CLEAN CODE: REFACTORING

PRESENTED BY

JEFF CAROUTH
@jcarouth

PROGRAMMING IS HARD

WHY IS PROGRAMMING DIFFICULT?

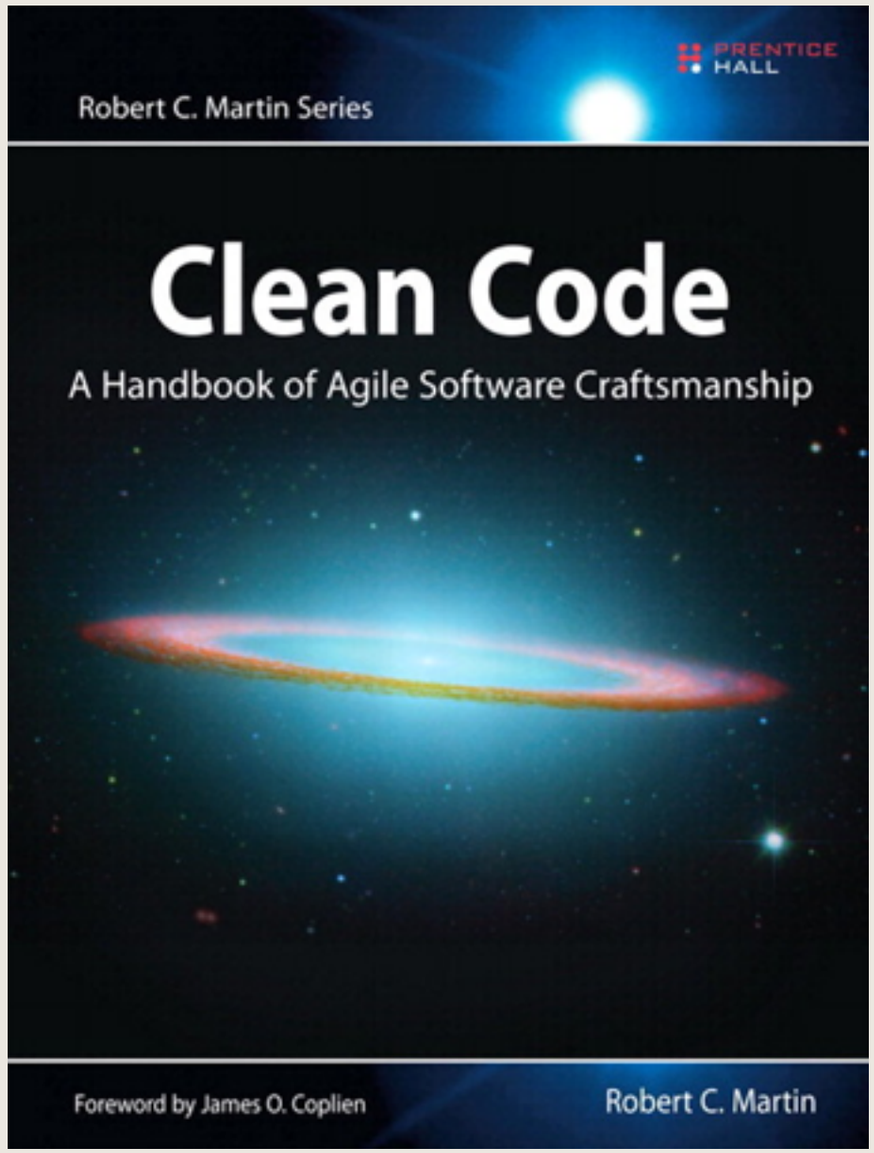
ARE YOU A COMPILER?

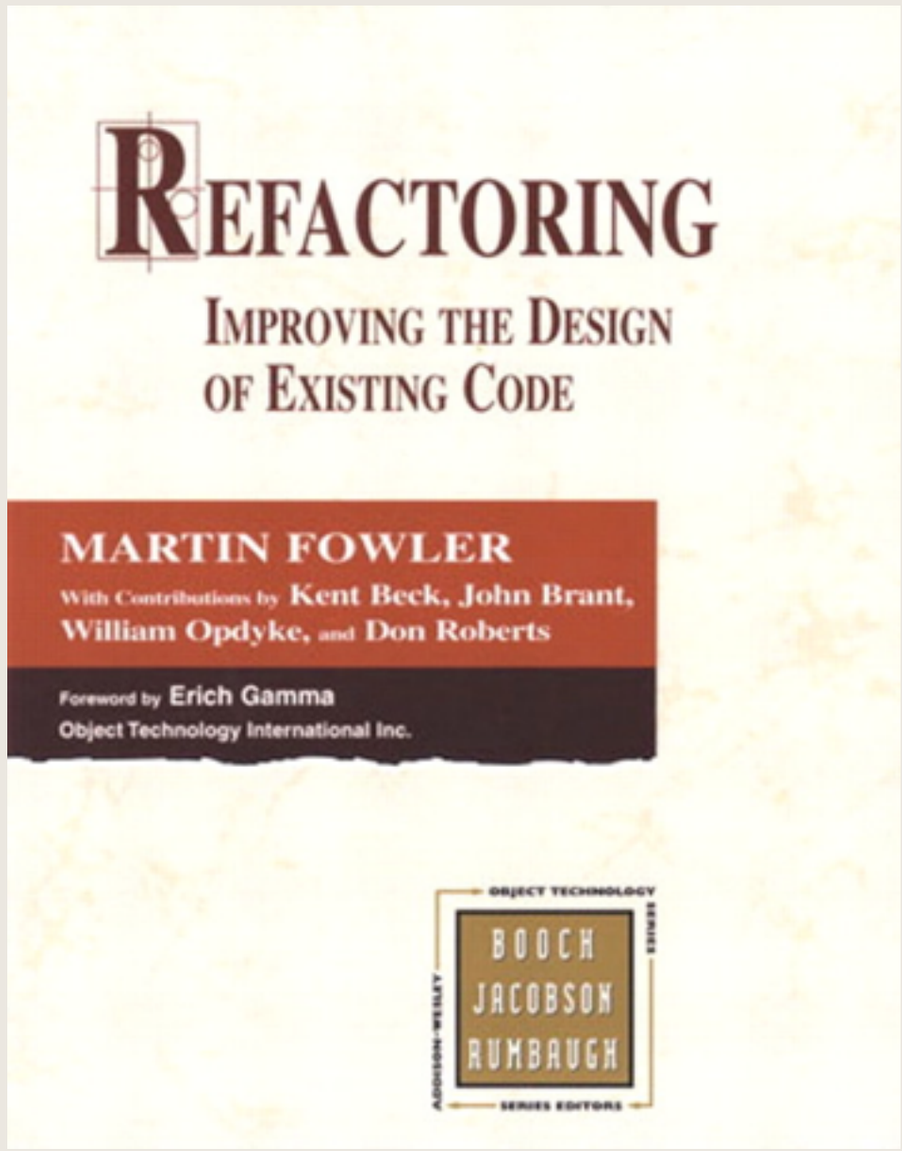
“ANY FOOL CAN WRITE CODE A COMPUTER CAN UNDERSTAND.”

“GOOD PROGRAMMERS WRITE CODE HUMANS CAN UNDERSTAND.”

– MARTIN FOWLER

READ THESE BOOKS





CLEAN CODE

Clean code is readable; it tells a story.

Clean code is cared for;
it is maintained.

Clean code is efficient; it is done right.

Clean code is extensible;
it is able to solve
tomorrow's problems.

Clean code is simple;
it is easy to understand
how it works and
what it does.

THE TWO RULES OF CLEAN CODE

RULE #1

Write dirty code, then clean it.

RULE #1

Write dirty code, then clean it.

> You cannot write clean code at first.

RULE #1

Write dirty code, then clean it.

> You cannot write clean code at first.

> Your first pass at solving a problem will not be your best.

RULE #1

Write dirty code, then clean it.

> You cannot write clean code at first.

> Your first pass at solving a problem will not be your best.

> Successive refinement through refactoring is the key to clean code.

RULE #2

Leave the code cleaner than you found it.

RULE #2

Leave the code cleaner than you found it.

> Shamelessly borrowed from the Boy Scouts of America.

RULE #2

Leave the code cleaner than you found it.

> Shamelessly borrowed from the Boy Scouts of America.

> Incremental changes are the way to go.

RULE #2

Leave the code cleaner than you found it.

> Shamelessly borrowed from the Boy Scouts of America.

> Incremental changes are the way to go.

> Even if the only change you make is cosmetic, you are maintaining clean code.

