**Pencil and Paper 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.

1. 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.
2. Supply examples to demonstrate this distinction. Submit a Word document with your answers.

**Answer:**

**1.**

The **Bridge Pattern** is a ***structural*** pattern which emphasises ‘How do we build a software component’. But the **Strategy Pattern** is a ***behavioral***  (dynamic) pattern which emphasises ‘How you want to run a behaviour in software’.

The syntax is similar but the goal are differents:

* Strategy: you 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;
* Bridge: you can split the hierarchy of interface and class join him with an abstract reference

**2.**

**Strategy:**

**Context tied to the Strategy:** The context Class (possibly Abstract but not really an interface! as u wish to encapsulate out a specific behavior and not the entire implementation) would know/contain the strategy interface reference and the **implementation** to invoke the strategy behavior on it.

Intent is ability to swap behavior at runtime

class Context {

IStrategy strategyReference;

void strategicBehaviour() {

strategyReference.behave();

}

}

**Bridge**

Abstraction not tied to the Implementation: The abstraction interface (or abstract class with most of the behavior abstract) would not know/contain the implementation interface reference

Intent is to completely decouple the Abstraction from the Implementation

interface IAbstraction {

void behaviour1();

.....

}

interface IImplementation {

void behave1();

void behave2();

.....

}

class ConcreteAbstraction1 implements IAbstraction {

IImplementation implmentReference;

ConcreteAbstraction1() {

implmentReference = new ImplementationA() // Some implementation

}

void behaviour1() {

implmentReference.behave1();

}

.............

}

class ConcreteAbstraction2 implements IAbstraction {

IImplementation implmentReference;

ConcreteAbstraction1() {

implmentReference = new ImplementationB() //Some Other implementation

}

void behaviour1() {

implmentReference.behave2();

}

.............

}