



Lesson 10

24.03.2025

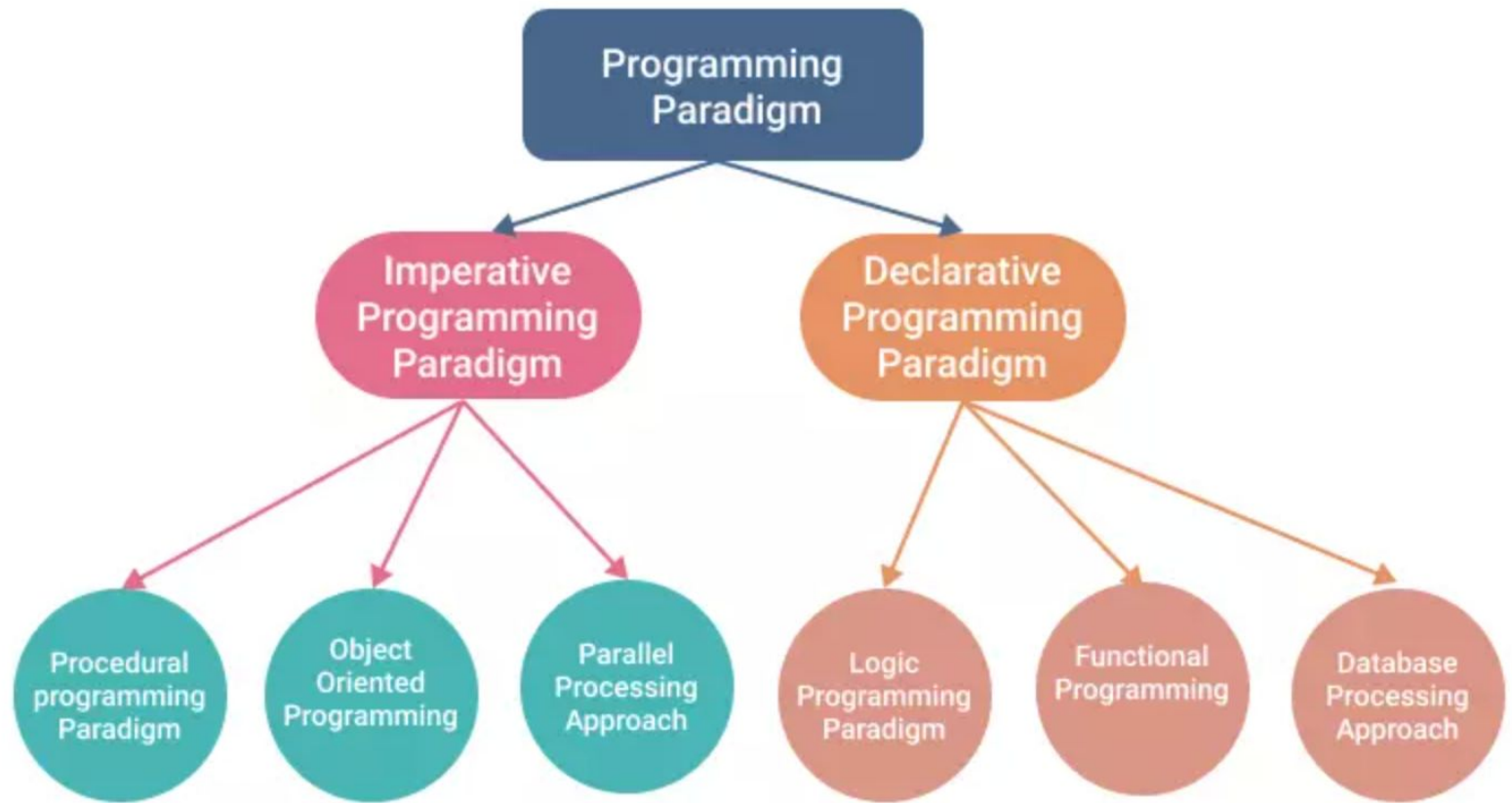
```
public class Task1 {  
    public static void main(String[] arguments) {  
        int turtle = 10 * (2 + (3 + 2) / 5);  
        int hare = turtle < 5 ? 10 : 25;  
        System.out.print(turtle < hare ? "Hare wins!" : "Turtle wins!");  
    }  
}
```

```
public class Task2 {  
    public static void main(String[] args) {  
        int characters = 5;  
        int story = 3;  
        double movieRating = characters ≤ 4 ? 3 : story > 1 ? 2 : 1;  
        System.out.println(movieRating);  
    }  
}
```

```
public class Task3 {  
    public static void main(String[] args) {  
        String letters = "";  
        while (letters.length() != 2)  
            letters += "a";  
        System.out.println(letters);  
    }  
}
```

```
public class Task4 {  
    public static void main(String[] args) {  
        System.out.print(5 + 6 + "7" + 8 + 9);  
    }  
}
```

```
public class Task5 {  
    public static void main(String[] args) {  
        int tiger = 2;  
        short lion = 3;  
        long winner = lion + 2 * (tiger + lion);  
        System.out.print(winner);  
    }  
}
```