AN EXAMPLE

```php
class AccountService
{
    public function find($id)
    {
        if (is_string($id) && strlen($id) == 36) {
            $statement = $this->db->prepare(
                "SELECT * FROM accounts WHERE id = :account_id"
            );
            $statement->execute(['account_id' => $id]);

            if ($statement->rowCount() == 1) {
                $data = $statement->fetch();

                $account = new Account();
                $account->setId($data['id']);
                $account->setName($data['name']);

                return $account;
            } else {
                return null;
            }
        } else {
            throw new \InvalidArgumentException('ID is not valid');
        }
    }
}
```

```php
class AccountService
{
    public function find($id)
    {
        if (is_string($id) && strlen($id) == 36) {
            $statement = $this->db->prepare(
                "SELECT * FROM accounts WHERE id = :account_id"
            );
            $statement->execute(['account_id' => $id]);

            if ($statement->rowCount() == 1) {
                $data = $statement->fetch();

                $account = new Account();
                $account->setId($data['id']);
                $account->setName($data['name']);

                return $account;
            } else {
                return null;
            }
        } else {
            throw new \InvalidArgumentException('ID is not valid');
        }
    }
}
```

REFACTORING

When you have a significant number of nested conditionals, you can refactor them to be more understandable using guard clauses.

```php
class AccountService
{
    public function find($id)
    {
        if (is_string($id) && strlen($id) == 36) {
            $statement = $this->db->prepare(
                "SELECT * FROM accounts WHERE id = :account_id"
            );
            $statement->execute(['account_id' => $id]);

            if ($statement->rowCount() == 1) {
                $data = $statement->fetch();

                $account = new Account();
                $account->setId($data['id']);
                $account->setName($data['name']);

                return $account;
            } else {
                return null;
            }
        } else {
            throw new \InvalidArgumentException('ID is not valid');
        }
    }
}
```

```php
class AccountService
{
    public function find($id)
    {
        if (!is_string($id) || strlen($id) != 36) {
            throw new \InvalidArgumentException('Account ID is not valid');
        }

        $statement = $this->db->prepare(
            "SELECT * FROM accounts WHERE id = :account_id"
        );
        $statement->execute(['account_id' => $id]);

        if ($statement->rowCount() == 1) {
            $data = $statement->fetch();

            $account = new Account();
            $account->setId($data['id']);
            $account->setName($data['name']);

            return $account;
        } else {
            return null;
        }
    }
}
```

REFACTORING

Complicated or boolean expressions using magic numbers or otherwise non-obvious logic should be refactored to a descriptively named helper method.

```php
class AccountService
{
    public function find($id)
    {
        if (!$this->isValidId($id)) {
            throw new \InvalidArgumentException('Account ID is not valid');
        }

        //snip – my favorite slide-only refactoring
    }

    private function isValidId($id)
    {
        return is_string($id) && strlen($id) == 36;
    }
}
```

```php
public function find($id)
{
    if (!$this->isValidId($id)) {
        throw new \InvalidArgumentException('Account ID is not valid');
    }

    $statement = $this->db->prepare(
        "SELECT * FROM accounts WHERE id = :account_id"
    );
    $statement->execute(['account_id' => $id]);

    if ($statement->rowCount() == 1) {
        $data = $statement->fetch();

        $account = new Account();
        $account->setId($data['id']);
        $account->setName($data['name']);

        return $account;
    } else {
        return null;
    }
}
```

REFACTORING

Extract what could be common functionality, or even functionality that belongs on a different object into methods.

```php
private function fetchAccountDataById($id)
{
    $statement = $this->db->prepare(
        "SELECT * FROM accounts WHERE id = :account_id"
    );
    $statement->execute(['account_id' => $id]);

    if ($statement->rowCount() == 1) {
        $data = $statement->fetch();
    } else {
        $data = array();
    }

    return $data;
}
```

```php
class AccountService
{
    public function find($id)
    {
        if (!$this->isValidId($id)) {
            throw new \InvalidArgumentException('Account ID is not valid');
        }

        $data = $this->fetchAccountDataById($id);

        if (empty($data)) {
            return null;
        }

        $account = new Account();
        $account->setId($data['id']);
        $account->setName($data['name']);

        return $account;
    }

    private function fetchAccountDataById($id)
    {
        //snip – slides are not very tall
    }
}
```

```php
class AccountService
{
    public function find($id)
    {
        if (!$this->isValidId($id)) {
            throw new \InvalidArgumentException('Account ID is not valid');
        }

        $data = $this->fetchAccountDataById($id);

        if (empty($data)) {
            return null;
        }

        $account = new Account();
        $account->setId($data['id']);
        $account->setName($data['name']);

        return $account;
    }
}
```

```php
class AccountService
{
    public function find($id)
    {
        if (!$this->isValidId($id)) {
            throw new \InvalidArgumentException('Account ID is not valid');
        }

        $data = $this->fetchAccountDataById($id);

        return $this->buildAccountFromData($data);
    }

    private function buildAccountFromData($data)
    {
        if (empty($data)) {
            return null;
        }

        $account = new Account();
        $account->setId($data['id']);
        $account->setName($data['name']);

        return $account;
    }

    // snip
}
```

