Projektowanie obiektowe oprogramowania
Zestaw 6

Wzorce czynnościowe

2017-04-04

Liczba punktów do zdobycia: **6/42**
Zestaw ważny do: 2017-04-19

1. (**1p**) (**Null Object**) Należy zaprojektować własny podsystem logowania, obsługujący zapis do pliku, na konsoli i brak logowania. Klient używa fabryki (singletona) do wydobycia odpowiedniego loggera. Brak logowania obsługiwany jest przez obiekt typu `Null Object`.

```
public interface ILogger
{
   void Log( string Message );
}

public enum LogType { None, Console, File }

public class LoggerFactory
{
   public ILogger GetLogger( LogType LogType, string Parameters = null )
   { ... }

   public static LoggerFactory Instance { ... }
}

// klient:

ILogger logger1 = LoggerFactory.GetLogger( LogType.File, "C:\foo.txt" );
logger1.Log( "foo bar" ); // logowanie do pliku

ILogger2 logger = LoggerFactory.GetLogger( LogType.None );
logger2.Log( "qux" );     // brak logowania
```

2. (**2p**) (**Interpreter**) Dostarczyć implementacji interpretera wyrażeń logicznych.

   Wyrażenia oparte powinny być na prostej gramatyce przewidującej binarne operatory koniunkcji i alternatywy logicznej i unarny operator negacji.

   Tokeny mogą być literałami `true`, `false` lub nazwami zmiennych. Kontekstem interpretera jest funkcja zadająca wartościowanie pewnych zmiennych - dla nazwy zmiennej funkcja zwraca informację o jej wartości logicznej.

   Interpreter powinien poprawnie wyliczać wartości wyrażeń, w których wszystkie symbole terminalne (zmienne) mają swoje wartości w zadanym kontekście oraz wyrzucać wyjątek jeśli podczas interpretacji jakaś zmienna nie ma wartości.

```
public class Context
{
   public bool GetValue( string VariableName ) { ... }
```

```
   public bool SetValue( string VariableName, bool Value ) { ... }
}

public abstract class AbstractExpression
{
   public abstract Interpret( Context context );
}

public class ConstExpression  : AbstractExpression { ... }
public class BinaryExpression : AbstractExpression { ... }
public class UnaryExpression  : AbstractExpression { ... }

// klient
Context ctx = new Context();
ctx.SetValue( "x", false );
ctx.SetValue( "y", true );

AbstractExpression exp = ....; // jakieś wyrażenie logiczne ze stałymi i zmiennymi

bool Value = exp.Interpret( context );
```

3. (**1+1p**) (**Visitor**) Dostarczyć implementacji visitorów dla drzewa binarnego `AbstractTree`, wyznaczających głębokość drzewa.

   Przechodzenie struktury drzewa powinno być zaimplementowane na dwa sposoby - w strukturze drzewa albo w strukturze visitorów, jak pokazano na wykładzie

4. (**1p**) (**Visitor**) Dostarczyć implementację takiego przeciążenia klasy `ExpressionVisitor` z biblioteki standardowej C#, które pokazuje na ekranie konsoli sformatowany obraz drzewa rozbioru wyrażenia typu `System.Linq.Expressions.Expression`.

   Uwaga, dla uproszczenia wystarczy obsłużyć węzły typu `BinaryExpression` i `LambdaExpression`. W praktyce oznacza to przeciążenie metod `VisitBinary` i `VisitLambda`.

   Przykładowe wywołanie

```
Expression<Func<int, bool>>  exp = x => x * 7 + 1 > 6;
ExampleExpressionVisitor visitor = new ExampleExpressionVisitor();

visitor.Visit(exp);
```

Wiktor Zychla