IMPERATIVE vs DECLARATIVE

WHAT DOES IT MEAN?



Follow steps

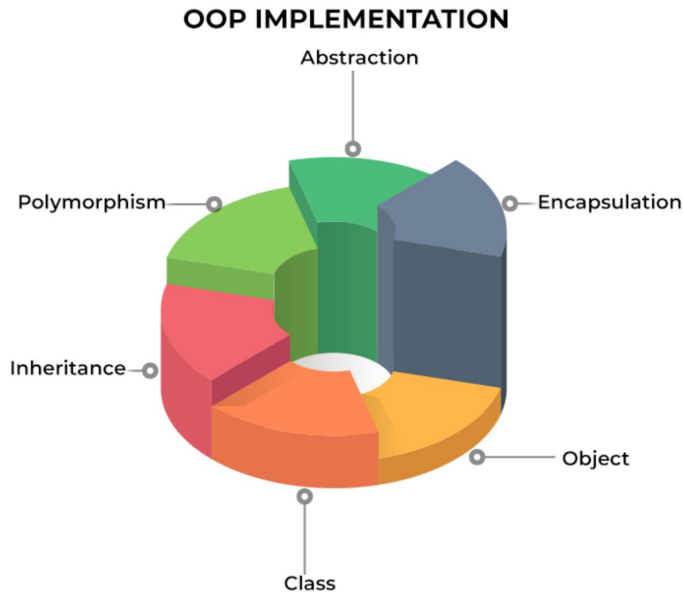
- ① Get a pot
- ② Pour water in pot
- ③ Heat pot over stove
- ④ Wait 5 minutes

Define what we need

- ① Fill kettle so it contains 1L water
- ② Heat water in kettle until temperature reaches 100°C

OOP

У цій парадигмі програмування програма розбивається на об'єкти – структури даних, що складаються з полів, що описують стан, та методів – підпрограм, що застосовуються до об'єктів для зміни чи запиту їх стану. Більшість об'єктно орієнтованих парадигм для опису об'єктів використовуються класи, об'єкти вищого порядку, що описують структуру та операції, пов'язані з об'єктами



class

car

methods

refuel() getFuel
setSpeed() getSpeed()
drive()

