|  |
| --- |
| LAB MANUAL |
| Software Design & Architecture |
| Course Code: SEL-322 |

|  |
| --- |
|  |

|  |
| --- |
| Department of Computer & Software Engineering  **Bahria University Karachi Campus**  13 National Stadium Road |

## Preface

As the size and complexity of software systems increases, the design problem goes beyond the algorithms and data structures of the computation: designing and specifying the overall system structure emerges as a new kind of problem. Structural issues include gross organization and global control structure; protocols for communication, synchronization, and data access; assignment of functionality to design elements; physical distribution; composition of design elements; scaling and performance; and selection among design alternatives.

The architecture & designs of software is main key in development. There is a considerable body of work on these topics, including module interconnection languages, templates and frameworks for systems that serve the needs of specific domains, and formal models of component integration mechanisms and the implementation of design patterns. In addition, an implicit body of work exists in the form of descriptive terms used informally to describe systems. And while there is not currently a well-defined terminology or notation to characterize architectural structures, good software engineers make common use of architectural principles when designing complex software. Many of the principles represent rules of thumb or idiomatic patterns that have emerged informally over time. Others are more carefully documented as industry and scientific standards. It is increasingly clear that effective software engineering requires facility in architectural software design and relevant design patterns.

In this lab manual we provide an introduction to the field of software architecture and will cover most of designs patterns as per their usage. The purpose is to illustrate the current state of the discipline and examine the ways in which these design can impact software design. The material presented here is selected from a semester course.  
The language used for implementations is C#. In implementing the assignments, a good programming style is very important, therefore some guide is given in Appendix A.

This guide is intended to be used by the students of Computer & Software Engineering Department of Bahria University Karachi Campus.

*C&SE Department, Bahria University Karachi Assist. Prof. Uswan Waheed and Engr. M. Mursaleen Javed*

#### **Software Design and Architecture**

## Course Description:

This course concentrates on the practical part of the course of Software Design and Architecture with OOP under C# Environment. This course allows students to understand practically the logical and physical representation of software design and architectural pattern their complexity, efficiency and their different variations. This part includes the implementation of design patterns, architectural pattern and their usage in real world scenarios.

## Course Objectives:

1. Extend the knowledge of software architecture and design patterns and improve the programming ability to apply them using an object-oriented language.
2. Analyze design patterns to determine their worth and usage.
3. Choose the appropriate data pattern to use in solving typical software engineering problems.

## Learning Outcomes:

After completing this course the student must demonstrate the Knowledge and ability to:

1. Demonstrate the application of software design patterns in software engineering.
2. Make students aware of the importance of software design and architectural patterns in developing software.
3. Student must know the systematic approach to study software design and architectural patterns, by focuses first on understanding the use of the pattern by analyzing it.

## List of Equipment/Requirements:

#### Hardware Requirements:

A personal computer

#### Software Requirements:

Visual Studio 2008/2010/2012/2013/2015

C#

## Lab Template Guidelines

### Objective: [Font: Candara, Font-Size: 12, Style: Heading 3]

Text starts from new line. [Font: Candara, Font-Size: 11]

**Write a procedure which finds the average of the values of Array elements.**

### Source Code: [Font: Candara, Font-Size: 12, Style: Heading 3]

Text starts from new line. [Font: Candara, Font-Size: 11]

[Line and Paragraph Spacing Rule: Apply Remove Line After Paragraph rule]

Using system

Using namespace std;

Class program {

Void main() {

………………………

}

}

### Result: [Font: Candara, Font-Size: 12, Style: Heading 3]

Text starts from new line. [Font: Candara, Font-Size: 11]

**The Array contains following elements:**

1,2 ,3 ,4, 5, 6, 7, 8 ,9,10

Sum = 55

Average = 5.5

Table of Contents

[**Preface ii**](#_Toc448192304)

[**Course Description: iii**](#_Toc448192305)

[**Course Objectives: iii**](#_Toc448192306)

[**Learning Outcomes: iii**](#_Toc448192307)

[**List of Equipment/Requirements: iii**](#_Toc448192308)

[**Lab Template Guidelines iv**](#_Toc448192309)

[**List of Figures viii**](#_Toc448192310)

[**SOFTWARE DESIGN & ARCHITECTURE VIA DOCUMENTATION 9**](#_Toc448192311)

[**Introduction 10**](#_Toc448192312)

[**Time Boxing 10**](#_Toc448192313)

**[Objectives 10](#_Toc448192314)**

[**Lab Tasks/Practical Work 10**](#_Toc448192315)

[**SOFTWARE DESIGN & ARCHITECTURE VIA LEGACY INTERFACE 11**](#_Toc448192316)

[**Introduction 12**](#_Toc448192317)

[**Time Boxing 13**](#_Toc448192318)

[**Objectives 13**](#_Toc448192319)

[**Lab Tasks/Practical Work 13**](#_Toc448192320)

[**TRANSACTION PATTERN 14**](#_Toc448192321)

[**Introduction 15**](#_Toc448192322)

[**Time Boxing 21**](#_Toc448192323)

[**Objectives 21**](#_Toc448192324)

[**Lab Tasks/Practical Work 22**](#_Toc448192325)

[**ARCHITECTURAL PATTERN: N-TIER 23**](#_Toc448192326)

[**Introduction 24**](#_Toc448192327)

[**Time Boxing 25**](#_Toc448192328)

[**Objectives 25**](#_Toc448192329)

[**Lab Tasks/Practical Work 25**](#_Toc448192330)

[**ARCHITECTURAL PATTERN: MVC (Model View Controller) 26**](#_Toc448192331)

[**Introduction 27**](#_Toc448192332)

[**Time Boxing 27**](#_Toc448192333)

[**Objectives 28**](#_Toc448192334)

[**Lab Tasks/Practical Work 28**](#_Toc448192335)

[**CREATIONAL DESIGN PATTERN: SINGLETON 29**](#_Toc448192336)

[**Introduction 30**](#_Toc448192337)

[**Time Boxing 30**](#_Toc448192338)

[**Objectives 30**](#_Toc448192339)

[**Lab Tasks/Practical Work 30**](#_Toc448192340)

[**CREATIONAL DESIGN PATTERN: BUILDER 31**](#_Toc448192341)

[**Introduction 32**](#_Toc448192342)

[**Time Boxing 32**](#_Toc448192343)

[**Objectives 33**](#_Toc448192344)

[**Lab Tasks/Practical Work 33**](#_Toc448192345)

[**CREATIONAL DESIGN PATTERN: PROTOTYPE 34**](#_Toc448192346)

[**Introduction 35**](#_Toc448192347)

[**Time Boxing 35**](#_Toc448192348)

[**Objectives 36**](#_Toc448192349)

[**Lab Tasks/Practical Work 36**](#_Toc448192350)

[**CREATIONAL DESIGN PATTERN: FACTORY [VIRTUAL CONSTRUCTOR] 37**](#_Toc448192351)

[**Introduction 38**](#_Toc448192352)

[**Time Boxing 38**](#_Toc448192353)

[**Objectives 39**](#_Toc448192354)

[**Lab Tasks/Practical Work 39**](#_Toc448192355)

[**CREATIONAL DESIGN PATTERN: ABSTRACT FACTORY PATTERN [KIT] 40**](#_Toc448192356)

[**Introduction 41**](#_Toc448192357)

[**Time Boxing 41**](#_Toc448192358)

[**Objectives 42**](#_Toc448192359)

[**Lab Tasks/Practical Work 42**](#_Toc448192360)

[**STRUCTURAL DESIGN PATTERN: PROXY [SURROGATE] 43**](#_Toc448192361)

[**Introduction 44**](#_Toc448192362)

[**Time Boxing 45**](#_Toc448192363)

[**Objectives 45**](#_Toc448192364)

[**Lab Tasks/Practical Work 45**](#_Toc448192365)

[**STRUCTURAL DESIGN PATTERN: COMPOSITE 46**](#_Toc448192366)

[**Introduction 47**](#_Toc448192367)

[**Time Boxing 48**](#_Toc448192368)

[**Objectives 48**](#_Toc448192369)

[**Lab Tasks/Practical Work 48**](#_Toc448192370)

[**STRUCTURAL DESIGN PATTERN: ADAPTER [WRAPPER] 49**](#_Toc448192371)

[**Introduction 50**](#_Toc448192372)

[**Time Boxing 51**](#_Toc448192373)

[**Objectives 51**](#_Toc448192374)

[**Lab Tasks/Practical Work 51**](#_Toc448192375)

[**STRUCTURAL DESIGN PATTERN: DECORATOR 52**](#_Toc448192376)

[**Introduction 53**](#_Toc448192377)

[**Time Boxing 53**](#_Toc448192378)

[**Objectives 54**](#_Toc448192379)

[**Lab Tasks/Practical Work 54**](#_Toc448192380)

[**STRUCTURAL DESIGN PATTERN: BRIDGE [HANDLE/BODY] 55**](#_Toc448192381)

[**Introduction 56**](#_Toc448192382)

[**Intent 56**](#_Toc448192383)

[**Time Boxing 56**](#_Toc448192384)

[**Objectives 57**](#_Toc448192385)

[**Lab Tasks/Practical Work 57**](#_Toc448192386)

[**STRUCTURAL DESIGN PATTERN: FAÇADE 58**](#_Toc448192387)

[**Introduction 59**](#_Toc448192388)

[**Time Boxing 59**](#_Toc448192389)

[**Objectives 59**](#_Toc448192390)

[**Lab Tasks/Practical Work 59**](#_Toc448192391)

[**STRUCTURAL DESIGN PATTERN: FLYWEIGHT 60**](#_Toc448192392)

[**Introduction 61**](#_Toc448192393)

[**Time Boxing 62**](#_Toc448192394)

[**Objectives 62**](#_Toc448192395)

[**Lab Tasks/Practical Work 62**](#_Toc448192396)

[**BEHAVIORAL DESIGN PATTERN: CHAIN OF RESPONSIBILITY PATTERN 63**](#_Toc448192397)

[**Introduction 64**](#_Toc448192398)

[**Time Boxing 65**](#_Toc448192399)

[**Objectives 65**](#_Toc448192400)

[**Lab Tasks/Practical Work 65**](#_Toc448192401)

[**BEHAVIORAL DESIGN PATTERN: VISITOR 66**](#_Toc448192402)

[**Introduction 67**](#_Toc448192403)

[**Time Boxing 68**](#_Toc448192404)

[**Objectives 68**](#_Toc448192405)

[**Lab Tasks/Practical Work 68**](#_Toc448192406)

[**BEHAVIORAL DESIGN PATTERN: OBSERVER [DEPENDENTS, PUBLISH-SUBSCRIBE] 69**](#_Toc448192407)

[**Introduction 70**](#_Toc448192408)

[**Time Boxing 70**](#_Toc448192409)

[**Objectives 71**](#_Toc448192410)

[**Lab Tasks/Practical Work 71**](#_Toc448192411)

[**BEHAVIORAL DESIGN PATTERN: STATE PATTERN [OBJECT OF STATES] 72**](#_Toc448192412)

[**Introduction 73**](#_Toc448192413)

[**Time Boxing 73**](#_Toc448192414)

[**Objectives 73**](#_Toc448192415)

[**Lab Tasks/Practical Work 73**](#_Toc448192416)

[**BEHAVIORAL DESIGN PATTERN: STRATEGY [POLICY] 74**](#_Toc448192417)

[**Introduction 75**](#_Toc448192418)

[**Time Boxing 75**](#_Toc448192419)

[**Objectives 75**](#_Toc448192420)

