International Journal of Emerging Engineering Research and Technology
Volume 3, Issue 1, January 2015, PP 41-44
ISSN 2349-4395 (Print) & ISSN 2349-4409 (Online)

Object Oriented Applications: Integration, Quality Assurance, and Deployment

Charles Edeki, Ph.D[1], Binit Datta[2]

[1]Bronx Community College, City University of New York, Department of Business and Information System, 2155 University Avenue, Bronx, New York 10453.
Charles.Edeki@bcc.cuny.edu
[2]School of Science and Technology, Department of Information Technology, American Military University, 111 W. Congress Street, Charles Town, WV 25414
binit.datta@mycampus.apus.edu

Abstract: *The article reviewed the founding principles of object oriented programming (OOP), unlike procedural programs, classes written in OOP do not normally provide access to the private instance variables to the external accessors. The advantages and disadvantages of OOP are discussed. Design patterns as described below, is a tried tested solution for a well-known problem. Design patterns do exists in the outside world and that is where they are applied to software development in general and OOP in specific. The best way to see design patterns at work is to design an application using them. The section for the design pattern does just that. While encapsulation is very important in OOP, high reusability through inheritance and composition is another great attraction of OOP. The section for reusability in OOP discusses that aspect. Finally the last section touches upon the relationship between UML and Object Oriented Programming.*

Keywords: *OOP, UML, design pattern, inheritance and composition.*

1. Introduction

Among all the philosophies for computer programming, Object Oriented Programming is probably the closest to real life. In real life, one can experience objects without even learning how to program in OOP. If a person holds an account in a bank, he / she cannot get access to the money in his / her account. That money lies secured with other account holder's money in the banks vault as the bank's private property. However, the bank offers public access to the account holder money through online banking website and in store tellers. Now, when the same bank's operation is modelled in UML and programmed in OOP, one can see what exact instance variables need to secured with private access modifiers and public accessors and mutators. OOP language like Java and C++ checks access violation for a classes' private instance variables and generates compilation errors for the violating code. This aspect of protecting private instance variables has a name in OOP called Encapsulation.

Scheldt said that encapsulationis the mechanism that binds together code and the data it manipulates, andkeeps both safe from outside interference and misuse. One way to think about encapsulationis as a protective wrapper that prevents the code and data from being arbitrarily accessedby other code defined outside the wrapper. Access to the code and data inside the wrapperis tightly controlled through a well-defined interface. To relate this to the real world, considerthe automatic transmission on an automobile. It encapsulates hundreds of bits of informationabout your engine, such as how much you are accelerating, the pitch of the surface you areon, and the position of the shift lever. You, as the user, have only one method of affecting this complex encapsulation: by moving the gear-shift lever. You can't affect the transmissionby using the turn signal or windshield wipers, for example. Thus, the gear-shift lever is awell-defined (indeed, unique) interface to the transmission. Further, what occurs inside thetransmission does not affect objects outside the transmission. For example, shifting gearsdoes not turn on the headlights! Because an automatic transmission is encapsulated, dozensof car manufacturers can implement one in any way they please. However, from thedriver'spoint of view, they all work the same. This same idea can be applied to programming.The power of encapsulated code is that everyone knows how to access it and thus can useit regardless of the implementation details—and without fear of unexpected side effects.In Java, the basis of encapsulation is the class. Although the class will be examined in greatdetail later in this book, the following brief discussion will be helpful now. A class definesthe structure and behavior (data and code) that will be shared by a set of objects. Each objectof a given class contains the structure and behavior defined by the class, as if it were stampedout by a mold in the shape of the class. For this reason, objects are sometimes referred to asinstances of a class. Thus, a class is a logical construct; an object has physical reality.When you create a class, you will specify the code and data that constitute that class.Collectively, these elements are called members of the class. Specifically, the data defined bythe class are referred to as member variables or instance variables.