attributes

fuel
maxspeed

Encapsulation

Data Security

Inheritance

Code Reusability

OOP Program

Class

Object

Polymorphism

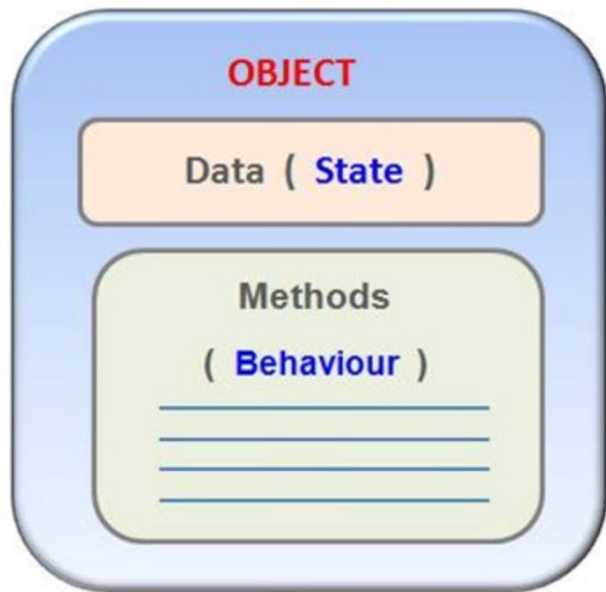
Code Reusability

Abstraction

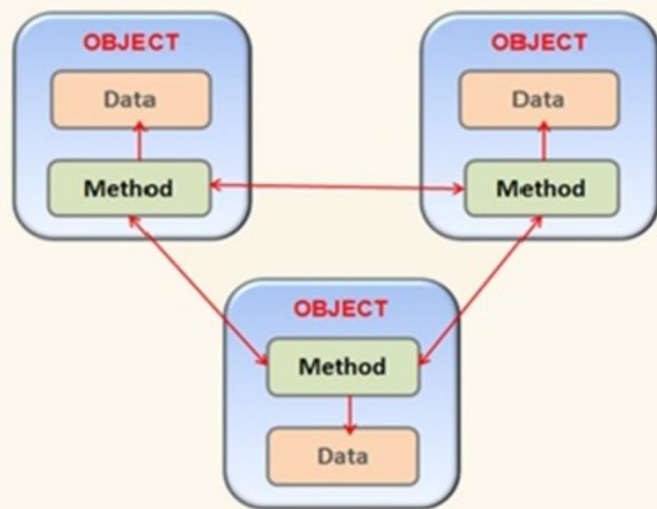
Hides Complexity



OOP - Concept Of Object




OPP - Program Organization





Class vs. Object

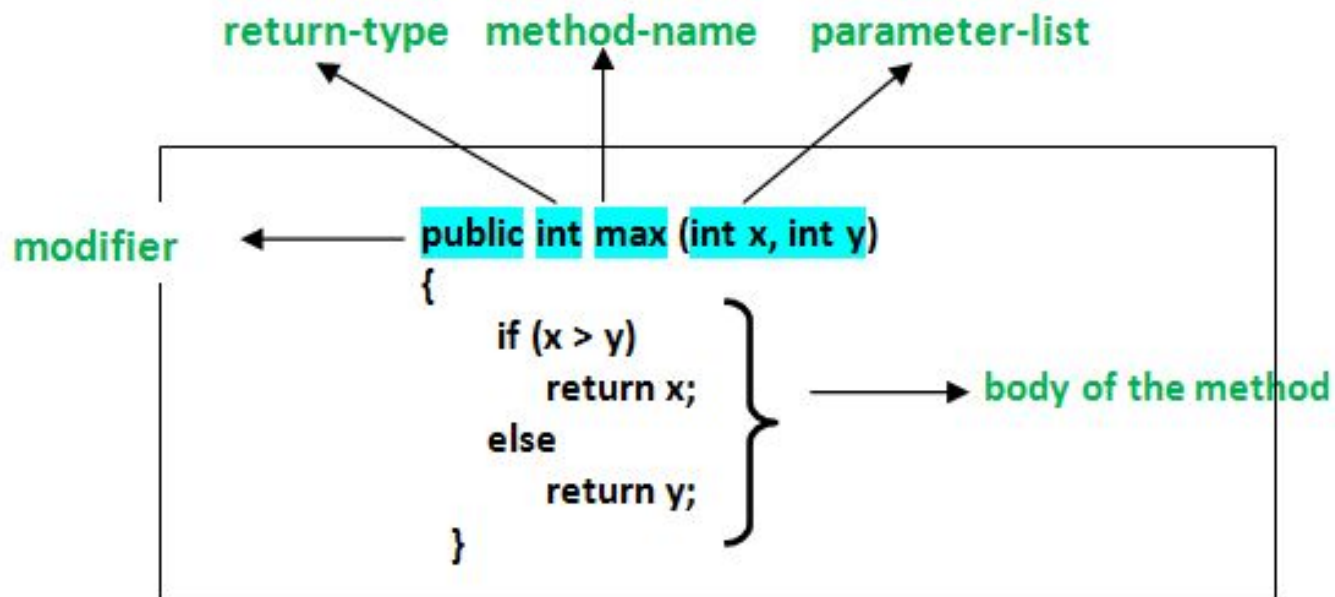


Функція – частина програми, яка має власне ім'я. Це ім'я можна використовувати у програмі як команду (така команда називається викликом функції). Під час виклику функції виконуються команди, з яких вона складається. Виклик функції може повертати значення (аналогічно операції) і тому може використовуватися у виразі поряд з операціями.

Метод - це функція, яка є частиною деякого класу, яка може виконувати операції над даними цього. У мові Java вся програма складається тільки з класів і функції можуть описуватись тільки всередині них. Саме тому всі функції мови Java є методами.

