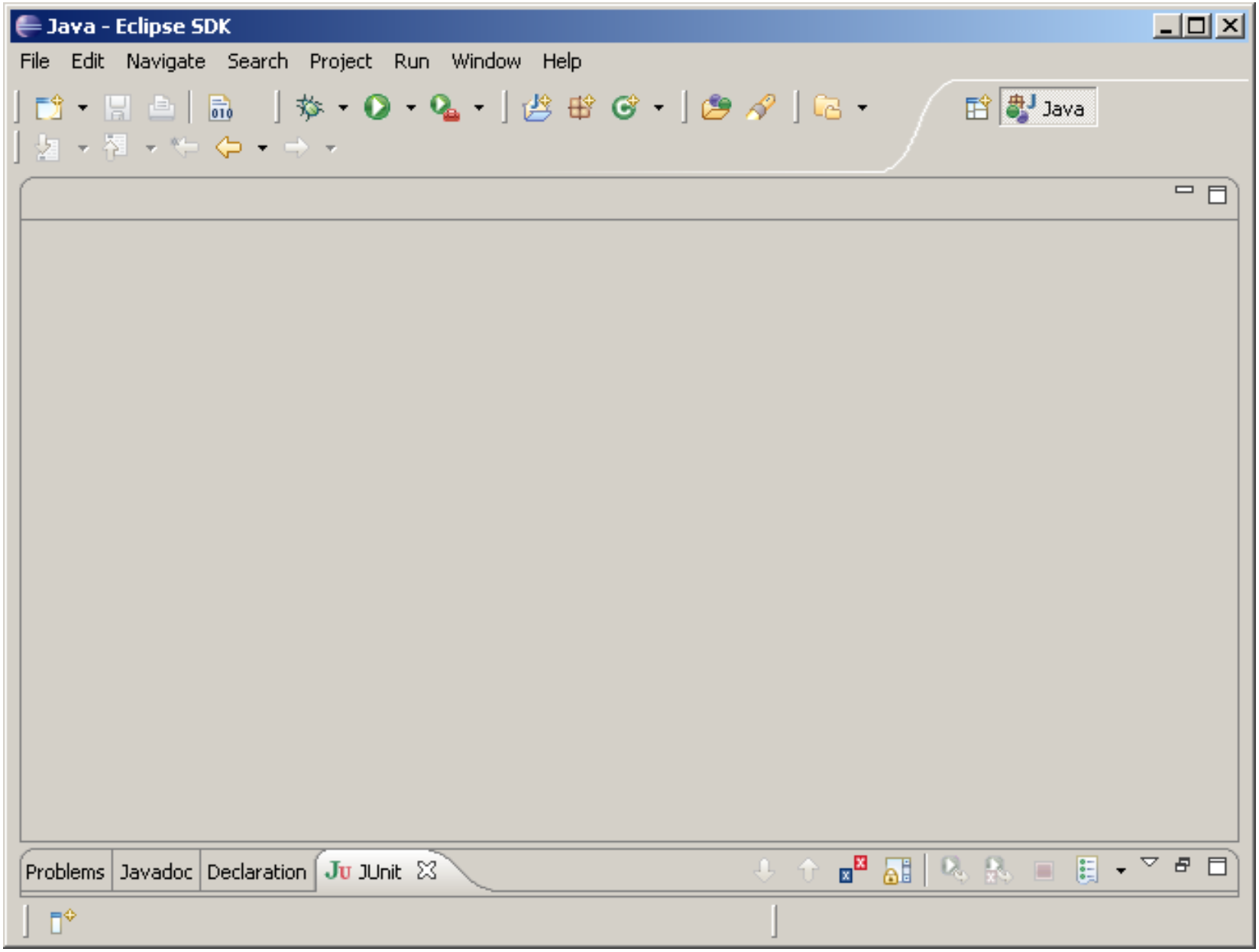
(ok, this design is not good... but it illustrates the point well)

The Bowling Kata, Summary

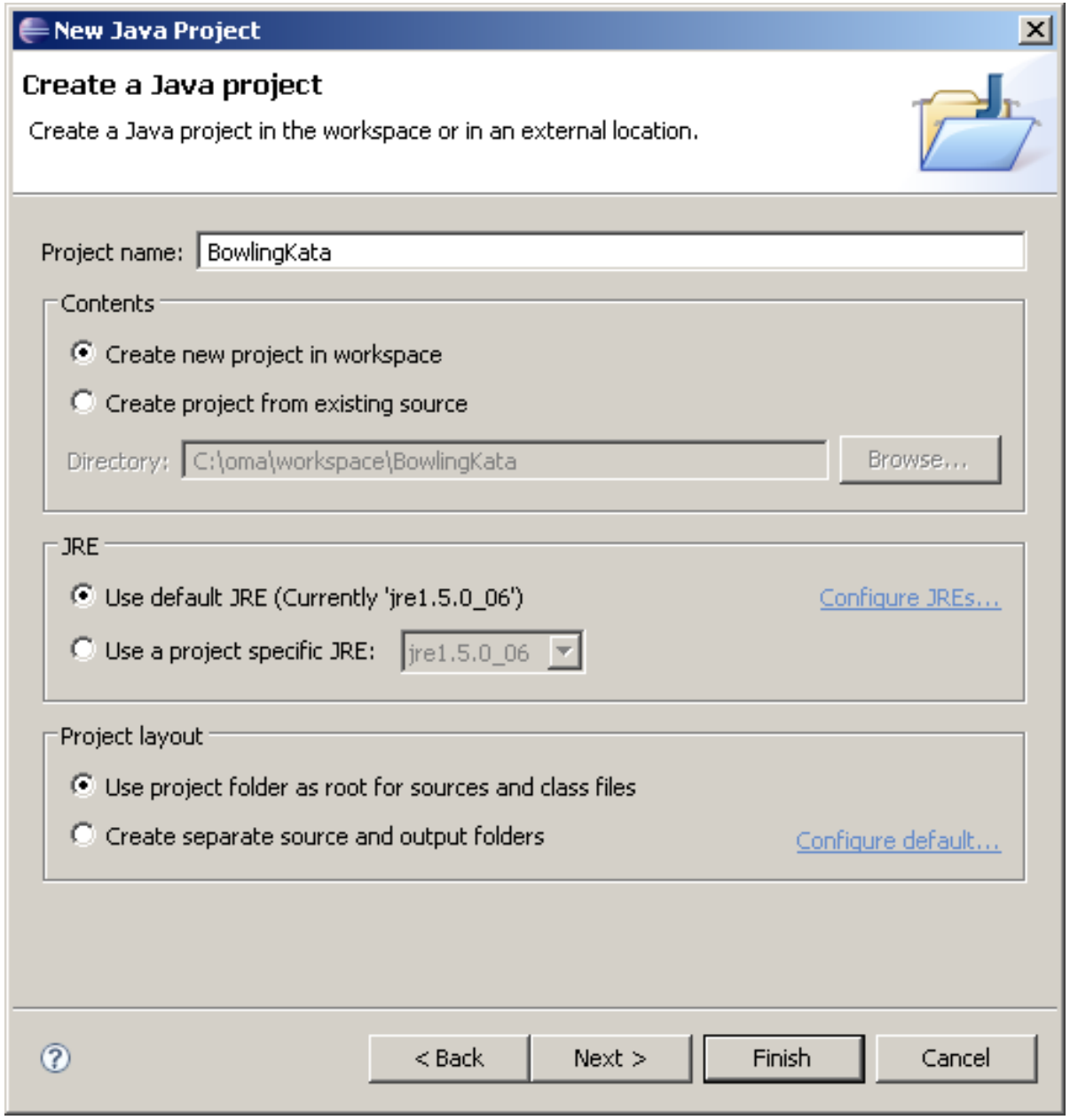
- TDD drives the design and implementation process
- Test coverage is often close to 100%
- TDD vs Debuggers

- Brief introduction to Test-Driven Development
- QUnit - A simple framework for unit testing in C++
- The Bowling Game Kata in C++
- **More TDD examples**
- Q&A

