




# Lesson 4

22.12.2024

```
public class Task1 {  
    public static void main(String[] args) {  
        int a = 7;  
        int b = 2;  
        System.out.println(a / b);  
    }  
}
```





```
public class Task2 {  
    public static void main(String[] args) {  
        int a = 7;  
        int b = 3;  
        System.out.println(a % b);  
    }  
}
```

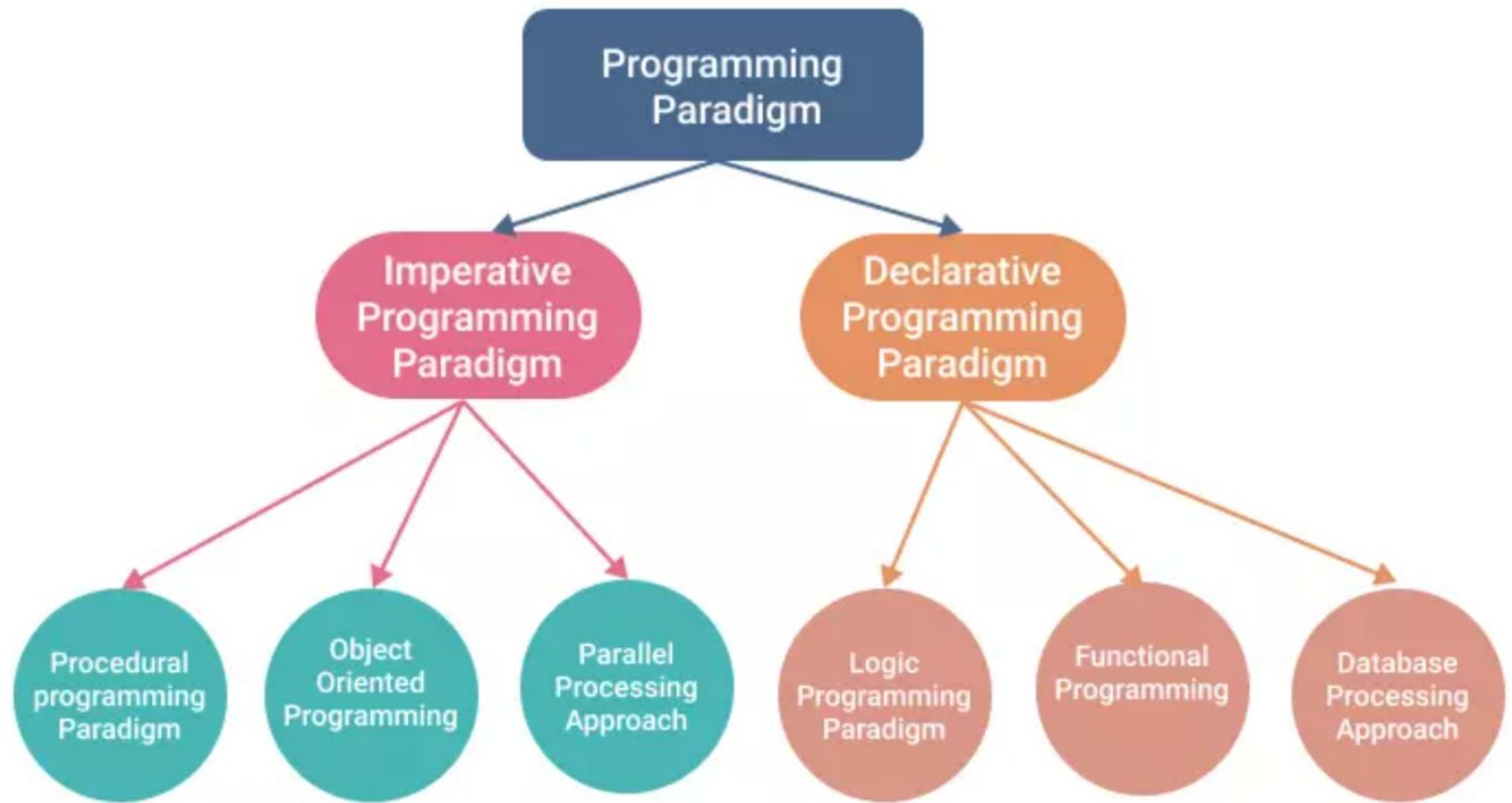


```
public class Task3 {  
    public static void main(String[] args) {  
        int a = 5;  
        a += 3;  
        System.out.println(a);  
        a--;  
        System.out.println(a);  
    }  
}
```

```
public class Task4 {  
    public static void main(String[] args) {  
        int a = 5;  
        System.out.println(a++);  
        System.out.println(++a);  
    }  
}
```

```
public class Task5 {  
    public static void main(String[] args) {  
        double x = 9;  
        double y = Math.sqrt(x);  
        System.out.println(Math.pow(y, 2));  
    }  
}
```

