

Design Patterns

Produced
by

Eamonn de Leastar
edeleastar@wit.ie

Department of Computing, Maths & Physics
Waterford Institute of Technology

<http://www.wit.ie>

<http://elearning.wit.ie>



Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE



Solver Example

Idiomatic, Template Method, Strategy

Case Study

- Examine a simple problem/solution from three perspectives
 - Idiomatic
 - Template Method Pattern
 - Strategy Pattern

Solver - Find the minima of a line

- Build test class first:

```
public class MinimaSolverTest
{

    private double[] line = { 1.0, 2.0, 1.0, 2.0, -1.0, 3.0, 4.0, 5.0, 4.0 };
    private MinimaSolver solver;

    @Test
    public void minima()
    {
        solver = new MinimaSolver();
        double[] result = solver.minima(line);
        assertTrue(result[0] == 1.1);
        assertTrue(result[1] == 2.2);
    }
}
```

Solver - the MinimaSolver class

- Some preprocessing and post processing is carried out (commented out in the code).
- Note that there are in fact two algorithms for finding the minima - our code just uses the `leastSquaresAlgorithm`.
- The algorithm implementations are replaced with stubs to keep the code focussed on the patterns

```
public class MinimaSolver
{
    public MinimaSolver()
    {
    }

    public double[] minima(double[] line)
    {
        // do some pre-processing
        double[] result = leastSquaresAlgorithm(line);
        // do some post-processing
        return result;
    }

    public double[] leastSquaresAlgorithm(double[] line)
    {
        return new double[] { 1.1, 2.2 };
    }

    public double[] newtonsMethodAlgorithm(double[] line)
    {
        return new double[] { 3.3, 4.4 };
    }
}
```

Modify to Allow Algorithm to be Selected

- Simple integer to select appropriate algorithm:
 - 0 - least squares
 - 1 - newtons method

```
@Test
public void leastSquaresAlgorithm()
{
    solver = new MinimaSolver(0);
    double[] result = solver.minima(line);
    assertTrue(result[0] == 1.1);
    assertTrue(result[1] == 2.2);
}

@Test
public void newtonsMethodAlgorithm()
{
    solver = new MinimaSolver(1);
    double[] result = solver.minima(line);
    assertTrue(result[0] == 3.3);
    assertTrue(result[1] == 4.4);
}
```

MinimaSolver with integer selection

- Simple if statement to select desired algorithm.

```
public class MinimaSolver
{
    private int algorithm;

    public MinimaSolver(int algorithm)
    {
        this.algorithm = algorithm;
    }

    public double[] minima(double[] line)
    {
        // do some pre-processing
        double[] result = null;
        if (algorithm == 0)
        {
            result = leastSquaresAlgorithm(line);
        }
        else if (algorithm == 1)
        {
            result = newtonsMethodAlgorithm(line);
        }
        // do some post-processing
        return result;
    }

    public double[] leastSquaresAlgorithm(double[] line)
    {
        return new double[] { 1.1, 2.2 };
    }

    public double[] newtonsMethodAlgorithm(double[] line)
    {
        return new double[] { 3.3, 4.4 };
    }
}
```

Idiomatic Improvement

- Eliminate 0 and 1 “magic numbers”
- Use constants
 - LeastSquares
 - NewtonsMethod

```
@Test
public void leastSquaresAlgorithm()
{
    solver = new MinimaSolver(MinimaSolver.LeastSquares);
    double[] result = solver.minima(line);
    assertTrue(result[0] == 1.1);
    assertTrue(result[1] == 2.2);
}
```

```
@Test
public void newtonsMethodAlgorithm()
{
    solver = new MinimaSolver(MinimaSolver.NewtonsMethod);
    double[] result = solver.minima(line);
    assertTrue(result[0] == 3.3);
    assertTrue(result[1] == 4.4);
}
```

```
public class MinimaSolver
{
    public static final int LeastSquares = 0;
    public static final int NewtonsMethod = 1;

    private int algorithm;

    public MinimaSolver(int algorithm)
    {
        this.algorithm = algorithm;
    }

    public double[] minima(double[] line)
    {
        // do some pre-processing
        double[] result = null;
        if (algorithm == LeastSquares)
        {
            return leastSquaresAlgorithm(line);
        }
        else if (algorithm == NewtonsMethod)
        {
            return newtonsMethodAlgorithm(line);
        }
        // do some post-processing
        return result;
    }
    ....
}
```


