Assignment 1: Detecting (Anti-)Patterns

Milan Lagae

Institution/University Name:	Vrije Universiteit Brussel
Faculty:	Sciences and Bioengineering Sciences
Course:	Software Quality Analysis

Contents

1 Intro

The assigned pattern for this assignment is the Template Method pattern as described in [1] and [2]. All indivual complete methods can be found in the Appendix section §4.

2 Discussion Point 1

This section will discuss the creating of a logic query in Ekeko for Discussion Point 1, executed on the DesignPatterns folder.

2.1 Abstract & Extending

The first step in the logic query is retrieving all abstract & extending classes (get-abstract-and-extending-classes), the complete method can be found in Listing 15.

The first step is retrieving all the ast nodes of type :TypeDeclaration and define this as the extending class. For each ast node, retrieve the type of the declaration and check if the ast-node is a class-type, shown in Listing 1.

```
(ast :TypeDeclaration ?extending)
(typedeclaration-type ?extending ?type)
(type|class ?type)
```

Listing 1: Extending class.

After the type of the ast-node is known to be a class-type, check that the extending class is not defined as an abstract class, Listing 2.

```
; The extending class cannot both be abstract & have a super type
(has :modifiers ?extending ?modifiers-extending)
(modifier|abstract ?mod-abstract)
(fails (contains ?modifiers-extending ?mod-abstract))
```

Listing 2: Modifiers extending class.

2.2 Checking methods

The list of abstract and extending class, is refined by filtering the results based on methods defined on the pair of abstract & extending class. Filtering is performed on both classes, this action is done by the check-methods method, which can be found in full in Listing 16.

The core of the method is surrounded by the logic operations: one & all, declaring that one valid result is required and all perations must succeed for all declared. The list of method declarations in a class is retrieved by using the logic query: typedeclaration-method, it retrieves all the :bodyDeclarations (methods) in the body of the given class, it can be found in Listing 17

2.2.1 Abstract method

Validating the retrieved method is an abstract method is the next step in finding Template Method patterns, the complete method can be found in Listing 18.

2.2.1.1 Method Body

According to the pattern definition, the body of an abstract method must be empty. Java allows for multiple definitions of abstract methods as illustrated in Listing 3. Detecting these two different examples must be managed and are explained in Listing 4.

```
public abstract methodDefinition();
// or
public abstract methodDefinition() {};
```

Listing 3: Java method examples

The first type of empty body can be detected in a Ekeko logic query by checking if the body property of a method has the null type.

The other type of empty body declaration requires a bit more code. Starting with a fails check, the first case is excluded. Following that, the statements property is retrieved from the body of the method, which is the list of expressions inside the body of the method. The list of statements is transformed to its raw value using: value-raw. Finally, the list of statements is checked to be empty: (equals 0 (count ?raw)).

```
; Get method body
(has :body ?abstract-method ?abstract-body)
; Method body is null or has no statements
(conde
  [(value|null ?abstract-body)]
  [
   (fails (value|null ?abstract-body))
   (has :statements ?abstract-body ?stmts)
   (value-raw ?stmts ?raw)
   (equals 0 (count ?raw))])

```

Listing 4: Abstract body must be empty.

2.2.1.2 Method Modifiers

Continuining from the body of the methods, the next part is checking the list of modifiers on the respective methods in Listing 5. Following the pattern definition, there are two scenarios of possible modifier combinations for abstract methods:

- public & abstract
- protected.

The list of modifiers is retrieved, and the two scenarios are detected by using a conde.

```
; Modifiers
(has :modifiers ?abstract-method ?modifiers)
; Abstract method is public + abstract or protected
(conde
  [
   (modifier|public ?mod-public)
   (contains ?modifiers ?mod-public)
   (modifier|abstract ?mod-abstract)
   (contains ?modifiers ?mod-abstract)]
  [
   (modifier|protected ?mod-protected)
   (contains ?modifiers ?mod-protected)])

```

Listing 5: Abstract method, required modifiers.

That concludes the code required for checking if a method is an abstract method.

2.2.2 Overriding method

For each abstract method there is an associated overriding method, detecting such a method is done by the method: is-overriding-method, the full Implementation can be found in Listing 19.

