Agile Software Development

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

eLearning support unit

Xtend Solver Example

Template Method

MinimaSolver

```java
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);
}
```

Java

```xtend
abstract class MinimaSolver
{
  new()
  {
  }

  def double[] minima(double[] line)
  {
    // do some pre-processing
    var double[] result = null
    result = algorithm(line)
    // do some post-processing
    result
  }

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

Xtend

MinimaSolver Algorithms

```java
public class BisectionSolver extends MinimaSolver
{
  public double[] algorithm(double[] line)
  {
    // Compute Minima on line
    //  - algorithm
    double x = 5.5;  // simulated result
    double y = 6.6;  // simulated result

    return new double[]{x, y};
  }
}
public class NewtonsMethodSolver extends MinimaSolver
{
  public double[] algorithm(double[] line)
  {
    // Compute Minima on line
    //  - algorithm
    double x = 3.3;  // simulated result
    double y = 4.4;  // simulated result

    return new double[]{x, y};
  }
}
```

Java

```
class BisectionSolver extends MinimaSolver
{
  override algorithm(double[] line)
  {
    // Compute Minima on line
    //  - algorithm
    val x = 5.5;  // simulated result
    val y = 6.6;  // simulated result
    #[x, y]
  }
}

class NewtonsMethodSolver extends MinimaSolver
{
  override algorithm(double[] line)
  {
    // Compute Minima on line
    //  - algorithm
    val x = 3.3;  // simulated result
    val y = 4.4;  // simulated result
    #[x, y]
  }
}
```

Xtend

MinimaSolver Test

```java
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 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);
  }

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

Java

```xtend
class MinimaSolverTest
{
  val line = #[ 1.0, 2.0, 1.0, 2.0,
                -1.0, 3.0, 4.0, 5.0, 4.0 ]
  var MinimaSolver solver

  @Test
  def newtonsMetod()
  {
    solver = new NewtonsMethodSolver
    val result = solver.minima(line)
    assertTrue(result.get(0) == 3.3)
    assertTrue(result.get(1) == 4.4)
  }

    @Test
  def leastSquares()
  {
    solver = new LeastSquaresSolver
    val result = solver.minima(line)
    assertTrue(result.get(0) == 1.1)
    assertTrue(result.get(1) == 2.2)
  }
  @Test
  def bisection()
  {
    solver = new BisectionSolver
    val result = solver.minima(line)
    assertTrue(result.get(0) == 5.5)
    assertTrue(result.get(1) == 6.6)
  }
}
```

Xtend

Strategy

MinimaSolver

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

```java
public class MinimaSolver
{
  private FindMinima strategy;

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

  double[] minima(double[] line)
  {
    double[] result = null;
   // do some pre-processing
    result = strategy.algorithm(line);
    // do some post-processing
    return result;
  }

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

Java

```xtend
public interface FindMinima
{
  def List<Double> algorithm(List<Double>line)
}
```

```xtend
class MinimaSolver
{
  @Property FindMinima findMinima

  new(FindMinima findMinima)
  {
    this.findMinima = findMinima
  }

  def double[] minima(double[] line)
  {
    // do some pre-processing
    val result = findMinima.algorithm(line)
    // do some post-processing
    result
  }
}
```

Xtend

MinimaSolver Algorithms

```java
public class BisectionStrategy implements FindMinima
{
  public double[] algorithm(double[] line)
  {
    // Compute Minima on line
    //  - algorithm
    double x = 5.5;  // simulated result
    double y = 6.6;  // simulated result

    return new double[]{x, y};
  }
}
```

```java
public class NewtonsMethodStrategy implements FindMinima
{
  public double[] algorithm(double[] line)
  {
    // Compute Minima on line
    //  - algorithm
    double x = 3.3;  // simulated result
    double y = 4.4;  // simulated result

    return new double[]{x, y};
  }
}
```

Java

```xtend
public class Bisection implements FindMinima
{
  override List<Double> algorithm(List<Double>line)
  {
    // Compute Minima on line
    //  - algorithm
    val x = 5.5;  // simulated result
    val y = 6.6;  // simulated result
    #[x, y]
  }
}
```

```xtend
public class NewtonsMethod implements FindMinima
{
  override List<Double> algorithm(List<Double>line)
  {
    // Compute Minima on line
    //  - algorithm
    val x = 3.3;  // simulated result
    val y = 4.4;  // simulated result
    #[x, y]
  }
}
```

Xtend

MinimaSolver Test

```java
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 leastSquares()
  {
    solver = new MinimaSolver(new LeastSquaresStrategy());
    double[] result = solver.minima(line);
    assertTrue(result[0] == 1.1);
    assertTrue(result[1] == 2.2);
  }