2. Design Pattern

Design Patterns are established, tried and tested solution for common problems. As an example right hand side driving is a design pattern that everyone follows in the US for the problem of driving a car in the streets and highways. Lane driving is also a design pattern for most of the world for the problem of safe driving. Likewise, using Intercepting Filter in Java Web Applications, is an established design patterns for request pre and post processing. It is largely ineffective to study design pattern without trying to apply them in the context of a large application. To be effective, the design of a University's Course Registration System website is discussed below, along with the architectural concerns, and design patterns used in building the website. First, though the following figure shows the Java Enterprise design patterns in a larger context of application design.

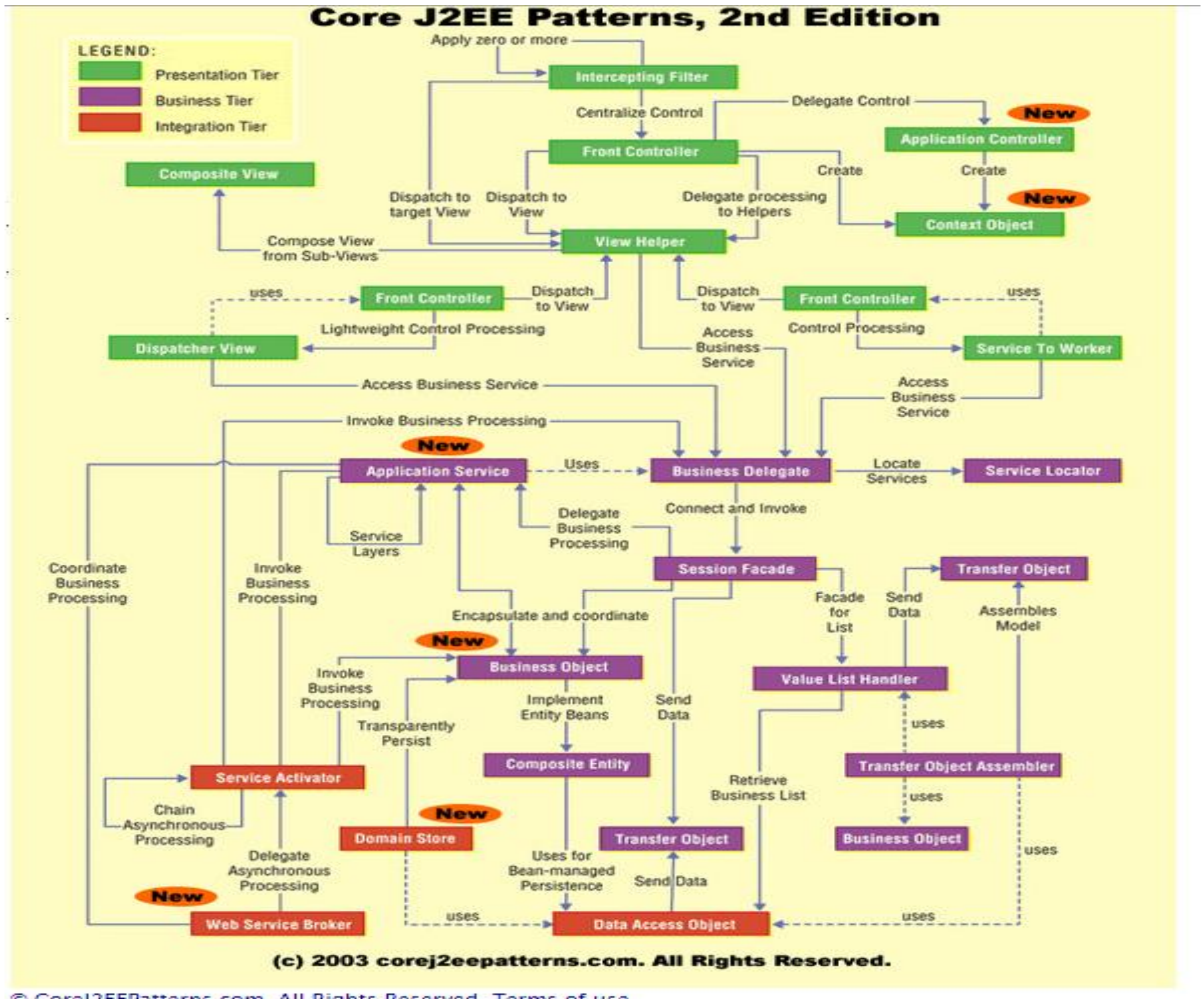

Fig1. *Java Enterprise Design Patterns Adapted from Core J2EE Design Patterns*

Architecture: The components that make the course registration website, are deployed in their own load balanced environments. The user's browser gets the IP address of a global load balancer, from the local DNS servers, which can route the traffic in a round robin manner between two Web Virtual IPs (VIPs). The Virtual IPs route the traffic the cluster of Apache Web Servers (5) that hosts the static assets like images, HTML, CSS and JavaScript for the actual Web GUI. When the Web GUI loads into the user's browser, it