Retrieving the connected overriding method with each abstract method is done by using the built Ekeko logic query: (methoddeclaration-methoddeclaration|overrides ?abstract-method ?overrider-method). For either grounded value, Ekeko will retrieve the accompaying method, in this case the overrider-method.

2.2.2.1 Class

Each retrieved method is checked so that it belongs to the extending class found earlier. For this the typedeclaration-method defined earlier is reused, code can be found in Listing 6.

```
; Get the overrider method from the abstract-method
(methoddeclaration-methoddeclaration|overrides ?abstract-method ?overrider-method)
; Check if parent class is the same as the extending class of the abstract
(typedeclaration-method ?extending ?overrider-method)
```

Listing 6: Retrieving overrider method from abstract method.

2.2.2.2 Method Body

For each overriding method, it's body cannot be empty or null, the code for this can be found in Listing 7. Retrieve the body of the method using the :body property value. After the body is returned, check the body is not null by using a fails.

```
; Overriding method body cannot be empty
(ast :MethodDeclaration ?overrider-method)
(has :body ?overrider-method ?overrider-body)
(fails (value|null ?overrider-body))
```

Listing 7: Checking body of overriding method.

2.2.2.3 Method Modifiers

The final check for the overriding method is checking the modifiers defined on the method, Listing 8. Retrieve the list of modifiers from the method using the :modifiers property value. Get the public modifier value from the modifier|public Ekeko built-in method. After that use contains to check for the presence of the public modifier in the list of modifiers.

```
; Overrider is public
(has :modifiers ?overrider-method ?modifiers)
(modifier|public ?mod-public)
(contains ?modifiers ?mod-public)
```

Listing 8: Checking list of modifiers on the overriding method.

2.2.3 Overrider & Template Method

This step checks if the overrider-method is present in the template-method of the abstract-class. Surround the query with a one for the ?template-method value.

Retrieve a method decleration from the abstract class, first check if the name is not the same as the abstract-method retrieved earlier. This is done by retrieving the :name property on both respective methods. Proceed to check if comparison of name fails.

If that is the case, check if the method is a template method using is-template-method, the full code of the method can be found in Listing 21.

```
; Check if the overrider method is present
; in the template method of the abstract class
(one (fresh [?template-method]
     (typedeclaration-method ?abstract ?template-method)
     ; Template method does not equal abstract-method
     (has :name ?template-method ?template-name)
     (has :name ?abstract-method ?abstract-name)
     (fails (name|simple-name|simple|same ?template-name ?
abstract-name))

     ; Is template method
     (is-template-method ?template-method ?template-body)

     (is-algorithm-step ?template-body ?overrider-method)
    ))
```

Listing 9: Validating template method & overrider method.

2.2.3.1 Template Method

The pattern defintion declares a 'Template Method' to adhere to the following properties: must be a public method, no other modifiers are allowed; the body of the method must also not be empty. Since this is the method that defines the steps of the algorithm, which are the abstract methods defined earlier, and which are later overriden by methods in the extending class. The name of the implementation method is: is-template-method and can be found in full in Listing 21.

The first step of checking if a given method is a template method is by validating the list of modifiers, the method must be public and cannot contain any other modifiers, code can be found in Listing 10.

```
; Public and non abstract
(has :modifiers ?template-method ?modifiers)

; Public method
(modifier|public ?mod-public)
(contains ?modifiers ?mod-public)
; Must be public method only
(value-raw ?modifiers ?raw)
(equals 1 (count ?raw))
```

Listing 10: Modifier check on the template method.

Retrieve the :modifiers property from the method using the has query. Retrieving a value for the public modifier using modifier|public to than check using contains if the value is present in the list of modifiers. The length of the list of modifiers is also checked to see if it's length is 1, e.g: only contains the public modifier.

After the modifiers are checked, validating if the body of the method is not empty is in Listing 11.

```
; Non empty body
(has :body ?template-method ?template-body)
(fails (value|null ?template-body))
```

Listing 11: Template mehod non-empty body.

Validating if a method is a template method in the sense is not complete without the next method named: is-algorithm-step.

