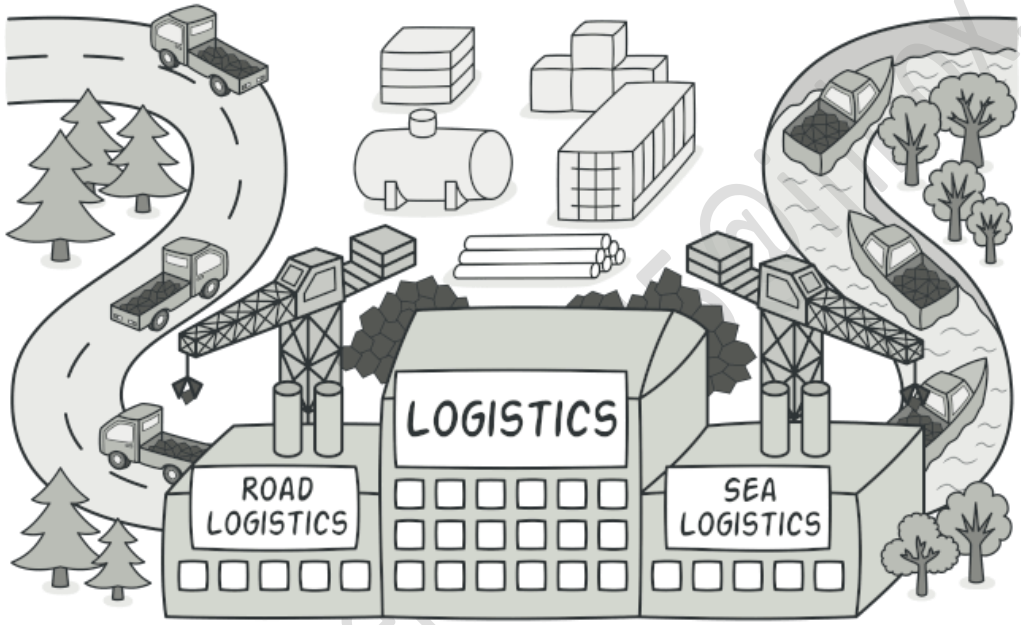
Factory Method

Boshqacha nomi: **Virtual Constructor**

Maqsadi

Factory method – bu creational design pattern guruhiga kiruvchi pattern bo'lib, u asosiy ota klass asosida obyektlarni yaratish uchun interfeyslarni e'lon qilishdan iborat. Ammo, bunda bola klasslarga yaratilayotgan obyektlarning tipini o'zgartirishga imkon beradi.



☹ Muammo:

Faraz qiling siz logistikani boshqaruvchi ilova yaratyapsiz. Ilovangizning birinchi versiyasida faqat yuk mashinalari bilan tashishni boshqarish mumkin. Bu esa asosiy katta kodlaringiz *Truck* klassida joylashadi degani.

Biroz vaqtdan keyin, ilovangiz ancha ommalashib ketadi. Har kuni sizga ko'plab yuk tashuvchi kompaniyalardan ilovangizga qo'shilish uchun so'rovlar kela boshlaydi.

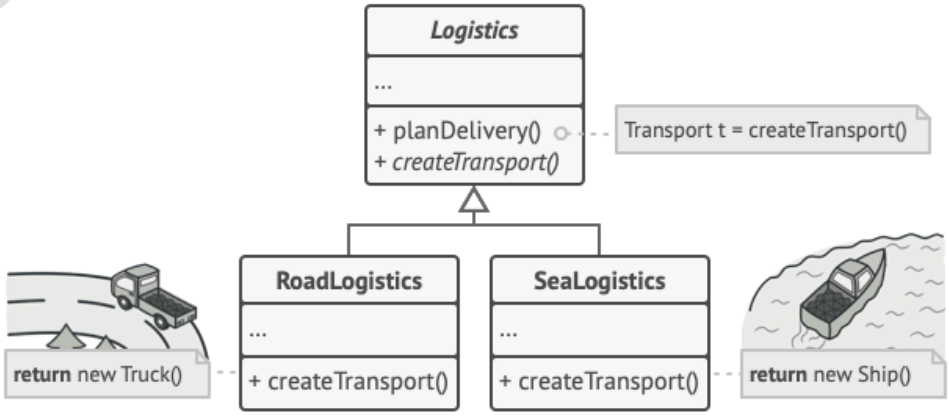
Agar kodning boshqa qismlari asosiy klass bilan kuchli bog'lanib ketgan bo'lsa, yangi klassni qo'shish unchalik ham oson bo'lmaydi