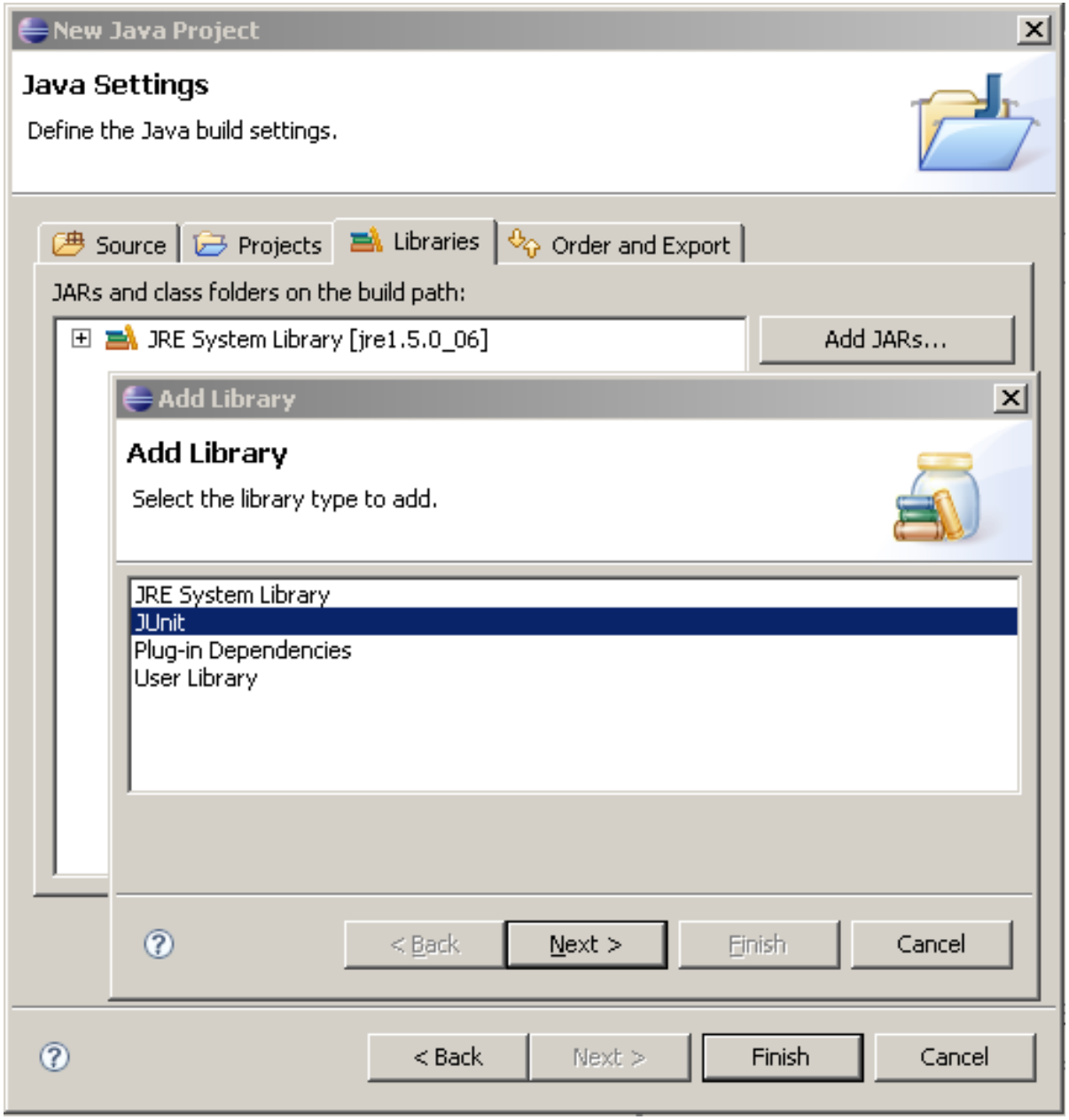
Eclipse


Java - Eclipse SDK
File Edit Navigate Search Project Run Window Help
Java
Problems Javadoc Declaration JUnit

