Open/closed principle

*„Software entities (classes, modules, functions, etc.) should be **open for extension**, but **closed for modification**”*

Basics

- Bertrand Meyer is generally credited for having originated the term *open/closed principle (OCP)*, which appeared in his 1988 book *Object Oriented Software Construction*.

✓ **We should strive to write code that doesn’t have to be changed every time the requirements change**

✗ When a single change to a program results in a cascade of changes to dependent modules – OCP is broken

https://www.exceptionnotfound.net/simply-solid-the-open-closed-principle/

Why?

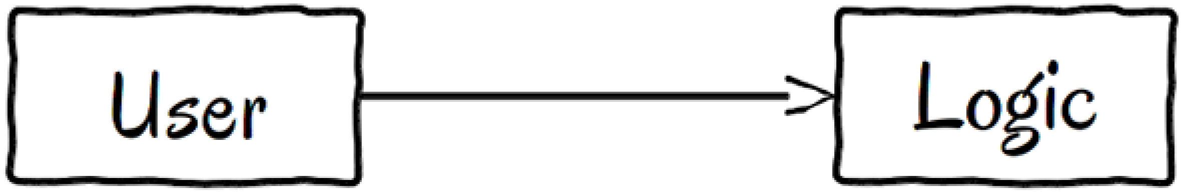
- Changes in the existing code should be minimized, since it's assumed that the existing code is already unit tested and changes in already written code might affect the existing functionality in an unwanted manner.

Single responsibility vs Open/closed

- „When we have code that has a single reason to change, introducing a new feature will create a secondary reason for that change. So both SRP and OCP would be violated. In the same way, if we have code that should only change when its main function changes and should remain unchanged when a new feature is added to it, thus respecting OCP, will mostly respect SRP also”.

https://code.tutsplus.com/tutorials/solid-part-2-the-openclosed-principle--net-36600

How to?



```php
function testItCanGetTheProgressOfAFileAsAPercent() {
    $file = new File();
    $file->length = 200;
    $file->sent = 100;

    $progress = new Progress($file);

    $this->assertEquals(50, $progress->getAsPercent());
}
```

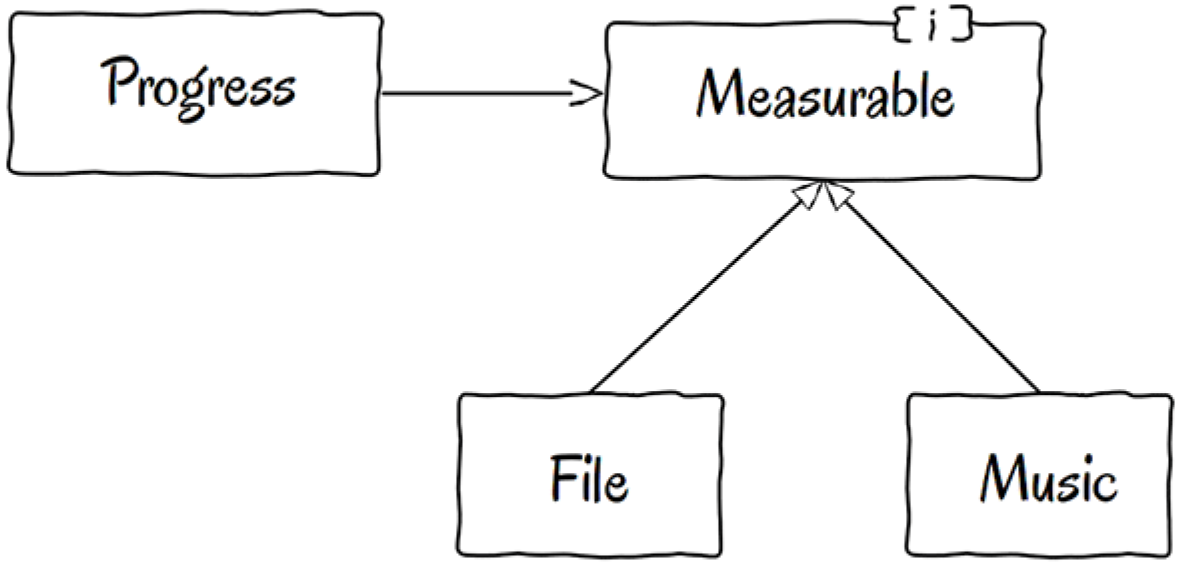
```php
class Progress {

    private $file;

    function __construct(File $file) {
        $this->file = $file;
    }

    function getAsPercent() {
        return $this->file->sent * 100 / $this->file->length;
    }

}
```

```php
class File {
    public $length;
    public $sent;
}
```

