## Observer

### Questions

**What is it?**
many objects need to be notified of changes in the state of one object

**Where have I seen it before?**
RSS feeds or any pub/sub system you might have used/coded

**Ok, this is cool. What do I need to implement it?**
1. A Subject Abstract Class and an Observer Abstract Class
2. Concrete subject and observer class that implement above pattern.

**Example**
A referee (concrete subject) notifies all the players and commentators (concrete observers) about changes in the state of a Soccer match. Each player must be notifiable.

### Memento

**Questions**

**What problem does it solve?**
you want to save the state of an object so that you can restore it later.

**Where have I seen it before?**
Git or any version control system for that matter. A memento makes rollbacks possible.

**Ok, this is cool. What do I need to implement it?**
1. An originator class (class that has a state that needs to be remembered)
2. A caretaker class (class that wants to modify the state of the originator)
3. A memento class that holds originator information that can't be modified by any other class. It is merely a container.

**Example**
A programmer (caretaker) asks for a copy (memento) of the code (originator) he/she is modifying. Later he/she decides he doesn't like the new state of the code so he restores it with the copy it still has.

## Interpreter

### Questions

**What problem does it solve?**
A language must be parsed and interpreted.

**Where have I seen it before?**
Parsers

**Ok, this is cool. What do I need to implement it?**
1. A Context class that contains the input.
2. An AbstractExpression class, a composite object of terminals and non-terminals.
3. The client passes the context to the abstract expression, which calls the interpret() function on its children.

**Example**
A roman numeral (context) is converted into decimal notation by the parser (Abstract Expression).

\[
\text{Derivation: LIV} \Rightarrow 50 + IV \Rightarrow 50 + (-1 + 5) \Rightarrow 50 + 4 \Rightarrow 54
\]
### Design Patterns: Observer, Interpreter, Memento Cheat Sheet

**Advantages**

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Advantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observer</td>
<td>minimal coupling; easy addition and removal of observers</td>
</tr>
<tr>
<td>Memento</td>
<td>it is an encapsulated copy so it avoids exposing its info; the storage burden is on the caretaker, not on originator</td>
</tr>
<tr>
<td>Interpreter</td>
<td>easy to change/extend/implement/evaluate a language</td>
</tr>
</tbody>
</table>

**Disadvantages**

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Disadvantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observer</td>
<td>Possible memory leak; Objects might need to work hard to deduce what changed in the subject.</td>
</tr>
<tr>
<td>Memento</td>
<td>Copy operation to a memento can be costly for the originator; Caretaker might have large storage costs.</td>
</tr>
<tr>
<td>Interpreter</td>
<td>Complex grammars are hard to maintain and debug.</td>
</tr>
</tbody>
</table>

Published 2nd February, 2015.
Last updated 3rd February, 2015.
Page 2 of 2.