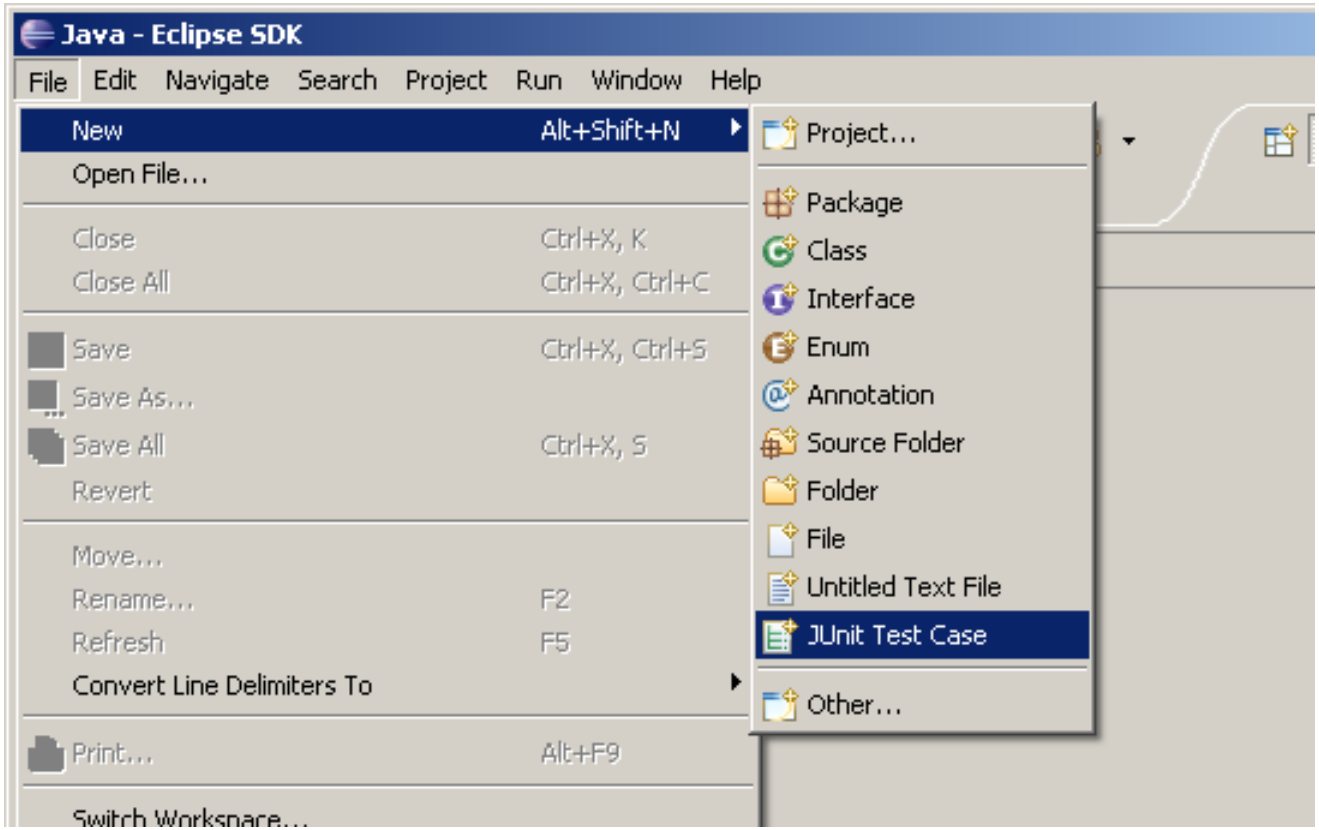
Eclipse - New Java Project



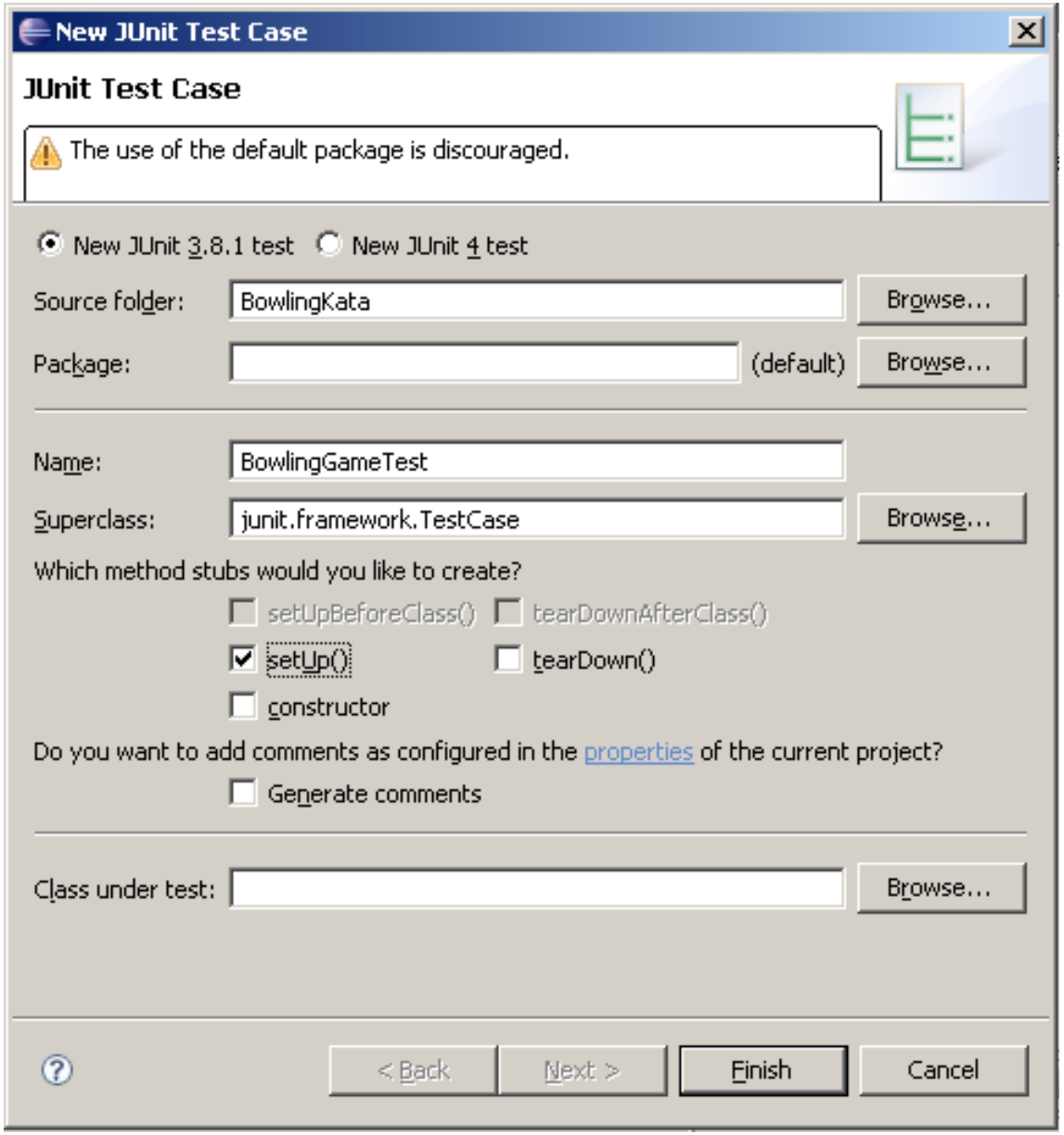
Eclipse - Add JUnit Library



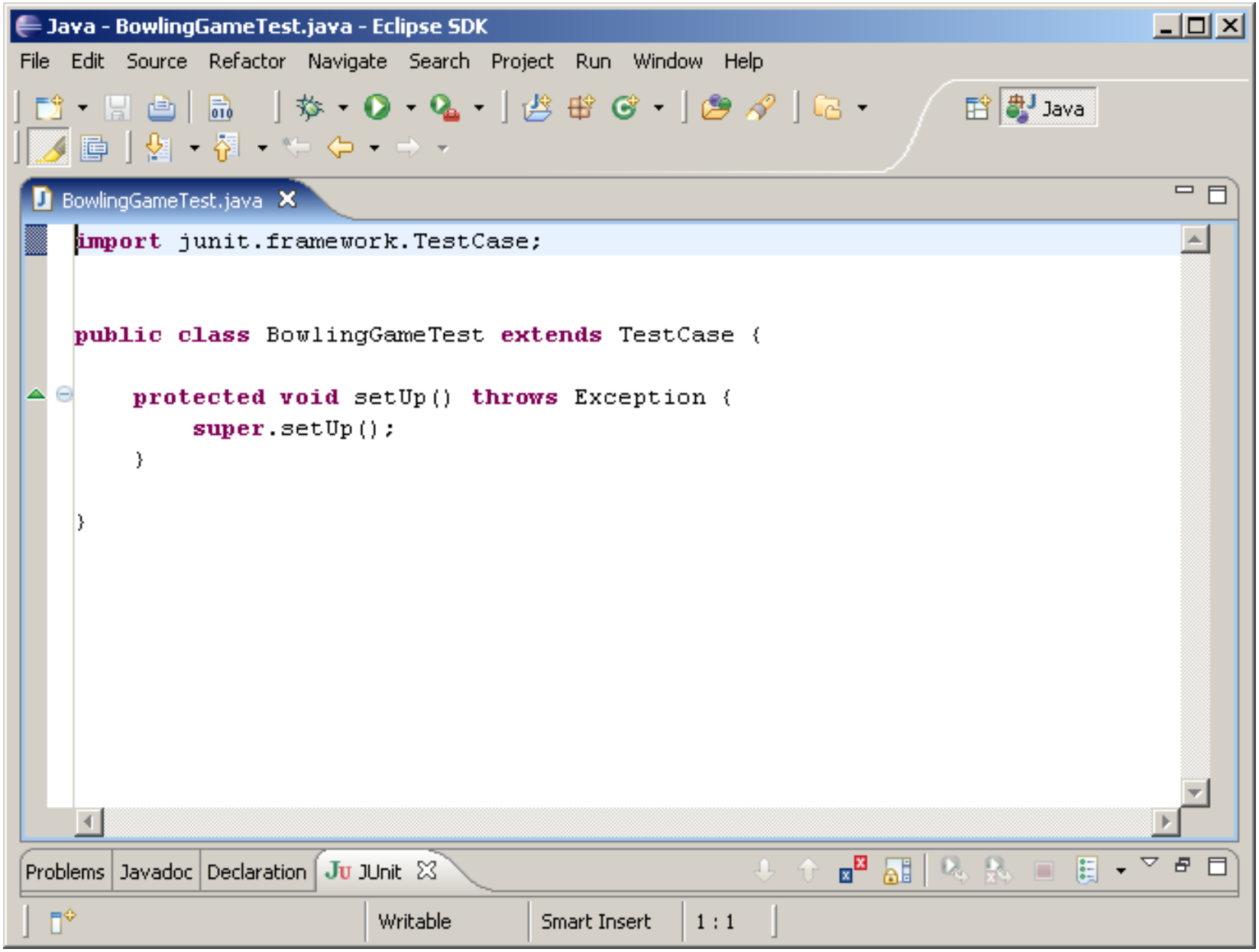