2.2.4 Algorithm Step

Pattern definition declares that there must be at least one template method per pattern implementation [2]. Thus the purpose of this method named: is-algorithm-step which can be found in full in Listing 20, is to check the supposed template method for presence of each of the abstract<=>overriding method pair.

The method starts of with retrieving the name of the overrider-method method, the :statements property is also retrieved from the template-body, as illustrated in Listing 12.

```
; Overrider method name
(has :name ?overrider-method ?overrider-name)
; Extracts rhs method invocation from the body of the template
method
(has :statements ?template-body ?stmts)
```

Listing 12: Method name & body statements.

Each statement in the list of statements is than iterated using contains, for each, the name of the invocation expression is compared with the name of the given overrider-name. The list of queries is surrouned with a one, indicating that at least one match for that particular method must be found in the body of the template method.

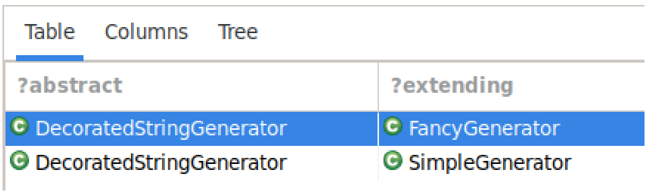
```
; We need to at least have one match
(one (fresh  [?stmt ?expression ?rhs ?inv-name]
        ; Iterate each stmt
        (contains ?stmts ?stmt)
        ; Extract name
        (has :expression ?stmt ?expression)
        (has :rightHandSide ?expression ?rhs)
        (has :name ?rhs ?inv-name)

        ; 'Compare' with overrider method name
        (name|simple-name|simple|same ?overrider-name ?inv-name)
        )
```

Listing 13: Comparing method name & invocation name in method body.

This completes the list of methods used for querying the Template Method pattern on the DesignPatterns folder.

2.3 Results

Executing the query as defined above, results in the following result defined in Figure 1.



?abstract	?extending
DecoratedStringGenerator	FancyGenerator
DecoratedStringGenerator	SimpleGenerator

Figure 1: Discussion Point 1 - Result

All the expected results for the Template Method pattern in the DesignPatterns folder are present in the result of the query.

3 Discussion Point 2

Executing the query defined in Discsussion Point 1, without modification on the JhotDraw folder, results in no results. To gather more results, part of the query was removed/ commented out.

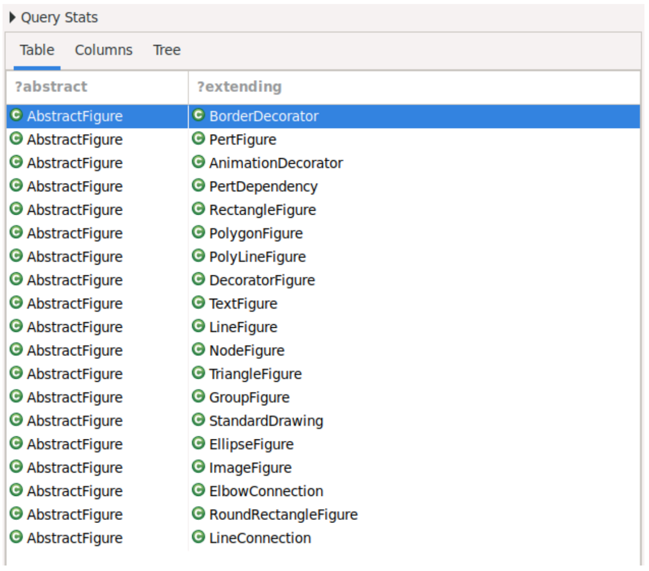
3.1 Comparison

First start by removing the comparison query line in the is-algorithm-step method, namely: name|simple-name|simple|same, which can be found in Listing 14.

```
; We need to at least have one match
(one (fresh  [?stmt ?expression ?rhs ?inv-name]
        (...)

        ; 'Compare' with overrider method name
        (name|simple-name|simple|same ?overrider-name ?inv-name))
```

Listing 14: Comparing method name & invocation name in method body.

Executing the remaining query which this line removed results in the following as defined in Figure 2.



