A Solution to the Square-Rectangle Problem Within the Framework of Object Morphology

Zbyněk Šlajchrt1

1 Department of Information Technologies, Faculty of Informatics and Statistics,

University of Economics, Prague

W. Churchill Sq. 4, 130 67 Prague 3

zslajchrt@gmail.com

**Abstract:** Insert abstract in English here. Please follow the instructions for authors and respect maximum abstract length, which it should be between 100 – 200 words in length. Do not cite references to literature in the abstract.

**Keywords:** Max seven keywords or phrases state here, separated by commas.

# Introduction

The square-rectangle problem (SRP) exposes a couple of flaws inherent to object-oriented programming (OOP). In particular, the problem is closely related to subtyping and inheritance and manifests itself as a violation of the Liskov substitution principle (LSU).

To solve SRP, one attempts to determine the proper subtyping, resp. inheritance relationship between the rectangle and the square. In mathematics, a square is a degenerated version of a rectangle having the same width and height. It follows that a square **is a** rectangle and thus the “IS-A” relationship should be used to model the relationship between squares and rectangles; in other words the square inherits from the rectangle (or the square is a subtype of the rectangle).

If the rectangle class exposes methods for setting the width and height of a rectangle then also the square class automatically inherits those methods. Given that according to the Liskov substitution principle squares are substitutable for rectangles, any square may be used in any routine processing rectangles, including the routines modifying the state of the processed rectangles using the public width and height setter methods. However, in case those setters are invoked on a square the resulting state of the square may be in violation of the constraint stipulating that the width and the height of a square must be the same.

This problem exemplifies a more general modeling problem, in which a class hierarchy consists of a base class and of subclasses that are constrained versions of the base class. The point here is that some methods of the base class can modify the object’s state so that the modified state violates the constraints of a subclass. Such a modeling situation may occur quite often and may lead to serious design-related issues in the final application as long as the problem is not identified and solved early in the modeling phase of the development.

Although the problem looks quite simple at first sight, its solution is harder than it looks [Shang, Martin&Ottinger]. There are a number of possible solutions of this problem and some of them are described in the literature review section. Broadly speaking, all those solutions do not solve the problem comprehensively and often introduce secondary problems.

The purpose of this paper is to present a novel approach to the above-mentioned problem based on a conceptual framework called object morphology (OM), which is being developed by the author’s doctoral thesis [X]. In OM an object is in principle a mutable entity, while the mutation does not affect the object’s state only, but also its type. There are no class or type hierarchies in OM; instead, objects are modeled by the so-called morph models describing permitted structural mutations of the objects.

The presented approach entirely shifts the view on the constraint violations. In this view, a violation of constraints caused by the mutated object’s state is no longer considered a necessarily faulty situation; instead the violation may simply be interpreted as a signal to mutate the object to a different form (type), with which the mutated state does not conflict. To put it in the context of SRP, a violation of the square constraints caused by setting the width and the height properties to different values would lead to mutating the square instance into a rectangle, while preserving the identity of the object.

The solution is demonstrated on an extended version of SRP using Morpheus, a proof-of-concept implementation of OM in the Scala language.

This paper focuses on the conceptual side of the problem, leaving aside other important aspects of the solutions, such as performance, mainly on account of the fact that Morpheus is still in the proof-of-concept stage.

# Related works

A number of attempts to solve SRP have been made so far. While some could be described rather as “quick fixes” the other delve deeper to the core of the problem.

In [Wiki-ECP] a couple of simple possible solutions try to get rid of the problem by changing the contract associated with the modifying methods that cause constraint violations. One solution suggests raising an exception from the problematic method in the affected subclass. A similar solution alters the previous one so that instead of throwing an exception the modifier would return the modified state. Provided that the inherited modifier cannot change the state due to a possible violation of the constrains it would return the unchanged state. A client of this modifier would acknowledge the contract that the returned data represents the actual effect of the modifier invocation. Another solution proposes imposing preconditions on modifiers; this approach is usable in the languages with support for method preconditions, such as Eiffel, Sather, Transframe [Eiff, Sather, Trans].

While the above-mentioned solutions remain at the method level, other solutions approach the problem from a structural point of view. One of such solutions is based on a popular rule, according to which a concrete class must be derived from an abstract class and never from a concrete one [Meyer, Martin&Ottinger, Grosberg]. Using this rule there would an abstract ancestor of both the square and the rectangle classes containing the “harmless” methods only, such as read-only methods. However, this approach may result in complicated hierarchies and may also break encapsulation of data declared in the abstract ancestor as explained in [Kazimir].

Other solutions address mutability of classes either by avoiding mutability completely or by factoring out modifiers to other classes [Wiki-ECP].

A more profound approach can be found in [Kazimir]. The authors come with an interesting concept of inverse inheritance, according to which certain methods defined in a subclass, which represents a subset of the set represented by its superclass, would be automatically and implicitly (inversely) inherited by its superclass. And conversely, only certain methods defined in the superclass would be automatically inherited by the subclass. A typical example of an inversely inherited method is a method assigning a new state to the inherited properties. Since such a method is defined in the subclass it may be assumed that the method does not corrupt the constraints of the subclass. It follows, however, that this same method would work in the superclass and thus it may be inversely inherited by the superclass.

In [Boult] the author present the concept of dynamic attributes. It separates inheritance of object state from inheritance of storage, which leads to more flexible object derivations. Each object property is associated with one dynamic attribute, which comes in several forms differing in how they store or retrieve the value etc. Importantly, the form of an attribute in a subclass may differ from the form of the same attribute in the superclass. The author demonstrates the usefulness of dynamic attributes on the eclipse-circle problem, which is just another version of CRP.

The common ground of the above-mentioned solutions is that they consider violations of constrains as an error. In contrast to them the solution presented in this paper is based on the assumption that a constraint violation may be a signal to mutate the object so that the new state no longer violates any constraint.

[Boult]: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.52.1584&rep=rep1&type=pdf>

[Kazimir]: http://kazimirmajorinc.com/Documents/1998,-Majorinc,-Ellipse-circle-dilemma-and-inverse-inheritance.pdf

# Research method

The solution of SRP described in the paper is developed in the framework of object morphology, which is a novel approach to object-oriented modeling. Therefore, in order to make the explanation of the solution method clearer, a brief introduction to OM is given.

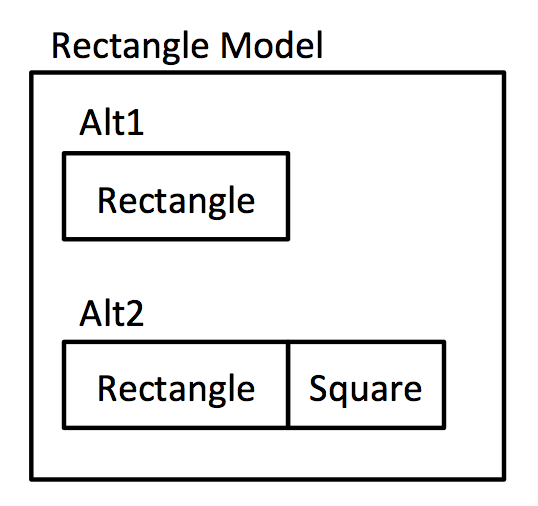
## Object morphology

Object morphology is a general approach to modeling primarily the so-called protean objects. A *protean object* is a term referring to a phenomenon occurring in a multitude of forms and defying the traditional Aristotelian class-based categorization [Madsen]. The concepts (abstractions) of such objects may often be only loosely defined, e.g. by means of family resemblance rather than by specifying strict rules for class membership.

Examples are fetal development, insect metamorphosis, phase transitions, autopoietic (self-maintaining and self-reproducing) systems such as cells, roles in society, crisis and other biological, social or economic phenomena [Some refs from diss.].