REFACTORING

Replace instances of returning `null` in place of an object with returning instances of a `NullObject` where possible and appropriate.

```php
$accountService = new AccountService(new MyPdoWrapper(...));

$account = $accountService->fetch('...');

if (null === $account) {
    //bail. we don't have a valid account

}

if (!$account->isActive()) {
    //error the account is not active
}
```

```php
class NullAccount extends Account
{
    public function isActive()
    {
        return false;
    }
}
```

```php
class AccountService
{
    public function find($id)
    {
        if (!$this->isValidId($id)) {
            throw new \InvalidArgumentException('Account ID is not valid');
        }

        $data = $this->fetchAccountDataById($id);

        return $this->buildAccountFromData($data);
    }

    private function buildAccountFromData($data)
    {
        if (empty($data)) {
            return new NullAccount();
        }

        $account = new Account();
        $account->setId($data['id']);
        $account->setName($data['name']);

        return $account;
    }

    // snip
}
```

REFACTORING

Extract methods or logic into classes when the code isn't necessarily relevant to the object you are working in.

```php
class AccountService
{
    // snip

    private function fetchAccountDataById($id)
    {
        $statement = $this->db->prepare(
            "SELECT * FROM accounts WHERE id = :account_id"
        );
        $statement->execute(['account_id' => $id]);

        if ($statement->rowCount() == 1) {
            $data = $statement->fetch();
        } else {
            $data = array();
        }

        return $data;
    }

    // snip
}
```

```php
class AccountDataProvider
{
    public function __construct(PDO $pdo)
    {
        $this->dbh = $pdo;
    }

    public function fetchById($id)
    {
        $statement = $this->db->prepare(
            "SELECT * FROM accounts WHERE id = :account_id"
        );
        $statement->execute(['account_id' => $id]);

        if ($statement->rowCount() == 1) {
            $data = $statement->fetch();
        } else {
            $data = array();
        }

        return $data;
    }
}
```

```php
class AccountService
{
    // snip

    public function find($id)
    {
        if (!$this->isValidId($id)) {
            throw new \InvalidArgumentException('Account ID is not valid');
        }

        $data = $this->provider->fetchById($id);

        return $this->buildAccountFromData($data);
    }

    // snip
}
```

```php
$accountService = new AccountService(new AccountDataProvider($pdo));
$account = $accountService->fetch('...');

if (!$account->isActive()) {
    //error the account is not active
}
```

WHICH IS MORE READABLE?

```php
class AccountService
{
    public function find($id)
    {
        if (isset($id) && strlen($id) == 36) {
            $statement = $this->db->prepare(
                "SELECT * FROM accounts WHERE id = :account_id"
            );
            $statement->execute(['account_id' => $id]);

            if ($statement->rowCount() == 1) {
                $data = $statement->fetch();

                $account = new Account();
                $account->setId($data['id']);
                $account->setName($data['name']);

                return $account;
            } else {
                return null;
            }
        } else {
            throw new \InvalidArgumentException('ID is not valid');
        }
    }
}
```

```php
class AccountService
{
    public function find($id)
    {
        if (!$this->isValidId($id)) {
            throw new \InvalidArgumentException('Account ID is not valid');
        }

        $data = $this->provider->fetchById($id);

        return $this->buildAccountFromData($data);
    }

    // snip
}
```

TWO EXAMPLES ARE BETTER THAN ONE

```php
class Product
{
    public function createUrl($store, $isMobile = false, $includeDomain = true)
    {
        $url = "";

        if (!$store->isDefault()) {
            $trySSL = false;
            $forceToCore = false;

            if ($this->featureToggle) {
                if ($includeDomain) {
                    $url .= $this->config->secureProtocol.$this->config->storeDomainName;
                }
                $url .= "/checkout?product_id=".$this->productId;
                if ($this->isDated()) {
                    $url .= "&date=".date("Y-m-d", strtotime($this->date));
                }
            } else {
                if ($includeDomain) {
                    $url .= $store->GetFullUrl($trySSL, $isMobile);
                }
                $url .= "/resort_detail.php?ProductId=".$this->productId;
                if ($this->isDated()) {
                    $url .= "&StartDate=".date("Y-m-d", strtotime($this->date));
                }
            }
        } else {
            $domainName = $this->config->domainName;
            if (!$domainName) {
                $domainName = $this->config->desktopDomainName;
            }
            if ($this->featureToggle) {
                if ($includeDomain) {
                    $url .= $this->config->secureProtocol.$domainName;
                }
                $url .= "/checkout?product_id=".$this->productId;
                if ($this->isDated()) {
                    $url .= "&date=".date("Y-m-d", strtotime($this->date));
                }
            } else {
                if ($includeDomain) {
                    $url .= "http://".$domainName;
                }
                $url .= "/resort_detail.php?ProductId=".$this->productId;
                if ($this->isDated()) {
                    $url .= "&StartDate=".date("Y-m-d", strtotime($this->date));
                }
            }
        }
    }
}
```

```
{
    $url = "";

    if (!$store->isDefault()) {
        $trySSL = false;
        $forceToCore = false;

        if ($this->featureToggle) {
            if ($includeDomain) {
                $url .= $this->config->secureProtocol.$this->config->sto
            }
            $url .= "/checkout?product_id=".$this->productId;
            if ($this->isDated()) {
                $url .= "&date=".date("Y-m-d", strtotime($this->date));
            }
        } else {
            if ($includeDomain) {
                $url .= $store->GetFullUrl($trySSL, $isMobile);
            }
            $url .= "/product_detail.php?ProductId=".$this->productId;
            if ($this->isDated()) {
                $url .= "&StartDate=".date("Y-m-d", strtotime($this->dat
            }
        }
    } else {
        $domainName = $this->config->domainName;
        if (!$domainName) {
            $domainName = $this->config->desktopDomainName;
        }
        if ($this->featureToggle) {
```

```
                $url .= "&StartDate=".date("Y-m-d", strtotime($this->date
            }
        }
    } else {
        $domainName = $this->config->domainName;
        if (!$domainName) {
            $domainName = $this->config->desktopDomainName;
        }
        if ($this->featureToggle) {
            if ($includeDomain) {
                $url .= $this->config->secureProtocol.$domainName;
            }
            $url .= "/checkout?product_id=".$this->productId;
            if ($this->isDated()) {
                $url .= "&date=".date("Y-m-d", strtotime($this->date));
            }
        } else {
            if ($includeDomain) {
                $url .= "http://".$domainName;
            }
            $url .= "/product_detail.php?ProductId=".$this->productId;
            if ($this->isDated()) {
                $url .= "&StartDate=".date("Y-m-d", strtotime($this->date
            }
        }
    }
}
```

```
{
    $url = "";

    if (!$store->isDefault()) {
        $trySSL = false;
        $forceToCore = false;

        if ($this->featureToggle) {
            if ($includeDomain) {
                $url .= $this->config->secureProtocol.$this->config->sto
            }
            $url .= "/checkout?product_id=".$this->productId;
            if ($this->isDated()) {
                $url .= "&date=".date("Y-m-d", strtotime($this->date));
            }
        } else {
            if ($includeDomain) {
                $url .= $store->GetFullUrl($trySSL, $isMobile);
            }
            $url .= "/product_detail.php?ProductId=".$this->productId;
            if ($this->isDated()) {
                $url .= "&StartDate=".date("Y-m-d", strtotime($this->dat
            }
        }
    } else {
        $domainName = $this->config->domainName;
        if (!$domainName) {
            $domainName = $this->config->desktopDomainName;
        }
        if ($this->featureToggle) {
```

```
class Product
{
    public function createUrl($store, $isMobile = false, $includeDomain = tru
    {
        $url = "";

        if ($includeDomain) {
            if ($store->isDefault()) {
                $domainName = $this->config->domainName;
                if (!$domainName) {
                    $domainName = $this->config->desktopDomainName;
                }

                if ($this->featureToggle) {
                    $url .= $this->config->secureProtocol.$domainName;
                } else {
                    $url .= "http://".$domainName;
                }
            } else {
                if ($this->featureToggle) {
                    $url .= $this->config->secureProtocol.$this->config->stor
                } else {
                    $url .= $store->GetFullUrl($trySSL, $isMobile);
                }
            }
        }

        if (!$store->isDefault()) {
            $trySSL = false;
```

```php