[**Lab Tasks/Practical Work 75**](#_Toc448192421)

[**BEHAVIORAL DESIGN PATTERN: TEMPLATE METHOD 76**](#_Toc448192422)

[**Introduction 77**](#_Toc448192423)

[**Time Boxing 77**](#_Toc448192424)

[**Objectives 77**](#_Toc448192425)

[**Lab Tasks/Practical Work 78**](#_Toc448192426)

[**BEHAVIORAL DESIGN PATTERN: NULL OBJECT 79**](#_Toc448192427)

[**Introduction 80**](#_Toc448192428)

[**Time Boxing 81**](#_Toc448192429)

[**Objectives 81**](#_Toc448192430)

[**Lab Tasks/Practical Work 81**](#_Toc448192431)

[**BEHAVIORAL DESIGN PATTERN: MEDIATOR 82**](#_Toc448192432)

[**Introduction 83**](#_Toc448192433)

[**Time Boxing 84**](#_Toc448192434)

[**Objectives 84**](#_Toc448192435)

[**Lab Tasks/Practical Work 84**](#_Toc448192436)

[**BEHAVIORAL DESIGN PATTERN: ITERATOR [CURSOR] 85**](#_Toc448192437)

[**Introduction 86**](#_Toc448192438)

[**Time Boxing 86**](#_Toc448192439)

[**Objectives 87**](#_Toc448192440)

[**Lab Tasks/Practical Work 87**](#_Toc448192441)

[**Appendix 88**](#_Toc448192442)

[**References 96**](#_Toc448192443)

## List of Figures

[Figure 1Transaction Pattern 15](#_Toc448185532)

[Figure 2 Transaction sets from transaction pattern 16](#_Toc448185533)

[Figure 3 General Items 17](#_Toc448185534)

[Figure 4 Specific Item Transaction 17](#_Toc448185535)

[Figure 5 Specific Items With Line Items 18](#_Toc448185536)

[Figure 6 General and Specific Item Transaction 19](#_Toc448185537)

[Figure 7 Transaction based on subsequent transaction 19](#_Toc448185538)

[Figure 8 Transaction with line item and subsequent transaction without line item 20](#_Toc448185539)

[Figure 9 Transaction and subsequent transaction both with line item 20](#_Toc448185540)

[Figure 10 N-Tier Structure 24](#_Toc448185541)

[Figure 11 Model View Controller 27](#_Toc448185542)

[Figure 12: Singleton Design Pattern Structure 30](#_Toc448185543)

[Figure 13 Builder Pattern 32](#_Toc448185544)

[Figure 14 Prototype Pattern 35](#_Toc448185545)

[Figure 15 Factory Pattern 38](#_Toc448185546)

[Figure 16 Abstract Factory 41](#_Toc448185547)

[Figure 17 Proxy Pattern 44](file:///C:\Users\Mursaleen\Desktop\SDA%20Lab%20Manual.docx#_Toc448185548)

[Figure 18 Composie Pattern 47](#_Toc448185549)

[Figure 19 Adpater Pattern 50](#_Toc448185550)

[Figure 20 Decorator Pattern 53](#_Toc448185551)

[Figure 21 Bridge Pattern 56](#_Toc448185552)

[Figure 22 Façade Pattern 59](#_Toc448185553)

[Figure 23 Flyweight Pattern 61](#_Toc448185554)

[Figure 24 Chain of Responsibility 64](#_Toc448185555)

[Figure 25 Visitor Pattern 67](#_Toc448185556)

[Figure 26 Observer Pattern 70](#_Toc448185557)

[Figure 27 State Pattern 73](#_Toc448185558)

[Figure 28 Strategy Pattern 75](#_Toc448185559)

[Figure 29 Template Pattern 77](#_Toc448185560)

[Figure 30 Null Object Pattern 80](#_Toc448185561)

[Figure 31 Mediator Pattern 83](#_Toc448185562)

[Figure 32 Iterator Pattern 86](#_Toc448185563)

**Lab Manual for Software Design & Architecture**

**Lab No. 1**

# **SOFTWARE DESIGN & ARCHITECTURE VIA DOCUMENTATION**

Objectives

To have the knowledge of how software designs & architectures are developed through documentation

**LAB # 01**

**Software Design & Architecture via Documentation**

## **Introduction**

Software documentation is written text or illustration that accompanies computer software. It either explains how it operates or how to use it, and may mean different things to people in different roles.

Documentation is an important part of software engineering. Types of documentation include:

Requirements - Statements that identify attributes, capabilities, characteristics, or qualities of a system. This is the foundation for what will be or has been implemented.

Architecture/Design - Overview of software. Includes relations to an environment and construction principles to be used in design of software components.

Technical - Documentation of code, algorithms, interfaces, and APIs.

End user - Manuals for the end-user, system administrators and support staff.

Marketing - How to market the product and analysis of the market demand.

The main documentation required for designing a software or for software development.

1. Software Requirements Specification
   1. Software Design Description

### For more information

(Wikipedia, Software requirements specification)

(Wikipedia, Software Design Description)

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 5 mints | 5 mints |
| Walk through Theory & Tasks (Lecture) | 60 mints | 60 mints |
| Implement Tasks | 90 mints | 80 mints |
| Evaluation Time | 20 mints | 20 mints |
|  | Total Duration | 175 mints |

## **Objectives**

After completing this lab, student should be able to:

* Clearly understand the purpose and benefits that software design & architecture.
* Understand the concept of a Software Design & Architecture for development.

## **Lab Tasks/Practical Work**

1. By using business document identify:
   1. Participants & Transactions (entities & attributes)
   2. Item, line item (attributes & entities)
   3. Place other information (other associate)
   4. Object diagram (relationship, attributes)
   5. Missing fields / attributes
   6. Proposed business document
2. Propose structure and technologies for your system.

**Lab Manual for Software Design & Architecture**

**Lab No. 2**

# **SOFTWARE DESIGN & ARCHITECTURE VIA LEGACY INTERFACE**

Objectives

To have the knowledge of how software designs & architectures are developed through legacy interfaces

**LAB # 02**

**Software Design & Architecture via Legacy Interface**

## **Introduction**

In computing, a legacy system is an old method, technology, computer system, or application program, "of, relating to, or being a previous or outdated computer system." Often a pejorative term, referencing a system as "legacy" often implies that the system is out of date or in need of replacement.

The first use of the term legacy to describe computer systems probably occurred in the 1970s. By the 1980s it was commonly used to refer to existing computer systems to distinguish them from the design and implementation of new systems. Legacy was often heard during a conversion process, for example, when moving data from the legacy system to a new database.

While this term may indicate that some engineers may feel that a system is out of date, a legacy system may continue to be used for a variety of reasons. It may simply be that the system still provides for the users' needs. In addition, the decision to keep an old system may be influenced by economic reasons such as return on investment challenges or vendor lock-in, the inherent challenges of change management, or a variety of other reasons other than functionality. Backward compatibility (such as the ability of newer systems to handle legacy file formats and character encodings) is a goal that software developers often include in their work.

Even if it is no longer used, a legacy system may continue to impact the organization due to its historical role. Historic data may not have been converted into the new system format and may exist within the new system with the use of a customized schema crosswalk, or may exist only in a data warehouse. In either case, the effect on business intelligence and operational reporting can be significant. A legacy system may include procedures or terminology which are no longer relevant in the current context, and may hinder or confuse understanding of the methods or technologies used.

Organizations can have compelling reasons for keeping a legacy system, such as:

* The system works satisfactorily, and the owner sees no reason to change it.
* The costs of redesigning or replacing the system are prohibitive because it is large, monolithic, and/or complex.
* Retraining on a new system would be costly in lost time and money, compared to the anticipated appreciable benefits of replacing it (which may be zero).
* The system requires near-constant availability, so it cannot be taken out of service, and the cost of designing a new system with a similar availability level is high. Examples include systems to handle customers' accounts in banks, computer reservations systems, air traffic control, energy distribution (power grids), nuclear power plants, military defense installations, and systems such as the TOPS database.
* The way that the system works is not well understood. Such a situation can occur when the designers of the system have left the organization, and the system has either not been fully documented or documentation has been lost.
* The user expects that the system can easily be replaced when this becomes necessary.
* Newer systems perform undesirable (especially for individual or non-institutional users) secondary functions such as a) tracking and reporting of user activity and/or b) automatic updating that creates "back-door" security vulnerabilities and leaves end users dependent on the good faith and honesty of the vendor providing the updates. This problem is especially acute when these secondary functions of a newer system cannot be disabled.

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 5 mints | 5 mints |
| Walk through Theory & Tasks (Lecture) | 60 mints | 60 mints |
| Implement Tasks | 90 mints | 80 mints |
| Evaluation Time | 20 mints | 20 mints |
|  | Total Duration | 175 mints |

## **Objectives**

After completing this lab, student should be able to:

* Clearly understand the purpose and benefits that software design & architecture has to offer.
* Understand the concept of a software design & architecture for development.

## **Lab Tasks/Practical Work**

1. By using Interface identify:
   1. Participant
   2. Transaction
   3. Line Item
   4. Item
   5. Specific Item
   6. Place
   7. Subsequent Transaction
   8. Attributes
   9. Make Object Diagram.
2. Propose improvements, structure and technologies for your system.

**Lab Manual for Software Design & Architecture**

**Lab No. 3**

# **TRANSACTION PATTERN**

Objectives

To have the knowledge of transaction pattern and their implementation & structures.

**LAB # 03**

**Transaction Pattern**

## **Introduction**

A transaction is a recording or logging of any event of significance; hence, transaction is “a significant event remembered”. In business scenarios, event that involve currency, count, or date time are classified as a transaction.

#### Examples:

Deposit, order, payment, purchase, refund, rental, reservation, sale and withdrawal are examples of currency based transactions.

Courses enrollment, attendance register, books issued or returned and marks obtained are examples of count based transactions.

Time in time out, birth certificate, appointment, schedule, session logs are examples of date time based transactions.

While defining transactions multiple objects are used. These objects vary from transaction to transaction. To meet this objectives, a generalized business model is required that captures a wide range of business transactions and their objects.

Research selects transaction pattern, which supports large variety of business transaction situations. By arranging different transaction objects, different type of transactions are identified and named them *transaction sets*. Research is helpful for analyzing business applications and specially in designing different business processes. Different transaction sets are useful in validating transaction objects of specific business situation and validation their relationships.

### Transaction Sets

Transaction pattern is an analysis pattern that specifies a transaction with the help of players. *Players* are objects with special roles and associations in a transaction. Players include actor, participant, place, transaction line item, item, specific item, subsequent transaction, and subsequent transaction line item. A workflow consists of a sequence of transactions in which a transaction is followed by a subsequent transaction. In *figure-1* transaction pattern, with its players and relationship is presented.

Analysis of business applications indicates that not all transaction contains all players. It is also observed simplest transaction is based on only two players and that are participant and transaction. The set of players varies from transaction to transaction.

Place

Specific Item

Item

Participant

Actor

\*

1

0..\*

TransactionineItem

0..\*

1

SubsequentTransactionLineItem

1

0..\*

Transaction

0..\*

1

0..\*

1

1

0..\*

1

SubsequentTransaction

1

0..\*

1

0..