?abstract	?extending
AbstractFigure	BorderDecorator
AbstractFigure	PertFigure
AbstractFigure	AnimationDecorator
AbstractFigure	PertDependency
AbstractFigure	RectangleFigure
AbstractFigure	PolygonFigure
AbstractFigure	PolyLineFigure
AbstractFigure	DecoratorFigure
AbstractFigure	TextFigure
AbstractFigure	LineFigure
AbstractFigure	NodeFigure
AbstractFigure	TriangleFigure
AbstractFigure	GroupFigure
AbstractFigure	StandardDrawing
AbstractFigure	EllipseFigure
AbstractFigure	ImageFigure
AbstractFigure	ElbowConnection
AbstractFigure	RoundRectangleFigure
AbstractFigure	LineConnection

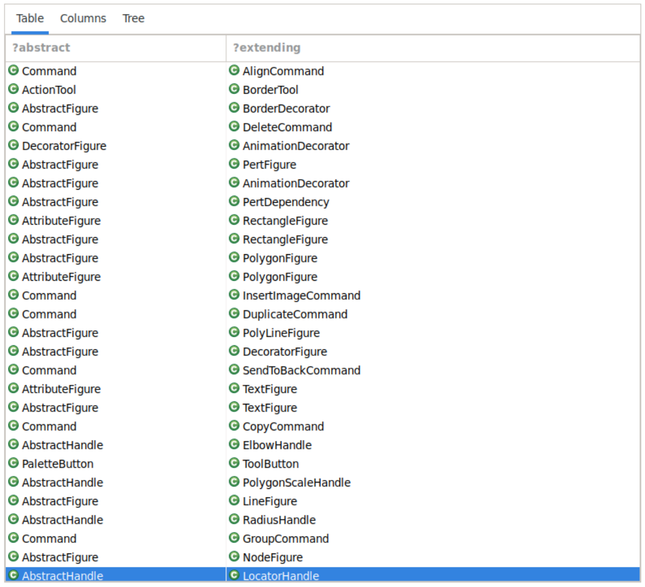
Figure 2: Discussion Point 2 - Comparison Removed Result

The result already includes more of the expected patterns. When comparing with the expected results as defined in the xml file, all results regarding the: AttributeFigure are missing, and for the results regarding AbstractFigure, there is **1** false positive: DecoratorFigure and 3 missing implementations (DiamondFigure, NumberTextFigure, BouncingDrawing).

3.2 Is Algorithm Step

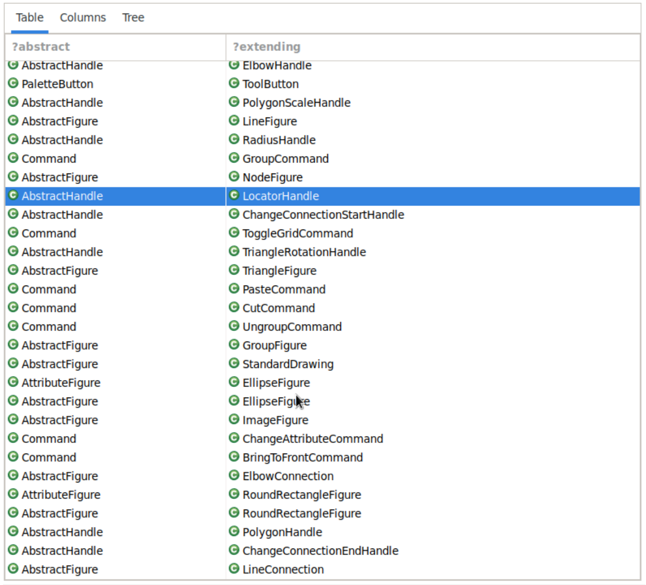
The purpose of the is-algorithm-step method was to check for the template-method in the abstract class of the pattern to check for the overrider methods that are defined as the steps of the algorithm as explained in [2].

When removing the calling of the is-algorithm-step method from the logic query, more results are show, but include more false/positives. The results can be seen in Figure 3 & Figure 4.