Eclipse - Create new JUnit Test Case



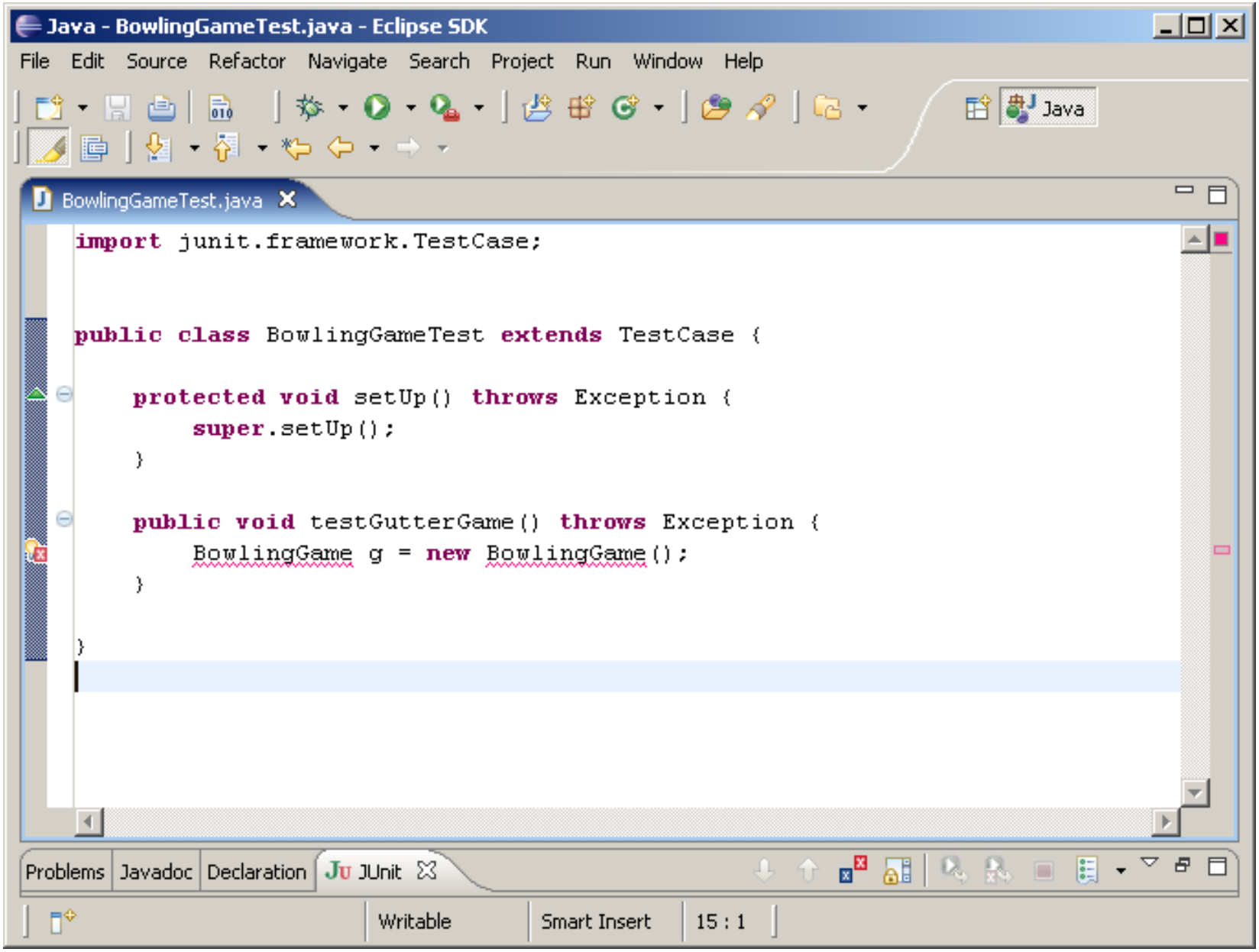
Eclipse - Create new JUnit Test Case



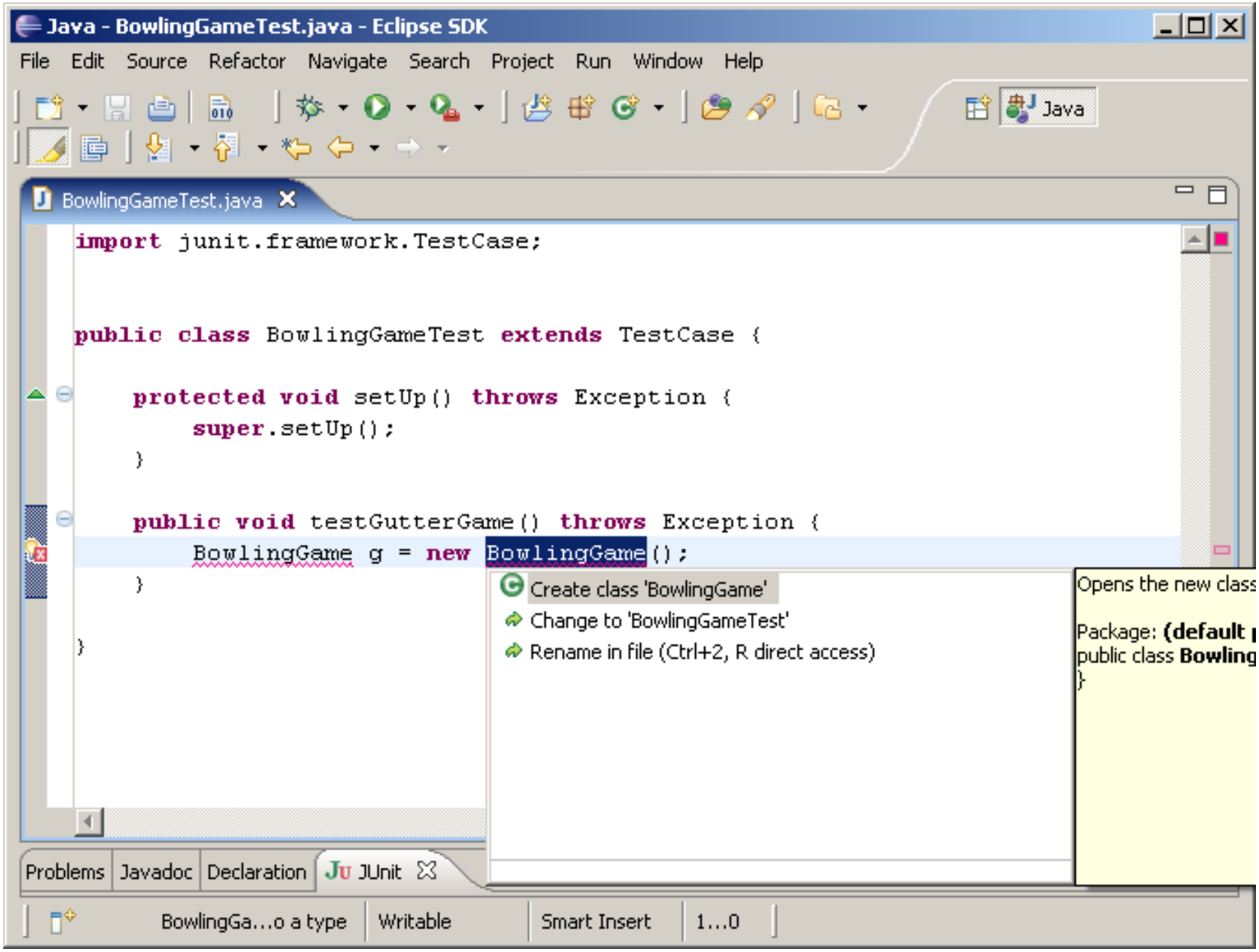
Eclipse - JUnit Test Case



