Clean Code

Peter Ruczynski

php BERKSHIRE

What is clean code?

171313

- Adherence to PSR standards?
  - PSR-1?
  - PSR-2?
- Comments?
- Spacing?
- Is that enough?

We can do much better than that!

- We are authors, even artists!
- We should set examples to other developers
- We should leave the code as clean or cleaner than we find it.
- So what can we do?

Make it better...

- Follow PSR standards yes, but also...
- Improve naming
- Use comments wisely
- Use spacing consistently and thoughtfully
- Organise our classes, methods and functions
- Improve error handling
- Look out for code ‘smells’ and refactor

Naming

- Make names meaningful and reveal the intention of what is named

```
class DataRcdT87
{
    private $crtd;
    private $updtd;
    private $dbidint = “87”;
    // …
}
```

```
class Employee
{
    private $createdTimestamp;
    private $updatedTimestamp;
    private $recordType = “87”;
    // …
}
```

Naming

- Make names meaningful and reveal the intention of what is named
  - Use full word, try not to abbreviate
  - Classes are things, use nouns
  - Methods ‘do’ things, use verbs
- You don’t need to add ‘Interface’ and ‘Abstract’, users don’t need to know this
- You don’t always need to add a context, only when necessary

Comments

```
/**
 * This is the employee class.
 * It contains employee information as well as when it was updated
 * and when it was created.
 */
class Employee
{
	// The time the record was created
	private $createdTimestamp;
	// The time the record was updated
	private $updatedTimestamp;
	// The type of the record as referenced by the HR module
	private $recordType = “87”;
	// The version of the record type
	private $info;

	// …
}
```

Comments

- Keep comments relevant
- Ensure they ‘add’ information and only when that information is obscured
- Comments don’t make bad code better or acceptable
- Make the code do the talking

Class and method organisation

- Keep them small!
  - Methods < 20 lines
  - Classes (harder) should:
    - have one responsibility!
    - have a small number of instance variables
    - be highly cohesive
- Read code from top to bottom
- Methods should do one thing and one thing only!
- Methods should have no side effects
- Avoid ‘switch’ if you can (they’re big and mean you’re doing lots of things)

Method arguments

- Methods should ideally have no arguments
- 1 argument
  - operating on that argument
  - avoid references (output arguments)
    - return transformations
    - if possible, change the state of the owning object
      - `$page.appendHeader()` rather than `appendHeader($page)`
- 2 arguments
  - harder to understand
  - can get arguments the wrong way round
  - rarely have cohesion or ordering
- Don’t pass flags, they mean your method does more than one thing!

Do or inform?

Methods should do something or tell you something, not both!

```
class Employee
{
	// ...

	public function set($attribute, $value)
	{
		// ...
		return true;
	}

	// ...

		if (this.set('name', 'elephant')) {...

	// ...
}
```

Do or inform?

- What does this mean?
- Are we checking the value of the attribute is already set to ‘elephant’?
- Are we checking that the attribute has been set correctly?
- ‘set’ is meant to be a verb, but ‘if’ changes that
- This is called command/query separation

Errors

- Prefer exceptions to returning errors
- Exceptions mean fewer checks in your code
  - Caller no longer needs to be responsible...
- Can move error handling out of caller
- If a method handles an error, that should be all it does!
  - Move other processing to another method called from the try block

Code ‘smells’

- Name doesn’t mean anything
- Large class
- Large method
- Too many method arguments
- Method output arguments
- Method flag arguments
- Duplication of code
- Method does more than one thing
- Lots of ‘+1’ or ‘-1’ - encapsulate in a variable (boundary condition)
- ...

References

- PHP Standards: https://github.com/php-fig/fig-standards
  - Basic Coding Standards: http://www.php-fig.org/psr/psr-1/
  - Coding Style: http://www.php-fig.org/psr/psr-2/
- Clean Code Book (Robert C Martin):
  - http://www.hive.co.uk/ebook/clean-code-a-handbook-of-agile-software-craftsmanship/17362277/
- Refactoring Book (Martin Fowler):
  - http://www.amazon.co.uk/Refactoring-Improving-Design-Existing-Technology/dp/0201485672/ref=sr_1_1?ie=UTF8&qid=1439840291&sr=8-1&keywords=refactoring
- S.O.L.I.D principles
  - https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)
- D.R.Y - Don't Repeat Yourself
  - https://en.wikipedia.org/wiki/Don%27t_repeat_yourself
- K.I.S.S - Keep It Simple Stupid!
  - https://en.wikipedia.org/wiki/KISS_principle
- Law of Demeter http://c2.com/cgi/wiki?LawOfDemeter
  - PeterVanRooijen posted the following description of the LawOfDemeter to Usenet:
    - You can play with yourself.
    - You can play with your own toys (but you can't take them apart),
    - You can play with toys that were given to you.
    - And you can play with toys you've made yourself.

php BERKSHIRE

Questions?

S.O.L.I.D Principles

- A class should have a **S**ingle responsibility
- **O**pen to extension and closed to modification
- **L**iskov substitution principle - objects should be replaceable with an instance of a subtype without altering the behaviour of the program
- **I**nterface segregation principle - keep interfaces small so clients only need to know about the methods that apply to them
- **D**ependency Inversion - depend on abstractions not details

Law of Demeter

PeterVanRooijen posted the following description of the LawOfDemeter to Usenet:

- You can play with yourself.
- You can play with your own toys (but you can't take them apart),
- You can play with toys that were given to you.
- And you can play with toys you've made yourself.

In Plain English:

- Your method can call other methods in its class directly
- Your method can call methods on its own fields directly (but not on the fields' fields)
- When your method takes parameters, your method can call methods on those parameters directly.
- When your method creates local objects, that method can call methods on the local objects