Лабораторна робота №2

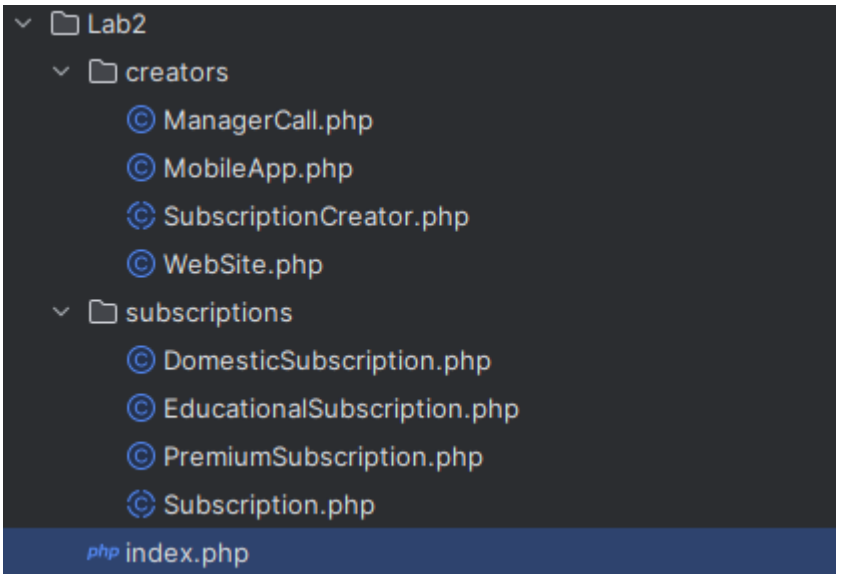
Тема: Породжувальні шаблони

Мета роботи: навчитися реалізовувати породжувальні шаблони проектування

Завдання 1: Фабричний метод.

1. Напишіть систему класів для реалізації функціоналу створення різних типів підписок для відео провайдера.
2. Кожна з підписок повинна мати щомісячну плату, мінімальний період підписки та список каналів й інших можливостей.
3. Види підписок: ***DomesticSubscription, EducationalSubscription, PremiumSubscription***.
4. Придбати (тобто створити) підписку можна за допомогою трьох різних класів: ***WebSite, MobileApp, ManagerCall***, кожен з них має реалізувати свою логіку створення підписок.
5. Покажіть правильність роботи свого коду запустивши його в головному методі програми.
6. Підготуйте діаграму створених у програмі класів та інтерфейсів за допомогою https://app.diagrams.net/, експортуйте та завантажте її до репозиторія.

Результат виконання:



index.php:

```php
<?php
require_once __DIR__ . '/subscriptions/Subscription.php';
require_once __DIR__ . '/subscriptions/DomesticSubscription.php';
require_once __DIR__ . '/subscriptions/EducationalSubscription.php';
require_once __DIR__ . '/subscriptions/PremiumSubscription.php';
require_once __DIR__ . '/creators/SubscriptionCreator.php';
require_once __DIR__ . '/creators/WebSite.php';
require_once __DIR__ . '/creators/MobileApp.php';
require_once __DIR__ . '/creators/ManagerCall.php';

use creators\WebSite;
use creators\MobileApp;
use creators\ManagerCall;

$website = new WebSite();
$mobileApp = new MobileApp();
$managerCall = new ManagerCall();

$sub1 = $website->createSubscription('domestic');
$sub2 = $mobileApp->createSubscription('premium');
$sub3 = $managerCall->createSubscription('educational');

print_r($sub1->getInfo());
print_r($sub2->getInfo());
print_r($sub3->getInfo());
```

клас WebSite:

```php
<?php

namespace creators;

use subscriptions\DomesticSubscription;
use subscriptions\EducationalSubscription;
use subscriptions\PremiumSubscription;

class WebSite extends SubscriptionCreator
{
   public function createSubscription(string $type):
\subscriptions\Subscription
   {
       return match ($type) {
           'domestic' => new DomesticSubscription(),
           'educational' => new EducationalSubscription(),
           'premium' => new PremiumSubscription(),
           default => throw new \Exception("Unknown subscription type")
       };
   }
}
```

клас SubscriptionCreator:

```php
<?php

namespace creators;

use subscriptions\Subscription;

abstract class SubscriptionCreator
{
   abstract public function createSubscription(string $type): Subscription;
}
```

клас MobileApp:

```php
<?php

namespace creators;

use subscriptions\DomesticSubscription;
use subscriptions\EducationalSubscription;
use subscriptions\PremiumSubscription;

class MobileApp extends SubscriptionCreator
{
   public function createSubscription(string $type):
\subscriptions\Subscription
   {
       return match ($type) {
           'domestic' => new DomesticSubscription(),
           'premium' => new PremiumSubscription(),
           default => throw new \Exception("Unknown subscription type")
       };
   }
}
```

клас ManagerCall:

```php
<?php

namespace creators;

use subscriptions\DomesticSubscription;
use subscriptions\EducationalSubscription;
use subscriptions\PremiumSubscription;

class ManagerCall extends SubscriptionCreator
{
   public function createSubscription(string $type):
\subscriptions\Subscription
   {
       return match ($type) {
           'domestic' => new DomesticSubscription(),
           'educational' => new EducationalSubscription(),
           'premium' => new PremiumSubscription(),
           default => throw new \Exception("Unknown subscription type")
       };
   }
}
```

клас Subscription:

```php
<?php

namespace subscriptions;

abstract class Subscription
{
   public $monthlyFee;
   public $minPeriod;
   public $channels;

   public function getInfo()
   {
       return [
           'monthlyFee' => $this->monthlyFee,
           'minPeriod'  => $this->minPeriod,
           'channels'   => $this->channels
       ];
   }
}
```

клас PremiumSubscription:

```php
<?php

namespace subscriptions;

class PremiumSubscription extends Subscription
{
    public function __construct()
    {
        $this->monthlyFee = 200;
        $this->minPeriod  = 1;
        $this->channels   = ['Усі канали', '4K', 'Спорт+', 'Кінотеатр'];
    }
}
```

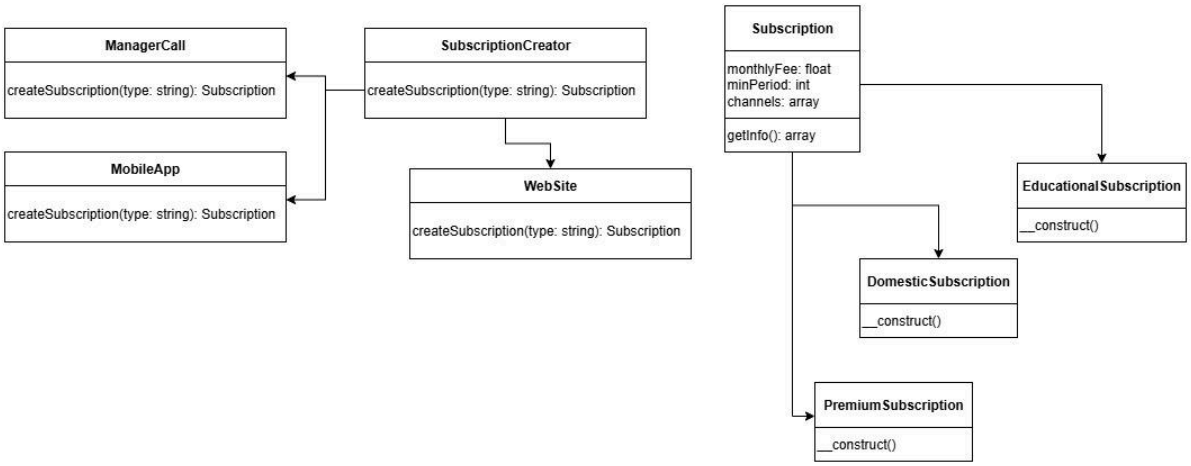
клас EducationalSubscription:

```php
<?php

namespace subscriptions;

class EducationalSubscription extends Subscription
{
    public function __construct()
    {
        $this->monthlyFee = 80;
        $this->minPeriod  = 3;
        $this->channels   = ['Наука', 'Документальні', 'Історія'];
    }
}
```

клас DomesticSubscription:

```php
<?php

namespace subscriptions;

class DomesticSubscription extends Subscription
{
    public function __construct()
    {
        $this->monthlyFee = 100;
        $this->minPeriod  = 1;
        $this->channels   = ['Новини', 'Спорт', 'Розваги'];
    }
}
```

Array ([monthlyFee] => 100 [minPeriod] => 1 [channels] => Array ([0] => Новини [1] => Спорт [2] => Розваги)) Array ([monthlyFee] => 200 [minPeriod] => 1 [channels] => Array ([0] => Усі канали [1] => 4K [2] => Спорт+ [3] => Кінотеатр)) Array ([monthlyFee] => 80 [minPeriod] => 3 [channels] => Array ([0] => Наука [1] => Документальні [2] => Історія))

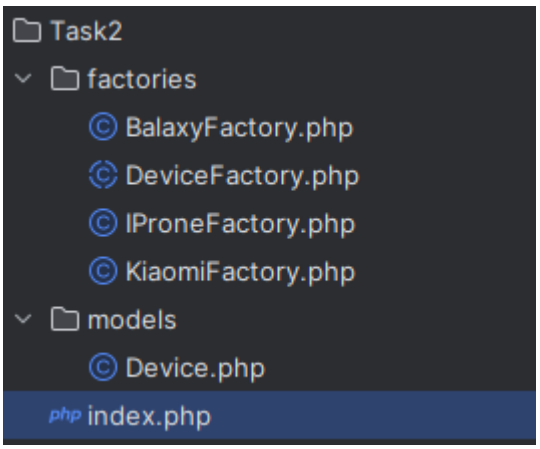


Завдання 2: Абстрактна фабрика.

1. Створіть фабрику виробництва техніки.
2. На фабриці мають створюватися різні девайси (наприклад, Laptop, Netbook, EBook, Smartphone) для різних брендів (IProne, Kiaomi, Balaxy).
3. Покажіть правильність роботи свого коду запустивши його в головному методі програми.

4. Підготуйте діаграму створених у програмі класів та інтерфейсів за допомогою https://app.diagrams.net/, експортуйте та завантажте її до репозиторія.

Результат виконання:



index.php:

```php
<?php

require_once __DIR__ . '/models/Device.php';
require_once __DIR__ . '/factories/DeviceFactory.php';
require_once __DIR__ . '/factories/IProneFactory.php';
require_once __DIR__ . '/factories/KiaomiFactory.php';
require_once __DIR__ . '/factories/BalaxyFactory.php';

function testFactory(DeviceFactory $factory) {
    $types = ['Laptop', 'Netbook', 'EBook', 'Smartphone'];

    foreach ($types as $type) {
        $device = $factory->createDevice($type);
        echo $device->getName() . "<br>";
    }
    echo "--------------------<br>";
}


$iprFactory = new IProneFactory();
$kiFactory = new KiaomiFactory();
$balaxyFactory = new BalaxyFactory();

testFactory($iprFactory);
testFactory($kiFactory);
testFactory($balaxyFactory);
```

клас Device:

```php
<?php

class Device {
    private string $brand;
    private string $type;

    public function __construct(string $brand, string $type) {
        $this->brand = $brand;
        $this->type = $type;
    }

    public function getName(): string {
        return "{$this->brand} {$this->type}";
    }
}
```

клас KiaomiFactory:

```php
<?php

require_once __DIR__ . '/../models/Device.php';
require_once 'DeviceFactory.php';

class KiaomiFactory extends DeviceFactory {
    public function __construct() {
        parent::__construct('Kiaomi');
    }
}
```

клас IProneFactory:

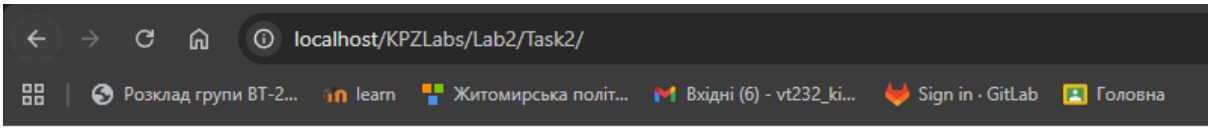
```php
<?php

require_once __DIR__ . '/../models/Device.php';
require_once 'DeviceFactory.php';

class IProneFactory extends DeviceFactory {
    public function __construct() {
        parent::__construct('IProne');
    }
}
```

клас DeviceFactory:

```php
<?php

abstract class DeviceFactory {
    protected string $brand;

    public function __construct(string $brand) {
        $this->brand = $brand;
    }

    public function createDevice(string $type): Device {
        return new Device($this->brand, $type);
    }
}
```

клас BalaxyFactory:

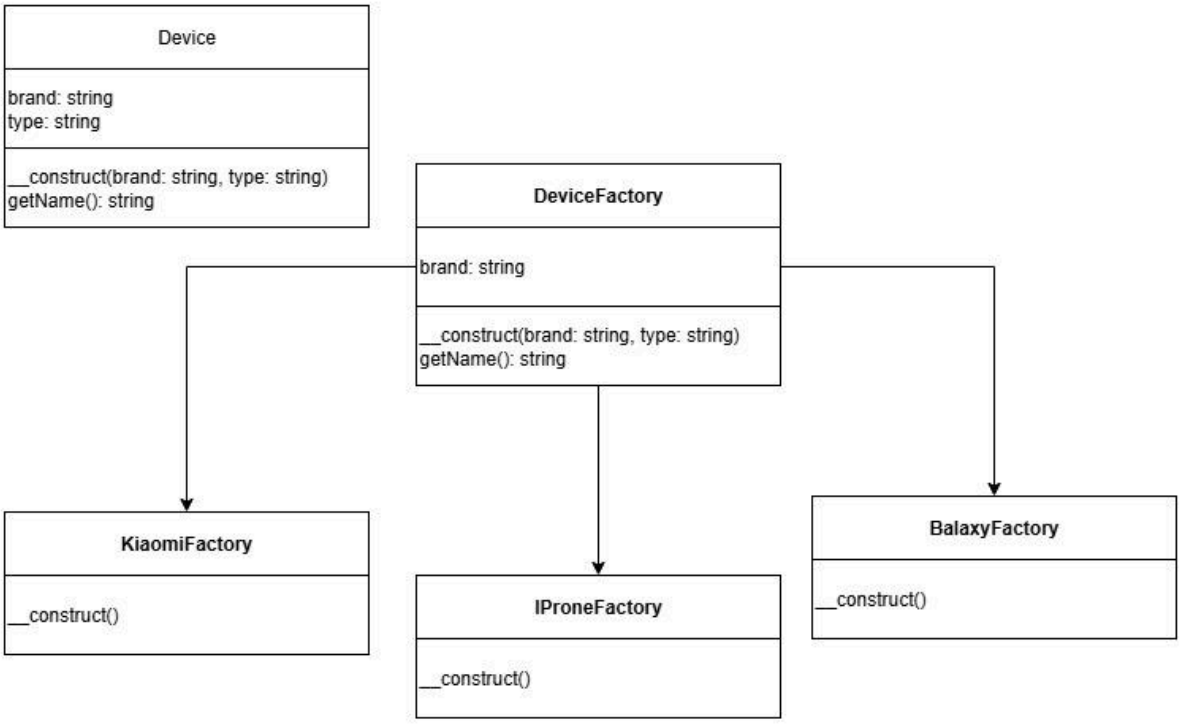
```php
<?php

require_once __DIR__ . '/../models/Device.php';
require_once 'DeviceFactory.php';

class BalaxyFactory extends DeviceFactory {
    public function __construct() {
        parent::__construct('Balaxy');
    }
}
```



IProne Laptop
IProne Netbook
IProne EBook
IProne Smartphone

Kiaomi Laptop
Kiaomi Netbook
Kiaomi EBook
Kiaomi Smartphone

Balaxy Laptop
Balaxy Netbook
Balaxy EBook
Balaxy Smartphone



Завдання 3: Одинак.

1. Створіть клас ***Authenticator*** таким чином, щоб бути впевненим, що цей клас може створити лише один екземпляр, незалежно від кількості потоків і класів, що його наслідують.
2. Покажіть правильність роботи свого коду запустивши його в головному методі програми.

Результат виконання:

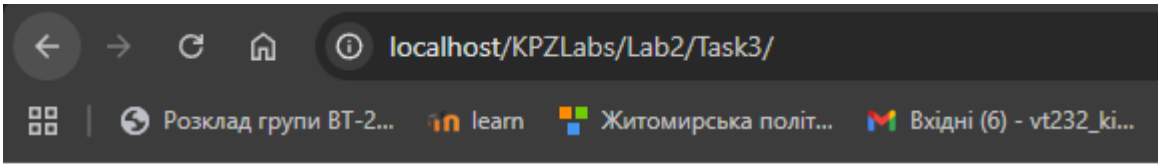
index.php:

```php
<?php

require_once 'Authenticator.php';

$auth1 = Authenticator::getInstance();
$auth2 = Authenticator::getInstance();

if ($auth1 === $auth2) {
    echo "Same instance\n";
} else {
    echo "Different instances\n";
}

if ($auth1->authenticate("admin", "1234")) {
    echo "Authentication successful\n";
} else {
    echo "Authentication failed\n";
}
```

клас Authenticator:

```php
<?php

final class Authenticator
{
    private static ?Authenticator $instance = null;

    private function __construct() {}

    private function __clone() {}

    public function __wakeup() {}

    public static function getInstance(): Authenticator
    {
        if (self::$instance === null) {
            self::$instance = new Authenticator();
        }
        return self::$instance;
    }
```

```
    public function authenticate(string $user, string $password): bool
    {
        return ($user === "admin" && $password === "1234");
    }
}
```



Same instance Authentication successful

Завдання 4: Прототип.

1. Створіть клас ***Virus***. Він повинен містити вагу, вік, ім'я, вид і масив дітей, екземплярів ***Virus***.
2. Створіть екземпляри для цілого "сімейства" вірусів (мінімум три покоління).
3. За допомогою шаблону Прототип реалізуйте можливість клонування наявних вірусів.
4. При клонуванні віруса-батька повинні клонуватися всі його діти.
5. Покажіть правильність роботи свого коду запустивши його в головному методі програми.

Результат виконання:

index.php:

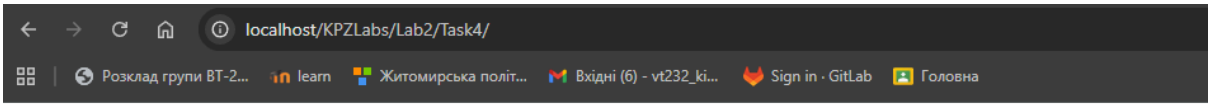
```
<?php

require_once 'Virus.php';

$rootVirus = new Virus(10.5, 5, "RootVirus", "Alpha");

$child1 = new Virus(5.2, 3, "ChildVirus1", "Beta");
$child2 = new Virus(4.8, 2, "ChildVirus2", "Gamma");

$grandChild1 = new Virus(2.1, 1, "GrandChildVirus1", "Delta");
$grandChild2 = new Virus(1.9, 1, "GrandChildVirus2", "Epsilon");

$child1->addChild($grandChild1);
$child2->addChild($grandChild2);
```

```php
$rootVirus->addChild($child1);
$rootVirus->addChild($child2);

echo "Original virus family:\n";
$rootVirus->print();

$clonedVirus = clone $rootVirus;

echo "\nCloned virus family:\n";
$clonedVirus->print();
```