makes Ajax calls back to the same Global VIP. The Apache servers see the Ajax calls and routes them to the two Tomcat clusters (each has 10 Tomcat instances) for the REST calls. The Tomcat REST service in turn makes calls to MongoDB and MySQL to fulfill the request.

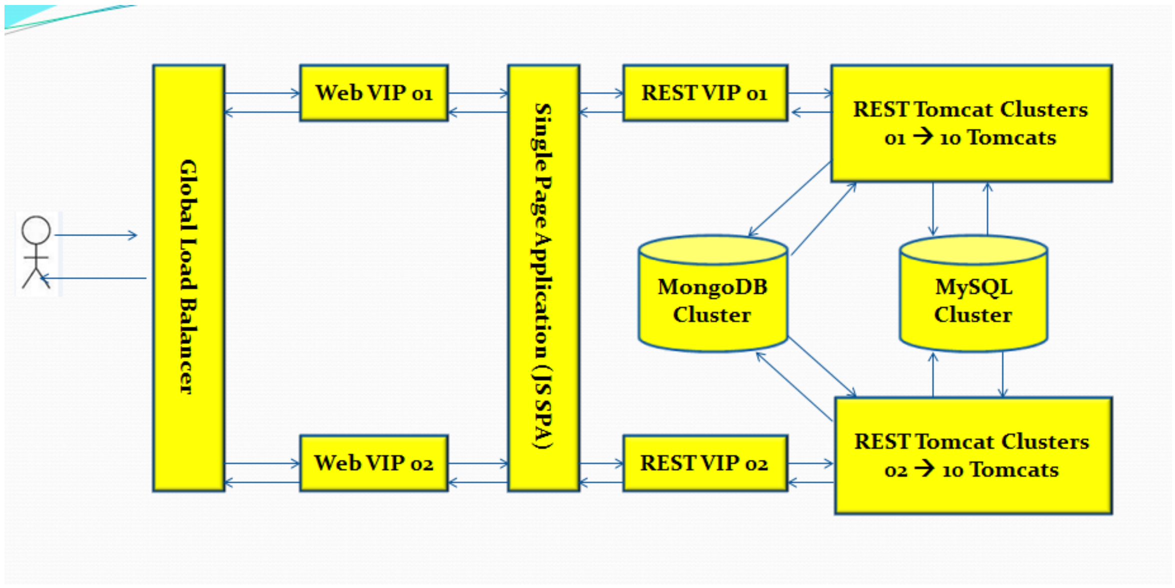
Global Load Balancer
Web VIP 01
Web VIP 02
Single Page Application (JS SPA)
REST VIP 01
REST VIP 02
REST Tomcat Clusters 01 → 10 Tomcats
MongoDB Cluster
MySQL Cluster
REST Tomcat Clusters 02 → 10 Tomcats

Fig2. *Architecture of the Course Registration System*

The intercepting filter design pattern is used when there is a need to centrally preprocess and post process the requests and responses. Filters are defined in a chain and are applied to the request in the defined order before the request is passed to the actual application component. When the response is sent from the application component, the same filter chain is applied on the response in the reverse order. Cross cutting concerns such as authentication, authorization, logging, compression are common tasks for this design pattern. You want to intercept and manipulate a request and a response before and after the request is processed.You want centralized, common processing across requests, such as checking the data-encoding scheme of each request, logging information about each request, or compressing an outgoing response.You want pre and post processing components loosely coupled with core request-handling services to facilitate unobtrusive addition and removal.