Функції використовуються у програмуванні, щоб зменшити його складність:

- 1.Замість того, щоб писати безперервну послідовність команд, якої незабаром перестав орієнтуватися, програму розбивають на підпрограми, кожна з яких вирішує невелике закінчене завдання, а потім велика програма складається з цих підпрограм (цей прийом називається декомпозицією).
- 2.Зменшується загальна кількість коду, тому що, як правило, одна функція використовується у програмі кілька разів.
- 3.Написана якимось і всебічно перевірена функція, можливо включена до бібліотеки функцій та використовуватись в інших програмах (при цьому не треба згадувати, як була запрограмована ця функція, достатньо знати, що вона робить).

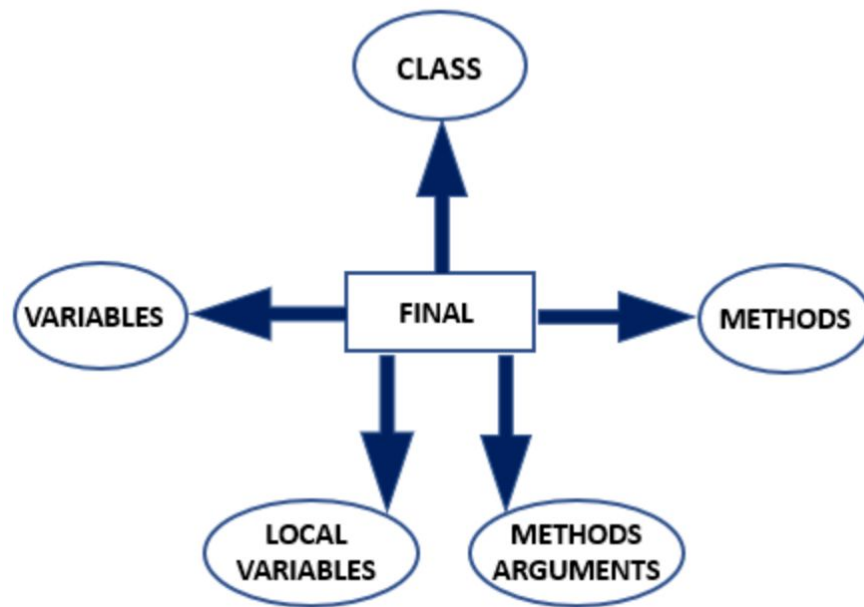




Access Modifiers	Non-Access Modifiers
<p>private default or No Modifier protected public</p>	<p>static final abstract synchronized transient volatile strictfp</p>



1. Final Non Access Modifiers



Final ключове слово використовується з класом, коли ми хочемо обмежити його успадкування будь-яким іншим класом. Наприклад, якщо у нас є **final** клас, то будь-яка спроба розширити цей клас може призвести до помилки під час компіляції.

2. Abstract Non-Access Modifier



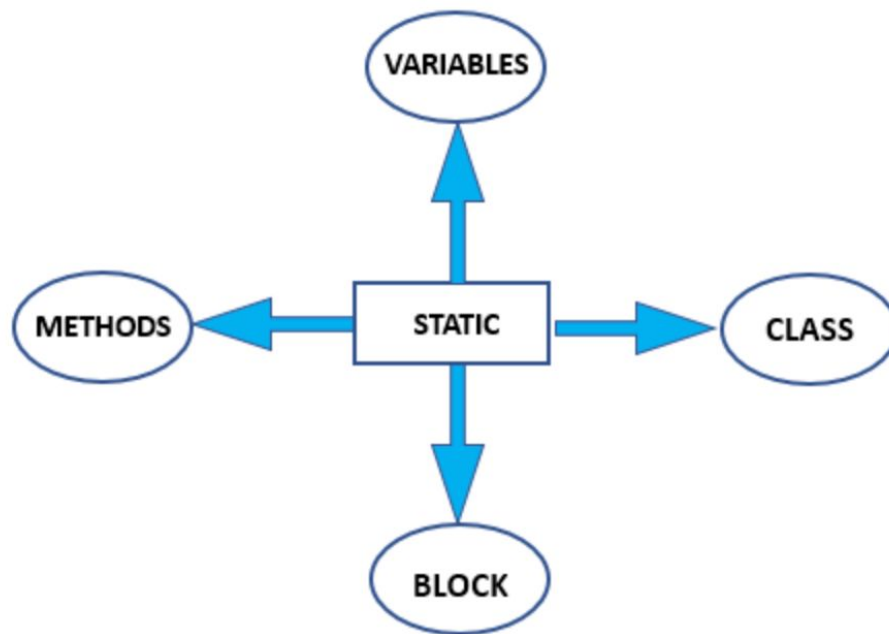
Клас оголошується як абстрактний, щоб вказати, що цей клас не може бути створений, що означає, що жодні об'єкти не можуть бути сформовані для цього класу, але можуть бути успадковані. Тим не менш, цей клас має конструктор, який буде викликатися всередині конструктора його підкласу. Він може містити як абстрактні, так і фінальні методи, де абстрактні методи будуть замінені в підкласі.