Enums

- Further enhance readability with enums.

```
@Test
public void leastSquaresAlgorithm()
{
    solver = new MinimaSolver(AlgorithmTypes.LeastSquares);
    double[] result = solver.minima(line);
    assertTrue(result[0] == 1.1);
    assertTrue(result[1] == 2.2);
}
```

```
@Test
public void newtonsMethodAlgorithm()
{
    solver = new MinimaSolver(AlgorithmTypes.NewtonsMethod);
    double[] result = solver.minima(line);
    assertTrue(result[0] == 3.3);
    assertTrue(result[1] == 4.4);
}
```

```
public class MinimaSolver
{
    public enum AlgorithmTypes
    {
        LeastSquares, NewtonsMethod
    }

    private AlgorithmTypes algorithm;

    public MinimaSolver(AlgorithmTypes algorithm)
    {
        this.algorithm = algorithm;
    }

    public double[] minima(double[] line)
    {
        // do some pre-processing
        double[] result = null;
        if (algorithm == AlgorithmTypes.LeastSquares)
        {
            return leastSquaresAlgorithm(line);
        }
        else if (algorithm == AlgorithmTypes.NewtonsMethod)
        {
            return newtonsMethodAlgorithm(line);
        }
        // do some post-processing
        return result;
    }
}
```

Introducing a new Algorithm

- Write the test first.
- It will fail to compile (as Bisection is not a valid enum)
- Once Bisection is introduced, the test will compile but fail to pass until we implement the method.

```
@Test
public void bisection()
{
    solver = new MinimaSolver(AlgorithmTypes.Bisection);

    double[] result = solver.minima(line);
    assertTrue(result[0] == 5.5);
    assertTrue(result[1] == 6.6);
}
```