Data Access Object Pattern is used to abstract the rest of application the details of accessing the data from integration tier. An application may receive its data from a relational database, a NoSQL database or other sources. Without the DAO pattern, code accessing the data using low level technology would be scatter throughout the application and would become a maintenance nightmare. Centralizing data access, thus provide proper decoupling of data sources from the rest of the application**.**

3. UML AND OOP

Inheritance and composition are two of the popular design technique available in Object oriented analysis and design (OOAD). Both offer several advantages including code reuse, reliability through tested software etc. Both also have a few disadvantages as well be analyzed in the following sections. Inheritance while the more natural between the two, becomes more rigid when it comes to maintenance. However, polymorphism, one of major advantages of OOP is much easier to achieve through inheritance.

All software programming is done by the human mind. There is software that itself, writes other software such as code generating tools. Even this type of special software is created by the human mind. The question is how the human mind normally prepares itself before it writes software. Software programing, unlike any other human tasks, demand an enormous amount of detailing to be effective. Design is process, through which the human mind prepares itself for writing the actual software. The advantage of the design process is that obvious mistakes committed during this phase can be removed and eliminated without a high cost, if those mistakes did not make their way into the actual creation itself. Design is also the process through which the human mind first programs itself to program a computer.

In the past, this design process was very much description oriented. While people used diagrams like Data Flow Diagrams (DFD) and Entity Relationship Diagrams (ERD) the design specification were primarily written and reviewed in big fat documents spanning hundreds of pages, for a big project. It was an uphill task for the human minds involved to first comprehend and then to remember each and every detail that were written in those big documents. Naturally the design process was cumbersome and ineffective. Unified Modeling Language is a collection of notations designed to ease the design problem. It is based on tried, tested and age old English phrase that "A picture is worth a thousand words". The rise and love for UML also happened during the same time, the love for Object Oriented programming grew among the information technology community. Several new set of pictures were introduced and each got their new name. As a result, we have several new UML diagrams such as Use Case diagram, sequence diagram, class diagram, collaboration diagram, component diagram, state chart diagram, activity diagram, and deployment diagram. Each diagram is a collection of a set of basic UML notations such as the Actor, use case, class and the association and dependencies notations, among others. As other craftsmen such as Carpenters has different tools, as there are multiple different types of screw drivers, hammers and ranches in a hardware toolbox, to accomplish different types of tasks, the set of UML diagrams are tools to the object oriented designer to hone his / her understanding of the requirement in question. They are the tools with which the designer first programs his / her mind to allow it to program computers.

UML in Object Oriented Analysis and Design (OOAD) is also simplistic in communication. While designers sent long e-mails and probably attached MS word documents for stakeholders to read for certain requirement / design aspect, with UML, now they can just draw pictures for everyone to understand. While only technical people understood jargons written in the design documents in the past, UML vastly broadens the understanding of the non-technical stakeholders in the organization value chain, by using simple notation in the diagrams.

How is UML used in the OOAD analysis and design process? Different diagrams are used at different point during the analysis and design. While high level business use cases and high level sequence diagrams are drawn and used to understand and communicate business requirements during the initial phase of the OOAD project, detailed use case, sequence diagrams can be drawn later in the detail design phase. Class diagrams also find their place during initial as well as detailed design phases. Other diagrams such as the Activity diagram is used in replacement of the old flow charting techniques to explain a process flow. Some other diagrams like the component, state chart and deployment diagrams are, however, are used mostly during the later construction and deployment phase of the project.

REFERENCES

[1]. Fowler, M. (2014) Front Controller retrieved from http://www.martinfowler.com/eaaCatalog/frontController.html

[2]. Alur, D, Crupi J, Malks, D (2006), Business Delegate retrieved from http://www.corej2eepatterns.com/BusinessDelegate.htm

[3]. Alur, D, Crupi J, Malks, D (2006), Intercepting Filter retrieved from http://www.corej2eepatterns.com/InterceptingFilter.htm

[4]. Alur, D, Crupi J, Malks, D (2006), Data Access Object retrieved from http://www.corej2eepatterns.com/DataAccessObject.htm

[5]. Scheldt, H (2007) Java™: The Complete Reference, Seventh Edition. New York : McGraw-Hill