Bridge vs Strategy Design Patterns

# **Explanation**

*The simple answer is “They are similar but different”. The implementations are similar but the intentions are different. To give an analogy, a city bus and school bus are both similar vehicles, but they are used for different purposes. One is used to transport people between various parts of the city as a commuter service. The other is used for transporting kids to schools.*

*The Bridge pattern is a structural pattern (HOW DO YOU BUILD A SOFTWARE COMPONENT?). The Strategy pattern is a dynamic pattern (HOW DO YOU WANT RUN A BEHAVIOUR IN SOFTWARE?).*

Let’s start with the definitions of each of these patterns

**Strategy Pattern:**

* Define a family of algorithms, encapsulate each one, and make them interchangeable.
* Strategy lets the algorithm vary independently from the clients that use it.
* Capture the abstraction in an interface, bury implementation details in derived classes
* A bind once behavioral pattern.

**Bridge Pattern:**

* Decouple an abstraction from its implementation so that the two can vary independently.
* Publish interface in an inheritance hierarchy, and bury implementation in its own inheritance hierarchy.
* A structural pattern.

# **Code Samples**

**Strategy**

**Context tied to the Strategy:** The context Class 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();

}

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

}