Ajoyib yangilik, to'g'rimi? Ammo kod haqida nima deyishimiz mumkin? Hozirda, kodingizning ko'p qismi *Truck* klassi bilan bog'langan. Ilovaga *Ships* klassini qo'shish butun ilova kodini o'zgartirishga olib kelishi mumkin. Shu bilan birga, keyinchalik siz ilovangizga boshqa turdagi transportni qo'shmoqchi bo'lsangiz, boyagi o'zgarishlarni yana qaytadan qilib chiqish kerak bo'ladi.

Natijada, siz tashish obyektlarining sinfiga qarab, ilovaning ishlashini o'zgartiradigan shartlarga ega bo'lgan juda yoqimsiz va aralashib ketgan kod bilan ishni yakunlaysiz.

Yechim

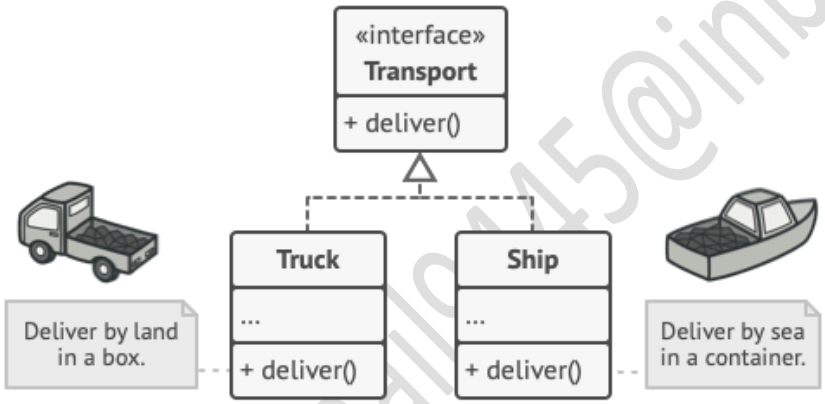
Factory method patternida siz to'g'ridan-to'g'ri obyekt kostruktorini chaqirish(new operatori orqali) o'rniga maxsus factory metod orqali chaqirishni bajarishingiz kerak bo'ladi. Xavotir olishga hojat yo'q: obyektlar hali ham new operatori bilan yaratiladi, ammo bunday yaratish factory method ichida amalga oshiriladi. Factory methoddan chiquvchi obyektlar ko'pincha "mahsulotlar" deb nomlanadi.



Bola klasslar obyekt klasslarini factory method orqali qaytaradi

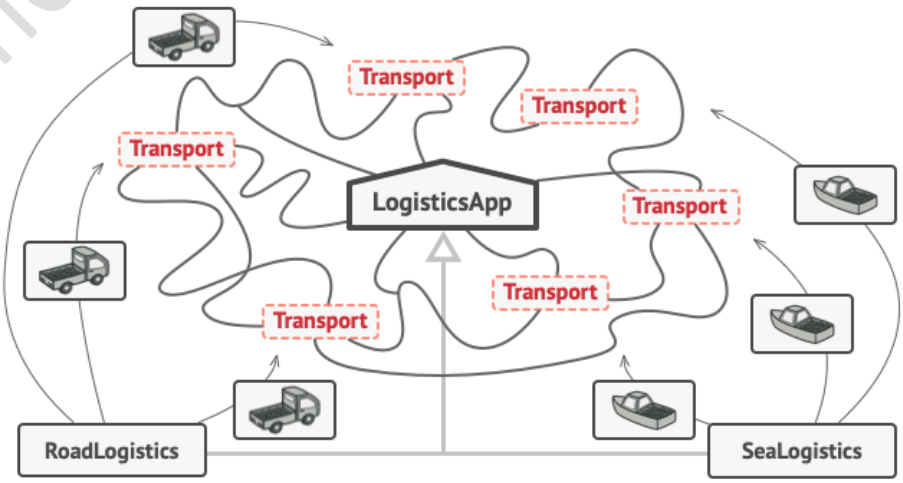
Bir qarashda, bunday o'zgartirish ma'nosizday ko'rinadi: biz konstruktorni chaqirishni bitta kod ichidan boshqasining ichiga ko'chirgandekmiz. Biroq, quyidagini ham hisobga olishimiz kerak: endi siz bola klassdagi factory methodni qaytadan e'lon qilib, method tomonidan yaratilayotgan mahsulotlarning klassini o'zgartirishingiz mumkin.

Ammo, biroz cheklov ham mavjud: agar bu mahsulotlar umumiy asos klass yoki interfeysga ega bo'lsagina bola klasslar har xil turdagi mahsulotlarni berishi mumkin. Shu bilan birga, asosiy klassdagi factory method interfeysida e'lon qilinganidek o'zining qaytarish tipiga ega bo'lishi kerak.



Barcha mahsulotlar bir xil interfeysga bo'ysunishi shart

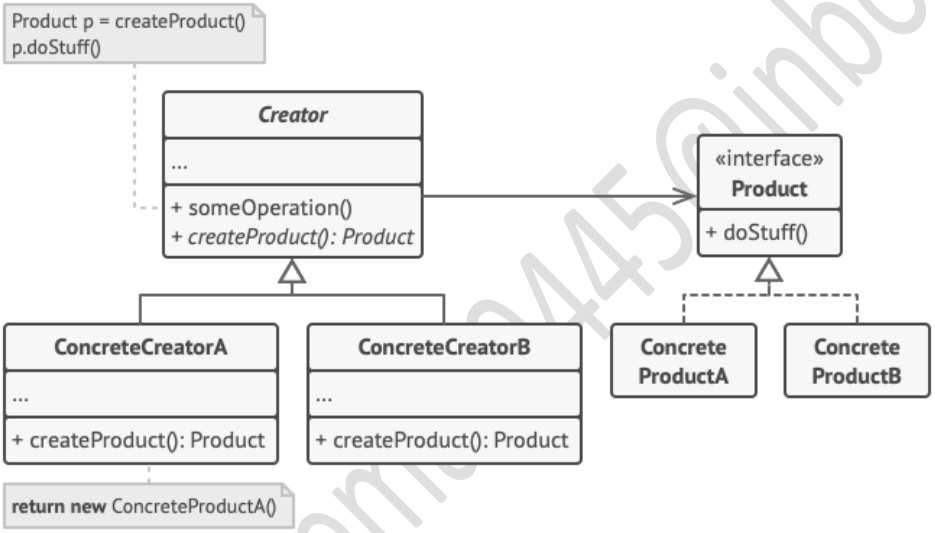
Masalan, ikkala *Truck* va *Ship* klasslari *deliver* deb nomlangan metodni e'lon qiladigan *Transport* interfeysini ishlatishi kerak. Har bitta klass bu interfeysni turlicha ishlatadi: yuk mashinalari yukni quruqlik orqali yetkazadi, kemalar esa dengiz orqali. *RoadLogistics* klassi yuk mashinasi obyektini qaytarsa, *SeaLogistics* klassi kema obyektini qaytaradi.



Barcha mahsulot klasslari umumiy interfeysni ishlatsa, ularning obyektlarini mijozning kodiga hech qanday yomon ta'sirlarsiz berish mumkin bo'ladi.

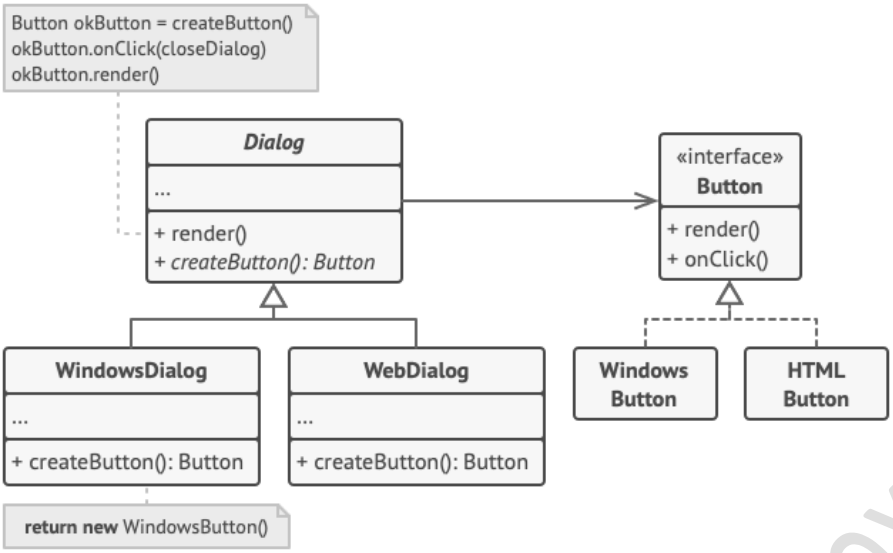
Factory methodni ishlatadigan kod (ko'pincha ***mijoz kodi*** deb nomlanadi) turli xildagi bola klasslardan keluvchi mahsulotlarda farqni sezmaydi. Mijoz barcha mahsulotlarga ***Transport*** abstrakti sifatida qaraydi va barcha transport obyektlarida deliver metodi borligini biladi. Ammo bu metodlarning qanday ishlashi uni qiziqtirmaydi.

Tuzilishi



1. Barcha obyektlar uchun umumiy bo'lgan Product interfeysi e'lon qilinadi.
2. Concrete products – bu mahsulot(Product) interfeysining turli ko'rinishda ishlatilishi.
3. Creator klassida yangi mahsulot obyektlarini qaytaruvchi factory methodi e'lon qilinadi. Bu methodning qaytariluvchi tipi mahsulot interfeysi bilan solishtirilishi muhim. Bola klasslar metodlarni majburiy e'lon qilishi uchun factory method ***abstract*** qilib e'lon qilinadi. Bunga alternativa sifatida, asos factory methodi odatiy (default) mahsulot turini qaytarishi ham mumkin.
4. Concrete Creatorlar asos factory methodlarni qaytadan yozadi, shu sababli ham ular turli xildagi mahsulot turlarini qaytaradi. **Yodda tuting**: factory method doim ham new operatori yordamida obyekt yaratib uni qaytarmaydi, u, shuningdek, keshda, obyektlar havzasida (object pool) yoki boshqa biror joyda mavjud bo'lgan obyektni olib qaytarishi ham mumkin.

 # Psevdokod



Kros-platforma dialogga misol

Asosiy dialog klassi o'zining oynasida turli xildagi UI elementlarini ishlatadi. Turli xildagi operatsion tizimlarda, bu elementlar turlicha ko'rinishi mumkin, ammo ular bir xil ishni bajaraveradi. Ya'ni Windowsdagi tugma Linuxda ham tugma bo'lib qolaveradi.

Factory method ishlatilganda, siz turli operatsion tizimlar uchun dialoglarni o'zgartirib yurmaysiz. Agar tugma yaratuvchi factory method asosiy dialog klassda e'lon qilinsa, keyinchalik windows stiliga mos keladigan tugmani yaratuvchi factory methodni qaytaradigan yangi dialog bola klassni yaratishimiz mumkin.

Qo'llanilishi

- Agar kodingiz ishlashi kerak bo'lgan klasslarning aniq turlari va qo'shimchalarini oldindan bilmasangiz Factory methoddan foydalaning
    - Factory method mahsulot konstruksiya kodini mahsulotni ishlatayotgan koddan ajratadi. Shu sababli, mahsulot konstruksiya kodini qolgan koddan mustaqil ravishda kengaytirish mumkin bo'ladi.
    - Masalan, ilovaga yangi turdagi mahsulotni qo'shishda, yangi yaratuvchi bola klassni yaratish va undagi factory metodni qaytadan o'zgartirib yozish yetarli.
- Factory methoddan sizning kutubxonangiz yoki freymvorkingizni boshqa foydalanuvchilarga kengaytirishga imkoniyat yaratishda ishlatish mumkin.
    - Freymvork yoki kutubxonaga o'zgartirish kiritishda eng oson yo'l – bu merosxo'rlik. Ammo, freymvork qanday qilib standart component o'rniga qo'shilgan bola klassni ishkatish kerakligini biladi.
    - Yechim quyidagicha: butun freymvorkdagi barcha komponentlarni quradigan kodlarni bitta factory methodga kiritish va komponentni

kengaytirish uchun hammaga bu methodni qayta o’zgartirib yozishga ruxsat berish kerak.

- Endi bu qanday ishlashini ko’raylik. Faraz qiling, siz ochiq kodli UI freymvorkdan foydalanib ilova yozyapsiz. Sizning ilovangizda aylana tugmalar bo’lishi kerak, ammo bu freymvorkda faqat to’rtburchak tugmalar mavjud. Bunda siz standart Button klassni kengaytirib RoundButton bola klassni olasiz. Ammo, siz freymvorkning UIFramework klassiga Button klass o’rniga o’zingiz yaratgan yangi RoundButton klassini ishlatishini aytishingiz kerak bo’ladi.
- Bunday qilish uchun esa asosiy freymvork klassidan meros olib UIWithRoundButton bola klassini yaratamiz va uning creatButton methodini qaytadan o’zgartirib yozamiz. Bunda sizning o’zgartirilgan createButton methodi RoundButton obyektini qaytaradigan qilinadi. Endi UIFramework klassi o’rniga UIWithRoundButtons ni ishlatsa bo’ldi.

- Factory methoddan obyektlarni har safar qaytadan yaratmasdan mavjudidan foydalanib tizim resurslarini tejamoqchi bo’linganda foydalaniladi.
  - Bunday holatga tarmoq manbaalari, fayl tizimlari, ma’lumotlar omboriga ulanish kabi katta va ko’p resurs talab qiladigan obyektlar bilan ishlaganda duch kelasiz.
  - Mavjud obyektlarni qaytadan foydalanish uchun nima qilish kerakligi haqida o’ylab ko’raylik
    - Birinchidan, yaratilgan obyektlarni saqlab turish uchun joy yaratish kerak.
    - Kimdir obyektni ishlatish uchun so’rov yuborganda, ilova obyektlar saqlanuvchi havzadan bo’sh turgan obyektni qidirish kerak.
    - Keyin esa topilgan obektni mijozga qaytarishi kerak.
    - Agar bo’sh obyekt bo’lmasa, ilova yangisini yaratib berishi kerak.
  - Shu bilan tamom! Endi siz ortiqcha takrorlangan kod bilan hamma yoqni “rasvo” qilmaysiz.
  - Mavjud obyektlarni qaytadan ishlatuvchi kod yoziladigan joy, albatta, constructorning ichi hisoblanadi. Biroq, konstructor doimo yangi obyekt yaratib beradi.
  - Shu sababli, sizda mavjud obyektni qaytaradigan, mavjud bo’lmasa yangisini yaratadigan maxsus method bo’lishi kerak. Bu method albatta factory metodi bo’ladi.

Qanday ishlatiladi

1. Barcha mahsulotlarni bitta interfeysga biriktiring. Bu interfeys hamma mahsulot uchun umumiy bo'lgan method e'lon qiladi.
2. Yaratuvchi klass ichida bo'sh factory method qo'shing. Bu methodning qaytaruvchi tipi asosiy mahsulot interfeysiga moslanadi.
3. Yaratuvchi klass kodida mahsulot construktorilariga qilingan barcha murojaatlarni toping. Ularni bitta-bittalab factory method chaqirilishi bilan o'zgartirib chiqing, mahsulot yaratishni esa factory method ichiga o'tkazib chiqing.

   Bunda siz factory methodga qaytariluvchi mahsulot turini boshqarish uchun vaqtinchalik parameter qo'shishingiz kerak bo'lishi mumkin.

   Shu nuqtada, factory methodning kodi juda xunuk ko'rinishi mumkin. Bu mahsulot turini tanlaydigan switch operatorining katta hajmli kodi sababli bo'ladi. Ammo, xavotir olmang, tezda buni to'g'rilaymiz.
4. Endi, factory methodda kelgan har bir mahsulot turi uchun yaratuvchi bola klasslar to'plamini yaratamiz. Bola klasslardagi factory methodlarni qayta o'zgartirib yozing va asosiy klassdagi mos konstruktorlarni chiqarib oling.
5. Agar juda ko'p mahsulot turlari mavjud bo'lsa va bu mahsulotlarning barchasi uchun bola klasslarni yaratishga ta'sir qilmasa, u holda asosiy klassdagi boshqaruvchi parametrini bola klasslarda qayta foydalanishingiz mumkin.

   Masalan, tasavvur qiling, sizda quyidagi klasslar iyerarxiyasi bor bo'lsin: Mail klassi AirMail va GroundMail bola klasslari bilan o'zaro bog'langan, Transport klassi Plane, Truck va Train klasslari bilan bog'langan. Bunda AirMail klassi Plane obyektlari bilan ishlasa, GroundMail Truck va Train obyektlari bilan ishlaydi. Ikkala holat bilan ishlash uchun siz yangi bola klass(aytaylik TrainMail) yaratishingiz mumkin, ammo boshqa yo'l ham mavjud. Buning uchun mijoz kodi GroundMail klassining factory methodiga qanday mahsulot qabul qilishni xohlayotganini bildiruvchi argument jo'natishi kerak bo'ladi.
6. Agar, barcha kodlar asosiy klassdan chiqarib olinganidan keyin, asosiy factory method bo'shab qolsa, siz uni abstract qilib e'lon qilishingiz mumkin. Agar biror narsa qolgan bo'lsa, uni methodning odatiy holati qilib qo'yasiz.