клас Virus:

```php
<?php

class Virus
{
   public float $weight;
   public int $age;
   public string $name;
   public string $type;
   /** @var Virus[] */
   public array $children = [];

   public function __construct(float $weight, int $age, string $name, string
$type)
   {
       $this->weight = $weight;
       $this->age = $age;
       $this->name = $name;
       $this->type = $type;
       $this->children = [];
   }

   public function addChild(Virus $child): void
   {
       $this->children[] = $child;
   }

   public function __clone()
   {
       $clonedChildren = [];
       foreach ($this->children as $child) {
           $clonedChildren[] = clone $child;
       }
       $this->children = $clonedChildren;
   }

   public function print(int $level = 0): void
   {
       echo str_repeat("  ", $level) . "- Virus: {$this->name}, Type:
{$this->type}, Age: {$this->age}, Weight: {$this->weight}<br>";
```

```
        foreach ($this->children as $child) {
            $child->print($level + 1);
        }
    }
}
```



Original virus family: - Virus: RootVirus, Type: Alpha, Age: 5, Weight: 10.5
- Virus: ChildVirus1, Type: Beta, Age: 3, Weight: 5.2
- Virus: GrandChildVirus1, Type: Delta, Age: 1, Weight: 2.1
- Virus: ChildVirus2, Type: Gamma, Age: 2, Weight: 4.8
- Virus: GrandChildVirus2, Type: Epsilon, Age: 1, Weight: 1.9
Cloned virus family: - Virus: RootVirus, Type: Alpha, Age: 5, Weight: 10.5
- Virus: ChildVirus1, Type: Beta, Age: 3, Weight: 5.2
- Virus: GrandChildVirus1, Type: Delta, Age: 1, Weight: 2.1
- Virus: ChildVirus2, Type: Gamma, Age: 2, Weight: 4.8
- Virus: GrandChildVirus2, Type: Epsilon, Age: 1, Weight: 1.9

Завдання 5: Будівельник.

1. Створіть клас ***HeroBuilder***, який буде створювати персонажа гри, поступово додаючи до нього різні ознаки, наприклад зріст, статуру, колір волосся, очей, одяг, інвентар тощо (можете включити фантазію).
2. Створіть клас ***EnemyBuilder***, який буде реалізовувати єдиний інтерфейс з ***HeroBuilder***. Відмінністю між ними можуть бути спеціальні методи для творення добра або зла, а також списки добрих і злих справ відповідно.
3. За допомогою свого білдера і класу-директора створіть героя (або героїню) своєї мрії 🙂, а також свого найзапеклішого ворога.
4. Зверніть увагу, що Ваші білдери повинні реалізовувати текучий интерфейс (fluent interface).

5. Покажіть правильність роботи свого коду запустивши його в головному методі програми.
6. Підготуйте діаграму створених у програмі класів та інтерфейсів за допомогою https://app.diagrams.net/, експортуйте та завантажте її до репозиторія.

Результат виконання:

index.php:

```php
<?php

require_once 'builders/BuilderInterface.php';
require_once 'builders/HeroBuilder.php';
require_once 'builders/EnemyBuilder.php';
require_once 'Character.php';
require_once 'Director.php';

$director = new Director();

$heroBuilder = new HeroBuilder();
$hero = $director->buildHero($heroBuilder);
$hero->describe();

$enemyBuilder = new EnemyBuilder();
$enemy = $director->buildEnemy($enemyBuilder);
$enemy->describe();
```

клас Director:

```php
<?php

class Director
{
   public function buildHero(BuilderInterface $builder): Character
   {
       return $builder
           ->setHeight(180)
           ->setBuild("Athletic")
           ->setHairColor("Blond")
           ->setEyeColor("Blue")
           ->setClothes("Armor")
           ->addInventory("Sword")
           ->addInventory("Shield")
           ->addAction("Save villagers")
           ->addAction("Protect kingdom")
           ->getCharacter();
   }

   public function buildEnemy(BuilderInterface $builder): Character
   {
       return $builder
           ->setHeight(190)
           ->setBuild("Muscular")
           ->setHairColor("Black")
           ->setEyeColor("Red")
           ->setClothes("Dark Robe")
           ->addInventory("Axe")
           ->addInventory("Poison")
```

```
            ->addAction("Steal treasures")
            ->addAction("Destroy villages")
            ->getCharacter();
    }
}
```

клас Character:

```
<?php

class Character
{
    public ?int $height = null;
    public ?string $build = null;
    public ?string $hairColor = null;
    public ?string $eyeColor = null;
    public ?string $clothes = null;
    public array $inventory = [];
    public string $type; // "Hero" або "Enemy"
    public array $actions = [];

    public function __construct(string $type)
    {
        $this->type = $type;
    }

    public function describe(): void
    {
        echo "{$this->type} Character:<br>";
        echo "Height: {$this->height}<br>";
        echo "Build: {$this->build}<br>";
        echo "Hair Color: {$this->hairColor}<br>";
        echo "Eye Color: {$this->eyeColor}<br>";
        echo "Clothes: {$this->clothes}<br>";
        echo "Inventory: " . implode(", ", $this->inventory) . "<br>";
        echo ($this->type === 'Hero' ? "Good deeds: " : "Evil deeds: ") .
implode(", ", $this->actions) . "<br><br>";
    }
}
```

інтерфейс BuilderInterface:

```php
<?php

interface BuilderInterface
{
   public function setHeight(int $height): self;
   public function setBuild(string $build): self;
   public function setHairColor(string $color): self;
   public function setEyeColor(string $color): self;
   public function setClothes(string $clothes): self;
   public function addInventory(string $item): self;
   public function addAction(string $action): self;
   public function getCharacter(): Character;
}
```

клас EnemyBuilder:

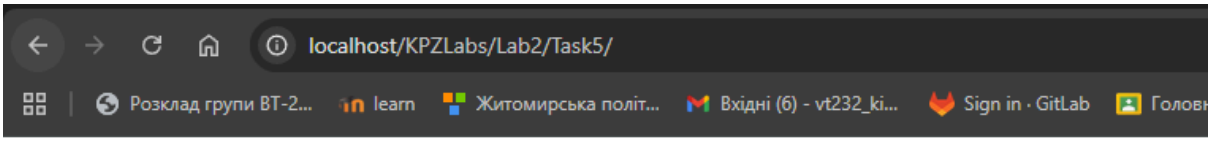
```php
<?php
require_once __DIR__ . '/../Character.php';
require_once 'BuilderInterface.php';

class EnemyBuilder implements BuilderInterface
{
   private Character $character;

   public function __construct()
   {
       $this->character = new Character("Enemy");
   }
   public function setHeight(int $height): self
   {
       $this->character->height = $height;
       return $this;
   }
   public function setBuild(string $build): self
   {
       $this->character->build = $build;
       return $this;
   }
   public function setHairColor(string $color): self
   {
       $this->character->hairColor = $color;
       return $this;
   }
   public function setEyeColor(string $color): self
   {
       $this->character->eyeColor = $color;
       return $this;
   }
   public function setClothes(string $clothes): self
   {
       $this->character->clothes = $clothes;
       return $this;
```

```
    }
    public function addInventory(string $item): self
    {
        $this->character->inventory[] = $item;
        return $this;
    }
    public function addAction(string $action): self
    {
        $this->character->actions[] = $action;
        return $this;
    }
    public function getCharacter(): Character
    {
        return $this->character;
    }
}
```

клас HeroBuilder:

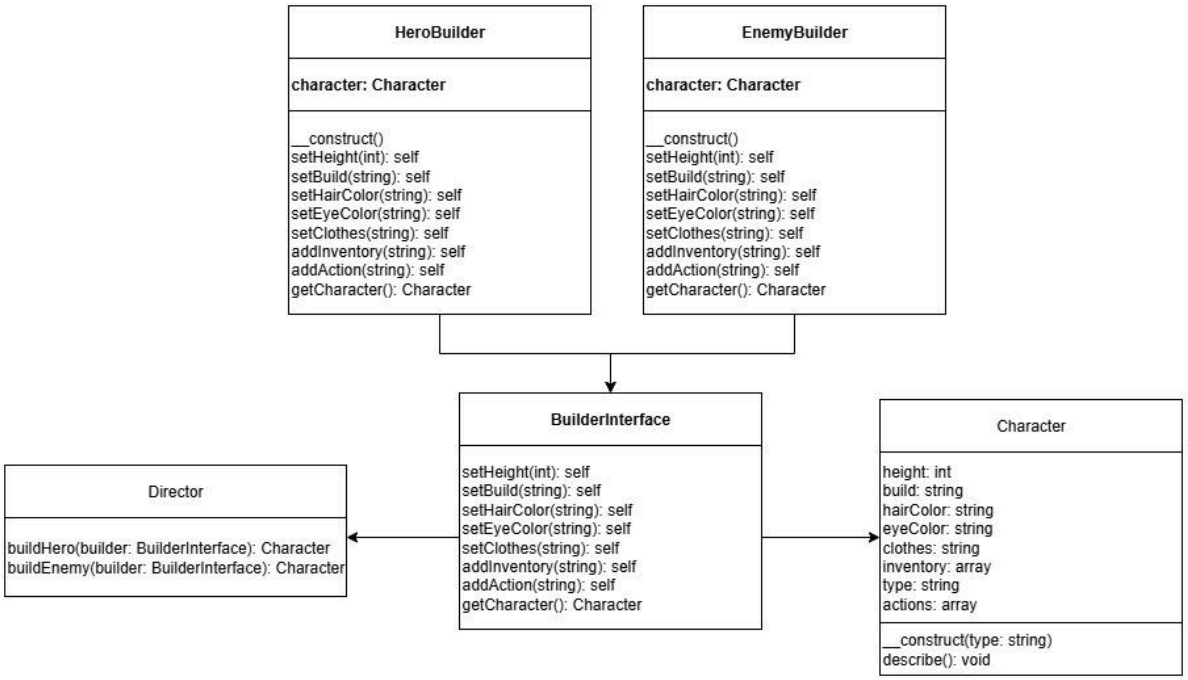
```
<?php
require_once __DIR__ . '/../Character.php';
require_once 'BuilderInterface.php';

class HeroBuilder implements BuilderInterface
{
    private Character $character;

    public function __construct()
    {
        $this->character = new Character("Hero");
    }
    public function setHeight(int $height): self
    {
        $this->character->height = $height;
        return $this;
    }
    public function setBuild(string $build): self
    {
        $this->character->build = $build;
        return $this;
    }
    public function setHairColor(string $color): self
    {
        $this->character->hairColor = $color;
        return $this;
    }
    public function setEyeColor(string $color): self
    {
        $this->character->eyeColor = $color;
        return $this;
    }
    public function setClothes(string $clothes): self
    {
```

```
        $this->character->clothes = $clothes;
        return $this;
    }
    public function addInventory(string $item): self
    {
        $this->character->inventory[] = $item;
        return $this;
    }
    public function addAction(string $action): self
    {
        $this->character->actions[] = $action;
        return $this;
    }
    public function getCharacter(): Character
    {
        return $this->character;
    }
}
```



Hero Character:
Height: 180
Build: Athletic
Hair Color: Blond
Eye Color: Blue
Clothes: Armor
Inventory: Sword, Shield
Good deeds: Save villagers, Protect kingdom

Enemy Character:
Height: 190
Build: Muscular
Hair Color: Black
Eye Color: Red
Clothes: Dark Robe
Inventory: Axe, Poison
Evil deeds: Steal treasures, Destroy villages



Посилання на репозиторій: https://github.com/KuzminIvan232/KPZLabs