New Algorithm - Bisection

- new enum for bisection.
- new method to implement the algorithm.
- new clause in the if statement to dispatch to bisection if selected.

```
public enum AlgorithmTypes
{
    LeastSquares, NewtonsMethod, Bisection
}
```

```
public double[] bisectionAlgorithm(double[] line)
{
    return new double[] { 5.5, 6.6 };
}
```

```
public double[] minima(double[] line)
{
    // do some pre-processing
    double[] result = null;
    if (algorithm == AlgorithmTypes.LeastSquares)
    {
        return leastSquaresAlgorithm(line);
    }
    else if (algorithm == AlgorithmTypes.NewtonsMethod)
    {
        return newtonsMethodAlgorithm(line);
    }
    else if (algorithm == AlgorithmTypes.Bisection)
    {
        return bisectionAlgorithm(line);
    }
    // do some post-processing
    return result;
}
```

```

public class MinimaSolver
{
    public enum AlgorithmTypes
    {
        LeastSquares, NewtonsMethod, Bisection
    }

    private AlgorithmTypes algorithm;

    public MinimaSolver(AlgorithmTypes algorithm)
    {
        this.algorithm = algorithm;
    }

    public double[] minima(double[] line)
    {
        // do some pre-processing
        double[] result = null;
        if (algorithm == AlgorithmTypes.LeastSquares)
        {
            return leastSquaresAlgorithm(line);
        }
        else if (algorithm == AlgorithmTypes.NewtonsMethod)
        {
            return newtonsMethodAlgorithm(line);
        }
        else if (algorithm == AlgorithmTypes.Bisection)
        {
            return bisectionAlgorithm(line);
        }
        // do some post-processing
        return result;
    }

    public double[] leastSquaresAlgorithm(double[] line)
    {
        return new double[] { 1.1, 2.2 };
    }

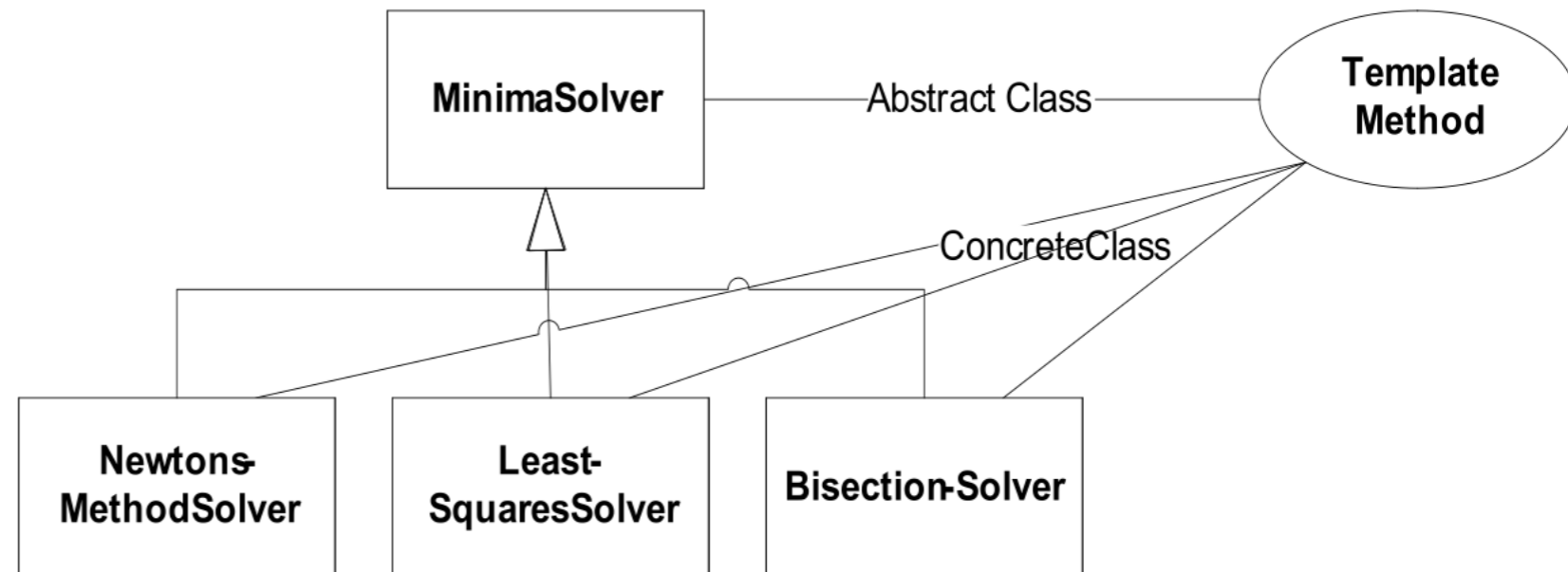
    public double[] newtonsMethodAlgorithm(double[] line)
    {
        return new double[] { 3.3, 4.4 };
    }

    public double[] bisectionAlgorithm(double[] line)
    {
        return new double[] { 5.5, 6.6 };
    }
}

```

Template Method Pattern

- Improve the maintainability of the solution.
- Enable new algorithms to be introduced without modifying the MinimaSolver class.
- Continue to improve readability



Template Method - Abstract Class

- Remove all of the algorithm implementations.
- Replace with an abstract method algorithm.
- Adjust minima method to invoke this algorithm.
- Mark class as abstract.

```
public abstract class MinimaSolver
{
    public MinimaSolver()
    {
    }

    double[] minima(double[] line)
    {
        // do some pre-processing
        double[] result = null;

        result = algorithm(line);

        // do some post-processing
        return result;
    }

    public abstract double[] algorithm(double[] line);
}
```

Template Method - Concrete Classes

- Algorithms extend MinimaSolver and implement algorithm method.

```
public class LeastSquaresSolver extends MinimaSolver
{
    public double[] algorithm(double[] line)
    {
        return new double[]{1.1, 2.2};
    }
}
```

```
public class NewtonsMethodSolver extends MinimaSolver
{
    public double[] algorithm(double[] line)
    {
        return new double[]{3.3, 4.4};
    }
}
```