class Product
{
    public function createUrl($store, $isMobile = false, $includeDomain = tru
    {
        $url = "";

        if ($includeDomain) {
            if ($store->isDefault()) {
                $domainName = $this->config->domainName;
                if (!$domainName) {
                    $domainName = $this->config->desktopDomainName;
                }

                if ($this->featureToggle) {
                    $url .= $this->config->secureProtocol.$domainName;
                } else {
                    $url .= "http://".$domainName;
                }
            } else {
                if ($this->featureToggle) {
                    $url .= $this->config->secureProtocol.$this->config->stor
                } else {
                    $url .= $store->GetFullUrl($trySSL, $isMobile);
                }
            }
        }

        if (!$store->isDefault()) {
            $trySSL = false;
```

```
class Product
{
    public function createUrl($store, $isMobile = false, $includeDomain = tru
    {
        $url = "";
        $protocol = $this->config->secureProtocol;

        if ($includeDomain) {
            if ($store->isDefault()) {

                $domainName = $this->config->domainName;
                if (!$domainName) {
                    $domainName = $this->config->desktopDomainName;
                }

                if ($this->featureToggle) {
                    $url .= $protocol.$domainName;
                } else {
                    $url .= "http://".$domainName;
                }
            } else {
                if ($this->featureToggle) {
                    $url .= $protocol.$this->config->storeDomainName;
                } else {
                    $url .= $store->GetFullUrl($trySSL, $isMobile);
                }
            }
        }
```

```php
class Product
{
    public function createUrl($store, $isMobile = false, $includeDomain = tru
    {
        $url = "";
        $protocol = $this->config->secureProtocol;

        if ($includeDomain) {
            if ($store->isDefault()) {
                $domainName = $this->config->domainName;

                if (!$domainName) {
                    $domainName = $this->config->desktopDomainName;
                }

                if (!$this->featureToggle) {
                    $protocol = "http://";
                }

                $url .= $protocol.$domainName;
            } else {
                if ($this->featureToggle) {
                    $url .= $protocol.$this->config->storeDomainName;
                } else {
                    $url .= $store->GetFullUrl($trySSL, $isMobile);
                }

            }
        }
```

```php
class Product
{
    public function createUrl($store, $isMobile = false, $includeDomain = tru
    {
        $url = "";
        $protocol = $this->config->secureProtocol;

        if ($includeDomain) {
            if ($store->isDefault()) {
                $domainName = $this->config->domainName;

                if (!$domainName) {
                    $domainName = $this->config->desktopDomainName;
                }

                if (!$this->featureToggle) {
                    $protocol = "http://";
                }

                $url .= $protocol.$domainName;
            } else {
                if ($this->featureToggle) {
                    $url .= $protocol.$this->config->storeDomainName;
                } else {
                    $url .= $store->GetFullUrl($trySSL, $isMobile);
                }

            }
        }
```

```php
class Product
{
    public function createUrl($store, $isMobile = false, $includeDomain = tru
    {
        $url = "";
        $protocol = $this->config->secureProtocol;

        if ($includeDomain) {
            $domainName = $this->config->storeDomainName;

            if ($store->isDefault()) {
                $domainName = $this->config->domainName;
                if (!$domainName) {
                    $domainName = $this->config->desktopDomainName;
                }
            }

            if ($store->isDefault()) {
                if (!$this->featureToggle) {
                    $protocol = "http://";
                }

                $url .= $protocol.$domainName;
            } else {
                if ($this->featureToggle) {
                    $url .= $protocol.$domainName;
                } else {
                    $url .= $store->GetFullUrl($trySSL, $isMobile);
                }
```

```
class Product
{
    public function createUrl($store, $isMobile = false, $includeDomain = tru
    {
        $url = "";
        $protocol = $this->config->secureProtocol;

        if ($includeDomain) {
            $domainName = $this->config->storeDomainName;

            if ($store->isDefault()) {
                $domainName = $this->config->domainName;
                if (!$domainName) {
                    $domainName = $this->config->desktopDomainName;
                }
            }
        }

        if ($store->isDefault() && !$this->featureToggle) {
            $protocol = "http://";
        }

        $url .= $protocol.$domainName;

        if (!$store->isDefault()) {
            if ($this->featureToggle) {
                $url .= $protocol.$domainName;
            } else {
                $url .= $store->GetFullUrl($trySSL, $isMobile);
            }
        }
```

```
$protocol = $this->config->secureProtocol;

if ($includeDomain) {
    $domainName = $this->config->storeDomainName;

    if ($store->isDefault()) {
        $domainName = $this->config->domainName;
        if (!$domainName) {
            $domainName = $this->config->desktopDomainName;
        }
    }

    if ($store->isDefault() && !$this->featureToggle) {
        $protocol = "http://";
    }

    $url .= $protocol.$domainName;

