Improving Type safety in the Command Pattern

Jabir Al Fatah
Date of birth: 1991-12-30
Software Engineering-2
Date: 2015-10-21

Abstract

In this report I will discuss about Java's generic mechanism that can improve Type Safety in the Command Design patter. The discussion will be based on a scientific article *''Using Java's Generics Mechanism to Improve Type Safety in the Command Pattern''* [1]. Particularly it explains how generic can be used to specify return and receiver types.

Introduction

According to the article, a java based platform or system named *Moxaic* was developed for presenting ultra-high resolution graphics on a collection of 'commodity' displays. In the application, the developers have introduced type safety problems. Type safety is an extent to which a programing language prevents type errors [2]. A type error is undesirable behavior caused by a discrepancy for the differing data types for the program constant, variables and methods, e.g., treating an integer (int) as a floating number (float) [2]. The purpose of this paper is to provide an overview of the kinds of type safety problems that can arise when using the command pattern and the ways in which these problems can be remedied [1].

Discussion

The first type of type safety is *Type Safety Of Returned Values.* It can be understood by comparing the command pattern to the ordinary interpretation of method calls. Command objects have a name, parameters, and receiver (which is specified by the parameters of execute () method). One problem with the classic implementation of command is that the return type of the execute is *void*. That is it doesn't return a value. However, it is possible to add a return type to the command pattern that is adding an 'outbound' parameter. In java this involves passing *Container* object to the *execute()* method. Container's job is to encapsulate a reference to an object. During the execution of the command, this reference is set as the return value.

Eventually client can read the result value from the container when the command is done with executing.

The second type of safety problem we can consider is *Type Safety Of receivers.* It is best understood by comparing the command pattern to the "message passing" interpretation of method calls. In this way, a message is sent to a receiving object that acts accordingly. When we use command pattern in this design, an actual command object is sent to the invoker and that must contain a typed receiver. The issue with this object is the generic command implementation suffers from the fact that no command may be defined that operates on a type that doesn't inherit from receiver. However, a straightforward solution is to omit the generic command type.

Conclusion

So far I have shown that several generic processes can be used to maintain type safety in this framework implemented with the command. In particular, several types' safety issues have been mentioned of the command pattern and also how they could be resolved in many ways. Moreover, its worth to add that command type can be parameterized with its receiver type. It overcomes the type safety problem introduced by having generic command and invoker types.

Reference

[1] HKR ACM Digital Library. (2015). http://www.hkr.se/en/english-start-page/. Retrieved 2015-10-21, from Using Java's Generics Mechanism to Improve Type Safety in the Command Pattern

http://delivery.acm.org/10.1145/1600000/1593165/p231-ridgeway.pdf?ip=194.47.41.70&id=1593165&acc=ACTIVE%20SERVICE&key=74F7687761D7AE37%2E3011C00F366D381C%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&CFID=723301083&CFTOKEN=47176222&__acm__=1445452003_5b19ad4c2420e43667c60ea4043f47d6

[2] Wikepedia. (2015). https://en.wikipedia.org/wiki/Main_Page. Retrieved 2015-10-21, from Type Safety

https://en.wikipedia.org/wiki/Type_safety