DESIGNING A COMMON FOUNDATION CLASS FOR ACCELERATOR CONTROL SYSTEM WITH UML

Jianjun Hu
IMP-CAS, Lanzhou, China
jjhu@eudoramail.com

Abstract

Through analyzing lots of class model suggested by experts in software field and frequently used in many mature software, combining with years programming experiences, the author design a hardware related class model which is expected to be a commonly used foundation class for OO based software development for particle accelerator control system. Java language will be used to be the programming language and UML (Unified Modeling Language) will be used to illustrate the architecture of the classes framework in this paper.

Keywords: Java, OO, class, UML

1 INTRODUCTION

OO (Object-Oriented) is an advanced programming technique and has been accepted by most programmers who work for the particle accelerator control systems. The classes are the foundation stones of OO-based software development. It is one of the most critical issues that how to analyze and organize the hardware related classes. In this paper we will address some common issues for establishing a hardware related class model. In this model, four thinkable aspects were considered, which are corresponded to the holistic layout, how to keep only one instance of a class, how to forward a request and how to implement request.

2 MODELING

2.1 Holistic Layout

When we begin to design a common class model for all the installation of the accelerator control system, we discover that it is necessary that some parts of an algorithm are well defined and can be implemented in the base class, while other parts may have several implementations and are best left to derived classes. Another main theme of this idea is that there are some basic parts of a class that can be put in a base class so that they do not need to be repeated in several subclasses.

The following codes present this idea.

```
public abstract class Installation {
        private String variable;
        public Installation() {
        }
        abstract void commonDo();
        public void setVariable(String variable){
                this.variable=variable;
        }
         public String getVariable(){
                 return variable;
        }
}

public class Magnet extends Installation
implements specificOperation{
        private String variable;
        public Magnet() {
        }
        void commonDo() {
        }
        public void specificDo(){
        }
        public void setVariable(String variable){
                this.variable=variable;
        }
        public String getVariable(){
                return variable;
        }
}

public interface specificOperation {
        void specificDo();
}
```

2.2 Keep Only One Instance of A Class

There are some occasions where we need to make sure that there can be one and only one instance of a class. For example, your system can have only one magnet under the control of the UI at one time. A better way is to create a class that throws an Exception when it is instantiated more than once. Let's create our own exception class for this case:

```
class SingleException extends runtimeException{
        public SingleException(){
                super();
        }
        public SingleException(String s){
```

```
        super(s);
    }
}
```

Note that other than calling its parent classes through the super() method, this new exception type doesn't do anything in particular. However, it is convenient to have our own named exception type so that the compiler will warn us of the type of exception we must catch when we attempt to create an instance of Magnet. Let's write the related skeleton code of our Magnet class; we'll omit all of the other methods and just concentrate on correctly implementing this idea.

```
public class Magnet extends Installation
implements specificOperation{
    .
    .
    static boolean instance_flag=false;
    public Magnet() throws SingleException{
        if (instance_flag) throw new
            SingleException("only one");
        else instance_flag = true;
    }
    public void finalize(){
        instance_flag = false;
    }
    .
    .
}
```

Remember that we must enclose every method that may throw an exception in a try - catch block when we use it.

In addition, another consequence of this approach is that you can easily change it to allow a small number of instances where they are allowable and meaningful.

2.3 Forwards A Request

When we want to forward a request to a specific module, it encloses a request for a specific action inside an object and gives it a known public interface. It lets you give the client the ability to make requests without knowing anything about the actual action that will be performed, and allows you to change that action without affecting the client program in any way.

```
class ConcreteCommand implements Command{
    private ActualOperation action;
    public ConcreteCommand
        (ActualOperation ao){
        action=ao;
    }
    public void execute(){
        action.externalDo();
    }
}

class ActualOperation{
    public void externalDo(){
        System.out.println("externalDo");
    }
}

public interface Command{
    public abstract void execute();
}
```

2.4 Implements A Request

This section is used to separate the interface of class from its implementation, so that either can be varied separately. It is designed to separate a class's interface from its implementation, so that you can vary or replace the implementation without changing the client code.

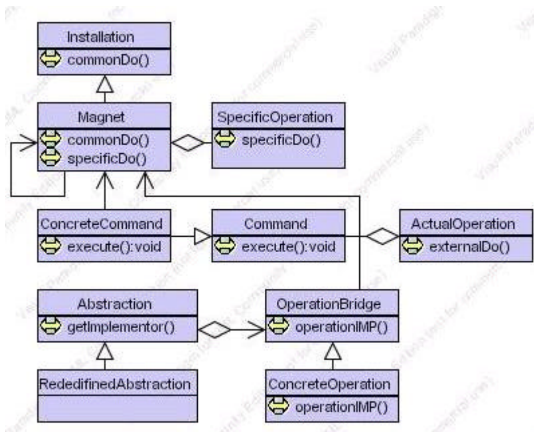
```
abstract class Abstraction{
    protected OperationBridge imp;
    public abstract OperationBridge
        getImplementor();
}

class RefinedAbstraction extends Abstraction{
    public OerationBridge getImplementor(){
        imp=new ConcreteOperation();
        return imp;
    }
}

class OperationBridge{
    public void operationImp(int actionType){
    }
}

class ConcreteOperation extends perationBridge{
    public void operationImp(int actionType){
        if(actionType==1) //Operation1();
        if(actionType==2) //Operation2();
    }
}
```

The consequences of this structure are:

1. It is intended to keep the interface to your client program constant while allowing you to change the actual kind of class you display or use. This can prevent you from recompiling a complicated set of user interface modules, and only require that you recompile the bridge itself and the actual end display class.

2. You can extend the implementation class and the bridge class separately, and usually without much interaction with each other.

The figure below is the class diagram for the Magnet class (actually, the classes collection). A visual UML tool Visual Paradigm for UML was used to make this figure.



Although the source code of this class model is made up of Java language, it can be transformed to another Object-Oriented programming language such as C++ which is also commonly used in software development for particle accelerator control.

The whole source codes for this class model can't be presented here because of the limited paper space. If you are interested in this topic, email provided at the title part of this article is the best way for you to contact the author.

3 CONCLUSIONS

In the preceding sections we discussed common foundation class for accelerator control issues. I hope it can solve the matching problems during your programming. To apply this class framework to actual programming, practices are needed. With current model, significant improvements in your programming for accelerator control system can be achieved if you accepted it. However, there are still many open problems and research issues to be solved, especially in the ease-to-use capability of this model——that is my current work.

4 ACKOWNLEDGEMENTS

The study was done when the author is at Control Group, Electronic Department, INFN-LNL. The author would like to thank Dr. Stefania Canella for her enthusiasm assistance and guidance. I am grateful to all the staff of Electronic Department leaded by Dr. Giorgio Bassato for their support and help. They are Dr. Andrea Battistella, Dr. Davide Carlucci, Dr. Alessandro Zanon et al. Moreover, many enthusiasm Italian give me direct or indirect help during I am at LNL. If I come out with their name, it must be a long list.

5 REFERENCES

[1] http://java.sun.com
[2] Laura Lemay, Rogers Cadenhead, Teach Yourself Java 2 In 21 Days, Second Edition. http://www.java21days.com
[3] http://www.omg.org/uml