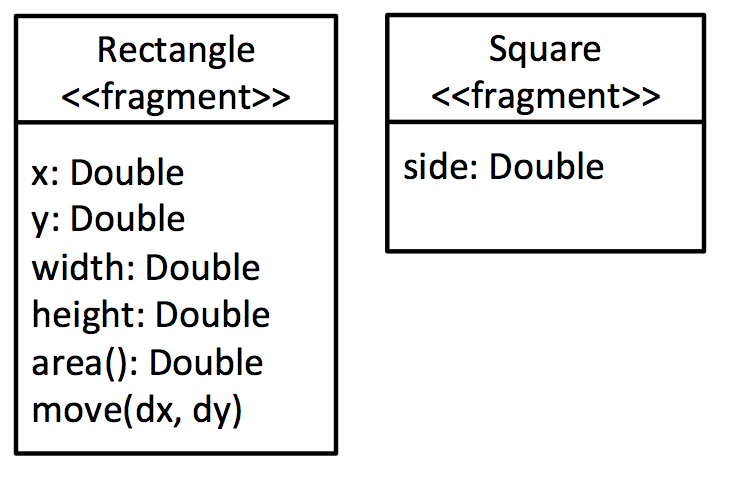
Instead of building type or class hierarchies, protean objects are modeled through the construction of morph models describing the forms that the protean objects may assume. The individual forms are called morph alternatives.

In its essence, a morph model is an abstraction (or concept) of related protean objects. The individual alternatives in the model are in fact the abstractions of the prototypical or exemplary instances among the abstracted protean objects (i.e. the concept’s extension). Each alternative consists of the so-called fragments, which represent properties or features of the protean objects (i.e. the concept’s intension) [Madsen].

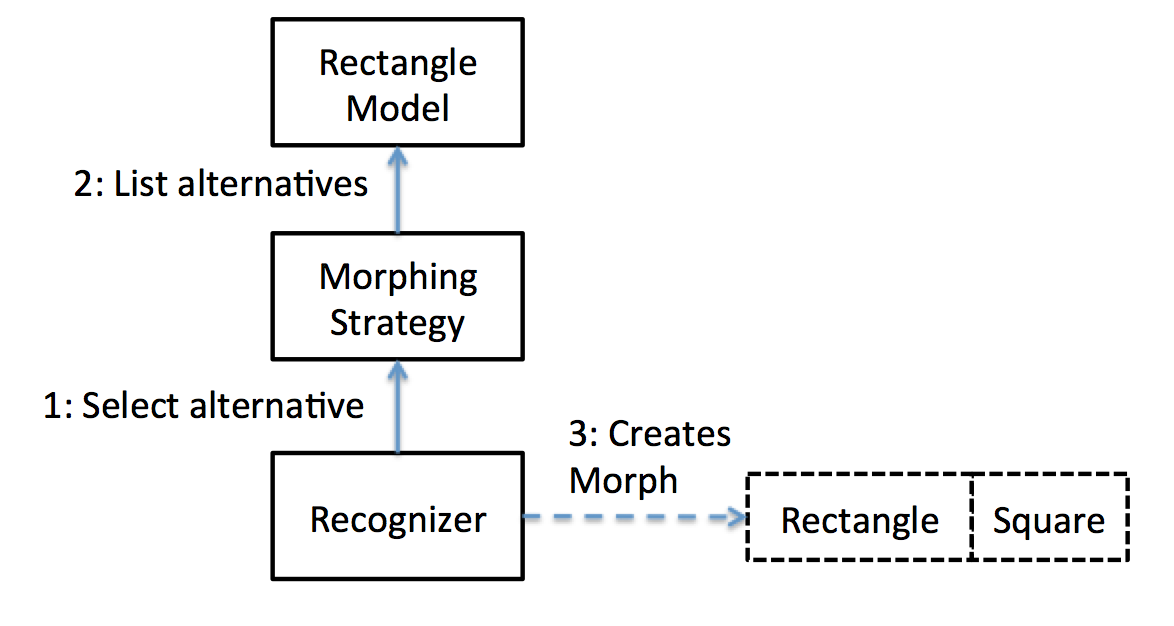


A morph model is an analogy to a class in the traditional (Aristotelian) OO programming. On a statically typed OO platform, the compiler may build the morph model by parsing the model’s type expression at compile-time. The compiler may analyze the morph type expression, build the model instance and perform various checks to guarantee that all alternatives in the model are complete and consistent.

A morph alternative describes one of the forms of a protean object and consists of one or more *fragments*. A fragment is a building block representing a typological, behavioral and structural element of protean objects. It represents a property or feature of a protean object and semantically corresponds to the concept of trait as defined in Scala or Groovy and as it has been used throughout the case study [Scala, Groovy traits].



Instances of alternatives are called *morphs*, which are created by a recognizer according to the alternative selected by the recognizer’s morphing strategy. On every morph instantiation the morph strategy evaluates all possible alternatives and selects the one that matches best the current state of the object or outer conditions such as input parameters.



It follows from the above that the recognizer can never instantiate an invalid composition of fragments. The only risk is that the strategy may be improperly configured or implemented, which may result in selecting inappropriate or invalid alternatives.

## Solution to CRP Using OM

**trait** Shape {  
 **def** shapeName: String  
 **def** area: Double  
 **def** move(dx: Double, dy: Double): Unit  
 **def** printShape(): Unit  
}

hhhhh

**trait** Rectangle **extends** Shape {  
 **var** x: Double = 0  
 **var** y: Double = 0  
 **var** width: Double = 0  
 **var** height: Double = 0  
  
 **override** **def** area: Double = width \* height  
 **override** **def** move(dx: Double, dy: Double): Unit = {  
 x += dx  
 y += dy  
 }  
  
 **def** printShape(): Unit = {  
 println(s"""$shapeName($x,$y,$width,$height)""")  
 }  
  
 **override** **def** shapeName: String = "Rectangle"  
}

hhhh

**trait** Square **extends** Rectangle {  
 **def** side = width  
 **def** side\_=(s: Double): Unit = {  
 **this**.width = s  
 **this**.height = s  
 }  
 **override** **def** shapeName: String = "Square"  
}

hhhh

**val** rect = **new** Rectangle **with** Square{}  
 rect.printShape()  
  
 rect.width = 200  
 rect.printShape()

hhhh

Square(0.0,0.0,0.0,0.0)  
Square(0.0,0.0,200.0,0.0)

Hhh

**val** rectModel = parse[Rectangle **with** (Unit | Square]   
  
**val** rectStg = promote[Square](rectModel)({  
 case None => Some(0)  
 case Some(rect) if rect.width == rect.height => Some(0)  
 case \_ => None  
})  
  
**val** rectRkg = singleton(rectModel, rectStg)  
**val** rect = rectRkg.~  
rect.printShape()  
  
rect.width = 200  
rect.remorph  
rect.printShape()

hhh

Square(0.0,0.0,0.0,0.0)  
Rectangle(0.0,0.0,200.0,0.0)

Hhh

rect.height = 200  
rect.remorph  
  
**val** sq: Square **with** Rectangle =   
 select[Square **with** Rectangle](rect).get  
  
sq.side = 80  
rect.printShape

hhh

Square(0.0,0.0,80.0,80.0)

Hhh

sq.height = 50  
 rect.remorph  
 rect.printShape  
  
 **try** {  
 sq.width = 100  
 } **catch** {  
 **case** e: StaleMorphException =>  
 println("No longer square")  
 }

hhh

Rectangle(0.0,0.0,80.0,50.0)

No longer square

# General information and styles

**This template is intended only for the contribution written in English! For Czech or Slovak written contribution, use the appropriate template.** Please use the following predefined formatting styles (Heading 1, Heading 2, main text, etc.) in this template.

Correctly formatted article according to this template should look like this:

[**EXAMPLE IN PDF**](http://aip.vse.cz/download/aip-template-en-example.pdf)

Articles can be published in English, Czech and Slovak. Articles are accepted only through [the journal publishing system](http://aip.vse.cz/index.php/aip/login) (if you have any questions, please contact the editor). Author will be informed about the acceptance of the paper by e-mail. The following communication (regarding the review process) goes on by the e-mail.

The minimum extent for the acceptance of the contribution as an original scientific paper (**Peer-reviewed papers section**) to the peer-review procedure is 4 pages and maximum is 20 pages. If you are interested in publishing longer paper, you have to inform the editor first. For contributions specified to the **section Peer-reviewed papers** is recommended this structure:

* Introduction
* Literature review and research methods
* Solutions and Results
* Discussion
* Conclusion
* Acknowledgement
* References

Parts can be structured into multiple subsections.

**Contributions to the non-reviewed sections** (Book reviews, Miscelanea and Reflections) at the end of the journal must be in the range from 2 to 6 pages.

# Heading 1 for the title of sections

Normal for the text of the paper.