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

Java

```
class MinimaSolverTest
{
  val line         = #[ 1.0, 2.0, 1.0, 2.0,
                        -1.0, 3.0, 4.0, 5.0, 4.0 ]
  var MinimaSolver solver

  @Test
  def newtonsMethod()
  {
   solver = new MinimaSolver (new NewtonsMethod)
    val result = solver.minima(line)
    assertTrue(result.get(0) == 3.3)
    assertTrue(result.get(1) == 4.4)
  }

  @Test
  def leastSquares()
  {
    solver = new MinimaSolver (new LeastSquares)
    val result = solver.minima(line)
    assertTrue(result.get(0) == 1.1)
    assertTrue(result.get(1) == 2.2)
  }
```

Xtend

Strategy + Lambdas

MinimaSolver

```
class MinimaSolver
{
  public (List<Double>)=>List<Double> findMinima

  new((List<Double>)=>List<Double> findMinima)
  {
    this.findMinima = findMinima
  }

  def List<Double> minima(double[] line)
  {
    // do some pre-processing
    val result = findMinima.apply(line)
    // do some post-processing
    result
  }
}
```

MinmaSolver Algorithms

```
class Algorithms
{
  public val bisection     = [ List<Double> line |
                               // Compute Minima on line
                               //  - algorithm
                               val x = 5.5  // simulated result
                               val y = 6.6  // simulated result
                               #[x, y]
                             ]

  public val newtonsMethod = [ List<Double> line |
                               // Compute Minima on line
                               //  - algorithm
                               val x = 3.3  // simulated result
                               val y = 4.4  // simulated result
                               #[x, y]
                             ]

  public val leastSquares = [ List<Double> line |
                               // Compute Minima on line
                               //  - algorithm
                               val x = 1.1  // simulated result
                               val y = 2.2  // simulated result
                               #[x, y]
                             ]
}
```

Xtend

MinmaSolver Test

```
class MinimaSolverTest
{
  val line       = #[ 1.0, 2.0, 1.0, 2.0, -1.0, 3.0, 4.0, 5.0, 4.0 ]
  val algorithms = new Algorithms

  @Test
  def newtonsMethod()
  {
    var solver = new MinimaSolver (algorithms.newtonsMethod)
    val result = solver.minima(line)
    assertTrue(result.get(0) == 3.3)
    assertTrue(result.get(1) == 4.4)
  }

  @Test
  def leastSquares()
  {
    var solver = new MinimaSolver (algorithms.leastSquares)
    val result = solver.minima(line)
    assertTrue(result.get(0) == 1.1)
    assertTrue(result.get(1) == 2.2)
  }

  @Test
  def bisection()
  {
    var solver = new MinimaSolver (algorithms.bisection)
    val result = solver.minima(line)
    assertTrue(result.get(0) == 5.5)
    assertTrue(result.get(1) == 6.6)
  }
}
```

Lambda Collections

```
class Algorithms
{
  public val bisection     = [ List<Double> line |
                               // Compute Minima on line
                               //  - algorithm
                               val x = 5.5  // simulated result
                               val y = 6.6  // simulated result
                               #[x, y]
                             ]

  public val newtonsMethod = [ List<Double> line |
                               // Compute Minima on line
                               //  - algorithm
                               val x = 3.3  // simulated result
                               val y = 4.4  // simulated result
                               #[x, y]
                             ]

  public val leastSquares = [ List<Double> line |
                               // Compute Minima on line
                               //  - algorithm
                               val x = 1.1  // simulated result
                               val y = 2.2  // simulated result
                               #[x, y]
                             ]
```

```
@Test
def algorithmList()
{
  var List <(List<Double>)=>List<Double>> list = new ArrayList

  list.add(algorithms.bisection)
  list.add(algorithms.newtonsMethod)
  list.add(algorithms.leastSquares)

  for ((List<Double>)=>List<Double> algorithm : list)
  {
    algorithm.apply(line)
  }
}
```

Single Abstract Method (SAM) Conversion

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

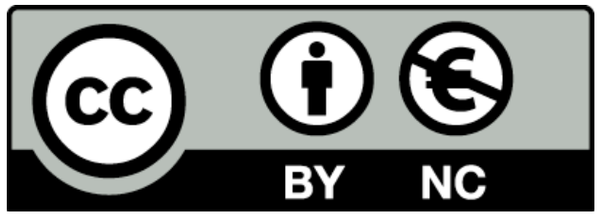
```
public class MinimaSolver
{
  private FindMinima strategy;

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

  double[] minima(double[] line)
  {
    double[] result = null;
   // do some pre-processing
    result = strategy.algorithm(line);
    // do some post-processing
    return result;
  }

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

```
val bisection  = [ List<Double> line |
                   // Compute Minima on line
                   //  - algorithm
                   val x = 5.5  // simulated result
                   val y = 6.6  // simulated result
                   #[x, y]
                 ]
@Test
def SAM()
{
  var solver = new MinimaSolver (bisection)
  val result = solver.minima(line)
  assertTrue(result.get(0) == 5.5)
  assertTrue(result.get(1) == 6.6)
}
```



Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit