**Bridge**

**Definition**

Decouple an abstraction or interface from its implementation so that the two can vary independently.

**Where to use & benefits**

* Want to separate abstraction and implementation permanently
* Share an implementation among multiple objects
* Want to improve extensibility
* Hide implementation details from clients
* Related patterns include
  + [Abstract Factory](http://www.javacamp.org/designPattern/abstractfactory.html), which can be used to create and configure a particular bridge.
  + [Adapter](http://www.javacamp.org/designPattern/adapter.html), which makes unrelated classes work together, whereas a bridge makes a clear-cut between abstraction and implementation.

**Examples**

If you have a question database, you may want to develop a program to display it based on the user selection. The following is a simple example to show how to use a Bridge pattern to decouple the relationship among the objects.

import java.util.\*;

//abstraction

interface Question {

public void nextQuestion();

public void priorQuestion();

public void newQuestion(String q);

public void deleteQuestion(String q);

public void displayQuestion();

public void displayAllQuestions();

}

//implementation

class QuestionManager {

protected Question questDB; //instantiate it later

public String catalog;

public QuestionManager(String catalog) {

this.catalog = catalog;

}

public void next() {

questDB.nextQuestion();

}

public void prior() {

questDB.priorQuestion();

}

public void newOne(String quest) {

questDB.newQuestion(quest);

}

public void delete(String quest) {

questDB.deleteQuestion(quest);

}

public void display() {

questDB.displayQuestion();

}

public void displayAll() {

System.out.println("Question Catalog: " + catalog);

questDB.displayAllQuestions();

}

}

//further implementation

class QuestionFormat extends QuestionManager {

public QuestionFormat(String catalog){

super(catalog);

}

public void displayAll() {

System.out.println("\n~~~~~~~~~~~~~~~~~~~~~~~~");

super.displayAll();

System.out.println("~~~~~~~~~~~~~~~~~~~~~~~~");

}

}

//decoupled implementation

class JavaQuestions implements Question {

private List<String> questions = new ArrayList<String>();

private int current = 0;

public JavaQuestions() {

//load from a database and fill in the container

questions.add("What is Java? ");

questions.add("What is an interface? ");

questions.add("What is cross-platform? ");

questions.add("What is UFT-8? ");

questions.add("What is abstract? ");

questions.add("What is Thread? ");

questions.add("What is multi-threading? ");

}

public void nextQuestion() {

if( current <= questions.size() - 1 )

current++;

}

public void priorQuestion() {

if( current > 0 )

current--;

}

public void newQuestion(String quest) {

questions.add(quest);

}

public void deleteQuestion(String quest) {

questions.remove(quest);

}

public void displayQuestion() {

System.out.println( questions.get(current) );

}

public void displayAllQuestions() {

for (String quest : questions) {

System.out.println(quest);

}

}

}

class TestBridge {

public static void main(String[] args) {

QuestionFormat questions = new QuestionFormat("Java Language");

questions.questDB = new JavaQuestions();//can be hooked up with other question class

//questions.questDB = new CsharpQuestions();

//questions.questDB = new CplusplusQuestions();

questions.display();

questions.next();

questions.newOne("What is object? ");

questions.newOne("What is reference type?");

questions.displayAll();

}

}

//need jdk1.5 to compile

| C:\ Command Prompt  C:\> javac TestBridge.java  C:\> java TestBridge  What is Java?  ~~~~~~~~~~~~~~~~~~~~~~~~  Question Catalog: Java Language  What is Java?  What is an interface?  What is cross-platform?  What is UFT-8?  What is abstract?  What is Thread?  What is multi-threading?  What is object?  What is reference type?  ~~~~~~~~~~~~~~~~~~~~~~~~  C:\> |
| --- |

Note that the JavaQuestion class can be launched independently and work as its own system. Here we just show you how to use Bridge pattern to decouple the interface from its implementation.