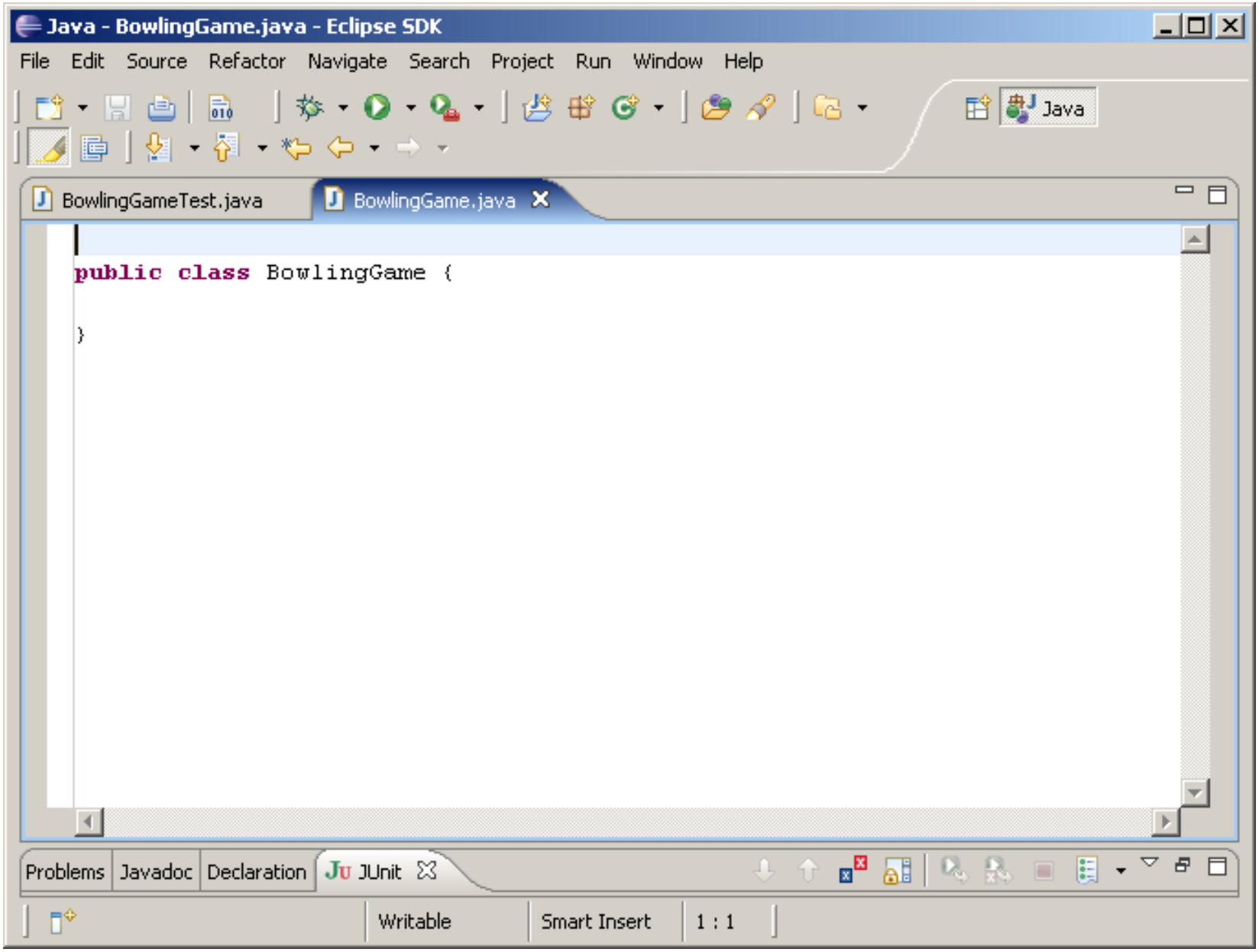
Eclipse - write the first test



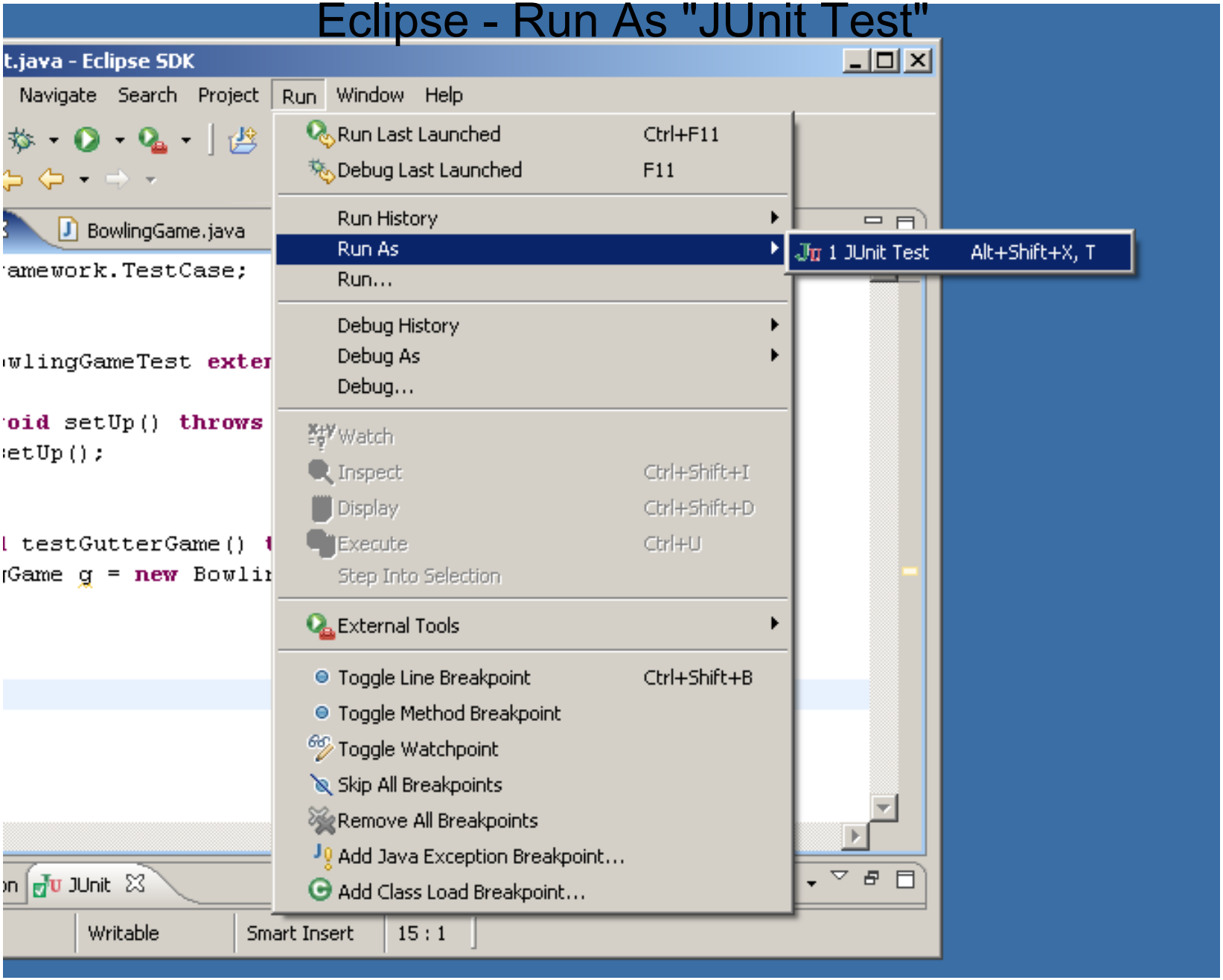
Eclipse - use class wizard to fix compile error



