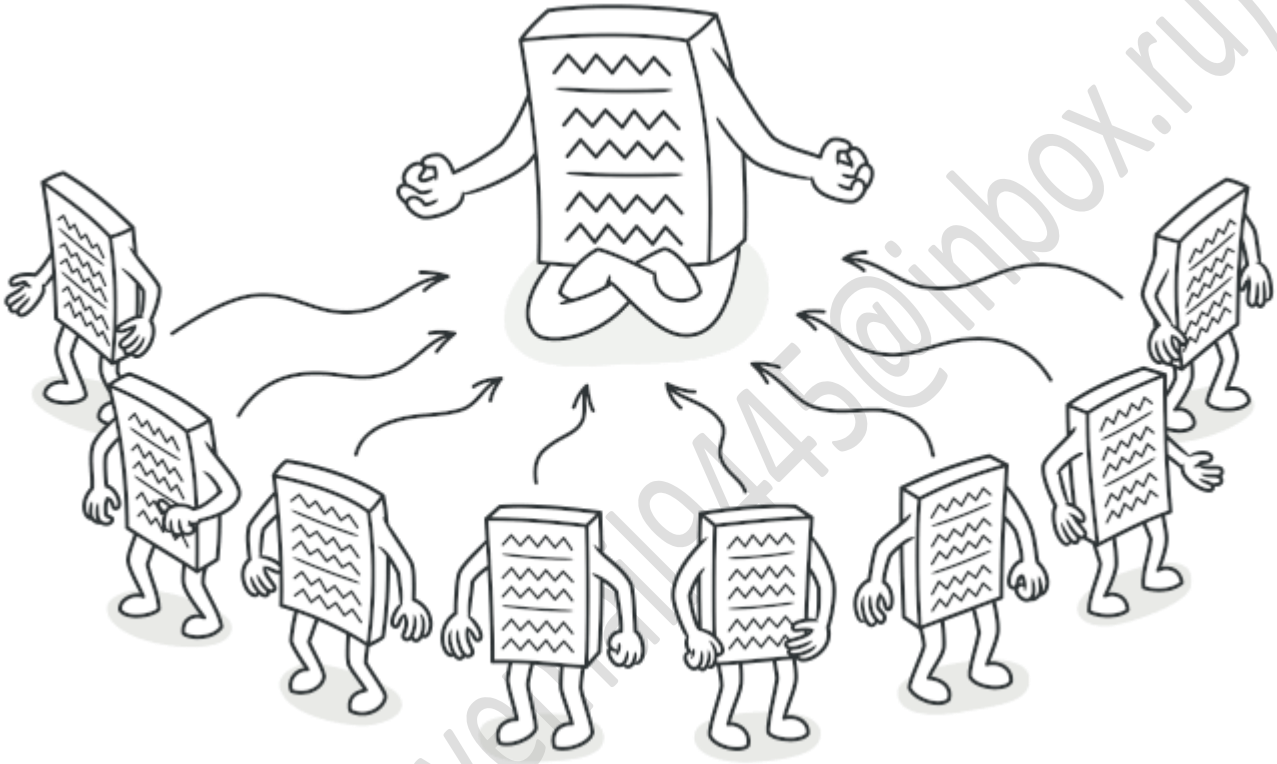
# Singleton

### Maqsadi

Singleton creational design pattern guruhiga mansub bo’lgan patern. U biror bir klassdan faqatgina bitta obyekt olish imkonini beradi.



