Bridge Pattern

aka handle/body

Dan Sullivan

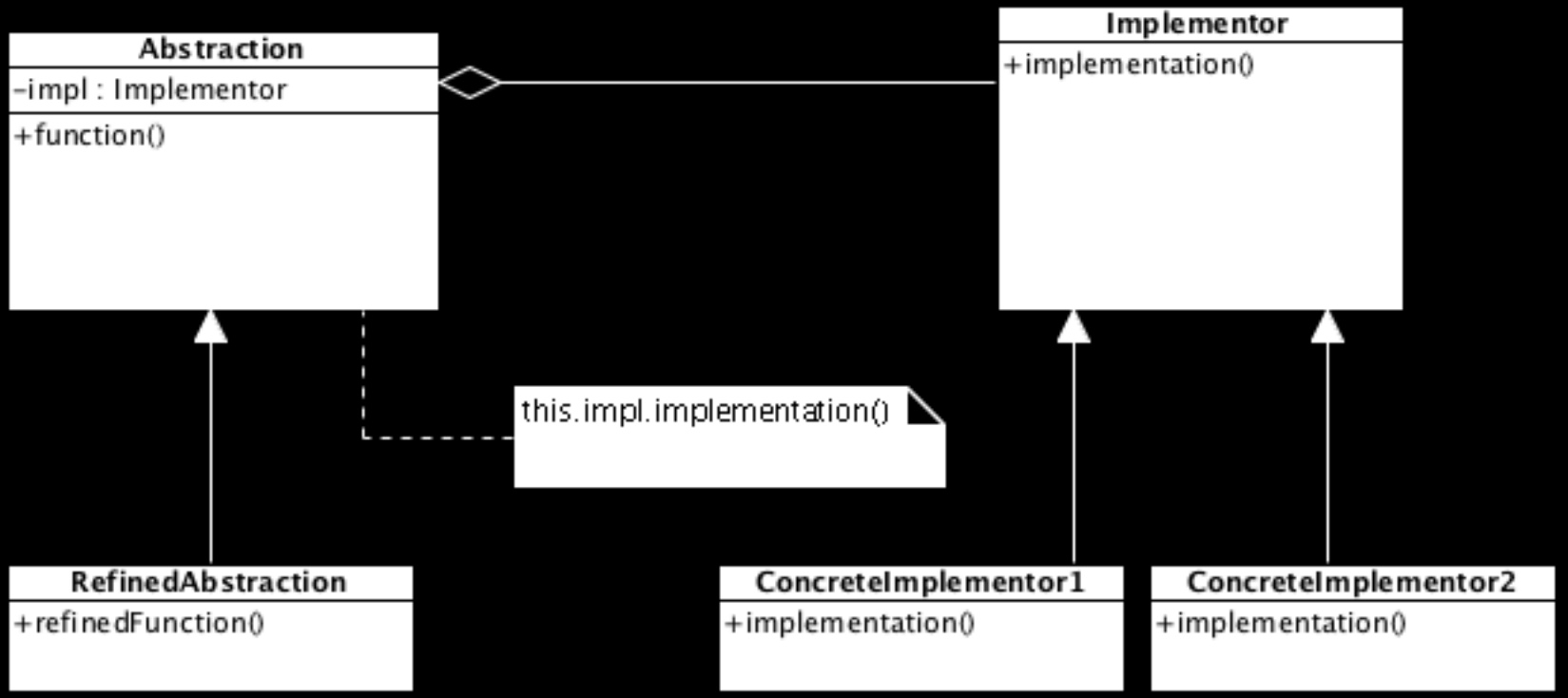
Pattern Description

- Meant to “decouple an abstraction from its implementation so that the two can vary independently”.
- Separates abstraction and implementation into separate class hierarchies.
- The bridge is the relationship between the abstraction and the implementation hierarchies.

More about the pattern

- It's a structural object pattern.
- Often confused with the adapter pattern, which uses multiple inheritance.
- Don't use it when you have a single implementor.

UML Representation



Source http://en.wikipedia.org/wiki/Bridge_pattern

Applications

- When an abstraction has varied implementations.
- When Implementations and abstractions both require distinct subclass extensibility.
- Where you don’t want changes in implementation to require recompilation of abstraction classes.

Abstractions

```
/** "Abstraction" */
interface Time {
   public void tell();                                  // low-level
}

/** "Refined Abstraction" */
class normalTime implements Time {
   private int hr, min, seconds;
   private TimeAPI drawingAPI;
   public normalTime(int hr, int min, int seconds, TimeAPI drawingAPI)
{
        this.hr = hr;  this.min = min;  this.seconds = seconds;
        this.drawingAPI = drawingAPI;
   }

   // low-level i.e. Implementation specific
   public void tell() {
         drawingAPI.tellTime(hr, min, seconds);
   }
}
```

Implementors

```
/** "Implementor" */
interface TimeAPI {
    public void tellTime(int hr, int min, int seconds);
}

/** "ConcreteImplementor" 1/3 */
class militaryTime implements TimeAPI {
   public void tellTime(int hr, int min, int seconds) {
            System.out.format("%02d%02d\n", hr,min);
   }
}

/** "ConcreteImplementor" 2/3 */
class civilianTime implements TimeAPI {
   public void tellTime(int hr, int min, int seconds) {
           String ampm;
           if (hr < 12) {
                   ampm = "AM";
           } else {
                   ampm = "PM";
           }
           System.out.printf("%d:2%d %s\n", hr, min, ampm);
   }
}
```

Client

```
public class Bridge {
/**Client Class**/
public static void main(String[] args) {
     Time[] timeCollection = new Time[] {
          new normalTime(1, 42, 1, new militaryTime()),
          new normalTime(10, 4, 33, new civilianTime()),
          new timeWithZone(22, 1, 0, 7, new civilianTimeWithSeconds()),
          new timeWithZone(13, 4, 1, 7, new militaryTime()),
     };
     for (Time timeItem : timeCollection) {
          timeItem.tell();
     }
}
}
```

Choosing The Implementor Object

- By passing parameters to the abstractions
- Assign a default implementor and change it later
- Delegate the decision to another object such as a factory class.

Using a bridge gets you..

- The ability to select implementation used at runtime.
- Compilation perks.
- Encourages programmatic layering, high-level components only need to know about the abstraction and implementor
- Implementation details can be masked from clients