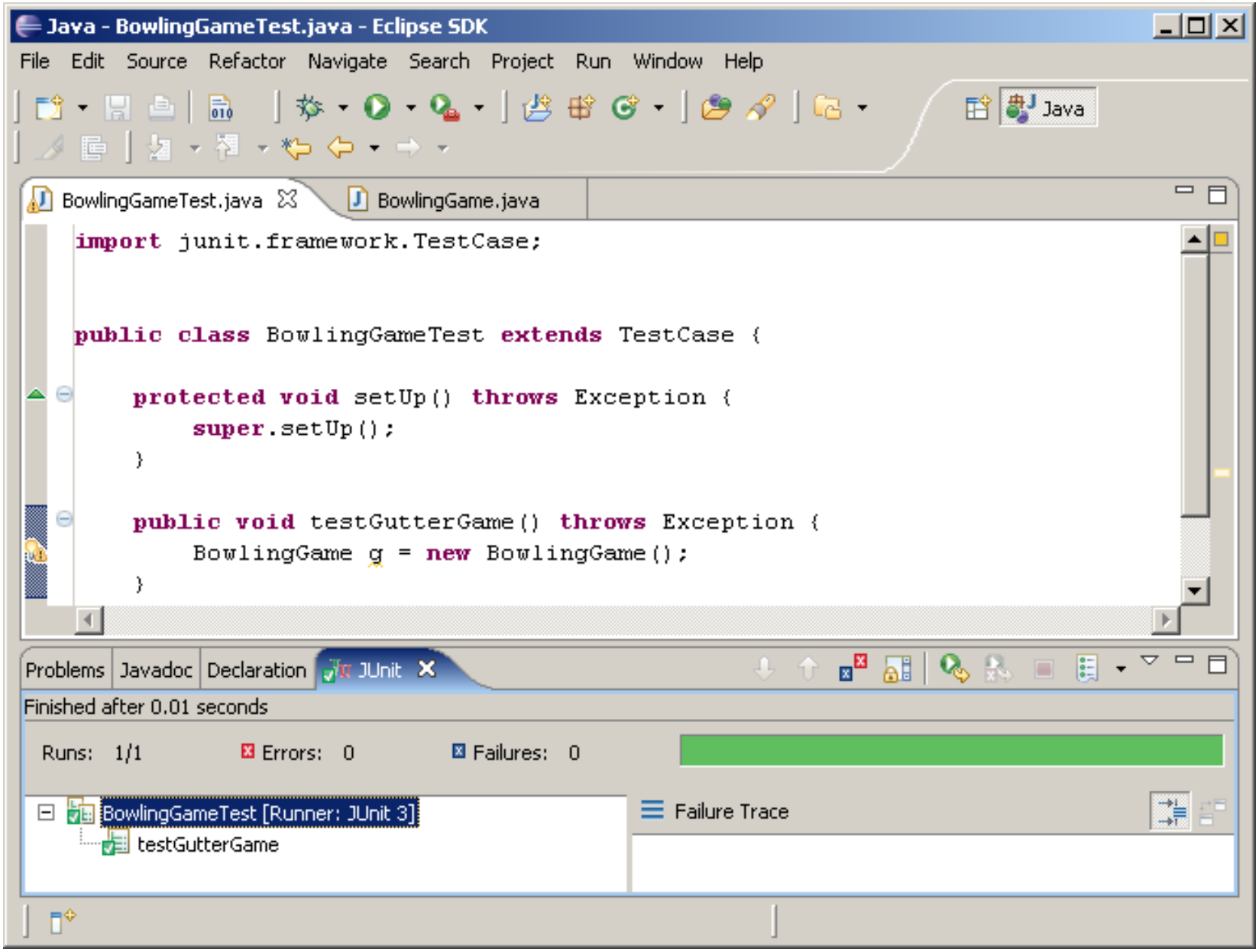
Eclipse - a new class is created



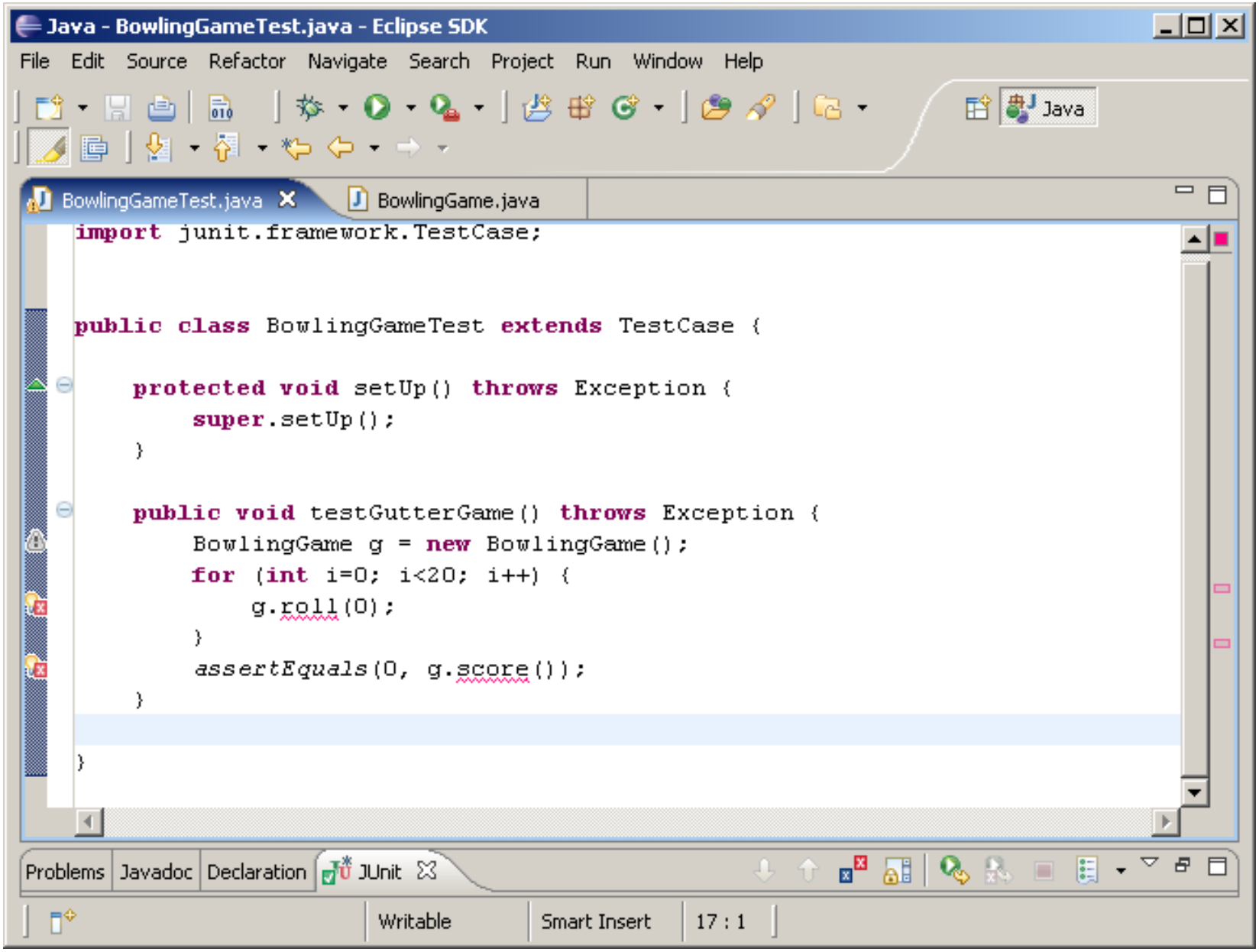
Eclipse - Run As "JUnit Test"