Template Method Test

- Tests now instantiate appropriate MinimaSolver subclass - and test as before.

```
@Test
public void leastSquaresAlgorithm()
{
    solver = new LeastSquaresSolver();
    double[] result = solver.minima(line);
    assertTrue(result[0] == 1.1);
    assertTrue(result[1] == 2.2);
}

@Test
public void newtonsMethodAlgorithm()
{
    solver = new NewtonsMethodSolver();
    double[] result = solver.minima(line);
    assertTrue(result[0] == 3.3);
    assertTrue(result[1] == 4.4);
}
```


Template Method - Introducing new Algorithm

- Just define new class (no need to modify MinimaSolver base class)
- And test by instantiating an object of this class
- Contrast with previous mechanism for introducing new algorithm

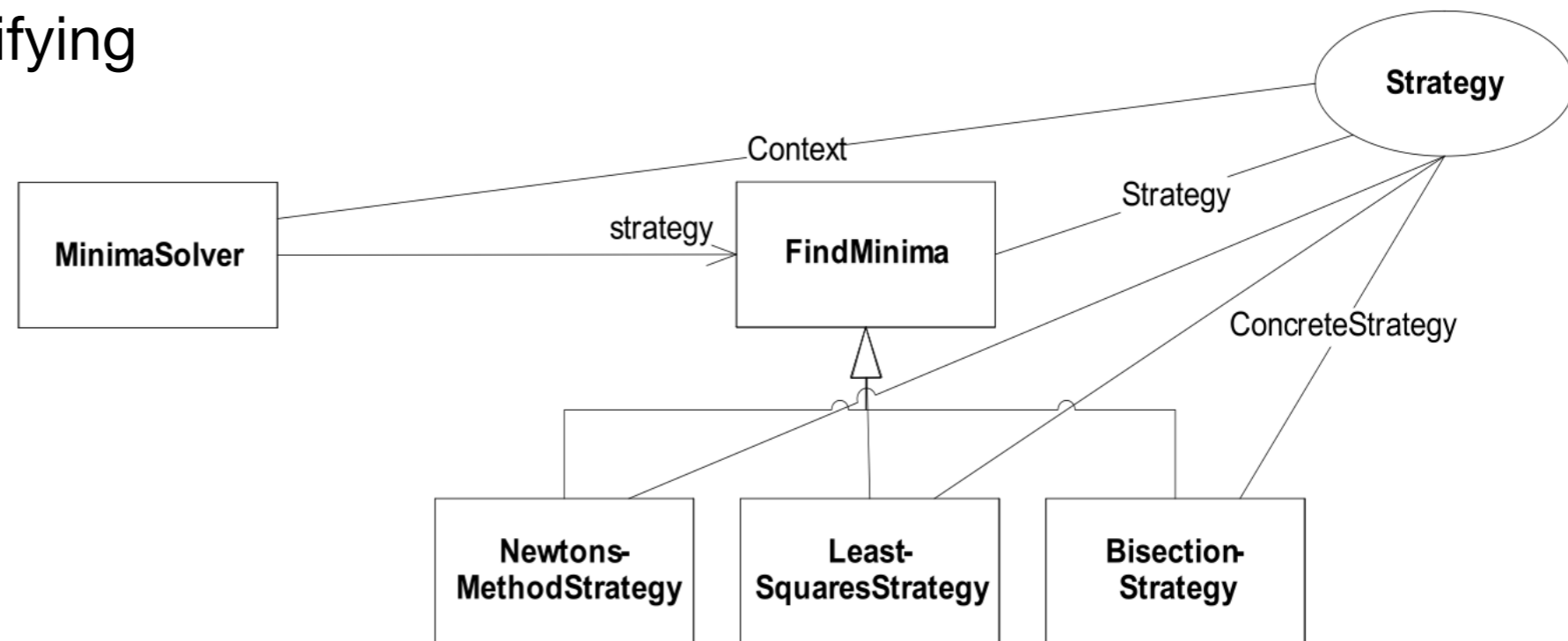
```
public class BisectionSolver extends MinimaSolver
{
    public double[] algorithm(double[] line)
    {
        return new double[]{5.5, 6.6};
    }
}
```

```
@Test
public void bisection()
{
    solver = new BisectionSolver();

    double[] result = solver.minima(line);
    assertTrue(result[0] == 5.5);
    assertTrue(result[1] == 6.6);
}
```

Strategy Pattern

- Improve the variability of the solution.
- Enable new algorithms to be introduced without modifying the MinimaSolver class.
- Continue to improve readability
- Enable algorithms to be changed at run time.