Ознака	Java	C
 <b>Розмір масиву</b>	Задається під час створення через <code>new</code> , фіксований	Фіксований або динамічний (через <code>malloc</code> )
 <b>Перевірка меж</b>	Так — <code>ArrayIndexOutOfBoundsException</code>	Ні — можливий вихід за межі (UB)
 <b>Структура</b>	Масив — це об'єкт у купі	Масив — це послідовна область в пам'яті
 <b>Типи даних</b>	Примітиви або об'єкти	Примітиви або структури
 <b>Арифметика указівників</b>	Немає	Є — можна пересуватися по масиву
 <b>Керування пам'яттю</b>	Автоматичне (GC)	Ручне ( <code>malloc</code> / <code>free</code> )
 <b>Многовимірність</b>	Масив масивів (зубчастий)	Справжній багатовимірний масив
 <b>Функціональність</b>	<code>Arrays.sort</code> , <code>stream()</code> тощо	Немає — все вручну





# IMPERATIVE vs DECLARATIVE

WHAT DOES IT MEAN?



Follow steps

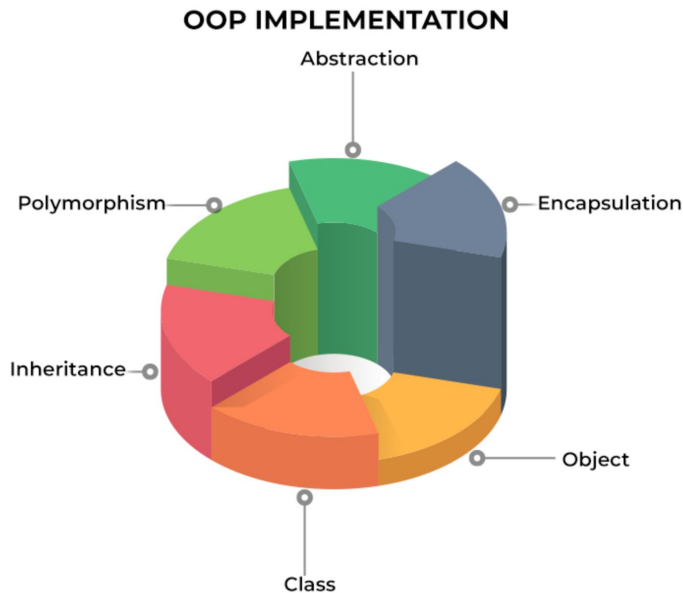
- ① Get a pot
- ② Pour water in pot
- ③ Heat pot over stove
- ④ Wait 5 minutes