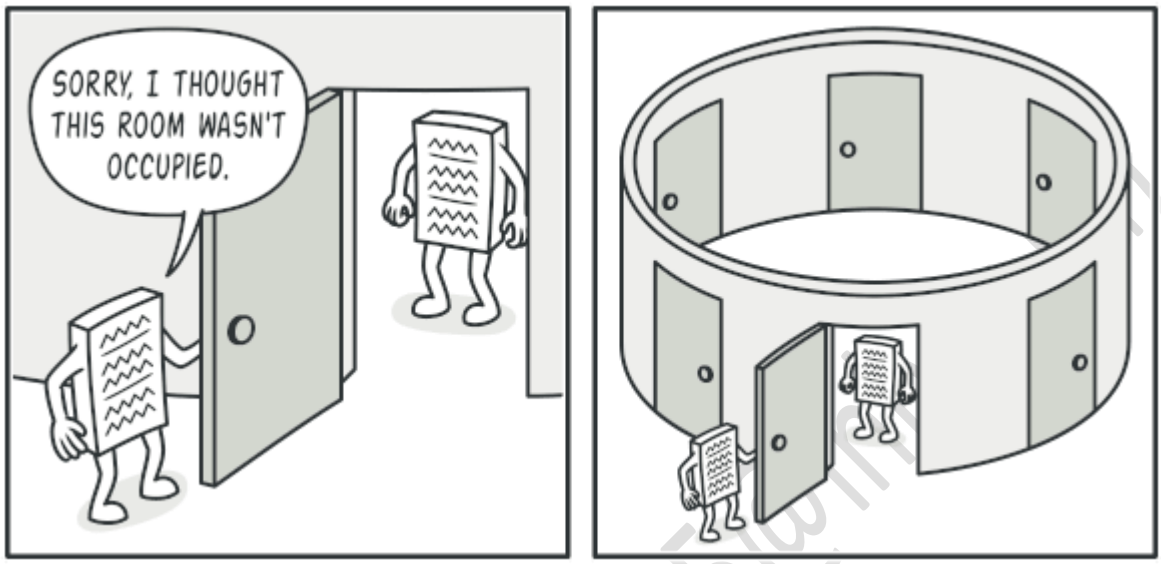
### Muammo

Singleton paterni Yagona Javobgarlik Tamoyilidan chetga chiqqan holda ikkita muammoni yechadi:

1. Klassning yagona obyekti bo’lishini kafolatlaydi. Nega biror klassdan olingan obyektlar sonini boshqarish kerak o’zi? Bunga eng asosiy sabab ma’lumotlar bazasi yoki fayl kabi umumiy ishlatiluvchi resurslarga murojaat qilishni boshqarishdir.

   Bu quyidagicha ishlaydi: faraz qiling siz obyekt yaratdingiz, ammo bir ozdan keyin yana bitta yangi shunday klass yaratmoqchi bo’ldingiz. Lekin, siz yangi obyekt yaratmasdan, oldin yaratib qo’ygan obyektingizdan foydalansangiz bo’ladi.

Shuni yodda tutingki, yangi obyekt yaratmasdan eskisini ishlatishni odatdagi konstruktorni ishlatib amalga oshirib bo'lmaydi, chunki konstruktor doimo klassning yangi obyektini qaytaradi.




*Mijoz doim faqat bitta obyekt bilan ishlayotganini tasavvur ham qilmaydi*

2. Yaratilgan obyektga global holda murojaat qilishni ta'minlab beradi. Global o'zgaruvchilar ishlatishga juda qulay, shu bilan birga, ular juda xavfli hisoblanadi, chunki har qanday kod bunday global o'zgaruvchilarning kontentini o'zgartirib ilovani ishlamay qolishiga olib kelishi mumkin.

   Global o'zgaruvchilarga o'xshab, Singleton paterni sizga ayrim obyektlarga dasturning har qanday joyidan turib murojaat qilish imkonini beradi. Biroq, yana u shu obyektlarni boshqa kod tomonidan qayta yozilishdan ham himoya qiladi.

   Muammoning boshqa tomoni ham bor: siz 1-muammoni hal qiluvchi kodni butun dastur bo'ylab sochilib yotishini xohlamaysiz. Bu kod bitta klassning ichida bo'lishi ancha qulay, ayniqsa agar qolgan kodlar shu kodga bog'langan bo'lsa.

Hozirgi kunda, Singleton paterni shunaqangi darajada mashhur bo'lib ketganki, hattoki berilgan muammolarning bittasini hal qilsa ham, uni dasturchilar singleton deb atashadi.

### Yechim

Singletonning ishlatilishi quyidagi ikkita qadamdan iborat:

- Singleton klassdan new operatori bilan yangi obyektni olishni cheklash uchun konstruktorni private qilib e’lon qilinadi.
- Konstruktor sifatida ishlaydigan static metod yaratiladi. Bu metod private konstruktordan yangi obyekt oladi va uni static xususiyatga saqlab qo’yadi. Singletondan obyekt olishda static metod avval static xususiyatni tekshiradi, agar unda obyekt saqlangan bo’lsa shu obyektni qaytaradi, aks holda private konstruktordan yangi obyekt olib uni static xususiyatga saqlab, shu obyektni qaytaradi.