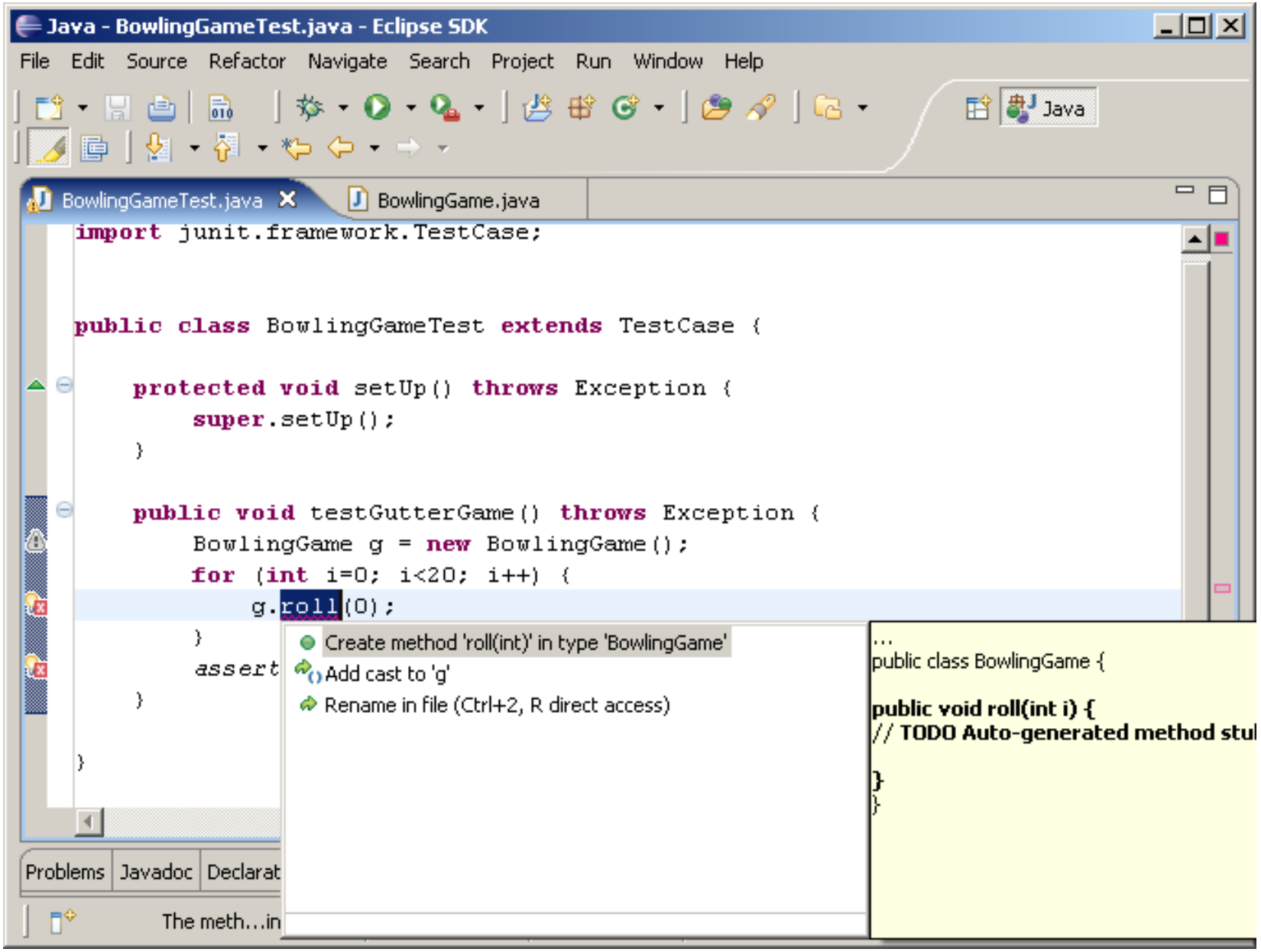
Eclipse - First successful build



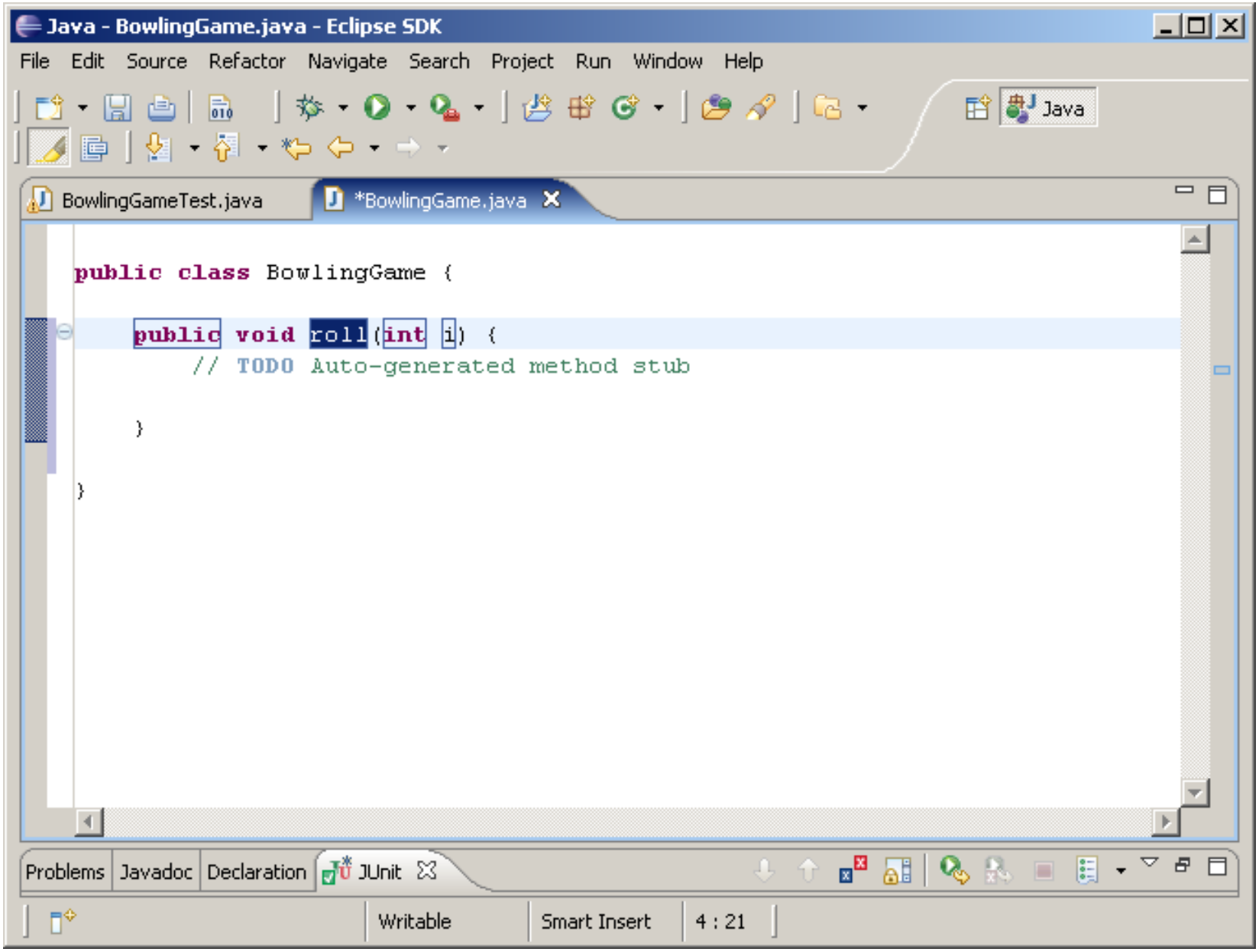
Eclipse - Add a test



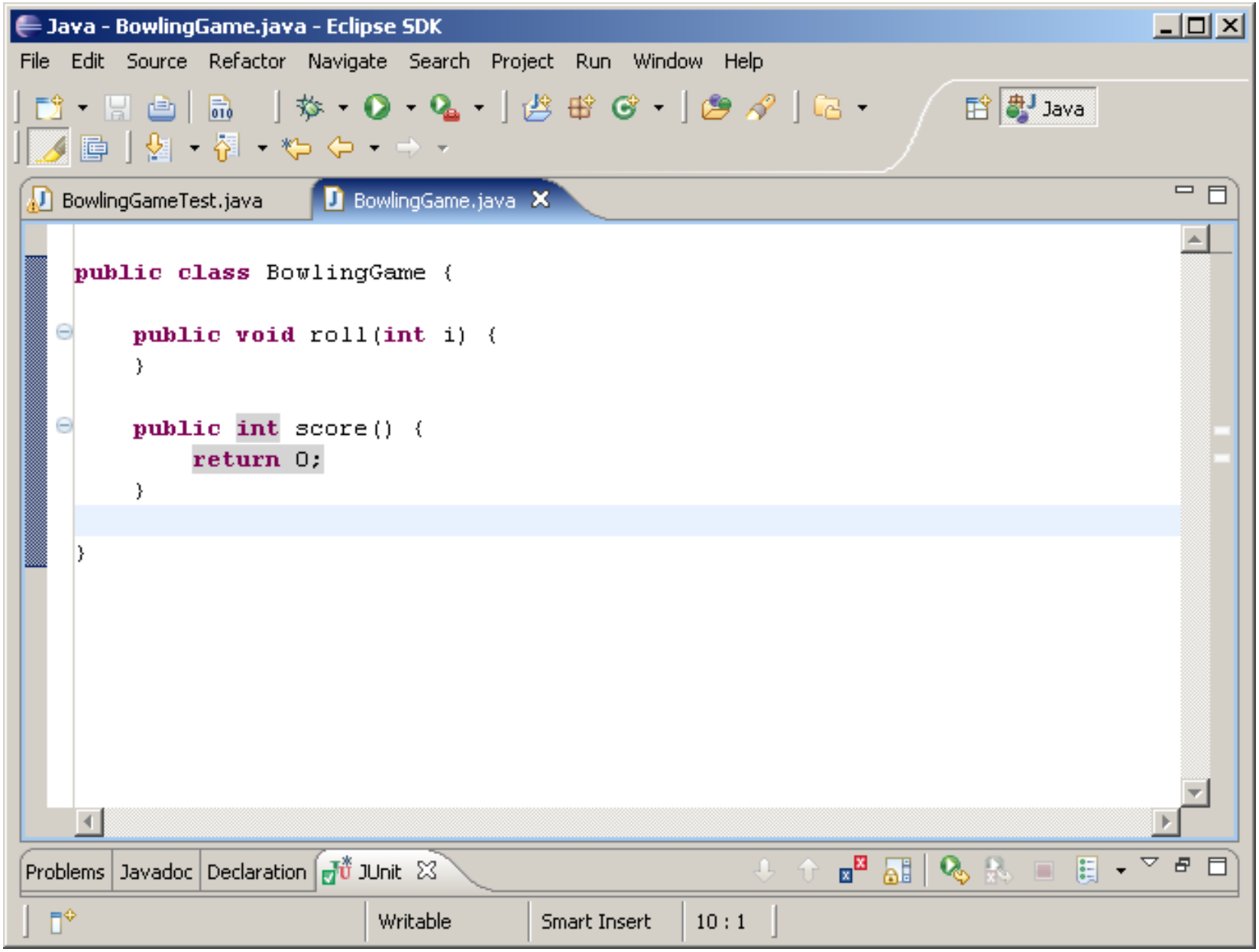
Eclipse - Resolve compile issues



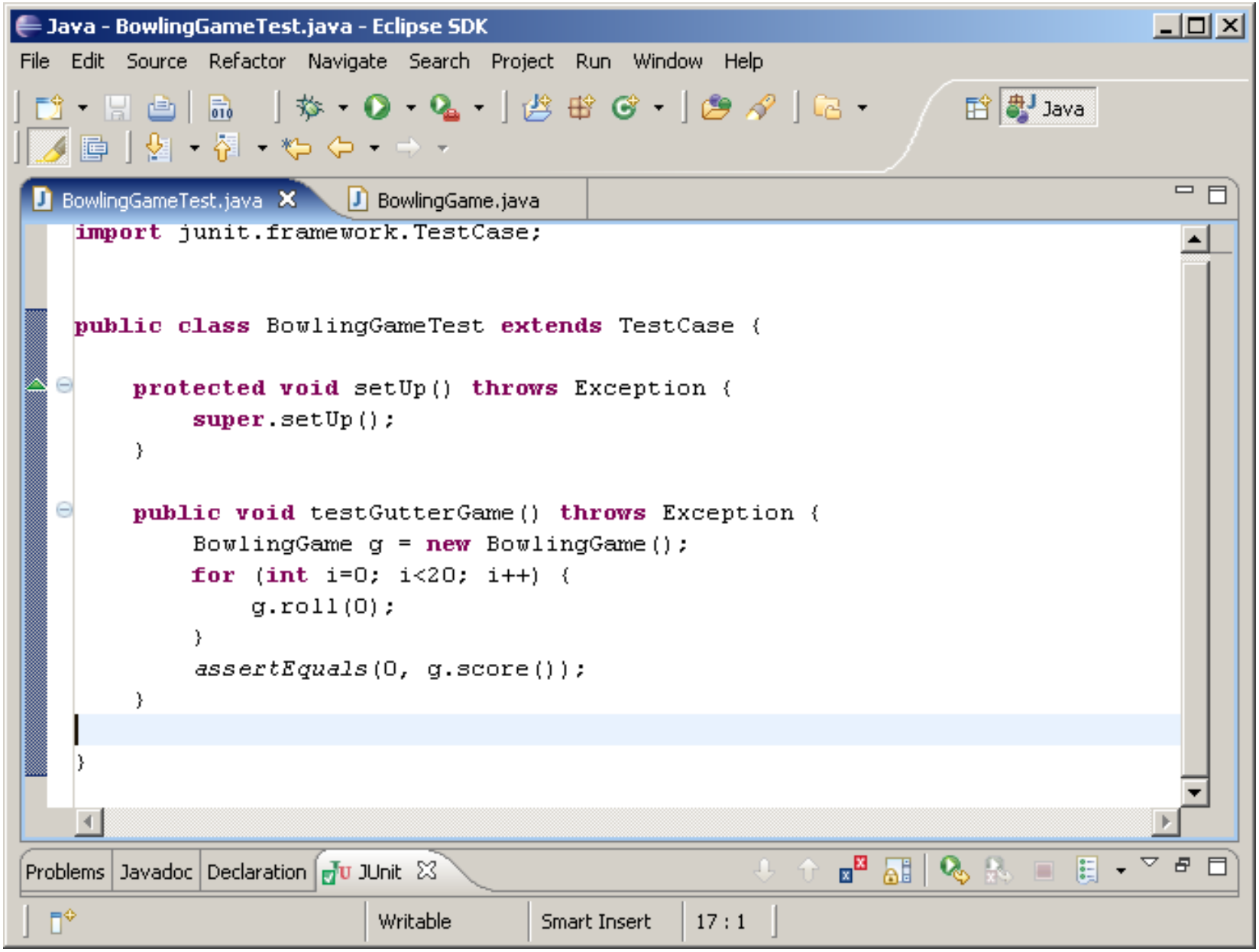
Eclipse - using wizard to create the method



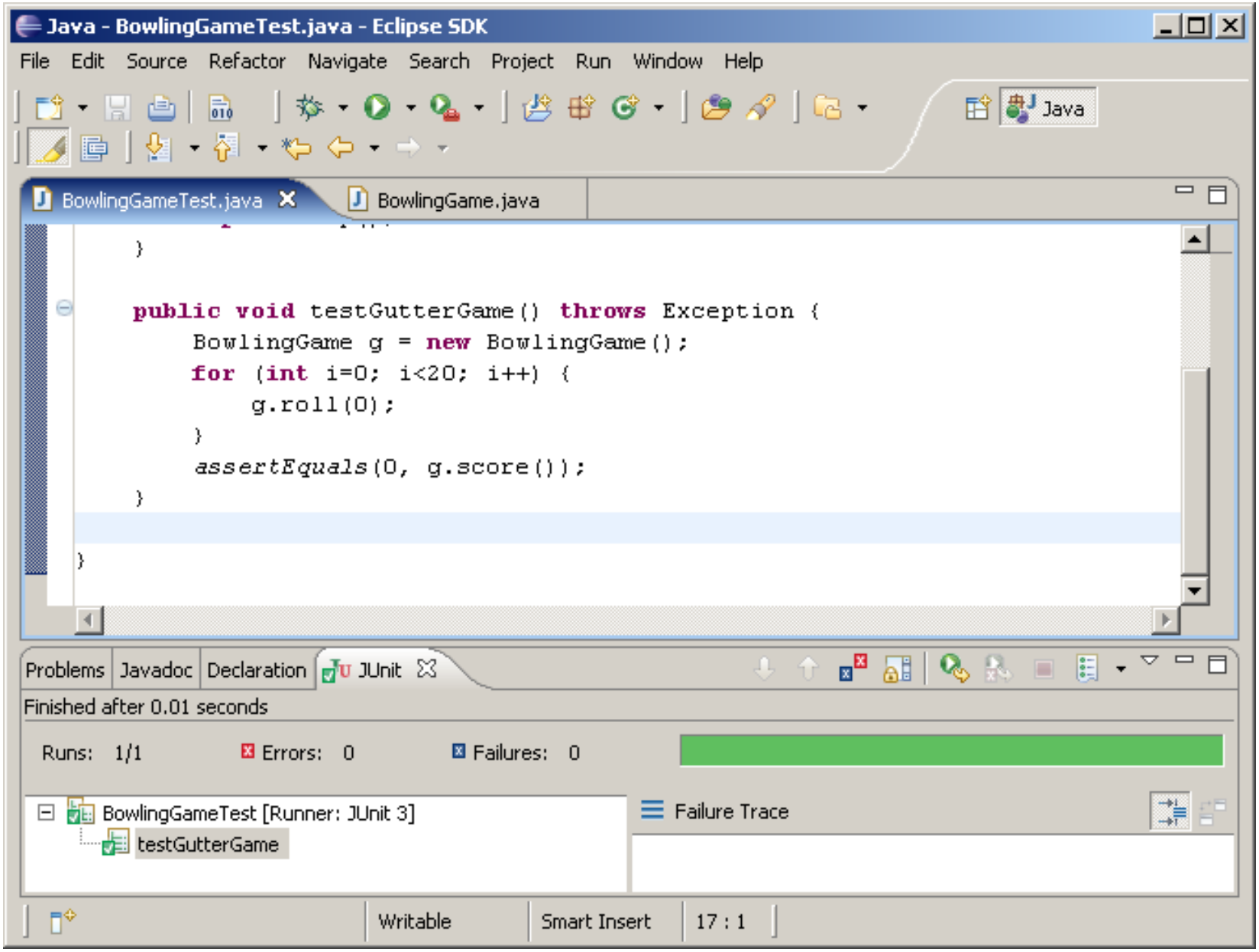
Eclipse - using wizard to create the method



Eclipse - compile ok



Eclipse - test ok



Behaviour-Driven Development

In computer science **Behavior Driven Development** (or BDD) is a programming technique that questions the behavior of an application before and during the development process. By asking questions such as "What should this application do?" or "What should this part do?" developers can identify gaps in their understanding of the problem domain and talk to their peers or domain experts to find the answers. By focusing on the behavior of applications, developers try to create a common language that's shared by all stakeholders: management, users, developers, project management and domain experts.

[source: Wikipedia]

BDD is TDD done correctly.
[source: unknown]

BDD - Bowling Score with RSpec (Ruby)

```ruby
require 'spec'

context "A bowling score calculator" do

  setup do
    @game = Game.new
  end

  specify "should score 0 for an all gutter game" do
    (1..20).each { @game.roll(0) }
