

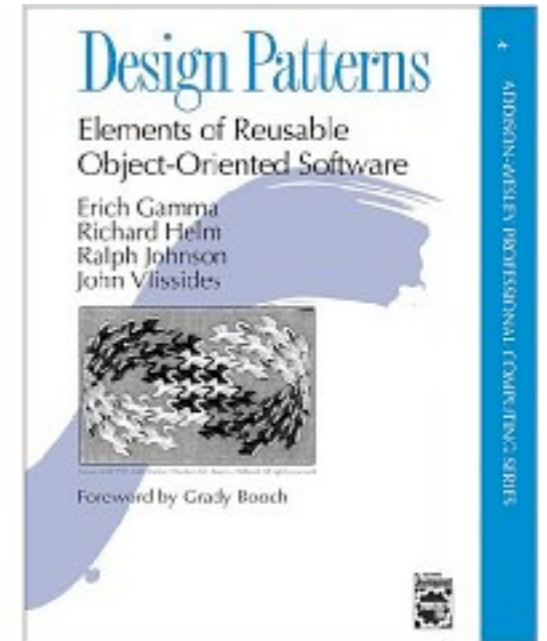


# Factory Method Pattern

Brian DeShong  
January 8, 2009

# What is it?

- A creational design pattern
  - Creates objects
- Used to create objects of the same type
  - Allows developer to rely on having a consistent interface for any objects returned
- Your code never specifies the class name to instantiate
  - That's handled by the factory method!
  - Determination of which class to instantiate happens at runtime



# Your PHP code

- Your factory method
  - It's always a static method
  - Under normal use, it returns an object
  - Throw an exception if the class can't be determined
- Your classes
  - You should always have an interface or an abstract class
  - Concrete classes should implement the interface
  - This is what helps ensure a consistent usage by the developer!

# Warning signs

```
// Let's just get it from the query string.  
$class = $_GET['class'];  
  
switch ($class) {  
    case 'foo':  
        $object = new Foo();  
        break;  
    case 'bar':  
        $object = new Bar();  
        break;  
    default:  
        exit("uh oh, that's a bogus class type!");  
}
```

# Factor out the commonality!

```
// Get user's favorite sport  
$sport = User::getFavoriteSport('bdeshong'); // "baseball"
```

```
switch ($sport) {  
    case 'baseball':  
        $object = new Sport_Baseball();  
        break;  
    case 'hockey':  
        $object = new Sport_Hockey();  
        break;  
    default:  
        exit("you wanna' play what?!");  
}
```

**Examples, of course!**

# Interface for all cars

```
interface Car_Interface
{
    public function accelerate();
    public function brake();
    public function getTopSpeed();
}
```

# Fast car: Ferrari

```
class Car_Ferrari implements Car_Interface
{
    public function accelerate()
    {
        return "is that all you've got?";
    }

    public function brake()
    {
        return "wimp!";
    }

    public function getTopSpeed()
    {
        return 250;
    }
}
```

# Slow car: Yugo

```
class Car_Yugo implements Car_Interface
{
    public function accelerate()
    {
        return "please, stop...I can't take it!";
    }

    public function brake()
    {
        return "ah, what a relief";
    }

    public function getTopSpeed()
    {
        return 35;
    }
}
```

# Car factory: by speed class

```
class Car_Factory
{
    const TYPE_FAST = 1;
    const TYPE_SLOW = 2;

    public static function createBySpeedClass($type)
    {
        switch ($type) {
            case self::TYPE_FAST:
                return new Car_Ferrari();
            case self::TYPE_SLOW:
                return new Car_Yugo();
            default:
                throw new Car_Exception(
                    "unrecognized car type; " .
                    "type = {$type}");
        }
    }
}
```

# Car factory: by country

```
class Car_Factory
{
    const COUNTRY_JAPAN = 'jp';
    const COUNTRY_USA = 'us';

    public static function createByCountry($country)
    {
        switch ($country) {
            case self::COUNTRY_USA:
                return new Car_Ford();
            case self::COUNTRY_JAPAN:
                return new Car_Honda();
            default:
                throw new Car_Exception(
                    "unrecognized country; " .
                    "country = {$country}");
        }
    }
}
```

# Example usage

```
63 try {
64     $car = Car_Factory::createBySpeedClass(Car_Factory::TYPE_FAST);
65 } catch (Car_Exception $e) {
66     echo $e->getMessage() . "\n";
67     exit();
68 }
69
70 echo "You're driving a " . get_class($car) . "\n";
71 echo "Car's max speed is: " . $car->getTopSpeed() . "\n";
72 $timeToBrake = time() + 30; // 30 seconds
73
74 // Accelerate for 30 seconds, woo!
75 while (time() < $timeToBrake) {
76     echo $car->accelerate() . "\n";
77 }
78
79 echo $car->brake() . "\n";
```

# Output: fast and slow!

```
$ php ./drive.php
```

```
You're driving a Car_Ferrari  
Car's max speed is: 250  
is that all you've got?  
is that all you've got?  
is that all you've got?  
is that all you've got?  
...  
wimp!
```

```
$ php ./drive.php
```

```
You're driving a Car_Yugo  
Car's max speed is: 35  
please, stop...I can't take it!  
please, stop...I can't take it!  
please, stop...I can't take it!  
please, stop...I can't take it!  
...  
ah, what a relief!
```

# What have I used a factory for?

- Credit card billing processors
- Copyright scanning vendor interfaces
- Image manipulation classes (GD vs. imagick)
- Newsfeed vendor content consumption (pre-RSS days)

# Where else is it seen?

- Zend Framework
  - Zend\_Db
  - Zend\_Cache
  - Zend\_Uri
  - Zend\_Paginator
- Where else?

# Summary

- Factories are your friend!
- If you're using a switch or if-else to determine which class to use...
  - ...make it a Factory!
  - Especially if you're copying and pasting that code!

# Thanks!

<http://www.deshong.net/>  
[brian@deshong.net](mailto:brian@deshong.net)

<http://www.schematic.com/>  
[bdeshong@schematic.com](mailto:bdeshong@schematic.com)