3. Synchronized Non-Access Modifier



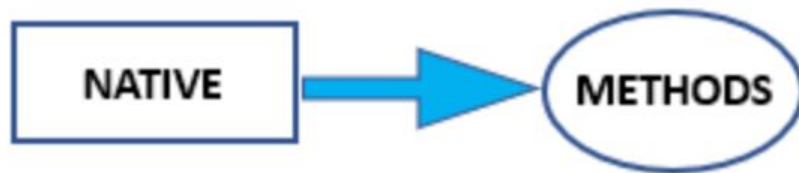
Це ключове слово допомагає запобігти одночасному доступу кількох потоків до одного методу, таким чином синхронізуючи потік програми та виводячи бажані результати за допомогою функції багатопоточності.

4. Static Non-Access Modifier



Ця змінна використовується для керування пам'яттю та для першого посилання під час завантаження класу. До цих членів ставляться на рівні класу; таким чином, їх не можна викликати за допомогою об'єкта; натомість ім'я класу використовується для посилання на них.

5. Native Non Access Modifier



Ключове слово `native` використовується лише з методами, щоб вказати, що конкретний метод написаний залежно від платформи. Вони використовуються для покращення продуктивності системи, а існуючий застарілий код можна легко використовувати повторно.

6. Strictfp Non-Access Modifier



Це ключове слово використовується для того, щоб результати операції над числами з плаваючою комою виводили однакові результати на кожній платформі. Це ключове слово не можна використовувати з абстрактними методами, змінними або конструкторами, оскільки вони не повинні містити операції.

7. Transient Non-Access Modifier

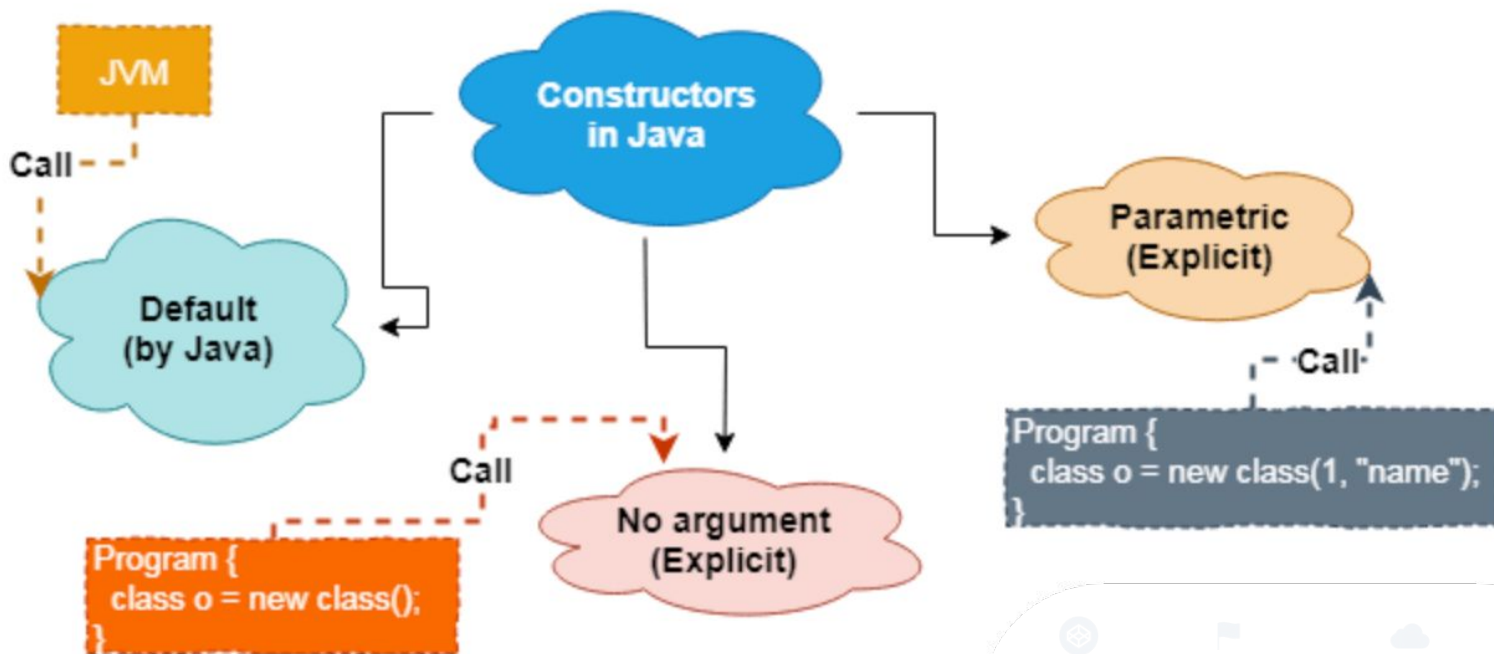
Під час передачі даних від одного кінця до іншого через мережу, вони повинні бути серіалізовані для успішного отримання даних, що означає перетворення в потік байтів перед надсиланням і перетворення його назад на кінці прийому. Щоб повідомити JVM про учасників, які не потребують серіалізації замість того, щоб бути втраченими під час передачі, з'являється тимчасовий модифікатор.



