

SE322: Software Design and Architecture

Activity07: Creational Design Patterns

Objective: In this activity, the students will learn how to analyze an existing software architecture or design and choose appropriate creational design pattern.

Expected Outcome: After completing this exercise, the students will be able to:

1. Choose appropriate creational design pattern.
2. Justify their selection.

Course Learning Outcome:

1. CLO 1: Recognize the principles, strategies, and process of software design and architecture.
2. CLO 2: Analyze an existing software architecture or design.
3. CLO 3: Differentiate how various architectural styles and design patterns enhance and degrade a system's functional-and non-functional properties.
4. CLO 4: Justify architecture and/or design given a collection of requirements.

Tool needed:

Visio, Lucidchart or StarUml

Question1: Computer component

You are writing a diagnostic utility to test the various components of a computer system to ensure that they will work with your big, expensive enterprise application. You want to write one class for each computer component you diagnose: **CPUTest**, **MemoryTest**, and **DiskTest**. However, your utility needs to be cross-platform, and each class contains some platform-specific code. So, for each type of test, you will have several implementations, one for each platform: **PCMemoryTest**, **MacMemoryTest**, **LinuxMemoryTest**, etc. You'd like your diagnostic utility to create and use the right test classes for the platform its running on, without having to change a lot of code each time you add a new platform. You recall that there is a design pattern to solve this problem.

1. What design pattern would you use?

I would use the **Abstract Factory** design pattern.

Justification: The utility needs to create a **family of related objects** for each platform: a CPU test, a memory test and a disk test. The pattern defines an abstract `TestFactory` with

`createCPUTest()`, `createMemoryTest()` and `createDiskTest()`. Each platform then gets its own concrete factory: `PCTestFactory`, `MacTestFactory` and `LinuxTestFactory`.

The utility picks the right factory once, based on the platform it's running on. After that it works only through the abstract interfaces (`CPUTest`, `MemoryTest`, `DiskTest`), so it always gets matching tests for that platform and never mixes them. Adding a new platform only means adding a new factory and its test classes, with no change to the existing code (Open-Closed Principle).

2. Draw a UML class diagram showing the relationships among participants in your design pattern.