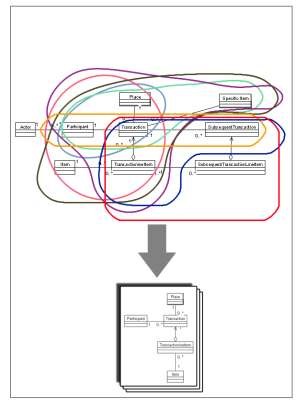
~~\*~~

1..\*

Figure 1Transaction Pattern

The research defines *transaction set* by grouping of players relating to different type of transactions. In some transaction few players are involved such as an appointment transaction requires three players (participant, transaction and place), where participant is patient, transaction is appointment and place is clinic. There are some transaction that involve more players such as order transaction which requires five players (participant, transaction, transaction line item, item, and place), where participant is customer, transaction is order, transaction line item is order detail and place is factory. By varying the selection of players, different transaction sets are generated.

Figure 2 Transaction sets from transaction pattern



The phenomenon of grouping different players and defining their transaction set is shown in figure-2. Different transaction sets represent different analysis and design situation. By varying the selection of player different transaction types are generated. Transaction pattern is a superset that contains many subsets of transaction sets. Each subset is representing a separate transaction type.

Eight transaction sets are presented and are divided in three broad categories.

There can be more transaction sets by varying players.

##### Transaction based on participant

##### Transaction based on items

##### Transaction based on subsequent transaction.

##### Transaction based on participant

A simple transaction may contain a few players such as participant and transaction, where place is an optional player.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | | Participant | |  | |  | | Transaction  Place  0..\*  1  0..\*  1 |

Examples*:*

Patient - appointment - clinic

Student - attendance

Visitor - gate pass – department Driver – driving license

Transaction Based on Items

Second type of transaction is based on items. Item is further classified as general item or specific item.

### General item transaction

These transactions are based on item and typically involve a count or quantity of the item. Players of these transactions are participants, transaction, transaction line item, item and place.

Participant

Transaction

Place

0..\*

1

0..\*

1

1

TransactionineItem

1

0..\*

Item

Figure 3 General Items

#### Examples:

Customer- order –order line item -product

Buyer –payment –payment line item -goods

Vendor – purchase- purchase line item - products

### Specific item transaction

These transactions are based on items whose individual identity is important to track such as the serial number. These transactions are designed for a single item and do not involve quantity as in general item transaction. Players of these transactions are participant, transaction and specific item, where place is an optional player.

Participant

0..\*

1

Transaction

Place

1

0..\*

0..\*

1

Specific Item

Figure 4 Specific Item Transaction

#### Examples:

Investor- payment- Specific aircraft

Agent – loading - specific container - port

Shipper - shipment - specific ship,

Buyer – rental- specific vehicle

Member – subscribe –specific credit card

Specific item transaction with line item

These transactions support multiple specific items. These transactions use transaction line item to store list of specific items. Players of these transactions are participant, transaction, transaction line item and specific item where place is optional.

Participant

1

0..\*

Transaction

0..\*

1

Place

1

TransactionineItem

0..\*

1

Specific Item

Figure 5 Specific Items With Line Items

#### Examples:

Student – issue –issue line item –specific book

Customer –purchase –purchase line item – electric goods

Account holder – deposit –deposit line item – cash bonds - bank branch

Buyer – purchase –purchase line item – computer accessories

### General and specific item transaction

These transactions are based on both kind of items, general items and specific items. These transactions use line item to store list of specific items and general item. Players of these transactions are participant, transaction, transaction line item, specific item, item and place.

Participant

1

0..\*

Transaction

0..\*

1

Place

1

TransactionineItem

0..\*

1

Specific Item

1

0..\*

Item

Figure 6 General and Specific Item Transaction

#### Examples:

Customer –purchase –purchase line item – electric goods - goods

Buyer – purchase –purchase line item – computer accessories – printer pages -department Customer- order – order line item – product – specific product

### Transaction based on subsequent transaction

These transactions are based on two transactions, one existing and other subsequent transaction. There are three possible combinations.

Transaction and subsequent transaction both without line item

These types of transactions involve simple transaction and subsequent transaction. Players are participant, transaction and subsequent transaction. The relationship between these players is many to many. In relational data model there must exist association table.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Participant | 1 |  | Transaction |  |  | SubsequentTransaction |
|  |  |  |  | 1 |  |  |
|  |  |  |

0..\* 0..\*

Figure 7 Transaction based on subsequent transaction

#### Examples:

Patient – appointment – treatment /admission

Student - attendance – examination

Visitor - gate pass – entry

Driver – driving license- renewal

### Transaction with line item and subsequent transaction without line item

These types of transactions involve transaction with line item and subsequent transaction without line item. Players are transaction, transaction line item and subsequent transaction. The relationship between transaction and subsequent transaction player is many to many.

Transaction

SubsequentTransaction

0..\*

0..\*

1

TransactionineItem

Figure 8 Transaction with line item and subsequent transaction without line item

#### Examples:

Order –order line item -payment

Purchase- purchase line item – packing

### Transaction and subsequent transaction both with line item

These types of transactions involve transaction and subsequent transaction both with line item. Players are transaction, transaction line item, subsequent transaction and subsequent transaction line item. The relationship among transaction, subsequent transaction and their line items is many to many. In relational data model there must exist association tables.

Transaction

0..\*

0..\*

SubsequentTransaction

1

TransactionineItem

0..\*

0..\*

SubsequentTransactionLineItem

Figure 9 Transaction and subsequent transaction both with line item

#### Examples:

Order –order line item –payment – payment line

item Payment –payment line item –shipment-shipment line item

Purchase- purchase line item – packing –packing line item

#### Attributes

Each Player contain attributes.

Essential attributes of each player are given in *table-1*

*Table-1 Attributes of players*

|  |  |
| --- | --- |
| *Players* | *Attributes* |
| Actor | Name,a ddress, phone |
| Participant | Number, start date, end date |
|  | authorization level, password |
| Transaction | Number, date, time, status |
| Transaction line item | Number, quantity, status |
| Item | Number, name , description, price, default value |
| Specific item | Serial number, purchase date, custom value |
| Place | Number, name, address |

Transaction pattern defines only essential atributes that significantly defines the role of player. An attribute **AboutMe** isavailable in every player that indicates that other user defined fields can be added to further describe the player.

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 5 mints | 5 mints |
| Walk through Theory & Tasks (Lecture) | 60 mints | 60 mints |
| Implement Tasks | 90 mints | 80 mints |
| Evaluation Time | 20 mints | 20 mints |
|  | Total Duration | 175 mints |

## **Objectives**

After completing this lab, student should be able to:

* Clearly understand the purpose and benefits of transaction patterns.
* Understand the usage and implementation of transaction pattern.
* Idea of different types of transaction patterns.

## **Lab Tasks/Practical Work**

1. Everyone have assigned a different topic you have to perform following steps
   1. Select only 5-7 Data models from category assigned to you. (contains 10+ entities)
   2. Identify transaction pattern Players (actor, participant, transaction, transaction line item, item, place, associate, other associate)
   3. Add more attributes from transaction pattern to assigned model.
   4. Write suggestion paragraph that recommend improvements in data model.

Source: <http://www.databaseanswers.org/data_models/>

**Lab Manual for Software Design & Architecture**

**Lab No. 4**

# **ARCHITECTURAL PATTERN: N-TIER**

Objectives

Understand the structure, implementation and usage of N-tier Architectural Pattern.

**LAB # 04**

**Architectural Pattern: N-Tier**

## **Introduction**

In software engineering, multi-tier architecture (often referred to as n-tier architecture) is a client–server architecture in which presentation, application processing, and data management functions are physically separated. The most widespread use of multi-tier architecture is the three-tier architecture.

N-tier application architecture provides a model by which developers can create flexible and reusable applications. By segregating an application into tiers, developers acquire the option of modifying or adding a specific layer, instead of reworking the entire application. A three-tier architecture is typically composed of a presentation tier, a domain logic tier, and a data storage tier.

While the concepts of layer and tier are often used interchangeably, one fairly common point of view is that there is indeed a difference. This view holds that a layer is a logical structuring mechanism for the elements that make up the software solution, while a tier is a physical structuring mechanism for the system infrastructure

### Structure

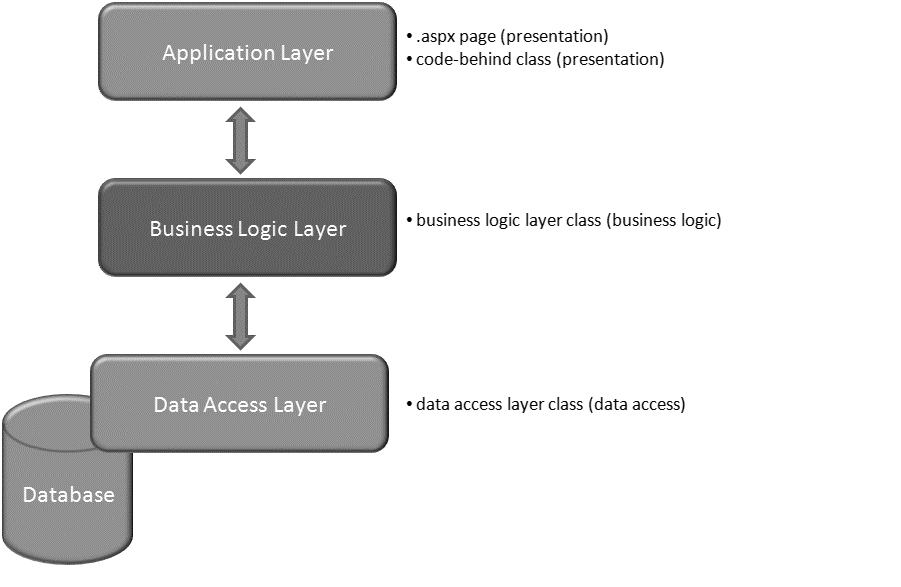


Figure 10 N-Tier Structure

### Participant

* Presentation Tier
* Middle Tier
* Data Tier

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 5 mints | 5 mints |
| Walk through Theory & Tasks (Lecture) | 60 mints | 60 mints |
| Implement Tasks | 90 mints | 80 mints |
| Evaluation Time | 20 mints | 20 mints |
|  | Total Duration | 175 mints |

## **Objectives**

After completing this lab, student should be able to:

* Clearly understand the purpose and benefits of N-Tier Architecture.
* Implementation and structure of N-Tier pattern.

## **Lab Tasks/Practical Work**

1. Develop a project by using n-tier pattern and implement student information system scenario.

**Lab Manual for Software Design & Architecture**

**Lab No. 5**

# **ARCHITECTURAL PATTERN: MVC (Model View Controller)**

Objectives

To understand the usage, structure and implementation of MVC pattern.

**LAB # 05**

**Architectural Pattern: MVC (Model View Controller)**

## **Introduction**

**Model–view–controller (MVC) is a software architectural pattern for implementing user interfaces. It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user.**

The *Model-View-Controller (MVC)* pattern separates the modeling of the domain, the presentation, and the actions based on user input into three separate classes:

#### Model

The model manages the behavior and data of the application domain, responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller).

#### View

The view manages the display of information.

#### Controller

The controller interprets the mouse and keyboard inputs from the user, informing the model and/or the view to change as appropriate.

### Structure

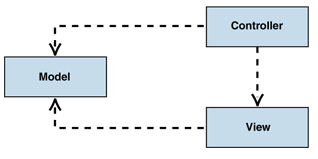


Figure 11 Model View Controller

### Participant

* Model
* View
* Controller

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 5 mints | 5 mints |
| Walk through Theory & Tasks (Lecture) | 60 mints | 60 mints |
| Implement Tasks | 90 mints | 80 mints |
| Evaluation Time | 20 mints | 20 mints |
|  | Total Duration | 175 mints |

## **Objectives**

After completing this lab, student should be able to:

* Clearly understand the purpose and benefits of MVC Pattern.
* The knowledge regarding implementation, usage and structure.