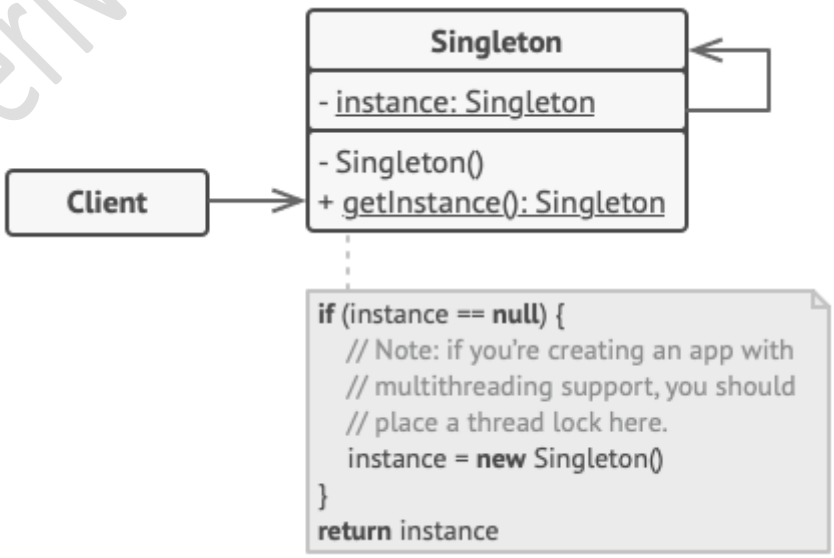
### Hayotiy misol

Hukumat Singleton klassiga ajoyib misol bo’ladi. Mamlakatda faqat bitta qonuniy hukumat bo’lishi mumkin.

### Tuzilishi

1. Singleton klass o’zining yagona obyektini qaytaruvchi getInstance deb nomlangan metodini e’lon qiladi.

   Singletonning konstrutori mijoz kodidan yashirilgan bo’lishi kerak. getInstance metodi Singleton klassning obyektini olishning yagona vositasi bo’lishi kerak.



### Psevdokod

Quyidagi misolda ma’lumotlar bazasiga bog’lanuvchi klass Singleton sifatida ishlaydi. Bu klassda public konstruktor bo’lmaydi, shuning uchun ham uning obyektini faqatgina getInstance metodi bilan olish mumkin. Bu metod birinchi yaratilgan obyektni saqlab qo’yadi va uni keyingi so’rovlarda qaytaradi.

```
// The Database class defines the `getInstance` method that lets
// clients access the same instance of a database connection
// throughout the program.
class Database is
    // The field for storing the singleton instance should be
    // declared static.
    private static field instance: Database

    // The singleton's constructor should always be private to
    // prevent direct construction calls with the `new`
    // operator.
    private constructor Database() is
        // Some initialization code, such as the actual
        // connection to a database server.
        // ...

    // The static method that controls access to the singleton
    // instance.
    public static method getInstance() is
        if (Database.instance == null) then
            acquireThreadLock() and then
                // Ensure that the instance hasn't yet been
                // initialized by another thread while this one
                // has been waiting for the lock's release.
                if (Database.instance == null) then
                    Database.instance = new Database()
        return Database.instance

    // Finally, any singleton should define some business logic
    // which can be executed on its instance.
    public method query(sql) is
        // For instance, all database queries of an app go
        // through this method. Therefore, you can place
        // throttling or caching logic here.
        // ...
```

```
class Application is
    method main() is
        Database foo = Database.getInstance()
        foo.query("SELECT ...")
        // ...
        Database bar = Database.getInstance()
        bar.query("SELECT ...")
        // The variable `bar` will contain the same object as
        // the variable `foo`.
```

### Amaliy qo’llanilishi

- Singletondan dasturingizda barcha mijozlar biror klassning faqat bitta obyektini ishlatishlari kerak bo’lsa ishlatiladi. Masalan, yagona ma’lumotlar bazasiga bog’lanish obyekti butun dastur bo’ylab ishlatiladi.
- Global o’zgaruvchilarni qat’iyroq boshqarish kerak bo’lganda Singletondan foydalaning

### Ishlatilishi

1. Singleton obyektni saqlab turish uchun klassga private static xususiyat qo’shing.
2. Singleton obyektni olish uchun public static metodni e’lon qiling.
3. Static metod ichida “lazy initialization”ni ishlating. U birinchi marta obyektni olish paytida yangi obyekt yaratadi va yaratilgan obyektni static xususiyatga saqlab qo’yadi. Keyingi ishlatilishlarda metod doim saqlab qo’yilgan obyektni qaytaradi.
4. Konstruktor metodini private qilib qo’ying. Shunda klassning static metodi konstruktorni ishlataveradi, lekin tashqaridan uni boshqa obyektlar ishlata olmaydi.
5. Mijoz kodida singleton klassning obyektini olish uchun static metodini ishlating.

