

# Design Patterns

## Observer

Let's consider an `Employee` object that has a `salary` property. We'd like to be able to change their salary and keep the payroll system informed about any modifications. The simplest way to achieve this is passing a reference to payroll and inform it whenever we modify the employee `salary`:

```
class Employee
  attr_reader :name, :title
  attr_reader :salary

  def initialize( name, title, salary, payroll)
    @name = name
    @title = title
    @salary = salary
    @payroll = payroll
  end

  def salary=(new_salary)
    @salary = new_salary
    @payroll.update(self)
  end
end
```

## Decorator

Here is an implementation of an object that simply writes a text line to a file.

At some point, we might need to print the line number before each one, or a timestamp or a checksum. We could achieve this by adding new methods to the class that performs exactly what we want, or by creating a new subclass for each use case. However, none of these solutions is optimal.

```
class SimpleWriter
  def initialize(path)
    @file = File.open(path, 'w')
  end

  def write_line(line)
    @file.print(line)
    @file.print("\n")
  end

  def close
    @file.close
  end
end
```

This worksheet is based on <https://github.com/davidgf/design-patterns-in-ruby>.

# Design Patterns

## Factory

Imagine that you are asked to build a simulation of life in a pond that has plenty of ducks. But how would we model our `Pond` if we wanted to have frogs instead of ducks? In the implementation above, we are specifying in the `Pond`'s initializer that it should be filled up with ducks.

```
class Pond
  def initialize(number_ducks)
    @ducks = number_ducks.times.inject([]) do |ducks, i|
      ducks << Duck.new("Duck#{i}")
    end
  end

  def simulate_one_day
    @ducks.each {|duck| duck.speak}
    @ducks.each {|duck| duck.eat}
    @ducks.each {|duck| duck.sleep}
  end
end
```

## Singleton

Let's consider the implementation of a logger class. Logging is a feature used across the whole application, so it makes sense that there should only be a single instance of the logger.

```
class SimpleLogger
  attr_accessor :level
  ERROR, WARNING, INFO = 1, 2, 3

  def initialize
    @log = File.open("log.txt", "w")
    @level = WARNING
  end

  def error(msg)
    ..
  end

  def warning(msg)
    ..
  end

  def info(msg)
    ..
  end
end
```