## **Lab Tasks/Practical Work**

1. Develop a program to implement MVC pattern based on student information system scenario.

**Lab Manual for Software Design & Architecture**

**Lab No. 6(A)**

# **CREATIONAL DESIGN PATTERN: SINGLETON**

Objectives

To understand singleton design pattern, structure, implementation, applicability and uses.

**LAB # 06 (A)**

**Creational Design Pattern: Singleton**

## **Introduction**

#### Definition

Ensure a class only has one instance, and provide a global point of access to it.

#### Where to use

When only one instance or a specific number of instances of a class are allowed. Facade objects are often Singletons because only one Facade object is required.

#### Benefits

* Controlled access to unique instance.
* Reduced name space.
* Allows refinement of operations and representations.

#### Drawbacks/consequences

Singleton pattern is also considered an anti-pattern by some people, who feel that it is overused, introducing unnecessary limitations in situations where a sole instance of a class is not actually required.

#### Structure

#### 

Figure 12: Singleton Design Pattern Structure

#### Participant

##### Singleton

* Defines an Instance operation that lets clients access its unique instance. Instance is a class operation (that is, a class method in Smalltalk and a static member function in C++).
* May be responsible for creating its own unique instance.

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 5 mints | 5 mints |
| Walk through Theory & Tasks (Lecture) | 60 mints | 60 mints |
| Implement Tasks | 90 mints | 80 mints |
| Evaluation Time | 20 mints | 20 mints |
|  | Total Duration | 175 mints |

## **Objectives**

After completing this lab, student should be able to:

* Clearly understand the purpose and benefits of singleton design pattern.
* Understand the concept of a Software Design & Architecture and algorithms.

## **Lab Tasks/Practical Work**

* 1. Create a program which implement a singleton pattern using pizza restaurant scenario.

**Lab Manual for Software Design & Architecture**

**Lab No. 6 (B)**

# **CREATIONAL DESIGN PATTERN: BUILDER**

Objectives

To understand builder design pattern, structure, implementation, applicability and uses.

**LAB # 06 (B)**

**Creational Design Pattern: Builder**

## **Introduction**

### Intent

Separate the construction of a complex object from its representation so that the same construction process can create different representations**.**

### Structure

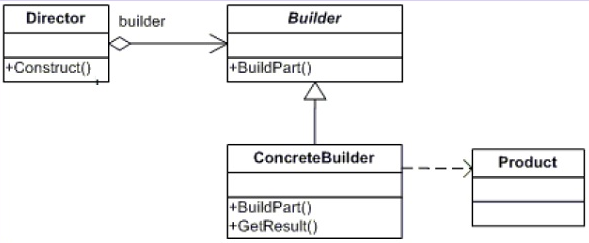
**

Figure 13 Builder Pattern

### Partipicant

Now lets see what each class in the above class diagram is meant for:

##### ConcreteBuilder

Concrete classes that will create the complex Product. This will keep track of what Product it has created i.e. assembled what parts and this will be used by the client to get the Product object.

##### Builder

This is the interface for creating the actual products

##### Director

This is the Client code that will specify the parts needs to be put together to create the actual concrete Product.

##### Product

Thats the object that will be created by assembling many parts.

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 5 mints | 5 mints |
| Walk through Theory & Tasks (Lecture) | 60 mints | 60 mints |
| Implement Tasks | 90 mints | 80 mints |
| Evaluation Time | 20 mints | 20 mints |
|  | Total Duration | 175 mints |

## **Objectives**

After completing this lab, student should be able to:

* Clearly understand the purpose and benefits of Builder design pattern.
* Understand the concept, structure and implementation of Builder pattern.

## **Lab Tasks/Practical Work**

1. Create a program which implement builder pattern by using Shape Creation scenario. (MS Paint)

**Lab Manual for Software Design & Architecture**

**Lab No. 7 (A)**

# **CREATIONAL DESIGN PATTERN: PROTOTYPE**

Objectives

To understand prototype design pattern, structure, implementation, applicability and uses.

**LAB # 07 (A)**

**Creational Design Pattern: Prototype**

## **Introduction**

### Intent

Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

### Structure

Figure 14 Prototype Pattern

### Participant

#### Prototype

Declares an interface for cloning itself.

#### ConcretePrototype

Implements an operation for cloning itself.

#### Client

Creates a new object by asking a prototype to clone itself.

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 5 mints | 5 mints |
| Walk through Theory & Tasks (Lecture) | 60 mints | 60 mints |
| Implement Tasks | 90 mints | 80 mints |
| Evaluation Time | 20 mints | 20 mints |
|  | Total Duration | 175 mints |

## **Objectives**

After completing this lab, student should be able to:

* Clearly understand the purpose and benefits of Prototype design pattern.
* Understand the concept, structure and implementation of prototype pattern.

## **Lab Tasks/Practical Work**

1. Create a program which implement prototype pattern by using Age of Empire game scenario.

**Lab Manual for Software Design & Architecture**

**Lab No. 7 (B)**

# **CREATIONAL DESIGN PATTERN: FACTORY [VIRTUAL CONSTRUCTOR]**

Objectives

To understand builder factory pattern, structure, implementation, applicability and uses.

**LAB # 07 (B)**

**Creational Design Pattern: Factory [Virtual Constructor]**

## **Introduction**

### Intent

**Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.**

### Structure

Figure 15 Factory Pattern

### Participant

#### Product

* Defines the interface of objects the factory method creates.

#### ConcreteProduct

* Implements the Product interface.

#### Creator

* Declares the factory method, which returns an object of type Product. Creator may also define a default implementation of the factory method that returns a default ConcreteProduct object.
* May call the factory method to create a Product object.

#### ConcreteCreator

* Overrides the factory method to return an instance of a ConcreteProduct.

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 5 mints | 5 mints |
| Walk through Theory & Tasks (Lecture) | 60 mints | 60 mints |
| Implement Tasks | 90 mints | 80 mints |
| Evaluation Time | 20 mints | 20 mints |
|  | Total Duration | 175 mints |

## **Objectives**

After completing this lab, student should be able to:

* Clearly understand the purpose and benefits of factory design pattern.
* Understand the concept, structure and implementation of factory pattern.

## **Lab Tasks/Practical Work**

1. Write a program that implement a factory pattern on Vehicle Factory Scenario.

**Lab Manual for Software Design & Architecture**

**Lab No. 8 (A)**

# **CREATIONAL DESIGN PATTERN: ABSTRACT FACTORY PATTERN [KIT]**

Objectives

To understand abstract factory design pattern, structure, implementation, applicability and uses.

**LAB # 08**

**Creational Design Pattern: Abstract Factory [KIT]**

## **Introduction**

### Intent

**Provide an interface for creating families of related or dependent objects without specifying their concrete classes.**

### Structure

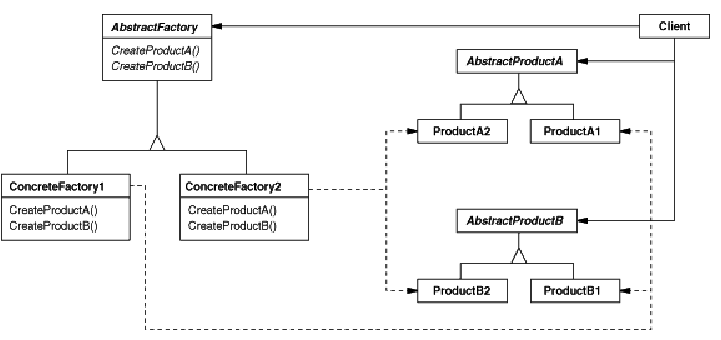


Figure 16 Abstract Factory

### Participant

##### AbstractFactory

* Declares an interface for operations that create abstract product objects.

##### ConcreteFactory

* Implements the operations to create concrete product objects.

##### AbstractProduct

* Declares an interface for a type of product object.

##### ConcreteProduct

* Defines a product object to be created by the corresponding concrete factory. Implements the AbstractProduct interface.

##### Client

* Uses only interfaces declared by AbstractFactory and AbstractProduct classes.

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 5 mints | 5 mints |
| Walk through Theory & Tasks (Lecture) | 60 mints | 60 mints |
| Implement Tasks | 90 mints | 80 mints |
| Evaluation Time | 20 mints | 20 mints |
|  | Total Duration | 175 mints |

## **Objectives**

After completing this lab, student should be able to:

* Clearly understand the purpose and benefits of abstract factory design pattern.
* Understand the concept, structure and implementation of abstract factory pattern.

## **Lab Tasks/Practical Work**

1. Write a program which uses abstract factory pattern for a super factory scenario which contains multiple vehicle factories and produces the end results.

**Lab Manual for Software Design & Architecture**

**Lab No. 9 (A)**

# **STRUCTURAL DESIGN PATTERN: PROXY [SURROGATE]**

Objectives

To understand proxy pattern, structure, implementation, applicability and uses.

**LAB # 09 (A)**

**Structural Design Pattern: Proxy [Surrogate]**

## **Introduction**

### Intent

**Provide a surrogate or placeholder for another object to control access to it.**

### Structure

### Participant

Figure 17 Proxy Pattern

##### Proxy (ImageProxy)

* Maintains a reference that lets the proxy access the real subject. Proxy may refer to a Subject if the realsubject and Subject interfaces are the same.
* Provides an interface identical to Subject's so that a proxy can by substituted for the real subject.
* Controls access to the real subject and may be responsible for creating and deleting it.
* Other responsibilities depend on the kind of proxy:
  + Remote proxies are responsible for encoding a request and its arguments and for sending the encoded request to the real subject in a different address space.
  + Virtual proxies may cache additional information about the real subject so that they can postpone accessing it. For example, the imageproxy from the motivation caches the real image's extent.
  + Protection proxies check that the caller has the access permissions required to perform a request.

##### Subject (Graphic)

* Defines the common interface for realsubject and Proxy so that a Proxy can be used anywhere a realsubject is expected.

##### RealSubject (Image)

* Defines the real object that the proxy represents.

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 5 mints | 5 mints |
| Walk through Theory & Tasks (Lecture) | 60 mints | 60 mints |
| Implement Tasks | 90 mints | 80 mints |
| Evaluation Time | 20 mints | 20 mints |
|  | Total Duration | 175 mints |

## **Objectives**

After completing this lab, student should be able to:

* Clearly understand the purpose and benefits of proxy design pattern.
* Understand the concept, structure and implementation of proxy pattern.

## **Lab Tasks/Practical Work**

1. Write a program to reduce memory footprint of Real Image object loading by using Proxy Pattern.

**Lab Manual for Software Design & Architecture**

**Lab No. 9 (B)**

# **STRUCTURAL DESIGN PATTERN: COMPOSITE**

Objectives

To understand composite pattern, structure, implementation, applicability and uses.

**LAB # 09 (B)**

**Structural Design Pattern: Composite**

## **Introduction**

### Intent

**Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.**

### Structure

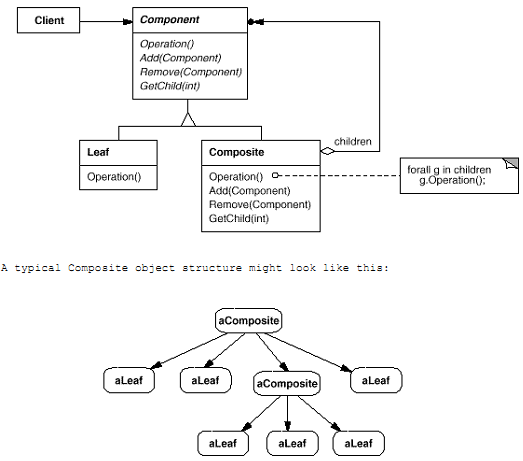


Figure 18 Composie Pattern

### Participant

##### Component (Graphic)

Declares the interface for objects in the composition.

