# Department of Computing

**CS 213: Advance Programming**

**Class: BSCS 6 AB**

# Lab 4: Design Patterns

**Date: Thu 4th October, 2018**

**Time: Thursday (10:00-12:50 & 14:00 – 16:50)**

# Instructor: Abid Rauf

**Lab Engineer: Ayesha Asif**

# 

# Lab 4: Design Patterns

## Introduction

In this lab the students have to implement Strategy Pattern. The Strategy Pattern can be used to encapsulating access control. The use of vendor-supplied software introduces a security problem in which how we can be sure that the supplied software does not include a trap door.

## Objectives

* Focuses on Strategy Pattern and shows how it defines a family of algorithms, encapsulate each one and make them interchangeable.
* A hierarchy that encapsulates: many possible "platforms", and the construction of a suite of "products".

**Description**

There are several goals when a designer wants to design an Object in Object oriented approach for security aspect. To achieve a higher level of security one can, choose an encryption algorithm dynamically in run-time, based on security requirements and computing time constraints. Security requirements describe functional and non-functional requirements that need to be satisfied in order to achieve the security attributes of applications.

The motivation to Strategy Pattern is that many algorithms exist for layout management of Graphical User Interface (GUI) components. Hard-wiring all such algorithms into the classes that require them aren't desirable for several reasons:

(a) Clients that need layout management get more complex if they include the layout management code. That makes clients bigger and harder to maintain, especially if they support multiple layout management algorithms

(b) Different algorithms will be appropriate at different times. We don't want to support multiple layout management algorithms if we don't use them all;

(c) It's difficult to add new algorithms and vary existing ones when layout management is an integral part of a client.

We can avoid these problems by defining classes that encapsulate different layout management algorithms. An algorithm that's encapsulated in this way is called a strategy. The Context is that a class can benefit from different variants for an algorithm. Clients sometimes want to replace standard algorithms with custom versions.

The intent of the Strategy Pattern, as mentioned above, suggests that this pattern is applicable when you have multiple algorithms and you want to treat them as independent objects that can be interchanged dynamically at runtime to achieve **high**[**cohesion**](https://en.wikipedia.org/wiki/Cohesion_%28computer_science%29) and **loose**[**coupling**](https://en.wikipedia.org/wiki/Coupling_%28computer_programming%29) in your application.

**Lab Task**

In enterprise applications, you will often have objects that use multiple algorithms to implement some business requirements. A common example is a number sorting class that supports multiple sorting algorithms, such as bubble sort, merge sort, and quick sort. Similarly, a file compression class can support different compression algorithm, such as ZIP, GZIP, LZ4, or even a custom compression algorithm. Another example can be a data encryption class that encrypts data using different encryption algorithms, such as AES, TripleDES, and Blowfish.

You need to implement for this lab the **Strategy Pattern for data encryption** in this way:

* **Strategy (EncryptionStrategy)**: Is an interface common to all supported algorithm-specific classes.
* **ConcreteStrategy** (**AesEncryptionStrategy, BlowfishEncryptionStrategy** and **TripleDESEncryptionStrategy**): Implements the algorithm using the **Strategy** interface.
* **Context** (**Encryptor**): Provides the interface to client for encrypting data. The **Context** maintains a reference to a **Strategy** object and is instantiated and initialized by clients with a **ConcreteStrategy** object.

**EncryptionStrategy.java**

**public interface EncryptionStrategy {**

**void encryptData(String plainText);**

**}**

**AesEncryptionStrategy.java**

**public class AesEncryptionStrategy implements EncryptionStrategy{**

**\\ Your own code**

}

**BlowfishEncryptionStrategy.java**

**public class BlowfishEncryptionStrategy implements EncryptionStrategy{**

**\\ Your own code**

}

**TripleDESEncryptionStrategy.java**

**public class TripleDESEncryptionStrategy implements EncryptionStrategy{**

**\\ Your own code**

**}**

**Encryptor.java**

**public class Encryptor {**

**\\ Your own code**

**}**

## Also write an EncryptorTest.java

**public class EncryptorTest {**

**\\ Your own code**

}

## Deliverables

* + Each submission is individual based with the Source Code in one Zip file strictly followed Strategy Pattern

## Grade Criteria



This lab will be graded on the following rubric: