

Bridge Pattern

Sometimes an abstraction should have different implementations; consider an object that handles persistence of objects over different platforms using either relational databases or file system structures (files and folders). A simple implementation might choose to extend the object itself to implement the functionality for both file system and RDBMS. However this implementation would create a problem; Inheritance binds an implementation to the abstraction and thus it would be difficult to modify, extend, and reuse abstraction and implementation independently.

The intent of this pattern is to decouple abstraction from implementation so that the two can vary independently.

Bridge Pattern

Motivation

Sometimes an abstraction should have different implementations; consider an object that handles persistence of objects over different platforms using either relational databases or file system structures (files and folders). A simple implementation might choose to extend the object itself to implement the functionality for both file system and RDBMS. However this implementation would create a problem; Inheritance binds an implementation to the abstraction and thus it would be difficult to modify, extend, and reuse abstraction and implementation independently.

Intent

- The intent of this pattern is to decouple abstraction from implementation so that the two can vary independently.

Implementation

The figure below shows a UML class diagram for the Bridge Pattern:

The participants classes in the bridge pattern are:

- Abstraction - Abstraction defines abstraction interface.
- AbstractionImpl - Implements the abstraction interface using a reference to an object of type Implementor.
- Implementor - Implementor defines the interface for implementation classes. This interface does not need to correspond directly to abstraction interface and can be very different. Abstraction imp provides an implementation in terms of operations provided by Implementor interface.
- ConcreteImplementor1, ConcreteImplementor2 - Implements the Implementor interface.

Description

An Abstraction can be implemented by an abstraction implementation, and this implementation does not depend on any concrete implementers of the Implementor interface. Extending the abstraction does not affect the Implementor. Also extending the Implementor has no effect on the Abstraction.

Applicability & Examples

The bridge pattern applies when there is a need to avoid permanent binding between an abstraction and an implementation and when the abstraction and implementation need to vary independently. Using the bridge pattern would leave the client code unchanged with no need to recompile the code.

Example - Object Persistence API Example

As discussed previously a persistence API can have many implementations depending on the presence or absence of a relational database, a file system, as well as on the underlying operating system.

Source: [Click here to see java source code](#)

Specific problems and implementation

Graphical User Interface Frameworks

Graphical User Interface Frameworks use the bridge pattern to separate abstractions from platform specific implementation. For example GUI frameworks separate a Window abstraction from a Window implementation for Linux or Mac OS using the bridge pattern.

Related Patterns

- Abstract Factory Pattern - An Abstract Factory pattern can be used create and configure a particular Bridge, for example a factory can choose the suitable concrete implementor at runtime.

Consequences

Known Uses:

- Decoupling interface and implementation. An implementation is not bound permanently to an interface. The implementation of an abstraction can be configured and even switched at run-time.

- Abstraction and Implementor hierarchies can be extended independently.

Known Uses:

- GUI frameworks as discussed previously.

- Persistence Frameworks as discussed previously.