Implements default behavior for the interface common to all classes, as appropriate.

Declares an interface for accessing and managing its child components.

(optional) Defines an interface for accessing a component's parent in the recursive structure, and implements it if that's appropriate.

##### Leaf (Rectangle, Line, Text, etc.)

Represents leaf objects in the composition. A leaf has no children.

Defines behavior for primitive objects in the composition.

##### Composite (Picture)

Defines behavior for components having children.

Stores child components.

Implements child-related operations in the component interface.

##### Client

Manipulates objects in the composition through the Component interface.

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 5 mints | 5 mints |
| Walk through Theory & Tasks (Lecture) | 60 mints | 60 mints |
| Implement Tasks | 90 mints | 80 mints |
| Evaluation Time | 20 mints | 20 mints |
|  | Total Duration | 175 mints |

## **Objectives**

After completing this lab, student should be able to:

* Clearly understand the purpose and benefits of composite design pattern.
* Understand the concept, structure and implementation of composite pattern.

## **Lab Tasks/Practical Work**

1. Write a program to manage a software company hierarchy by implementing composite pattern.

**Lab Manual for Software Design & Architecture**

**Lab No. 9 (C)**

# **STRUCTURAL DESIGN PATTERN: ADAPTER [WRAPPER]**

Objectives

To understand adapter pattern, structure, implementation, applicability and uses.

**LAB # 09 (C)**

**Structural Design Pattern: Adapter [Wrapper]**

## **Introduction**

### Intent

**Convert the interface of a class into another interface client expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.**

### Structure

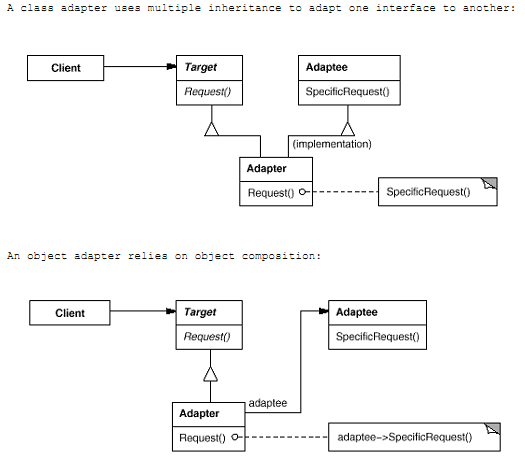


Figure 19 Adpater Pattern

### Participant

##### Target (Shape)

* Defines the domain-specific interface that client uses.

##### Client (DrawingEditor)

* Collaborates with objects conforming to the Target interface.

##### Adaptee (TextView)

* Defines an existing interface that needs adapting.

##### Adapter (TextShape)

* Adapts the interface of Adaptee to the Target interface.

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 5 mints | 5 mints |
| Walk through Theory & Tasks (Lecture) | 60 mints | 60 mints |
| Implement Tasks | 90 mints | 80 mints |
| Evaluation Time | 20 mints | 20 mints |
|  | Total Duration | 175 mints |

## **Objectives**

After completing this lab, student should be able to:

* Clearly understand the purpose and benefits of adapter design pattern.
* Understand the concept, structure and implementation of adapter pattern.

## **Lab Tasks/Practical Work**

1. Write a program to implement Adapter pattern by using a Media Player Legacy and New Interface, consist of different play methods for multiple extension files.

**Lab Manual for Software Design & Architecture**

**Lab No. 10 (A)**

# **STRUCTURAL DESIGN PATTERN: DECORATOR**

Objectives

To understand decorator pattern, structure, implementation, applicability and uses.

**LAB # 10 (A)**

**Structural Design Pattern: Decorator [Wrapper]**

## **Introduction**

### Intent

**Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.**

### Structure

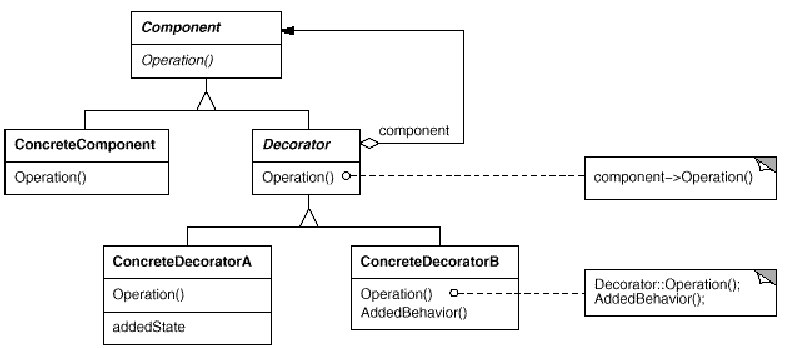


Figure 20 Decorator Pattern

### Participant

##### Component (VisualComponent)

* Defines the interface for objects that can have responsibilities added to them dynamically.

##### ConcreteComponent (TextView)

* Defines an object to which additional responsibilities can be attached.

##### Decorator

* Maintains a reference to a Component object and defines an interface that conforms to Component's interface.

##### ConcreteDecorator (BorderDecorator, ScrollDecorator)

* Adds responsibilities to the component.

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 5 mints | 5 mints |
| Walk through Theory & Tasks (Lecture) | 60 mints | 60 mints |
| Implement Tasks | 90 mints | 80 mints |
| Evaluation Time | 20 mints | 20 mints |
|  | Total Duration | 175 mints |

## **Objectives**

After completing this lab, student should be able to:

* Clearly understand the purpose and benefits of decorator design pattern.
* Understand the concept, structure and implementation of decorator pattern.

## **Lab Tasks/Practical Work**

1. Write a program to implement decorator pattern using a shape canvas scenario.

**Lab Manual for Software Design & Architecture**

**Lab No. 10 (B)**

# **STRUCTURAL DESIGN PATTERN: BRIDGE [HANDLE/BODY]**

Objectives

To understand bridge pattern, structure, implementation, applicability and uses.

**LAB # 10 (B)**

**Structural Design Pattern: Bridge (Handle/Body)**

## **Introduction**

## Intent

**Decouple an abstraction from its implementation so that the two can vary independently.**

### Structure

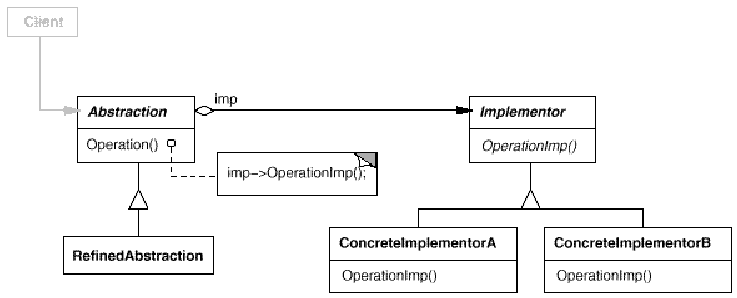


Figure 21 Bridge Pattern

### Participant

##### Abstraction

* Defines the abstraction's interface.
* Maintains a reference to an object of type Implementor.

##### RefinedAbstraction

* Extends the interface defined by Abstraction.

##### Implementor

* Defines the interface for implementation classes. This interface doesn't have to correspond exactly to Abstraction's interface; in fact the two interfaces can be quite different. Typically, the Implementor interface provides only primitive operations, and Abstraction defines higher-level operations based on these primitives.

##### ConcreteImplementor

* Implements the Implementor interface and defines its concrete implementation.

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 5 mints | 5 mints |
| Walk through Theory & Tasks (Lecture) | 60 mints | 60 mints |
| Implement Tasks | 90 mints | 80 mints |
| Evaluation Time | 20 mints | 20 mints |
|  | Total Duration | 175 mints |

## **Objectives**

After completing this lab, student should be able to:

* Clearly understand the purpose and benefits of bridge design pattern.
* Understand the concept, structure and implementation of bridge pattern.

## **Lab Tasks/Practical Work**

1. Write a program to draw a shape in different colors using same abstract class method but different bridge implementer classes.

**Lab Manual for Software Design & Architecture**

**Lab No. 11 (A)**

# **STRUCTURAL DESIGN PATTERN: FAÇADE**

Objectives

To understand Façadepattern, structure, implementation, applicability and uses.

**LAB # 09 (B)**

**Structural Design Pattern: Façade**

## **Introduction**

### Intent

**Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.**

### Structure

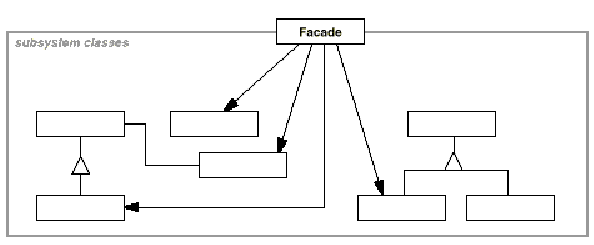


Figure 22 Façade Pattern

### Participant

##### Facade (Compiler)

* Knows which subsystem classes are responsible for a request.
* Delegates client requests to appropriate subsystem objects.

##### Subsystem classes (Scanner, Parser, ProgramNode, etc.)

* Implement subsystem functionality.
* Handle work assigned by the facade object.
* Have no knowledge of the facade; that is, they keep no references to it.

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 5 mints | 5 mints |
| Walk through Theory & Tasks (Lecture) | 60 mints | 60 mints |
| Implement Tasks | 90 mints | 80 mints |
| Evaluation Time | 20 mints | 20 mints |
|  | Total Duration | 175 mints |

## **Objectives**

After completing this lab, student should be able to:

* Clearly understand the purpose and benefits of composite Façade pattern.
* Understand the concept, structure and implementation of Façade pattern.

## **Lab Tasks/Practical Work**

1. Write a program to implement façade pattern using a shape maker scenario.

**Lab Manual for Software Design & Architecture**

**Lab No. 11 (B)**

# **STRUCTURAL DESIGN PATTERN: FLYWEIGHT**

Objectives

To understand flyweight pattern, structure, implementation, applicability and uses.

**LAB # 11 (B)**

**Structural Design Pattern: Flyweight**

## **Introduction**

### Intent

**Use sharing to support large numbers of fine-grained objects efficiently.**

### Structure

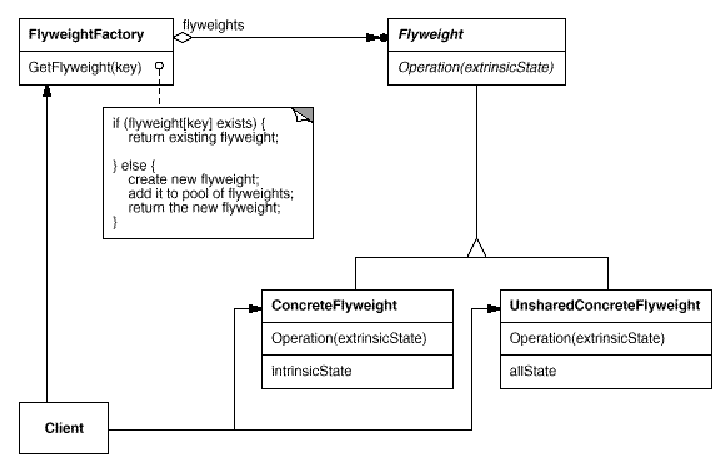
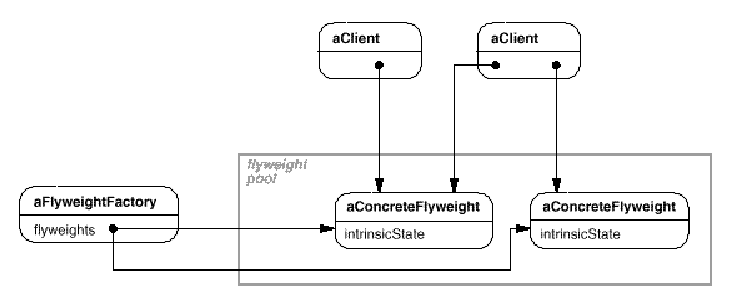


Figure 23 Flyweight Pattern

The following object diagram shows how flyweights are shared:



### Participant

##### Flyweight

* Declares an interface through which flyweights can receive and act on extrinsic state.

