Assignment 03

**Design Patterns**

(*Software Design Methodology)*

**COMSATS University Islamabad**

Sahiwal Campus



***Alina Raza***

FA17-BS(CS)-072-B

***Ahmad Shaf***

Software Design & Methodology

November 04, 2019

Role of Design Patterns in Software Engineering



1. **Definition of a Design Pattern**

In software engineering, a design pattern is a general reusable solution to a commonly occurring problem in software design. A design pattern is not a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations. Object-oriented design patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved.

Design patterns reside in the domain of modules and interconnections. At a higher level there are architectural patterns that are larger in scope, usually describing an overall pattern followed by an entire system.

# Types of Design Patterns

There are many types of design patterns:

* **Structural patterns**

It addresses concerns related to the high-level structure of an application being developed

* **Computational patterns**

It addresses concerns related to the identification of key computations. Algorithm strategy patterns address concerns related to high level strategies that describe how to exploit application characteristic on a computation platform**.**

* **Implementation strategy patterns**

It addresses concerns related to the realization of the source code to support how the program itself is organized and the common data structures specific to parallel programming**.**

* **Execution patterns**

It addresses concerns related to the support of the execution of an application, including the strategies in executing streams of tasks and building blocks to support the synchronization between tasks.

* **Object-oriented design patterns**

It typically shows relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved. Patterns that imply mutable state may be unsuited for functional programming languages, some patterns can be rendered unnecessary in languages that have built-in support for solving the problem they are trying to solve, and object-oriented patterns are not necessarily suitable for non-object-oriented languages.

# 2. Role of Design Patterns

Design patterns can speed up the development process by providing tested, proven development paradigms. Effective software design requires considering issues that may not become visible until later in the implementation. Freshly written code can often have hidden subtle issues that take time to be detected, issues that sometimes can cause major problems down the road. Reusing design patterns helps to prevent suchtle issue, and it also improves code readability for coders and architects who are familiar with the patterns.

## Singleton:

This is one of the most dangerous design patterns, when in doubt don't use it. Its main purpose is to guarantee that only one instance of a particular object exists. Possible applications are a printer manager or a database connection manager. It is useful when access to a limited resource needs to be controlled.

## Iterator:

Nowadays, the Iterator pattern is trivial: it allows you to go through a list of objects, starting at the beginning, iterating through the list one element after the other, until reaching the end.

## Template Method Edit:

Also, the Template Method pattern is rather simple: as soon as you define an abstract class, that forces its subclasses to implement some method, you are using a simple form of the Template pattern.

## Command:

To understand the idea behind the Command pattern, consider the following restaurant example: A customer goes to a restaurant and orders some food. The waiter takes the order (command, in this case) and hands it to the cook in the kitchen. In the kitchen the command is executed, and depending on the command different food or drink is being prepared.

## Adapter:

This allows incompatible classes to work together by converting the interface of one class into another. Think of it as a sort of translator: when two heads of states who don’t speak a common language meet, usually an interpreter sits between the two and translates the conversation, thus enabling communication.

## Strategy:

The strategy pattern allows grouping related algorithms under an abstraction, which allows switching out one algorithm or policy for another without modifying the client. Instead of directly implementing a single algorithm, the code receives runtime instructions specifying which of the group of algorithms to run.

## Observer:

This pattern is a one-to-many dependency between objects so that when one object changes state, all its dependents are notified. This is typically done by calling one of their methods.

Explain User Interface Design Patterns

# User Interface (UI) Design Patterns

User interface design patterns are descriptions of *best practices* within user interface design. They are general, reusable solutions to commonly occurring problems. As such, they form the backbone of “technical support.” However, as design patterns can be applied to a wide variety of instances, designers should adapt them to the specific context of use within each design project. User interface design patterns are descriptions of best practices within user interface design. They are general, reusable solutions to commonly occurring problems. As such, they form the backbone of “technical support.” ... Problem: The usability problem faced by the user when using the system.

* **Elements**
* **Problem**: The usability problem faced by the user when using the system.
* **Context of use**: The situation (in terms of the tasks, users, and context of use) giving rise to the usability problem.
* **Principle**: A pattern is usually based on one or more design principles, such as error management or the consistency of user guidance.
* **Solution**: A proven solution to the problem. A solution describes only the *core* of the problem, and the designer has the freedom to implement it in many ways.
* **Why**: How and why the pattern actually works, including an analysis of how it may affect certain attributes of usability.
* **Examples**: Each example shows how the pattern has been successfully applied in a real-life system. This is often accompanied by a screenshot and short description.
* **Implementation**: Some patterns provide implementation details.

**Types of User Interface:**

1. **Command Line Interface:** Command Line Interface provides a command prompt, where the user types the command and feeds to the system. The user needs to remember the syntax of the command and its use.

## CLI Elements:

* **Command Prompt**:

It is text-based notifier that is mostly shows the context in which the user is working. It is generated by the software system.

* **Cursor**:

It is a small horizontal line or a vertical bar of the height of line, to represent position of character while typing. Cursor is mostly found in blinking state. It moves as the user writes or deletes something.

* **Command:**

A command is an executable instruction. It may have one or more parameters. Output on command execution is shown inline on the screen. When output is produced, command prompt is displayed on the next line.

1. **Graphical User Interface:**

Graphical User Interface provides the simple interactive interface to interact with the system. GUI can be a combination of both hardware and software. Using GUI, user interprets the software.

### GUI Elements:

## Window:

An area where contents of application are displayed. Contents in a window can be displayed in the form of icons or lists, if the window represents file structure. It is easier for a user to navigate in the file system in an exploring window.

## Tabs:

If an application allows executing multiple instances of itself, they appear on the screen as separate windows

## Menu:

Menu is an array of standard commands, grouped together and placed at a visible place (usually top) inside the application window. The menu can be programmed to appear or hide on mouse clicks.

## Icon:

An icon is small picture representing an associated application. When these icons are clicked or double clicked, the application window is opened. Icon displays application and programs installed on a system in the form of small pictures.