Afzallik va kamchiliklari

Afzalliklari	Kamchiliklari
Yaratuvchi va mahsulot yaratuvchi(concrete product) alohida klasslar orasida kuchli bog'lanish(tight coupling)ni chetlab o'tishingiz mumkin.	Kodingiz juda murakkab bo'lib ketishi mumkin, chunki patternni ishlatish uchun yangi bola klasslarni e'lon qilishingiz kerak bo'ladi. Yaratuvchi klassning oldindan mavjud iyerarxiyasiga patternni

Yagona javobgarlik tamoyili. Mahsulot yaratish kodini ilovaning biror joyiga joylashtirish orqali kodni kengaytirishni osonlashtirish mumkin.

Ochiq/Yopiqlik tamoyili. Mijozning kodini buzmasdan turib yangi turdagi mahsulotlarni ilovaga qo'shishingiz mumkin.

qo'llaganingizda eng yaxshi holatga erishish mumkin.

Boshqa patternlar bilan bog'liqligi

- Ko'plab dizaynlar Factory Method(unchalik ham murakkab emas va bola klasslar yordamida o'zgartirish mumkin)ni qo'llash orqali boshlanadi va Abstract Factory, Prototype yoki Builder(ancha moslashuvchan, lekin biroz murakkabroq) patternlariga qarab rivojlanib boradi.
- Abstract Fatory klasslari ko'pincha Factory Methodlar to'plamiga asoslanadi, ammo bu klasslarda methodlarni yaratishda Prototype dan ham foydalansa bo'ladi.
- Bola klasslar to'plami to'plamga mos bo'lgan turli tipdagi iteratorlarni qaytarishiga imkon berish uchun Factory Methodni Iterator bilan birgalikda ishlatish mumkin.
- Prototype merosxo'rlikka asoslanmagan, shu sababli ham unda kamchiliklar yo'q. Boshqa tomondan esa, Prototype klon olingan obyektning murakkab initsializatsiyasini talab qiladi. Factory Method merosxo'rlikka asoslangan bo'lsa ham initsializatsiyani talab qilmaydi.
- Factory Method Template Methodning xususiy holi. Shu bilan bir birga, Factory Method katta Template Methodda qadam sifatida xizmat qilishi mumkin.

Kod misol

```php
<?php
abstract class Creator
{
    abstract public function factoryMethod(): Product;

    public function someOperation(): string
    {
        $product = $this->factoryMethod();
        $result = "Creator: The same creator's code has just worked with " .
            $product->operation();
        return $result;
    }
}
```

```php
class ConcreteCreator1 extends Creator
{
    public function factoryMethod(): Product
    {
        return new ConcreteProduct1;
    }
}

class ConcreteCreator2 extends Creator
{
    public function factoryMethod(): Product
    {
        return new ConcreteProduct2;
    }
}

interface Product
{
    public function operation(): string ;
}

class ConcreteProduct1 implements Product
{
    public function operation(): string
    {
        return "{Result of the ConcreteProduct1}";
    }
}

class ConcreteProduct2 implements Product
{
    public function operation(): string
    {
        return "{Result of the ConcreteProduct2}";
    }
}

function clientCode(Creator $creator) {
    echo "Client: I'm not aware of the creator's class, but it still works.\n"
    . $creator->someOperation();
}

echo "App: Launched with the ConcreteCreator1.\n";
echo clientCode(new ConcreteCreator1);
echo PHP_EOL . PHP_EOL;

echo "App: Launched with the ConcreteCreator2.\n";
echo clientCode(new ConcreteCreator2);
```

Natija:

App: Launched with the ConcreteCreator1.

Client: I'm not aware of the creator's class, but it still works.

Creator: The same creator's code has just worked with {Result of the ConcreteProduct1}

App: Launched with the ConcreteCreator2.

Client: I'm not aware of the creator's class, but it still works.

Creator: The same creator's code has just worked with {Result of the ConcreteProduct2}