##### ConcreteFlyweight (Character)

* Implements the Flyweight interface and adds storage for intrinsic state, if any. A ConcreteFlyweight object must be sharable. Any state it stores must be intrinsic; that is, it must be independent of the ConcreteFlyweight object's context.

##### UnsharedConcreteFlyweight (Row, Column)

* Not all Flyweight subclasses need to be shared. The Flyweight interface enables sharing; it doesn't enforce it. It's common for UnsharedConcreteFlyweight objects to have ConcreteFlyweight objects as children at some level in the flyweight object structure (as the Row and Column classes have).

##### FlyweightFactory

* Creates and manages flyweight objects.
* Ensures that flyweights are shared properly. When a client requests a flyweight, the flyweightfactory object supplies an existing instance or creates one, if none exists.

##### Client

* Maintains a reference to flyweight(s).
* Computes or stores the extrinsic state of flyweight(s).

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 5 mints | 5 mints |
| Walk through Theory & Tasks (Lecture) | 60 mints | 60 mints |
| Implement Tasks | 90 mints | 80 mints |
| Evaluation Time | 20 mints | 20 mints |
|  | Total Duration | 175 mints |

## **Objectives**

After completing this lab, student should be able to:

* Clearly understand the purpose and benefits of flyweight design pattern.
* Understand the concept, structure and implementation of flyweight pattern.

## **Lab Tasks/Practical Work**

1. Write a program to implement flyweight pattern for a color factory.

**Lab Manual for Software Design & Architecture**

**Lab No. 12 (A)**

# **BEHAVIORAL DESIGN PATTERN: CHAIN OF RESPONSIBILITY PATTERN**

Objectives

To understand chain of responsibility pattern, structure, implementation, applicability and uses.

**LAB # 12 (A)**

**Behavioral Design Pattern: Chain of Responsibility**

## **Introduction**

### Intent

**Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.**

### Structure

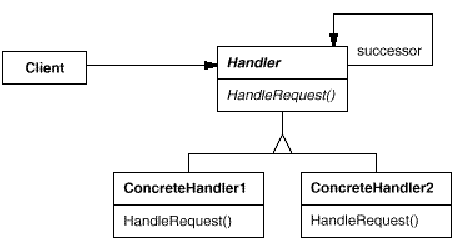


Figure 24 Chain of Responsibility

### Participant

##### Handler (HelpHandler)

* Defines an interface for handling requests.
* (optional) implements the successor link.

##### ConcreteHandler (PrintButton, PrintDialog)

* Handles requests it is responsible for.
* Can access its successor.
* If the concretehandler can handle the request, it does so; otherwise it forwards the request to its successor.

##### Client

Initiates the request to a ConcreteHandler object on the chain.

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 5 mints | 5 mints |
| Walk through Theory & Tasks (Lecture) | 60 mints | 60 mints |
| Implement Tasks | 90 mints | 80 mints |
| Evaluation Time | 20 mints | 20 mints |
|  | Total Duration | 175 mints |

## **Objectives**

After completing this lab, student should be able to:

* Clearly understand the purpose and benefits of chain of responsibility design pattern.
* Understand the concept, structure and implementation of chain of responsibility pattern.

## **Lab Tasks/Practical Work**

1. Create a logger (error, events occurred) by using chain of responsibility pattern.

**Lab Manual for Software Design & Architecture**

**Lab No. 12 (B)**

# **BEHAVIORAL DESIGN PATTERN: VISITOR**

Objectives

To understand visitor pattern, structure, implementation, applicability and uses.

**LAB # 12 (B)**

**Behavioral Design Pattern: Visitor**

## **Introduction**

### Intent

**Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates.**

### Structure

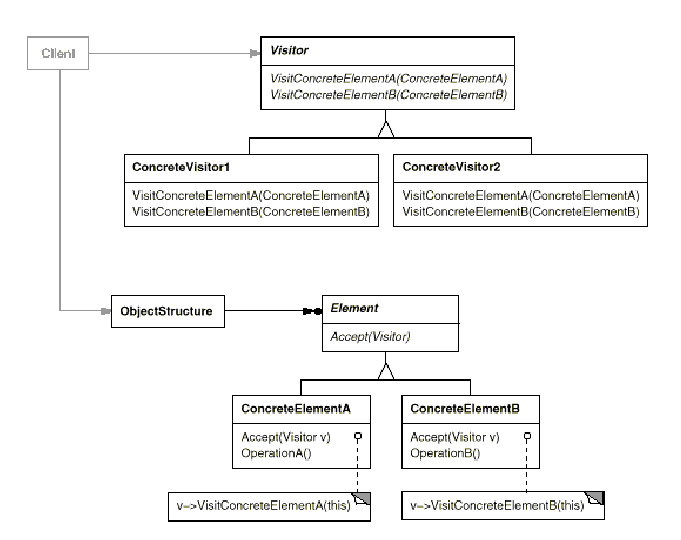


Figure 25 Visitor Pattern

### Participants

##### Visitor (NodeVisitor)

* Declares a Visit operation for each class of ConcreteElement in the object structure. The operation's name and signature identifies the class that sends the Visit request to the visitor. That lets the visitor determine the concrete class of the element being visited. Then the visitor can access the element directly through its particular interface.

##### ConcreteVisitor (TypeCheckingVisitor)

* Implements each operation declared by Visitor. Each operation implements a fragment of the algorithm defined for the corresponding class of object in the structure. ConcreteVisitor provides the context for the algorithm and stores its local state. This state often accumulates results during the traversal of the structure.

##### Element (Node)

* Defines an Accept operation that takes a visitor as an argument.

##### ConcreteElement **(**AssignmentNode,VariableRefNode**)**

* Implements an Accept operation that takes a visitor as an argument.

##### ObjectStructure (Program)

* Can enumerate its elements.
* May provide a high-level interface to allow the visitor to visit its elements.

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 5 mints | 5 mints |
| Walk through Theory & Tasks (Lecture) | 60 mints | 60 mints |
| Implement Tasks | 90 mints | 80 mints |
| Evaluation Time | 20 mints | 20 mints |
|  | Total Duration | 175 mints |

## **Objectives**

After completing this lab, student should be able to:

* Clearly understand the purpose and benefits of visitor design pattern.
* Understand the concept, structure and implementation of visitor pattern.

## **Lab Tasks/Practical Work**

1. Write a program to implement visitor pattern.

**Lab Manual for Software Design & Architecture**

**Lab No. 13 (A)**

# **BEHAVIORAL DESIGN PATTERN: OBSERVER [DEPENDENTS, PUBLISH-SUBSCRIBE]**

Objectives

To understand observer pattern, structure, implementation, applicability and uses.

**LAB # 13 (A)**

**Behavioral Design Pattern: Observer [Dependents, Publish-Subscribe]**

## **Introduction**

### Intent

**Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.**

### Structure

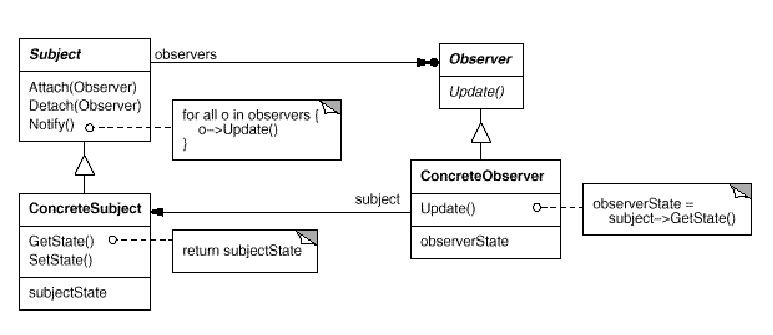


Figure 26 Observer Pattern

### Participants

##### Subject

* Knows its observers. Any number of Observer objects may observe a subject.
* Provides an interface for attaching and detaching Observer objects.

##### Observer

* Defines an updating interface for objects that should be notified of changes in a subject.

##### ConcreteSubject

* Stores state of interest to ConcreteObserver objects.
* Sends a notification to its observers when its state changes.

##### ConcreteObserver

* Maintains a reference to a ConcreteSubject object.
* Stores state that should stay consistent with the subject's.
* Implements the Observer updating interface to keep its state consistent with the subject's.

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 5 mints | 5 mints |
| Walk through Theory & Tasks (Lecture) | 60 mints | 60 mints |
| Implement Tasks | 90 mints | 80 mints |
| Evaluation Time | 20 mints | 20 mints |
|  | Total Duration | 175 mints |

## **Objectives**

After completing this lab, student should be able to:

* Clearly understand the purpose and benefits of observer design pattern.
* Understand the concept, structure and implementation of observer pattern.

## **Lab Tasks/Practical Work**

1. Use Observer pattern with your factory method program to observe the states of products creation.

**Lab Manual for Software Design & Architecture**

**Lab No. 13 (B)**

# **BEHAVIORAL DESIGN PATTERN: STATE PATTERN [OBJECT OF STATES]**

Objectives

To understand state pattern, structure, implementation, applicability and uses.

**LAB # 13 (B)**

**Behavioral Design Pattern: State [Object of States]**

## **Introduction**

### Intent

**Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.**

### Structure

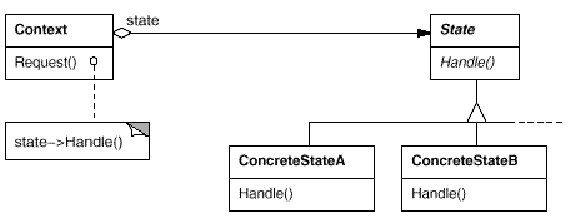


Figure 27 State Pattern

### Participant

##### Context (TCPConnection)

* Defines the interface of interest to clients.
* Maintains an instance of a concretestate subclass that defines the current state.

##### State (TCPState)

* Defines an interface for encapsulating the behavior associated with a particular state of the Context.

##### ConcreteState subclasses (TCPEstablished, TCPListen, TCPClosed)

* Each subclass implements a behavior associated with a state of the Context.

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 5 mints | 5 mints |
| Walk through Theory & Tasks (Lecture) | 60 mints | 60 mints |
| Implement Tasks | 90 mints | 80 mints |
| Evaluation Time | 20 mints | 20 mints |
|  | Total Duration | 175 mints |

## **Objectives**

After completing this lab, student should be able to:

* Clearly understand the purpose and benefits of state design pattern.
* Understand the concept, structure and implementation of state pattern.

## **Lab Tasks/Practical Work**

1. Write a program to implement State pattern.

**Lab Manual for Software Design & Architecture**

**Lab No. 14 (A)**

# **BEHAVIORAL DESIGN PATTERN: STRATEGY [POLICY]**

Objectives

To understand strategy pattern, structure, implementation, applicability and uses.

**LAB # 14 (A)**

**Behavioral Design Pattern: Strategy [Policy]**

## **Introduction**

### Intent

**Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.**

### Structure

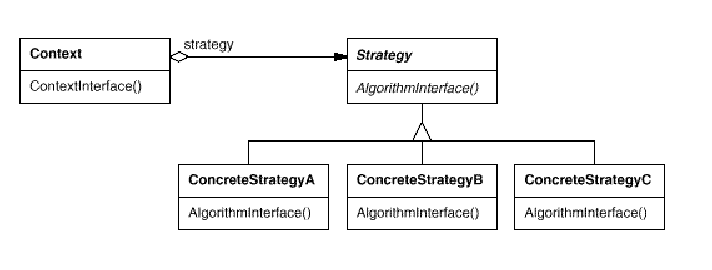


Figure 28 Strategy Pattern

### Participants

##### Strategy (Compositor)

* Declares an interface common to all supported algorithms. Context uses this interface to call the algorithm defined by a Concrete Strategy.

##### ConcreteStrategy **(**SimpleCompositor, TeXCompositor, ArrayCompositor**)**

* Implements the algorithm using the Strategy interface.

##### Context (Composition)

* Is configured with a ConcreteStrategy object.
* Maintains a reference to a Strategy object.
* May define an interface that lets Strategy access its data.

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 5 mints | 5 mints |
| Walk through Theory & Tasks (Lecture) | 60 mints | 60 mints |
| Implement Tasks | 90 mints | 80 mints |
| Evaluation Time | 20 mints | 20 mints |
|  | Total Duration | 175 mints |

## **Objectives**

After completing this lab, student should be able to:

* Clearly understand the purpose and benefits of strategy design pattern.
* Understand the concept, structure and implementation of strategy pattern.

## **Lab Tasks/Practical Work**

1. Write a program to manage a software company hierarchy by implementing composite pattern.

**Lab Manual for Software Design & Architecture**

**Lab No. 14 (B)**

# **BEHAVIORAL DESIGN PATTERN: TEMPLATE METHOD**

Objectives

To understand template method pattern, structure, implementation, applicability and uses.

**LAB # 14 (B)**

**Behavioral Design Pattern: Template Method**

## **Introduction**

### Intent

**Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.**

### Structure

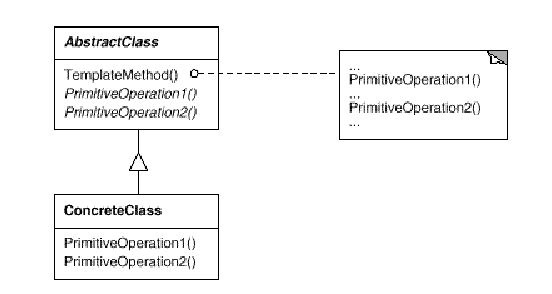


Figure 29 Template Pattern

### Participants

##### AbstractClass (Application)

* Defines abstract primitive operations that concrete subclasses define to implement steps of an algorithm.
* Implements a template method defining the skeleton of an algorithm. The template method calls primitive operations as well as operations defined in Abstract Class or those of other objects.

##### ConcreteClass (MyApplication)

* Implements the primitive operations to carry out subclass-specifi csteps of the algorithm.

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 5 mints | 5 mints |
| Walk through Theory & Tasks (Lecture) | 60 mints | 60 mints |
| Implement Tasks | 90 mints | 80 mints |
| Evaluation Time | 20 mints | 20 mints |
|  | Total Duration | 175 mints |

## **Objectives**

After completing this lab, student should be able to:

* Clearly understand the purpose and benefits of adapter design pattern.
* Understand the concept, structure and implementation of adapter pattern.

## **Lab Tasks/Practical Work**

1. Write a program to implement template pattern by using application life cycle template. (initialize, load, process, end).

**Lab Manual for Software Design & Architecture**

**Lab No. 14 (C)**

# **BEHAVIORAL DESIGN PATTERN: NULL OBJECT**

Objectives

To understand null object pattern, structure, implementation, applicability and uses.

**LAB # 14 (C)**

**Behavioral Design Pattern: Null Object**

## **Introduction**

### Intent

**A Null Object is to encapsulate the absence of an object by providing a substitutable alternative that offers suitable default do nothing behavior. In short, a design where "nothing will come of nothing"**

##### **Use the** Null **Object pattern when**

* **An object requires a collaborator. The Null Object pattern does not introduce this collaboration—it makes use of a collaboration that already exists**
* **Some collaborator instances should do nothing**
* **You want to abstract the handling of null away from the client**

### Structure

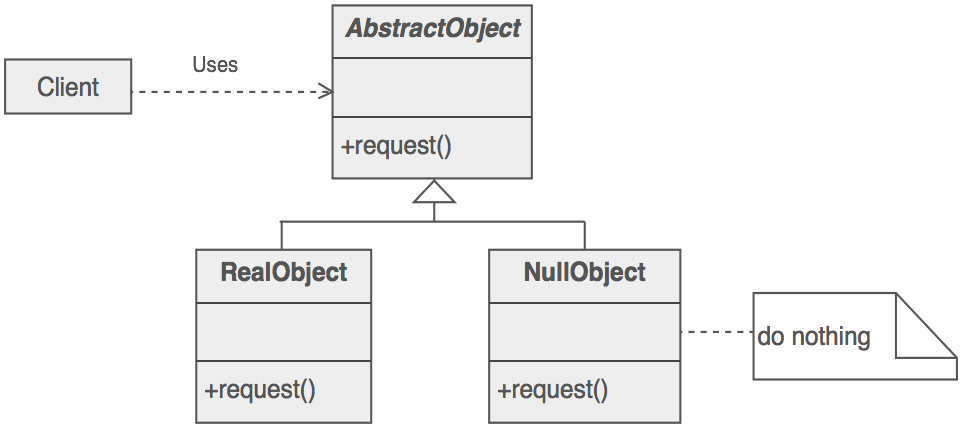


Figure 30 Null Object Pattern

### Participant

##### Client

* Requires a collaborator.

##### AbstractObject

* Declares the interface for Client's collaborator
* Implements default behavior for the interface common to all classes, as appropriate

##### RealObject

* Defines a concrete subclass of AbstractObject whose instances provide useful behavior that Client expects

##### NullObject

* Provides an interface identical to AbstractObject's so that a null object can be substituted for a real object.
* Implements its interface to do nothing. What exactly it means to do nothing depends on what sort of behavior Client is expecting. When there is more than one way to do nothing, more than one Null Object class may be required

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 5 mints | 5 mints |
| Walk through Theory & Tasks (Lecture) | 60 mints | 60 mints |
| Implement Tasks | 90 mints | 80 mints |
| Evaluation Time | 20 mints | 20 mints |
|  | Total Duration | 175 mints |

## **Objectives**

After completing this lab, student should be able to:

* Clearly understand the purpose and benefits of null object design pattern.
* Understand the concept, structure and implementation of null object pattern.

## **Lab Tasks/Practical Work**

1. Write a program to implement a null object pattern in a subscription plan scenario.

**Lab Manual for Software Design & Architecture**

**Lab No. 15 (A)**

# **BEHAVIORAL DESIGN PATTERN: MEDIATOR**

Objectives

To understand mediator pattern, structure, implementation, applicability and uses.

**LAB # 15 (A)**

**Behavioral Design Pattern: Mediator**

## **Introduction**

### Intent

**Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.**

### Structure

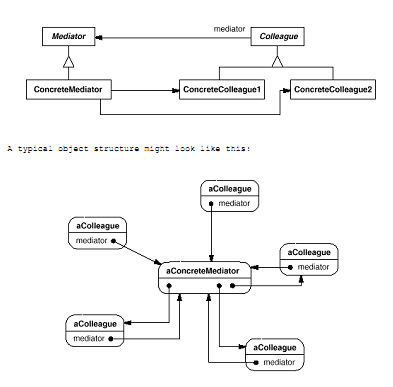


Figure 31 Mediator Pattern

### Participant

##### Mediator (DialogDirector)

* Defines an interface for communicating with Colleague objects.

##### ConcreteMediator (FontDialogDirector)

* Implements cooperative behavior by coordinating Colleague objects.
* Knows and maintains its colleagues.

##### Colleague classes (ListBox, EntryField)

* Each Colleague class knows its Mediator object.
* Each colleague communicates with its mediator whenever it would have otherwise communicated with another colleague.

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 5 mints | 5 mints |
| Walk through Theory & Tasks (Lecture) | 60 mints | 60 mints |
| Implement Tasks | 90 mints | 80 mints |
| Evaluation Time | 20 mints | 20 mints |
|  | Total Duration | 175 mints |

## **Objectives**

After completing this lab, student should be able to:

* Clearly understand the purpose and benefits of mediator design pattern.
* Understand the concept, structure and implementation of mediator pattern.

## **Lab Tasks/Practical Work**

1. Write a program for a chat room application by using mediator pattern.

**Lab Manual for Software Design & Architecture**

**Lab No. 15 (B)**

# **BEHAVIORAL DESIGN PATTERN: ITERATOR [CURSOR]**

Objectives

To understand iterator pattern, structure, implementation, applicability and uses.

**LAB # 15 (B)**

**Behavioral Design Pattern: Iterator [Cursor]**

## **Introduction**

### Intent

**Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.**

### Structure

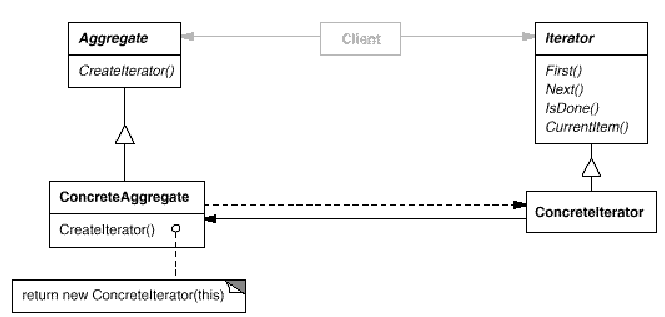


Figure 32 Iterator Pattern

### Participant

##### Iterator

* Defines an interface for accessing and traversing elements

##### ConcreteIterator

* Implements the Iterator interface.
* Keeps track of the current position in the traversal of the aggregate.

##### Aggregate

* Defines an interface for creating an Iterator object.

##### ConcreteAggregate

* Implements the Iterator creation interface to return an instance of the proper ConcreteIterator.

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 5 mints | 5 mints |
| Walk through Theory & Tasks (Lecture) | 60 mints | 60 mints |
| Implement Tasks | 90 mints | 80 mints |
| Evaluation Time | 20 mints | 20 mints |
|  | Total Duration | 175 mints |

## **Objectives**

After completing this lab, student should be able to:

* Clearly understand the purpose and benefits of iterator design pattern.
* Understand the concept, structure and implementation of iterator pattern.

## **Lab Tasks/Practical Work**

1. Write a program for student records file using iterator pattern.

# **Appendix**

The [C# Language Specification](http://go.microsoft.com/fwlink/?LinkId=199552) does not define a coding standard. However, the guidelines in this topic are used by Microsoft to develop samples and documentation.

Coding conventions serve the following purposes:

They create a consistent look to the code, so that readers can focus on content, not layout.

They enable readers to understand the code more quickly by making assumptions based on previous experience.

They facilitate copying, changing, and maintaining the code.

They demonstrate C# best practices.

### [Naming Conventions](javascript:void(0))

In short examples that do not include [using directives](https://msdn.microsoft.com/en-us/library/sf0df423.aspx), use namespace qualifications. If you know that a namespace is imported by default in a project, you do not have to fully qualify the names from that namespace. Qualified names can be broken after a dot (.) if they are too long for a single line, as shown in the following example.

var currentPerformanceCounterCategory = new System.Diagnostics.