- Ініціалізація статичних полів та блоків виконується під час завантаження класу
- Ініціалізація нестатичних елементів виконується під час створення об'єкта(при виклику конструктора)

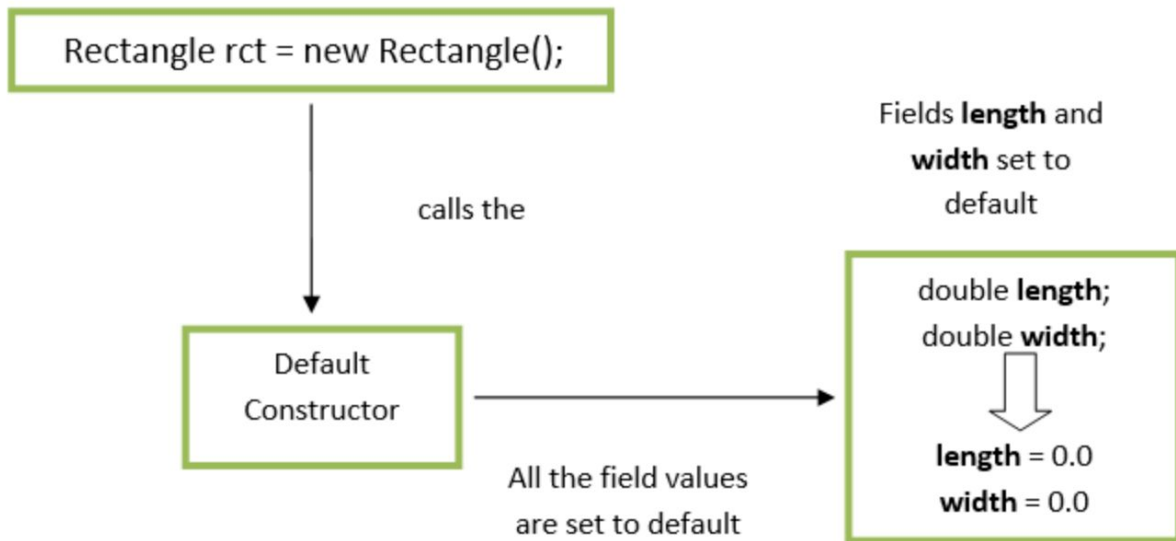


Конструктор – це метод, призначення якого полягає у створенні екземпляра класу.

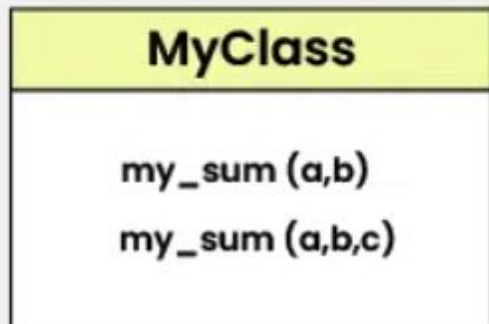


Послідовність дій під час виклику конструктора

- Усі поля даних ініціалізуються своїми значеннями, передбаченими за промовчанням (0, false або null).
- Ініціалізатори всіх полів та блоки ініціалізації виконуються в порядку їх перерахування в оголошенні класу.
- Якщо в першому рядку конструктора викликається інший конструктор, то виконується викликаний конструктор.
- Виконується тіло конструктора



Function Overloading in Programming



Function Overriding in Programming