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**

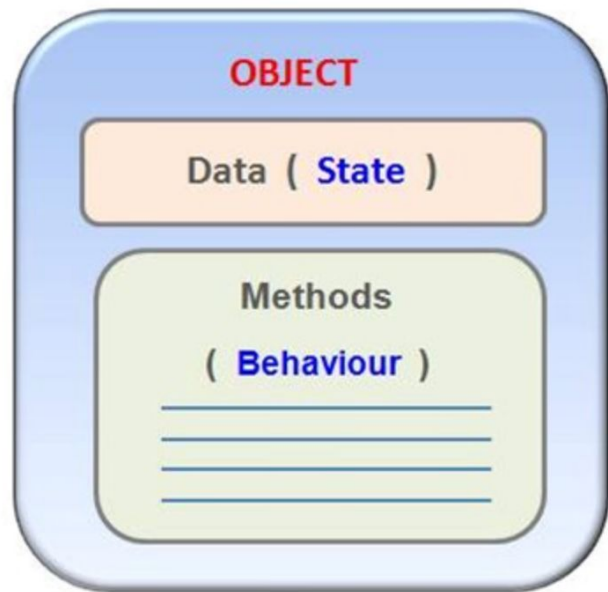
Code Reusability

## **Abstraction**

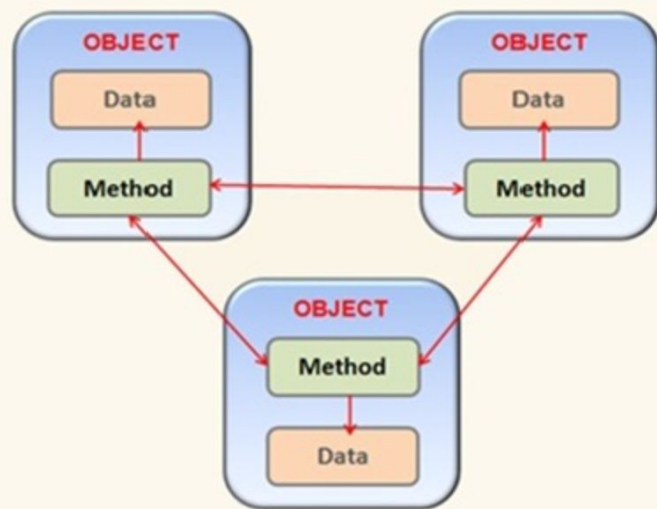
Hides Complexity



## OOP - Concept Of Object



## OPP - Program Organization





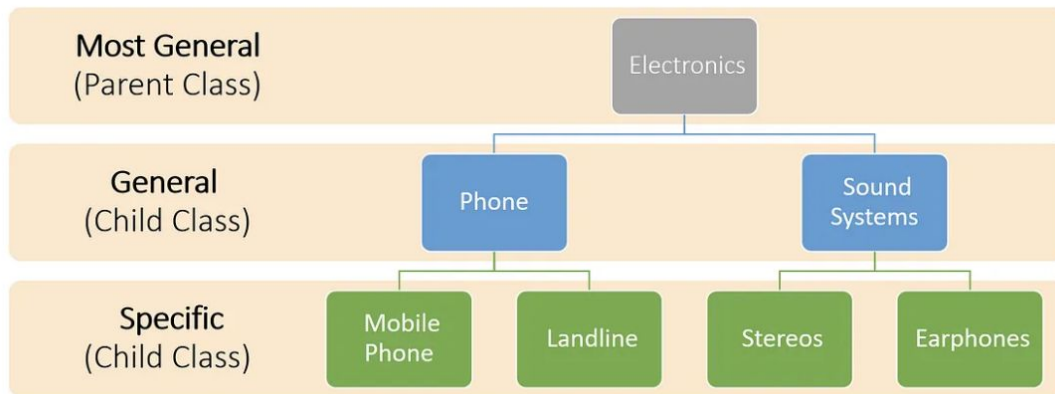
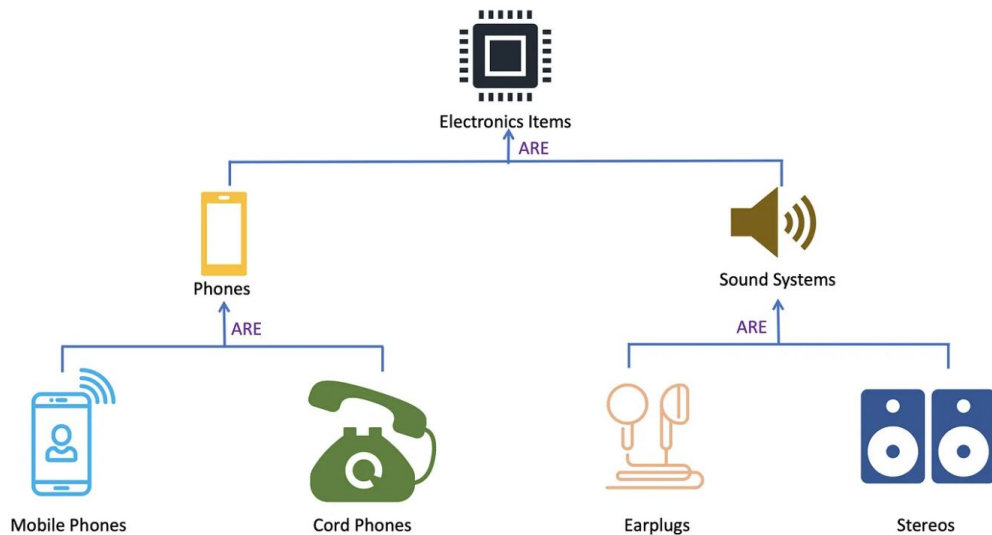
## Принципи ООП

**Інкапсуляція** це властивість системи, що дозволяє об'єднати дані та методи в класі, та приховати деталі реалізації від користувача.