Strategy Interface and Concrete Implementations

- Encapsulate algorithm in an interface.
- Realise algorithms as standalone classes implementing this interface

```
public interface FindMinima
{
    double[] algorithm(double[] line);
}
```

```
public class LeastSquaresStrategy implements FindMinima
{
    public double[] algorithm(double[] line)
    {
        return new double[]{1.1, 2.2};
    }
}
```

```
public class NewtonsMethodStrategy implements FindMinima
{
    public double[] algorithm(double[] line)
    {
        return new double[]{3.3, 4.4};
    }
}
```

Strategy Context

- MinimaSolver will now be initialised with the appropriate strategy object.
 - by the constructor...
 - or by a changeStrategy() method...
- minima() delegates algorithm to strategy object.

```
public class MinimaSolver
{
    private FindMinima strategy;

    public MinimaSolver(FindMinima strategy)
    {
        this.strategy = strategy;
    }

    double[] minima(double[] line)
    {
        // do some pre-processing
        double[] result = null;

        result = strategy.algorithm(line);

        // do some post-processing
        return result;
    }

    public void changeStrategy(FindMinima newStrategy)
    {
        strategy = newStrategy;
    }
}
```

Strategy Test

- MinimaSolver + the appropriate Strategy object need to be created.

```
@Test
public void leastSquaresAlgorithm()
{
    solver = new MinimaSolver(new LeastSquaresStrategy());
    double[] result = solver.minima(line);
    assertTrue(result[0] == 1.1);
    assertTrue(result[1] == 2.2);
}

@Test
public void newtonsMethodAlgorithm()
{
    solver = new MinimaSolver(new NewtonsMethodStrategy());
    double[] result = solver.minima(line);
    assertTrue(result[0] == 3.3);
    assertTrue(result[1] == 4.4);
}
```

Defining a new Algorithm

- Just provide a new implementation of FindMinima...
- ... and pass an implementation of this to the solver.

```
public class BisectionStrategy implements FindMinima
{
    public double[] algorithm(double[] line)
    {
        return new double[]{5.5, 6.6};
    }
}
```

```
@Test
public void bisection()
{
    solver = new MinimaSolver(new BisectionStrategy());
    double[] result = solver.minima(line);
    assertTrue(result[0] == 5.5);
    assertTrue(result[1] == 6.6);
}
```

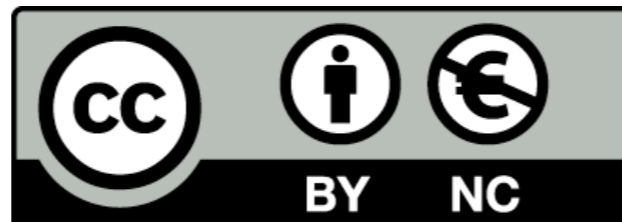
Changing the Strategy at Runtime

- Just call `changeStrategy()` with a new implementation.
- The same solver object is used in both tests here.
- This is not possible with Template Method.

```
@Test
public void testChangeAlgorithm()
{
    solver = new MinimaSolver(new LeastSquaresStrategy());

    double[] result = solver.minima(line);
    assertTrue(result[0] == 1.1);
    assertTrue(result[1] == 2.2);
    solver.changeStrategy(new BisectionStrategy());

    result = solver.minima(line);
    assertTrue(result[0] == 5.5);
    assertTrue(result[1] == 6.6);
}
```



Except where otherwise noted, this content is licensed under a Creative Commons Attribution-NonCommercial 3.0 License.

For more information, please see <http://creativecommons.org/licenses/by-nc/3.0/>