    if (!$store->isDefault()) {
        if ($this->featureToggle) {
            $url .= $protocol.$domainName;
        } else {
            $url .= $store->GetFullUrl($trySSL, $isMobile);
        }
    }
}

if (!$store->isDefault()) {
    $trySSL = false;
```

```
$protocol = $this->config->secureProtocol;

if ($includeDomain) {
    $domainName = $this->config->storeDomainName;

    if ($store->isDefault()) {
        $domainName = $this->config->domainName;
        if (!$domainName) {
            $domainName = $this->config->desktopDomainName;
        }
    }

    if ($store->isDefault() && !$this->featureToggle) {
        $protocol = "http://";
    }

    $url .= $protocol.$domainName;

    if (!$store->isDefault()) {
        if (!$this->featureToggle) {
            $url .= $store->GetFullUrl($trySSL, $isMobile);
        }
    }
}

if (!$store->isDefault()) {
    $trySSL = false;
    $forceToCore = false;
```

```
$protocol = $this->config->secureProtocol;

if ($includeDomain) {
    $domainName = $this->config->storeDomainName;

    if ($store->isDefault()) {
        $domainName = $this->config->domainName;
        if (!$domainName) {
            $domainName = $this->config->desktopDomainName;
        }
    }

    if ($store->isDefault() && !$this->featureToggle) {
        $protocol = "http://";
    }

    $url .= $protocol.$domainName;

    if (!$store->isDefault() && !$this->featureToggle) {
        $trySSL = false;
        $url .= $store->GetFullUrl($trySSL, $isMobile);
    }
}

if (!$store->isDefault()) {
    if ($this->featureToggle) {
        $url .= "/checkout?product_id=".$this->productId;
        if ($this->isDated()) {
            $url .= "&date=".date("Y-m-d", strtotime($this->date));
```

```
        }

        if (!$store->isDefault()) {
            if ($this->featureToggle) {
                $url .= "/checkout?product_id=".$this->productId;
                if ($this->isDated()) {
                    $url .= "&date=".date("Y-m-d", strtotime($this->date));
                }
            } else {
                $url .= "/product_detail.php?ProductId=".$this->productId;
                if ($this->isDated()) {
                    $url .= "&StartDate=".date("Y-m-d", strtotime($this->date
                }
            }
        } else {
            if ($this->featureToggle) {
                $url .= "/checkout?product_id=".$this->productId;
                if ($this->isDated()) {
                    $url .= "&date=".date("Y-m-d", strtotime($this->date));
                }
            } else {
                $url .= "/product_detail.php?ProductId=".$this->productId;
                if ($this->isDated()) {
                    $url .= "&StartDate=".date("Y-m-d", strtotime($this->date
                }
            }
        }
    }
}
```

```
        }

        if ($this->featureToggle) {
            if (!$store->isDefault()) {
                $url .= "/checkout?product_id=".$this->productId;
                if ($this->isDated()) {
                    $url .= "&date=".date("Y-m-d", strtotime($this->date));
                }
            } else {
                $url .= "/checkout?product_id=".$this->productId;
                if ($this->isDated()) {
                    $url .= "&date=".date("Y-m-d", strtotime($this->date));
                }
            }
        } else {
            if (!$store->isDefault()) {
                $url .= "/resort_detail.php?ProductId=".$this->productId;
                if ($this->isDated()) {
                    $url .= "&StartDate=".date("Y-m-d", strtotime($this->date
                }
            } else {
                $url .= "/resort_detail.php?ProductId=".$this->productId;
                if ($this->isDated()) {
                    $url .= "&StartDate=".date("Y-m-d", strtotime($this->date
                }
            }

        }
    }
}
```

```
        }

        if ($this->featureToggle) {
            $url .= "/checkout?product_id=".$this->productId;
        } else {
            $url .= "/resort_detail.php?ProductId=".$this->productId;
        }

        if ($this->isDated) {
            $dateParam = $this->featureToggle ? "start_date" : "StartDate";
            $url = "&{$dateParam}=" . date("Y-m-d", strtotime($this->date));
        }
    }
}
```

```
class Product
{
    public function createUrl($store, $isMobile = false, $includeDomain = tru
    {
        $url = "";
        $protocol = $this->config->secureProtocol;

        if ($includeDomain) {
            $domainName = $this->config->storeDomainName;

            if ($store->isDefault()) {
                $domainName = $this->config->domainName;
                if (!$domainName) {
                    $domainName = $this->config->desktopDomainName;
                }
            }

            if ($store->isDefault() && !$this->featureToggle) {
                $protocol = "http://";
            }

            $url .= $protocol.$domainName;

            if (!$store->isDefault() && !$this->featureToggle) {
                $trySSL = false;
                $url .= $store->GetFullUrl($trySSL, $isMobile);
            }
        }
```

```php
class Product
{
    public function createUrl($store, $isMobile = false, $includeDomain = tru
    {
        $url = "";
        $protocol = $this->config->secureProtocol;

        if ($includeDomain) {
            $domainName = $this->getDomainNameForStore($store);

            // snip
        }

        // snip
    }

    private function getDomainNameForStore($store)
    {
        $domainName = $this->config->storeDomainName;

        if ($store->isDefault()) {
            $domainName = $this->config->domainName;
            if (!$domainName) {
                $domainName = $this->config->desktopDomainName;
            }
        }
    }
}
```