**Наслідування** це властивість системи, що дозволяє описати новий клас на основі вже існуючого з частково або повністю запозиченої функціональністю

**Поліморфізм** – один інтерфейс, безліч методів”. Реалізації поліморфізму в мові Java - це навантаження та перевизначення методів, інтерфейси

**Абстракція** даних це спосіб виділити набір значних показників об'єкта, крім розгляду не значущі. Відповідно, абстракція – це набір всіх таких показників.



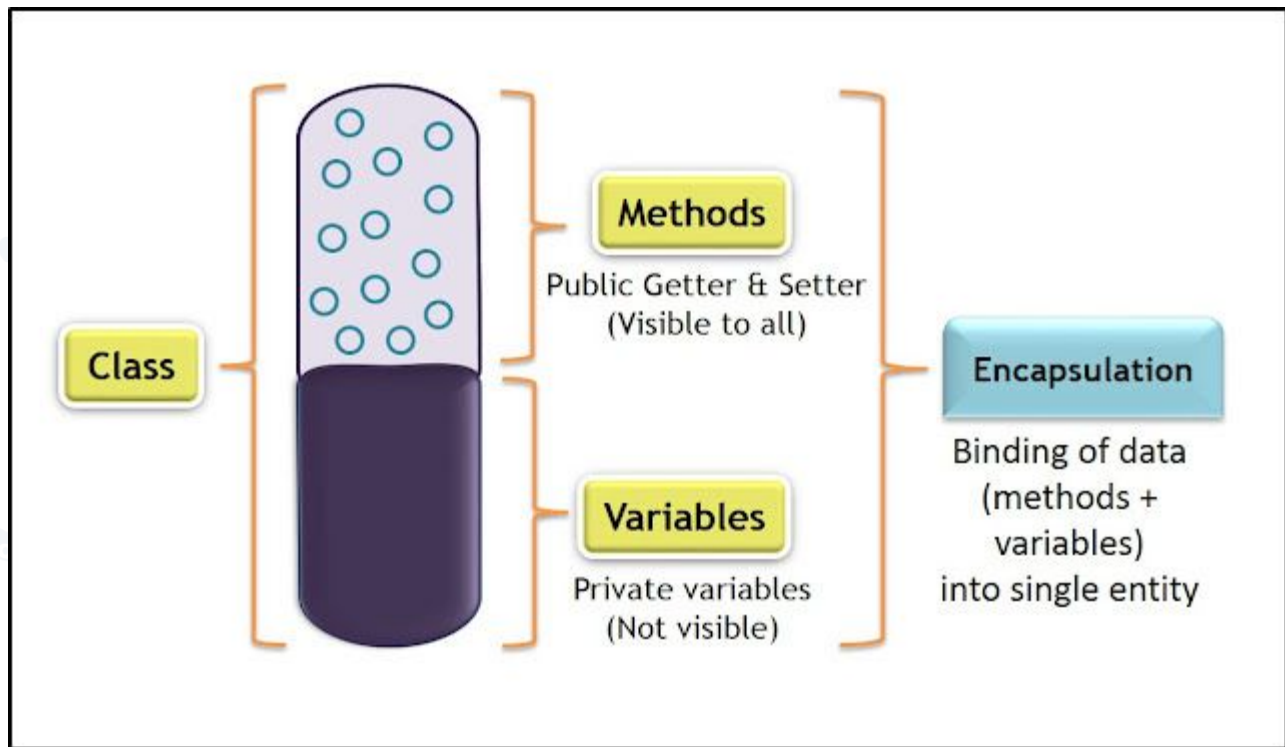
# Inheritance



you can create new type of animal  
changing or adding properties







# Incapsulation



every animal eats  
and then poop

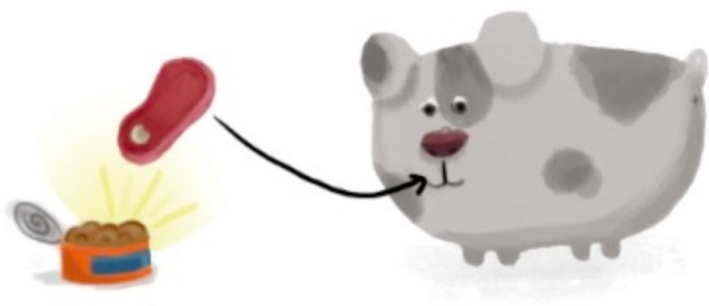




# Polymorphism



each animal can eat  
its own type of food





# ***Class vs. Object***



OBJECT	CLASS
Object is an instance of a class.	Class is a blue print from which objects are created
Object is a real world entity such as chair, pen, table, laptop etc.	Class is a group of similar objects.
Object is a physical entity.	Class is a logical entity.
Object is created many times as per requirement.	Class is declared once.
Object allocates memory when it is created.	Class doesn't allocate memory when it is created.
Object is created through new keyword. Employee ob = new Employee();	Class is declared using class keyword. class Employee{}
There are different ways to create object in java:- New keyword, newInstance() method, clone() method, And deserialization.	There is only one way to define a class, i.e., by using class keyword.



Breed: Bulldog  
Size: large  
Colour: light gray  
Age: 5 years

Dog1Object



Breed: Beagle  