PerformanceCounterCategory();

You do not have to change the names of objects that were created by using the Visual Studio designer tools to make them fit other guidelines.

### [Layout Conventions](javascript:void(0))

Good layout uses formatting to emphasize the structure of your code and to make the code easier to read. Microsoft examples and samples conform to the following conventions:

Use the default Code Editor settings (smart indenting, four-character indents, tabs saved as spaces). For more information, see [Options, Text Editor, C#, Formatting](https://msdn.microsoft.com/en-us/library/03864tbz.aspx).

Write only one statement per line.

Write only one declaration per line.

If continuation lines are not indented automatically, indent them one tab stop (four spaces).

Add at least one blank line between method definitions and property definitions.

Use parentheses to make clauses in an expression apparent, as shown in the following code.

if ((val1 > val2) && (val1 > val3))

{

// Take appropriate action.

}

### [Commenting Conventions](javascript:void(0))

Place the comment on a separate line, not at the end of a line of code.

Begin comment text with an uppercase letter.

End comment text with a period.

Insert one space between the comment delimiter (//) and the comment text, as shown in the following example.

// The following declaration creates a query. It does not run

// the query.

Do not create formatted blocks of asterisks around comments.

### [Language Guidelines](javascript:void(0))

The following sections describe practices that the C# team follows to prepare code examples and samples.

#### [String Data Type](javascript:void(0))

Use the **+** operator to concatenate short strings, as shown in the following code.

string displayName = nameList[n].LastName + ", " + nameList[n].FirstName;

To append strings in loops, especially when you are working with large amounts of text, use a [StringBuilder](https://msdn.microsoft.com/en-us/library/system.text.stringbuilder.aspx) object.

var phrase = "lalalalalalalalalalalalalalalalalalalalalalalalalalalalalala";

var manyPhrases = new StringBuilder();

for (var i = 0; i < 10000; i++)

{

manyPhrases.Append(phrase);

}

//Console.WriteLine("tra" + manyPhrases);

### [Implicitly Typed Local Variables](javascript:void(0))

Use [implicit typing](https://msdn.microsoft.com/en-us/library/bb384061.aspx) for local variables when the type of the variable is obvious from the right side of the assignment, or when the precise type is not important.

// When the type of a variable is clear from the context, use var

// in the declaration.

var var1 = "This is clearly a string.";

var var2 = 27;

var var3 = Convert.ToInt32(Console.ReadLine());

Do not use [var](https://msdn.microsoft.com/en-us/library/bb383973.aspx) when the type is not apparent from the right side of the assignment.

// When the type of a variable is not clear from the context, use an

// explicit type.

int var4 = ExampleClass.ResultSoFar();

Do not rely on the variable name to specify the type of the variable. It might not be correct.

// Naming the following variable inputInt is misleading.

// It is a string.

var inputInt = Console.ReadLine();

Console.WriteLine(inputInt);

Avoid the use of **var** in place of [dynamic](https://msdn.microsoft.com/en-us/library/dd264741.aspx).

Use implicit typing to determine the type of the loop variable in [for](https://msdn.microsoft.com/en-us/library/ch45axte.aspx) and [foreach](https://msdn.microsoft.com/en-us/library/ttw7t8t6.aspx) loops.

The following example uses implicit typing in a **for** statement.

var syllable = "ha";

var laugh = "";

for (var i = 0; i < 10; i++)

{

laugh += syllable;

Console.WriteLine(laugh);

}

The following example uses implicit typing in a **foreach** statement.

foreach (var ch in laugh)

{

if (ch == 'h')

Console.Write("H");

else

Console.Write(ch);

}

Console.WriteLine();

### [Unsigned Data Type](javascript:void(0))

In general, use **int** rather than unsigned types. The use of **int** is common throughout C#, and it is easier to interact with other libraries when you use **int**.

### [Arrays](javascript:void(0))

Use the concise syntax when you initialize arrays on the declaration line.

var localDistributors =

from customer in customers

join distributor in distributors on customer.City equals distributor.City

select new { Customer = customer, Distributor = distributor };

### [Delegates](javascript:void(0))

Use the concise syntax to create instances of a delegate type.

// Preferred syntax. Note that you cannot use var here instead of string[].

string[] vowels1 = { "a", "e", "i", "o", "u" };

// If you use explicit instantiation, you can use var.

var vowels2 = new string[] { "a", "e", "i", "o", "u" };

// If you specify an array size, you must initialize the elements one at a time.

var vowels3 = new string[5];

vowels3[0] = "a";

vowels3[1] = "e";

// And so on.

// First, in class Program, define the delegate type and a method that

// has a matching signature.

// Define the type.

public delegate void Del(string message);

// Define a method that has a matching signature.

public static void DelMethod(string str)

{

Console.WriteLine("DelMethod argument: {0}", str);

}

// In the Main method, create an instance of Del.

// Preferred: Create an instance of Del by using condensed syntax.

Del exampleDel2 = DelMethod;

// The following declaration uses the full syntax.

Del exampleDel1 = new Del(DelMethod);

### [try-catch and using Statements in Exception Handling](javascript:void(0))

Use a [try-catch](https://msdn.microsoft.com/en-us/library/0yd65esw.aspx) statement for most exception handling.

static string GetValueFromArray(string[] array, int index)

{

try

{

return array[index];

}

catch (System.IndexOutOfRangeException ex)

{

Console.WriteLine("Index is out of range: {0}", index);

throw;

}

}

Simplify your code by using the C# [using statement](https://msdn.microsoft.com/en-us/library/yh598w02.aspx). If you have a [try-finally](https://msdn.microsoft.com/en-us/library/zwc8s4fz.aspx) statement in which the only code in the **finally** block is a call to the [Dispose](https://msdn.microsoft.com/en-us/library/system.idisposable.dispose.aspx) method, use a **using** statement instead.

// This try-finally statement only calls Dispose in the finally block.

Font font1 = new Font("Arial", 10.0f);

try

{

byte charset = font1.GdiCharSet;

}

finally

{

if (font1 != null)

{

((IDisposable)font1).Dispose();

}

}

// You can do the same thing with a using statement.

using (Font font2 = new Font("Arial", 10.0f))

{

byte charset = font2.GdiCharSet;

}

### [&& and || Operators](javascript:void(0))

To avoid exceptions and increase performance by skipping unnecessary comparisons, use [&&](https://msdn.microsoft.com/en-us/library/2a723cdk.aspx) instead of [&](https://msdn.microsoft.com/en-us/library/sbf85k1c.aspx) and [||](https://msdn.microsoft.com/en-us/library/6373h346.aspx) instead of [|](https://msdn.microsoft.com/en-us/library/kxszd0kx.aspx) when you perform comparisons, as shown in the following example.

Console.Write("Enter a dividend: ");

var dividend = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter a divisor: ");

var divisor = Convert.ToInt32(Console.ReadLine());

// If the divisor is 0, the second clause in the following condition

// causes a run-time error. The && operator short circuits when the

// first expression is false. That is, it does not evaluate the

// second expression. The & operator evaluates both, and causes

// a run-time error when divisor is 0.

if ((divisor != 0) && (dividend / divisor > 0))

{

Console.WriteLine("Quotient: {0}", dividend / divisor);

}

else

{

Console.WriteLine("Attempted division by 0 ends up here.");

}

### [New Operator](javascript:void(0))

Use the concise form of object instantiation, with implicit typing, as shown in the following declaration.

var instance1 = new ExampleClass();

The previous line is equivalent to the following declaration.

ExampleClass instance2 = new ExampleClass();

Use object initializers to simplify object creation.

// Object initializer.

var instance3 = new ExampleClass { Name = "Desktop", ID = 37414,

Location = "Redmond", Age = 2.3 };

// Default constructor and assignment statements.

var instance4 = new ExampleClass();

instance4.Name = "Desktop";

instance4.ID = 37414;

instance4.Location = "Redmond";

instance4.Age = 2.3;

### [Event Handling](javascript:void(0))

If you are defining an event handler that you do not need to remove later, use a lambda expression.

public Form2()

{

// You can use a lambda expression to define an event handler.

this.Click += (s, e) =>

{

MessageBox.Show(

((MouseEventArgs)e).Location.ToString());

};

}

### [Static Members](javascript:void(0))

// Using a lambda expression shortens the following traditional definition.

public Form1()

{

this.Click += new EventHandler(Form1\_Click);

}

void Form1\_Click(object sender, EventArgs e)

{

MessageBox.Show(((MouseEventArgs)e).Location.ToString());

}

Call [static](https://msdn.microsoft.com/en-us/library/98f28cdx.aspx) members by using the class name: ClassName.StaticMember. This practice makes code more readable by making static access clear. Do not qualify a static member defined in a base class with the name of a derived class. While that code compiles, the code readability is misleading, and the code may break in the future if you add a static member with the same name to the derived class.

### [LINQQueries](javascript:void(0))

Use meaningful names for query variables. The following example uses seattleCustomers for customers who are located in Seattle.

var seattleCustomers = from cust in customers

where cust.City == "Seattle"

select cust.Name;

Use aliases to make sure that property names of anonymous types are correctly capitalized, using Pascal casing.

var localDistributors =

from customer in customers

join distributor in distributors on customer.City equals distributor.City

select new { Customer = customer, Distributor = distributor };

Rename properties when the property names in the result would be ambiguous. For example, if your query returns a customer name and a distributor ID, instead of leaving them as Name and ID in the result, rename them to clarify that Name is the name of a customer, and IDis the ID of a distributor.

Use implicit typing in the declaration of query variables and range variables.

Align query clauses under the [from](https://msdn.microsoft.com/en-us/library/bb383978.aspx) clause, as shown in the previous examples.

var seattleCustomers = from cust in customers

where cust.City == "Seattle"

select cust.Name;

var localDistributors2 =

from cust in customers

join dist in distributors on cust.City equals dist.City

select new { CustomerName = cust.Name, DistributorID = dist.ID };

Use [where](https://msdn.microsoft.com/en-us/library/bb311043.aspx) clauses before other query clauses to ensure that later query clauses operate on the reduced, filtered set of data.

var seattleCustomers2 = from cust in customers

where cust.City == "Seattle"

orderby cust.Name

select cust;

Use multiple **from** clauses instead of a [join](https://msdn.microsoft.com/en-us/library/bb311040.aspx) clause to access inner collections. For example, a collection of Student objects might each contain a collection of test scores. When the following query is executed, it returns each score that is over 90, along with the last name of the student who received the score.

// Use a compound from to access the inner sequence within each element.

var scoreQuery = from student in students

from score in student.Scores

where score > 90

select new { Last = student.LastName, score };

### [Security](javascript:void(0))

Follow the guidelines in [Secure Coding Guidelines](https://msdn.microsoft.com/en-us/library/8a3x2b7f.aspx).

# References

DoFactory. (n.d.). *Design Pattern Framework.* DoFactory.

four, G. o. (n.d.). *Design Pattern: Elements of Reusable Object-Oriented Software .*

Hyder, S. I., & Waheed, U. (2010). Transaction Patterns. *ResearchGate*, 3.

Point, T. (2016, 1 13). *Tutorials Point Design Pattern*. Retrieved from Tutorials Point: http://www.tutorialspoint.com/design\_pattern/

Shvetes, A. (n.d.). *Design Patterns Explained Simply.* SourceMaking.com.

Wikipedia. (n.d.). *Software Design Description.* Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Software\_design\_description

Wikipedia. (n.d.). *Software requirements specification.* Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Software\_requirements\_specification