Introduction to Design Patterns

Relevant Links

- Osmani's book up to summary table[1]
- Design Patterns (Gang of Four)[2] optional but great read
- Head First Design Patterns[3] optional, also great book on design patterns
- OO Design[4]
- SourceMaking Design Patterns[5]
- Hillside group[6]

Notes

A **Design Pattern** describe a problem that appears frequently in the open, followed by a solution to that problem, that describes the structure of the objects needed to implement the pattern. In general a pattern consists of various parts:

pattern name A short memorable name that is used to describe the problem and solution in one word. Once you are familiar with the patterns, these names become the language we can use to communicate an appication's design at a higher level.

problem A description of the problem that this pattern is meant to address.

solution What need to be implemented to carry out the pattern. This requires a description of the structure of the solution, and how the different parts of it come together. It may also include implementation details.

consequences How the choice to implement the pattern influences the future evolution of the application. Every decision involves tradeoffs.

Patterns are typically classified in one of three groupings according to their *purpose*:

creational These patterns concern themselves with object creation. Some typical patterns in this category include the *Factory Method*, the *Abstract Factory*, *Builder*, *Prototype* and *Singleton*.

structural These patterns describe ways to assemble objects. This category includes *Adapter*, *Bridge*, *Composite*, *Decorator*, *Facade*, *Flyweight* and *Proxy*.

behavioral These patterns describe how objects can cooperate to perform a task. These include *Observer*, *Visitor*, *Iterator*, *Template Method*, *Chain of Responsibility*, *Command*, *Iterator*, *Mediator*, *Memento*, *State*, *Strategy*.

[1] http://addyosmani.com/resources/essentialjsdesignpatterns/book/#summarytabledesignpatterns
[2] http://www.amazon.com/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612
[3] http://shop.oreilly.com/product/9780596007126.do
[4] http://www.oodesign.com/
[5] http://sourcemaking.com/design_patterns
[6] http://www.hillside.net/

Design patterns are closely related with a set of programming principles that have evolved over time. Here is just a partial list of these principles:

- **Program to interfaces (types), not to implementations**. Your code should rely on what objects can do, not on how they were implemented.
- **Document your interfaces**. In a dynamically typed language like Javascript, there is no explicit notion of an "interface": When describing a class you have to describe what methods it offers, what inputs they take and what outputs they produce.
- **Aim for a single point of entry**. When creating objects of a class, there should be at the end of the day a single constructor that everything else gets funnelled through. Makes maintenance easier down the line.
- **Classes should only try to do one thing**. This is a version of the "keep it simple" principle. Classes should do one thing, and they should do it well.
- **Favor composition over inheritance**. Inheritance makes your subclass depend on the implementation of the superclass, something that is best avoided.
- **Aim for loosely coupled designs**. The less your components know about each other, the better you can adapt to future changes in your application.
- **Encapsulate what varies**. If there is part of your design that might change in the future, it should be encapsulated and the rest of your application should be protected from knowing too much about it.

Most design patterns employ a number of these principles.

Pattern descriptions usually consist of the following parts:

name Make it part of your vocabulary!
classification What category the pattern is in.
intent What does the pattern do? What is its rationale? What issue does it address?
aka Some patterns are known by other names.
motivation A scenario illustrating the design problem and how the structures in the pattern address the problem.
applicability In what situations would we apply this pattern?
structure Graphical representation of the classes and objects involved in the pattern.
participants The classes and objects participating in the pattern.
collaborations How the participants work together to carry out their responsibilities.
consequences How does the pattern support its objectives? What are the trade-offs and results of using the pattern?
implementation Pitfalls, hints, techniques for implementing the pattern.
sample code Code fragments illustracting how the pattern might be implemented.
known uses Examples of patterns found in real systems.
related patterns What other patterns are closely related?