# Lab 13\_1

The Bridge pattern appears to have much in common with the Strategy pattern. Note the similarity between their GOF intent statements: Bridge Intent: “Decouple an abstraction from its implementation so that the two can vary independently.” Strategy Intent: “Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.” Also note the similarity between their UML diagrams.

a. Specify the deciding factors that determine which of these two patterns should be applied in a particular situation, emphasizing the reasons why the Bridge pattern is considered to be structural while the Strategy pattern is considered to be behavioral.

b. Supply examples to demonstrate this distinction. Submit a Word document with your answers.

|  |  |  |
| --- | --- | --- |
| Pattern | Bridge | Strategy |
| **Emphasis** | Bridge Pattern is a structural pattern which emphasizes **“How do we build a software component”**. | Strategy Pattern is a behavioral (dynamic) pattern which emphasizes **“How you want to run a behavior in software”**. |
| **Focus** | Bridge patterns suggest ways in which objects are composed or associated or inherited to forms larger objects i.e. they focus on object composition. | Strategy pattern deal with the algorithm or business logic (and not on the object creation itself) i.e. they focus on the collaboration between objects. |
| **Process** | We can split the hierarchy of interface and class join him with an abstract reference | We have more ways for doing an operation; with strategy you can choice the algorithm at run-time and you can modify a single Strategy without a lot of side-effects at compile-time |

A closer look at the implementation of the two patterns will reveal that in the bridge pattern one creates the concrete implementation of the object and then the call.

**Bridge Pattern:**

// Set implementation and call i.e. Returns (creates) the concrete implementation of the object,

// subsequently operation is called on the concrete implementation

// This is an abstract class that will represent numerous

// ways to work with each concrete implementor

**public** **abstract** **class** OurAbstractClass{

// A reference to a generic device using composition

**private** Implementor implementor;

:

:

**public** **void** operation() {

implementor.operation ();

}

}

**Strategy Pattern:**

Whereas in the case of the strategy pattern, one will not use the concrete implementation of the algorithm directly, instead he will create the context in which the strategy should execute,

// Set the context with a strategy i.e. Sets the concrete strategy into the context, concrete

// implementation of the class not directly available as a data object (only the algorithm is available).

Context context = new Context (new ConcreteStrategyA());

context.contextInterface();

// Strategy can be reused instead of creating a new instance every time it is used.

// Sort example

MergeSort mergeSort = **new** MergeSort();

QuickSort quickSort = **new** QuickSort();

context = **new** Context (mergeSort);

context.sort();

context = **new** Context(quickSort);

context.sort();

context = **new** Context (mergeSort);

context.sort();

Here, switching between the strategies at runtime.