?abstract	?extending
Command	AlignCommand
ActionTool	BorderTool
AbstractFigure	BorderDecorator
Command	DeleteCommand
DecoratorFigure	AnimationDecorator
AbstractFigure	PertFigure
AbstractFigure	AnimationDecorator
AbstractFigure	PertDependency
AttributeFigure	RectangleFigure
AbstractFigure	RectangleFigure
AbstractFigure	PolygonFigure
AttributeFigure	PolygonFigure
Command	InsertImageCommand
Command	DuplicateCommand
AbstractFigure	PolyLineFigure
AbstractFigure	DecoratorFigure
Command	SendToBackCommand
AttributeFigure	TextFigure
AbstractFigure	TextFigure
Command	CopyCommand
AbstractHandle	ElbowHandle
PaletteButton	ToolButton
AbstractHandle	PolygonScaleHandle
AbstractFigure	LineFigure
AbstractHandle	RadiusHandle
Command	GroupCommand
AbstractFigure	NodeFigure
AbstractHandle	LocatorHandle

Figure 3: Discussion Point 2 - Result 1



?abstract	?extending
AbstractHandle	ElbowHandle
PaletteButton	ToolButton
AbstractHandle	PolygonScaleHandle
AbstractFigure	LineFigure
AbstractHandle	RadiusHandle
Command	GroupCommand
AbstractFigure	NodeFigure
AbstractHandle	LocatorHandle
AbstractHandle	ChangeConnectionStartHandle
Command	ToggleGridCommand
AbstractHandle	TriangleRotationHandle
AbstractFigure	TriangleFigure
Command	PasteCommand
Command	CutCommand
Command	UngroupCommand
AbstractFigure	GroupFigure
AbstractFigure	StandardDrawing
AttributeFigure	EllipseFigure
AbstractFigure	EllipseFigure
AbstractFigure	ImageFigure
Command	ChangeAttributeCommand
Command	BringToFrontCommand
AbstractFigure	ElbowConnection
AttributeFigure	RoundRectangleFigure
AbstractFigure	RoundRectangleFigure
AbstractHandle	PolygonHandle
AbstractHandle	ChangeConnectionEndHandle
AbstractFigure	LineConnection

Figure 4: Discussion Point 2 - Result 2

Analysing the results when removing the method, is that more of the expected results are included, but also more false positives are included, mainly: Command, AbstractHandle, PaletteButton as the abstract classes, and their corresponding extending classes.

No further refined of the query was performed for discussion point 2.

4 Appendix

4.1 Discussion Point 1

clj

```clojure
; Abstract and extending classess
(defn get-abstract-and-extending-classes [?abstract ?extending]
 (fresh [?type ?supertype ?modifiers-abstract ?mod-abstract ?modifiers-extending]

   ; Intersection of abstract and extending class
   (ast :TypeDeclaration ?extending)
   (typedeclaration-type ?extending ?type)
   (type|class ?type)

   ; The extending class cannot both be abstract & have a super type
   (has :modifiers ?extending ?modifiers-extending)
   (modifier|abstract ?mod-abstract)
   (fails (contains ?modifiers-extending ?mod-abstract))

   ; Get the supertype of the type
   (type-type|super+ ?type ?supertype)

   ; Abstract class
   (type|class ?supertype)
   (typedeclaration-type ?abstract ?supertype)
   (ast :TypeDeclaration ?abstract)

   ; Abstract modifier
   (has :modifiers ?abstract ?modifiers-abstract)
   (contains ?modifiers-abstract ?mod-abstract)))
```

Listing 15: Method that retrieves a list of abstract and extending classes.

clj

```clojure
(defn check-methods [?abstract ?extending]
 (fresh [?overrider-body ?overrider-method ?template-method ?abstract-method
        ?template-body ?template-name ?abstract-name]

  (one (all
      ; Gets all the childs nodes of abstract class
      (typedeclaration-method ?abstract ?abstract-method)
      ; Abstract method
      (is-abstract-method ?abstract-method)

      ; Is overrider method?
      (is-overriding-method ?abstract-method ?extending ?overrider-method)

      ; Check if the overrider method is present
      ; in the template method of the abstract class
      (one (fresh [?template-method]
           (typedeclaration-method ?abstract ?template-method)
           ; Template method does not equal abstract-method
           (has :name ?template-method ?template-name)
           (has :name ?abstract-method ?abstract-name)
           (fails (name|simple-name|simple|same ?template-name ?abstract-name))

           ; Is template method
           (is-template-method ?template-method ?template-body)

           (is-algorithm-step ?template-body ?overrider-method))))))
```

Listing 16: Method responsible for checking method declerations on the pair of abstract and extending class.

clj

```clojure
; Method taken from WPO, get all method declerations
(defn typedeclaration-method [?class ?method]
 (child :bodyDeclarations ?class ?method))
```

Listing 17: Retrieves all method declerations from a given class.

clj

```clojure
  ; Is abstract method?
(defn is-abstract-method [?abstract-method]
 (fresh [?abstract-body ?modifiers ?mod-abstract ?mod-public ?mod-protected ?stmts ?raw]
   ; Method of the abstract class
   (ast :MethodDeclaration ?abstract-method)

   ; Get method body
   (has :body ?abstract-method ?abstract-body)
   ; Method body is null or has no statements
   (conde
     [(value|null ?abstract-body)]
     [(fails (value|null ?abstract-body))
      (has :statements ?abstract-body ?stmts)
      (value-raw ?stmts ?raw)
      (equals 0 (count ?raw))])


   ; Modifiers
   (has :modifiers ?abstract-method ?modifiers)
   ; Abstract method is public + abstract or protected
   (conde
     [(modifier|public ?mod-public)
      (contains ?modifiers ?mod-public)
      (modifier|abstract ?mod-abstract)
      (contains ?modifiers ?mod-abstract)]
     [(modifier|protected ?mod-protected)
      (contains ?modifiers ?mod-protected)])))
```

Listing 18: Checks if the given method is an abstract method.

clj

```clojure
  (defn is-overriding-method [?abstract-method ?extending ?overrider-method]
    (fresh [?overrider-body ?abstract-body ?modifiers ?mod-public]
      ; Get the overrider method from the abstract-method
      (methoddeclaration-methoddeclaration|overrides ?abstract-method ?overrider-method)
      ; Check if parent class is the same as the extending class of the abstract
      (typedeclaration-method ?extending ?overrider-method)

      ; Overriding method body cannot be empty
      (ast :MethodDeclaration ?overrider-method)
      (has :body ?overrider-method ?overrider-body)
      (fails (value|null ?overrider-body))

      ; Overrider is public
      (has :modifiers ?overrider-method ?modifiers)
      (modifier|public ?mod-public)
      (contains ?modifiers ?mod-public)))
```

Listing 19: Checks if the given method is an overriding method.

clj

```clojure
; Check with algorithm step in template-body
(defn is-algorithm-step [?template-body ?overrider-method]
  (fresh [?stmts ?overrider-name]

    ; Overrider method name
    (has :name ?overrider-method ?overrider-name)
    ; Extracts rhs method invocation from the body of the template method
    (has :statements ?template-body ?stmts)

    ; We need to at least have one match
    (one (fresh  [?stmt ?expression ?rhs ?inv-name]
                ; Iterate each stmt
                (contains ?stmts ?stmt)
                ; Extract name
                (has :expression ?stmt ?expression)
                (has :rightHandSide ?expression ?rhs)
                (has :name ?rhs ?inv-name)

                ; 'Compare' with overrider method name
                (name|simple-name|simple|same ?overrider-name ?inv-name)))))
```

Listing 20: Checks the body of the template-method to see if the overrider method is invoced in the body of the method.

clj

```clojure
; Is template method?
(defn is-template-method [?template-method ?template-body]
  (fresh [?mod-abstract ?mod-public ?modifiers ?raw]
    (ast :MethodDeclaration ?template-method)

    ; Public and non abstract
    (has :modifiers ?template-method ?modifiers)

    ; Public method
    (modifier|public ?mod-public)
    (contains ?modifiers ?mod-public)
    ; Must be public method only
    (value-raw ?modifiers ?raw)
    (equals 1 (count ?raw))

    ; Non empty body
    (has :body ?template-method ?template-body)
    (fails (value|null ?template-body))))
```

Listing 21: Checks if the given body is a template method.

Bibliography

[1] [Online]. Available: https://en.wikipedia.org/wiki/Template_method_pattern

[2] R. J. J. V. Erich Gamma Richard Helm, *Design Patterns Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1994.