    @game.score.should.be 0
  end

  specify "should score 20 for an all ones game" do
    (1..20).each { @game.roll(1) }
    @game.score.should.be 20
  end

  specify "should score 150 for an all fives game" do
    (1..21).each { @game.roll(5) }
    @game.score.should.be 150
  end

  specify "should score 300 for a perfect game" do
    (1..12).each { @game.roll(10) }
    @game.score.should.be 300
  end

end
```

```ruby
class Game

  def initialize
    @rolls = []
  end

  def roll(pins)
    @rolls.push pins
  end

  def score
    compute_score(1, @rolls)
  end

  def compute_score(frame, rolls)
    return 0 if frame > 10
    return do_strike(frame, rolls) if strike?(rolls)
    return do_spare(frame, rolls) if spare?(rolls)
    return do_regular_frame(frame, rolls)
  end

  def strike?(rolls)
    rolls[0] == 10
  end

  def spare?(rolls)
    rolls[0] + rolls[1] == 10
  end

  def do_strike(frame, rolls)
    10 + rolls[1] + rolls[2] + compute_score(frame + 1, rolls[1..-1])
  end

  def do_spare(frame, rolls)
    10 + rolls[2] + compute_score(frame + 1, rolls[2..-1])
  end

  def do_regular_frame(frame, rolls)
    rolls[0] + rolls[1] + compute_score(frame + 1, rolls[2..-1])
  end

end
```

- Brief introduction to Test-Driven Development
- QUnit - A simple framework for unit testing in C++
- The Bowling Game Kata in C++
- TDD in other languages
- Q&A

Q&A