New requirements

- Add new resource – *Music*, which would be streamed and we want to check proces progres
- Options to avoid changing od *Progress* class
  - Use dynamic typing
  - **Use abstraction**
    - **Interface**
    - Abstraction
    - Template Method

Define contract



Contract / methods

- Be careful while declaring interface method
  - Does *Progress* need to set the values? Probably not
  - If you would define the any method, you would force all of the server classes to implement it
- Avoid „I naming” (*IFile* or *FileInterface)* - interfaces belong to their clients, think about functionality and forget about the implementation

```
interface Measurable {
    function getLength();
    function getSent();
}
```

Naming

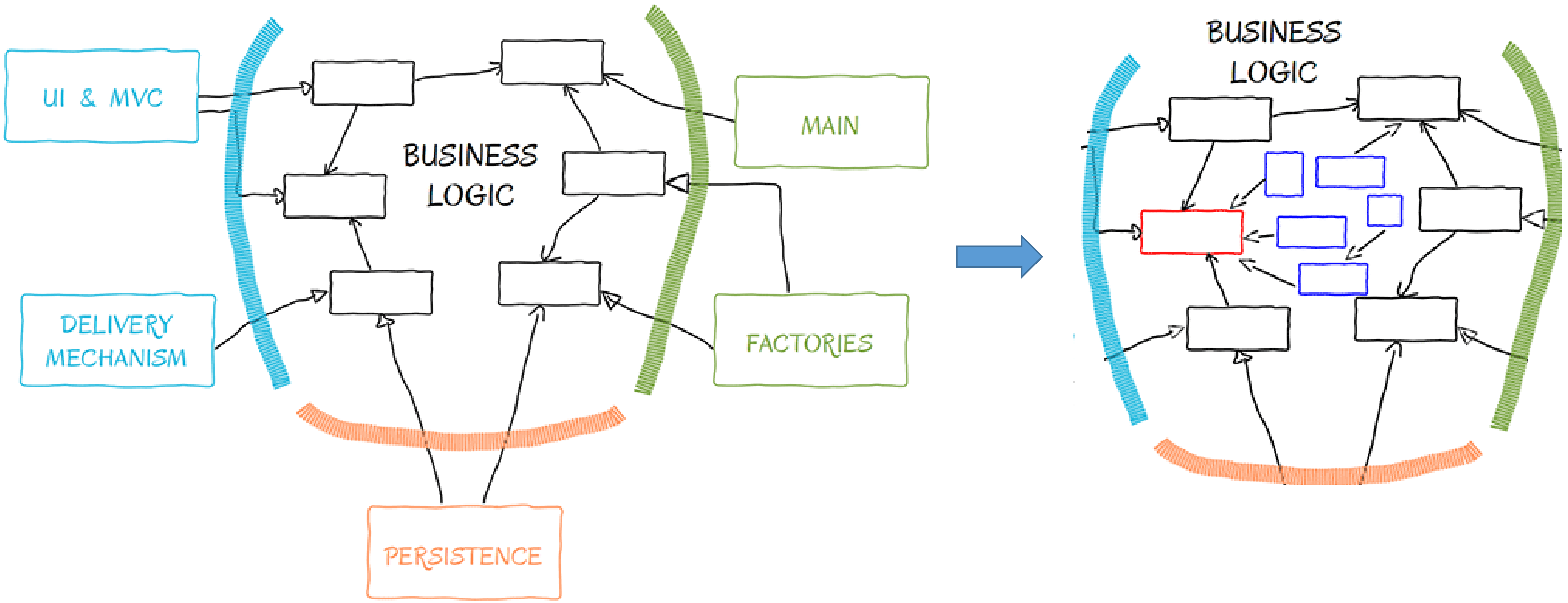
- Implementation can be from various domains
  - There are files and music, but *Progress* can be used in racing symulator to measuree classes like Speed, Fuel, etc.

```
interface Measurable {
    function getLength();
    function getSent();
}
```

$\rightarrow$

```
interface Measurable {
    function getTotalSize();
    function getMeasurement ();
}
```

Adding a new module with OPC



How to plan interfaces?

„Any exaggeration is bad.
If you think about everything upfront, it is bad.
If you think about nothing upfront, it is also bad.”

https://code.tutsplus.com/tutorials/solid-part-2-the-openclosed-principle--net-36600#comment-1565099179

Sources

- https://code.tutsplus.com/tutorials/solid-part-2-the-openclosed-principle--net-36600
    - https://code.tutsplus.com/tutorials/solid-part-2-the-openclosed-principle--net-36600#comment-1565099179 – nice comment describing what question should be asked to know when to introduce interfaces
- https://www.cs.duke.edu/courses/fall07/cps108/papers/ocp.pdf