Size: large  
Colour: orange  
Age: 6 years

Dog2Object



Breed: German Shepherd  
Size: large  
Colour: white & orange  
Age: 6 years

Dog3Object

Dog

**Fields**

Breed  
Size  
Colour  
Age

**Methods**

Eat()  
Run()  
Sleep()  
Name()

***Box myBox;***



***null***

***myBox = new Box();***



***Heap***

***Box***

***double height;***

***double depth;***

***double width;***



*stack*

*b1*

*b2*

*Heap*

***Box***

***double height;***

***double depth;***

***double width;***



У мові Java під час проектування класів прийнято обмежувати рівень доступу до змінним за допомогою модифікатора **private** і звертатися до них через гетери та сетери.

Існують правила оголошення таких методів, розглянемо їх:

- Якщо властивість НЕ типу `boolean`, префікс геттера має бути `get`. Наприклад: **getName()** - це коректне ім'я геттера для змінної `name`.
- Якщо властивість типу `boolean`, префікс імені геттера може бути `get` або `is`. Наприклад, **getPrinted()** або **isPrinted()** обидва є коректними іменами для змінних типу `boolean`.
- Ім'я сеттера повинне починатися з префікса `set`. Наприклад, **setName()** коректне ім'я для змінної `name`.
- Для створення імені геттера або сеттера, перша літера властивості має бути змінена на велику та додана до відповідного префіксу (`set`, `get` або `is`).
- Сеттер має бути `public`, повертати `void` тип і мати параметр відповідний типу змінної.
- Геттер метод повинен бути `public`, не мати параметрів методу, і повертати значення відповідне типу властивості



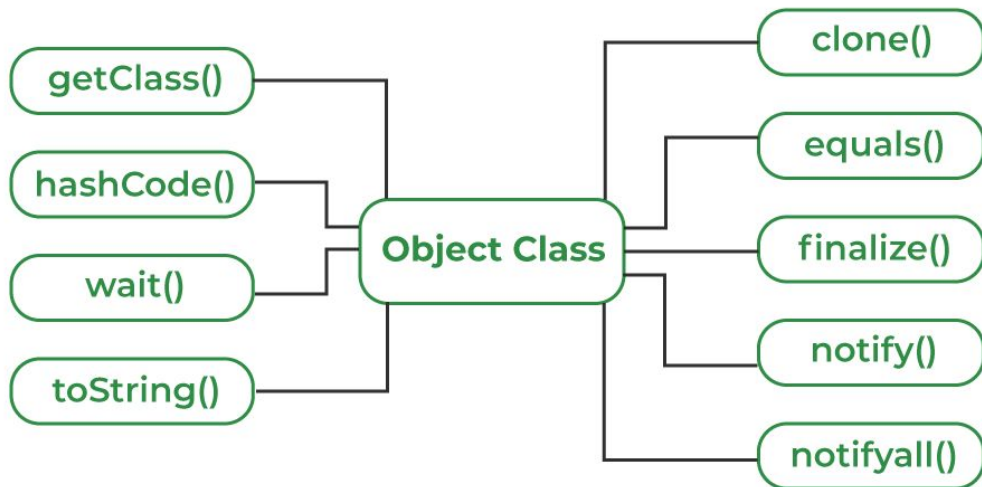
## Клас Object

Є суперкласом для всіх класів (включаючи масиви)