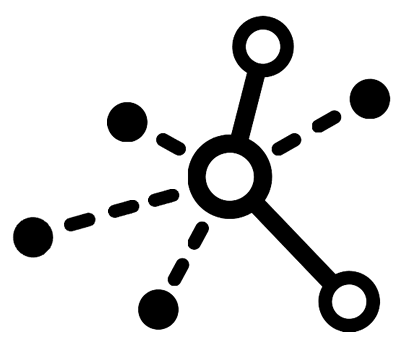
* Bullet A
* Bullet B
  + Bullet 1
  + Bullet 2
* Bullet C

## Heading 2 for the sub-titles of sections

### Heading 3

## Figure

All figures and tables must be centred and formatted according to the following examples.



**Fig. 1.** Caption for the captions of figures. Source (Berg, 2014, p. 57).

## Table

|  |  |  |
| --- | --- | --- |
| **AAA** | **BBB** | **CCC** |
| Aaa1 | Bbb1 | Ccc1 |
| Aaa2 | Bbb2 | Ccc2 |
| Aaa3 | Bbb3 | Ccc3 |
| Aaa4 | Bbb4 | Ccc4 |

**Tab. 1.** Caption for the captions of tables. Source (Berg, 2014).

## Source Code

#include <iostream>

using namespace std;

#define PI 3.14

int main ()

{

double r=4.0; // radius

double circle;

circle = 2 \* PI \* r;

cout << circle;

return 0;

}

## Equation

|  |  |
| --- | --- |
|  | **(1)** |

To create equations use the integrated editor "Equation editor" in MS Word. Equation insert in invisible table and each equation should be numbered.

# References

The contribution should primarily refer to the relevant scientific journals and conferences that are indexed in the Web of Knowledge and Scopus. References have to be **alphabetically sorted.** For links to references in the text, use the following examples using the author's surname and year of publication:

* **1 author** – Berg (2014, pp. 55-57) pointed out the problem… The problem is well-known (Magel, 2013a, p. 47) and further expanded in other papers (Lateg, 2013; Margel, 2013b; Apple, 2012).
* **2-3 authors** – Joergen a Jones (2009) improved used methods… Nevertheless in practice is used CUW method (Kang, Tucin & Kent, 2002).
* **More than 3 authors** – Skálová et al. (2010) provide the solution… The solution already exists (King et al., 2014).

**Examples of list of references**

To cite sources in the part of **References** use APA style (6th edition). The tutorial can be found here: <http://www.apastyle.org/learn/tutorials/basics-tutorial.aspx> In case of problems or doubts, we recommend to use a generator for APA citations on the internet. Below are examples of the most commonly used sources:

***Book:***

Berg, S. (2014). *Services marketing management*. London: Routledge.

Magel, J. (2013a) *Mastering data mining*. (1st ed.) Stockholm: Stockholm School of Economics.

Magel, J. (2013b) *Mastering GUHA. Stockholm*: Stockholm School of Economics.

***Article in journal:***

Joergen, P., & Jones, K. (2009). Random data analysis and measurement procedures. *Journal of Systems Integration*, 5(2), 55-85. Retrieved from <http://si-journal.org/joergen-jones-2009.pdf>

Kang, O., Tucin, J., & Kent, K. (2002). CUW methodics for marketing management. *Journal of Information Management*, 54(3), 1502-1535. doi: 10.7160/jim.2002.06784

***Article in other periodical (newspaper):***

Lateg, R. (2013, March 25). A survey of data provenance in e-science. *New York Review*, pp. 8-11.

***Conference contribution or chapter of the book:***

Skálová, U., Hopstal, H., Kuruc, T., & Krebs, W. (2010). The role of anomalous data in knowledge acquisition. In J. Jicinsky & P. Trejbal (Eds.), *5th International Conference on Informatics* (pp. 248-310). Kaunas: Walter Verlag.

King, U., Smith, L., Jones, E., Kain, W., & Wright, V. (2013). Power-law distributions in empirical data. In J. Rais (Ed.), *The design and analysis of spatial data structures* (1 ed., pp. 45-122). London: Research Life.

***Reference to the web page or e-document:***

Anders, Q. (2014, April 12). *Most-trusted brands*. Retrieved from <http://googleblog.blogspot.cz/anders-brands-2014>

Apple. (2012). *CFNetwork Programming Guide*. Retrieved from <https://develo-per.apple.com/library/mac/documentation/Networking/Conceptual/CFNetwork/CFNetwork.pdf>.