WHEW

```php
class Product
{
    public function createUrl($store, $isMobile = false, $includeDomain = true)
    {
        $url = "";

        if (!$store->isDefault()) {
            $trySSL = false;
            $forceToCore = false;

            if ($this->featureToggle) {
                if ($includeDomain) {
                    $url .= $this->config->secureProtocol.$this->config->storeDomainName;
                }
                $url .= "/checkout?product_id=".$this->productId;
                if ($this->isDated()) {
                    $url .= "&date=".date("Y-m-d", strtotime($this->date));
                }
            } else {
                if ($includeDomain) {
                    $url .= $store->GetFullUrl($trySSL, $isMobile);
                }
                $url .= "/resort_detail.php?ProductId=".$this->productId;
                if ($this->isDated()) {
                    $url .= "&StartDate=".date("Y-m-d", strtotime($this->date));
                }
            }
        } else {
            $domainName = $this->config->domainName;
            if (!$domainName) {
                $domainName = $this->config->desktopDomainName;
            }
            if ($this->featureToggle) {
                if ($includeDomain) {
                    $url .= $this->config->secureProtocol.$domainName;
                }
                $url .= "/checkout?product_id=".$this->productId;
                if ($this->isDated()) {
                    $url .= "&date=".date("Y-m-d", strtotime($this->date));
                }
            } else {
                if ($includeDomain) {
                    $url .= "http://".$domainName;
                }
                $url .= "/resort_detail.php?ProductId=".$this->productId;
                if ($this->isDated()) {
                    $url .= "&StartDate=".date("Y-m-d", strtotime($this->date));
                }
            }
        }
    }
}
```

```php
class Product
{
    public function createUrl($store, $isMobile = false, $includeDomain = true)
    {
        $url = "";
        $protocol = $this->config->secureProtocol;

        if ($includeDomain) {
            $domainName = $this->getDomainNameForStore($store);

            if ($store->isDefault() && !$this->featureToggle) {
                $protocol = "http://";
            }

            $url .= $protocol.$domainName;

            if (!$store->isDefault() && !$this->featureToggle) {
                $trySSL = false;
                $url .= $store->GetFullUrl($trySSL, $isMobile);
            }
        }

        if ($this->featureToggle) {
            $url .= "/checkout?product_id=".$this->productId;
        } else {
            $url .= "/product_detail.php?ProductId=".$this->productId;
        }

        if ($this->isDated) {
            $dateParam = $this->featureToggle ? "start_date" : "StartDate";
            $url = "&{$dateParam}=" . date("Y-m-d", strtotime($this->date));
        }
    }

    private function getDomainNameForStore($store)
```

RECAP

RECAP

Code should be understandable by humans.

RECAP

Code should be understandable by humans.

Clean code is readable, cared for, efficient, extensible, and simple.

RECAP

Code should be understandable by humans.

Clean code is readable, cared for, efficient, extensible, and simple.

Write dirty code and successively refine.