Design patterns - Introduction

Robert Lukoťka

November 25, 2015

What is a design pattern?

- General **reusable solution** to a commonly occurring problem within a given context **in software design**
- We already saw several examples: M-C-V, **strategy** , **decorator**, information expert;
- Design antipatterns - common response to a recurring problem that is usually ineffective and risks being highly counterproductive: circular dependency, poltergeist

Additional benefits of using DP

- Easier communication
- Allow to stay at design level without discussing details
- Improved speed of development

Do not overuse

Types of design paterns

- **creational patterns**: factory method, abstract factory
- **structural patterns**: decorator
- **behavioral patterns**: strategy, iterator, observer
- concurrency patterns: thread-specific storage
- architectural patterns: M-C-V
- ...

Strategy design pattern

- Enables an algorithm's behavior to be selected at runtime.
- Defines a family of algorithms,
- Encapsulates each algorithm, and
- Makes the algorithms interchangeable within that family.
- Class diagrams
- We had several examples during the course

Observer pattern

- Object wants to inform all it's observers.
- Class diagram
- Class diagram 2
- One of the most used DP in JDK.
- Potential memory leaks and performance decrease in garbage collecting languages can be solved by using weak references (inactive objects might not be collected because if a strong link exists)
- Implemented in Java: Observable class and Observer interface (but there are several drawbacks in using it: Observable is a class; setchanged() is protected)

Decorator pattern

- Allows behavior to be added to an individual object, either statically or dynamically, without affecting the behavior of other objects from the same class.
- Class diagram
- Can add small hard to understand classes
- Problems when code depends on specific type
- Example: Java I/O

Factory patterns

- Factory pattern uses factory methods to create objects without having to specify the exact class.
- Button okButton=new RectangularButton(); -coding to implementation not to interface
- With decorator DP:
  Button okButton=new RectangularButton(new BasicButton);

Factory patterns

- It might look like this in the actual code

  ```
  Button okButton;
  if (settings.shape == ButtonShape.square)
   okButton=new SquareButton();
   elseif (settings.shape == ButtonShape.rectangle)
    okButton=new RectangularButton();
    elseif (settings.shape == ButtonShape.triangle)
     okButton=new TriangularButton();
  ```

- You have to do changes here whenever new type is added.
- Solution:

Factory patterns

- It might look like this in the actual code

  ```
  Button okButton;
  if (settings.shape == ButtonShape.square)
   okButton=new SquareButton();
   elseif (settings.shape == ButtonShape.rectangle)
    okButton=new RectangularButton();
    elseif (settings.shape == ButtonShape.triangle)
     okButton=new TriangularButton();
  ```

- You have to do changes here whenever new type is added.
- Solution: Encapsulate what varies - interface with Factory method
- More flexibility:

Factory patterns

- It might look like this in the actual code

  ```
  Button okButton;
  if (settings.shape == ButtonShape.square)
   okButton=new SquareButton();
   elseif (settings.shape == ButtonShape.rectangle)
    okButton=new RectangularButton();
    elseif (settings.shape == ButtonShape.triangle)
     okButton=new TriangularButton();
  ```

- You have to do changes here whenever new type is added.
- Solution: Encapsulate what varies - interface with Factory method
- More flexibility: Abstract factory

Abstract factory example

- interface UICreator has abstract methods buttonCreate that creates an Button (Button is an interface, the buttonCreate can have arguments to define e.g. color), + some other abstract methods like checkBoxCreate;
- if some other functionality is associated with the type we might use abstract object instead of an interface; e.g. createMyFancyForm
- class GUICreator implements buttonCreator ... creates a GraphicalButton
- class TUICreator implements buttonCreator ... creates a TextButton
- at initialization of the program we choose whether we use GUICreator or TUIcreator, no need for changes in the remaining code.
- myUIcreator.createMyFancyForm();

Dependency inversion principle

Wikipedia
Applied to everything:

- No variable should hold a reference to concrete class
- No class should derive from concrete class
- No method should override an implemented method of any of its base classes

So you have to choose wisely.

Singleton DP

- Creates objects that have only one instance
- Private Constructor
- Static method that controls that only one object is created
- Careful in in multi-threaded applications

Command DP

- Used to encapsulate all information needed to perform an action or trigger an event at a later time
- Class diagram
- queuing
- logging
- scheduling
- Decorators: MacroComamand, LoggedCommand, SheduledCommand

Null object / Null DP

- Removes the responsibility of handling null from the client
- e.g. Null command

Adapter DP

- Object adapter: Object that uses method of another object to implement an interface
- Class adapter: Uses inheritance instead of composition
- Class apter can save code for methods that do not need to adapt, and faster.
- Object adapter is more flexible, e.g. can adapt subclasses
- Java example: old interface Enumeration, new interface Iterator

Facade pattern DP

- Adapter with intention of simplifying the interface
- Facade does not encapsulates
- Multiple facades possible
- Facades can be used to decouple client from subsystem

Principle of least knowledge

- wiki
- station.getThermometer().getTemperature()
- You may want to add station.getTemperature()
- If applied too much may lead to a lot of wrapper classes and methods.

Template method DP

- Defines a skeleton of an algorithm.
- Subclasses can redefine steps, but not the algorithm.
- Some methods called may the function may be abstract, some null (hook methods), and some implemented.
- A different example: general sort

Hollywood principle

- wiki
- Low-level components participate in tasks made by high-level components
- Low-level components do not call high-level components.

Other DP

- Iterator
- Composite pattern
- State pattern
- Proxy (remote proxy, virtual proxy, protection proxy, ...)
-