### Afzallik va kamchiliklari

| Afzalliklari | Kamchiliklari |
| --- | --- |
| Klassning faqat bitta obyekti bo’lishini | Yagona Javobgarlik Tamoyilini buzadi. |

| | |
|---|---|
| ta’minlaydi | Chunki singleton paterni bir vaqtning o’zida ikkita muammoni yechadi. |
| Bu yagona obyektga global holda murojaat qilish mumkin. | Yomon holatda qilingan dizaynni yashiradi. Masalan, dastur komponentlari bir biri haqida ko’p ma’lumotga ega bo’lgan paytda. |
| Singleton obyekt faqat birinchi marta ishlatishda yaratiladi. | Patern ko'p oqimli muhitda maxsus e’tiborni talab qiladi, shunda bir nechta oqim singleton ob'ektini bir necha marta yaratmaydi<br>Mijoz kodida Singletonni unit test qilish qiyinlashishi mumkin, chunki ko’pchilik testlovchi freymvorklar mock obyektlarni yaratishda merosxo’rlikdan foydalanadi. Singletonda esa konstruktor private bo’ladi va ko’pchilik tillar static metodlarni override qilish imkonini bermaydi. Singletondan mock obyektni olishning creative yo’lini topishingiz kerak bo’ladi. Yoki, shunchaki test yozmaysiz, yoki singleton ishlatmaysiz. |

**Boshqa paternlar bilan bog’liqligi**

1. Facade paterni klassi ko’pincha Singletonga o’zgartiriladi, chunki yagona facade obyekti ko’p hollarda yetarli bo’ladi.
2. Agar obyektlarning barcha umumiy holatlarini bitta flyweight obyektiga qisqartirishga erishsangiz Flyweight paterni Singletonga o’xshashi mumkin.
    a. Faqatgina bitta Singleton obyekti bo’ladigan bo’lsa, Flyweight klassda turli xil ichki holatga ega bo’lgan ko’plab obyektlarga ega bo’ladi.

b. Singleton obyekt mutable bo’ladi. Flyweight obyektlar esa immutable bo’ladi.

3. Abstract Factorylar, Builderlar va Prototypelarning barchasi Singleton sifatida ishlatilishi mumkin.

### Namuna kodi

```php
<?php

namespace RefactoringGuru\Singleton\Conceptual;

/**
 * The Singleton class defines the `GetInstance` method that serves as an
 * alternative to constructor and lets clients access the same instance of this
 * class over and over.
 */
class Singleton
{
    /**
     * The Singleton's instance is stored in a static field. This field is an
     * array, because we'll allow our Singleton to have subclasses. Each item in
     * this array will be an instance of a specific Singleton's subclass. You'll
     * see how this works in a moment.
     */
    private static $instances = [];

    /**
     * The Singleton's constructor should always be private to prevent direct
     * construction calls with the `new` operator.
     */
    protected function __construct() { }

    /**
     * Singletons should not be cloneable.
     */
    protected function __clone() { }

    /**
     * Singletons should not be restorable from strings.
     */
    public function __wakeup()
    {
```

```
        throw new \Exception("Cannot unserialize a singleton.");
    }

    /**
     * This is the static method that controls the access to the singleton
     * instance. On the first run, it creates a singleton object and places it
     * into the static field. On subsequent runs, it returns the client existing
     * object stored in the static field.
     *
     * This implementation lets you subclass the Singleton class while keeping
     * just one instance of each subclass around.
     */
    public static function getInstance(): Singleton
    {
        $cls = static::class;
        if (!isset(self::$instances[$cls])) {
            self::$instances[$cls] = new static();
        }

        return self::$instances[$cls];
    }

    /**
     * Finally, any singleton should define some business logic, which can be
     * executed on its instance.
     */
    public function someBusinessLogic()
    {
        // ...
    }
}

/**
 * The client code.
 */
function clientCode()
{
    $s1 = Singleton::getInstance();
    $s2 = Singleton::getInstance();
    if ($s1 === $s2) {
        echo "Singleton works, both variables contain the same instance.";
    } else {
        echo "Singleton failed, variables contain different instances.";
```

```
    }
}

clientCode();
```

```
Singleton works, both variables contain the same instance.
```