Змінна цього типу може посилатися будь-який об'єкт (але не змінну примітивного типу)

Його методи успадковуються усіма класами

Реалізує базові операції з об'єктами



Method	Purpose
Object clone( )	Creates a new object that is the same as the object being cloned.
boolean equals(Object <i>object</i> )	Determines whether one object is equal to another.
void finalize( )	Called before an unused object is recycled.
Class<?> getClass( )	Obtains the class of an object at run time.
int hashCode( )	Returns the hash code associated with the invoking object.
void notify( )	Resumes execution of a thread waiting on the invoking object.
void notifyAll( )	Resumes execution of all threads waiting on the invoking object.
String toString( )	Returns a string that describes the object.
void wait( ) void wait(long <i>milliseconds</i> ) void wait(long <i>milliseconds</i> , int <i>nanoseconds</i> )	Waits on another thread of execution.

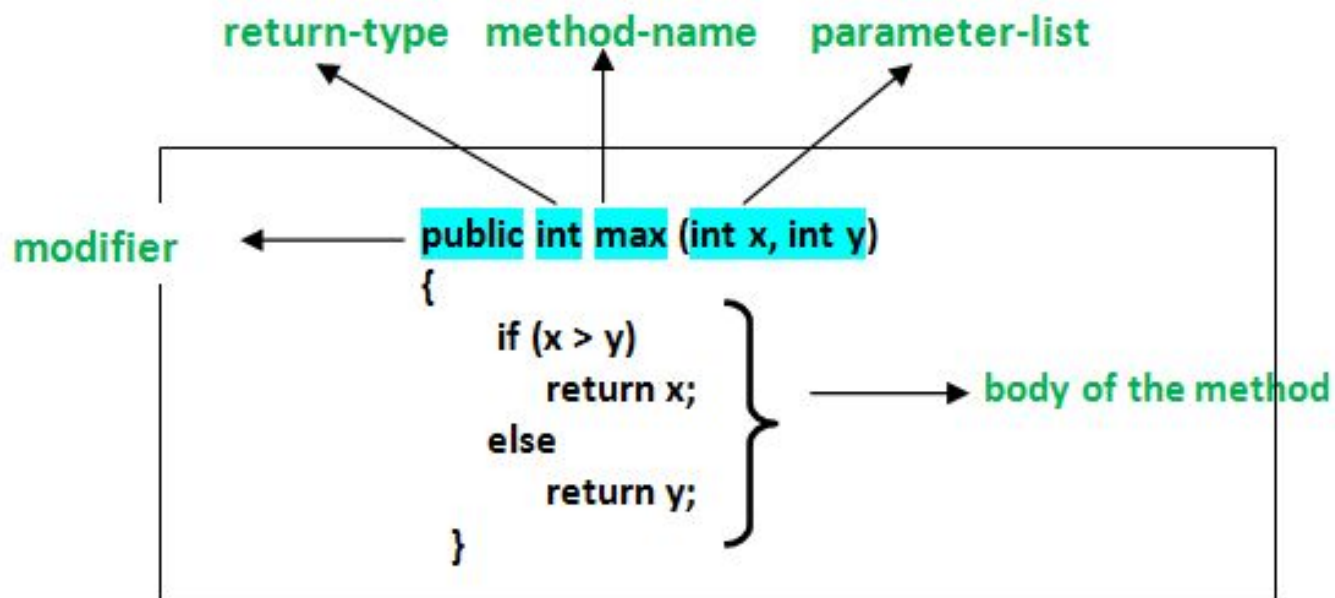


**Функція** – частина програми, яка має власне ім'я. Це ім'я можна використовувати у програмі як команду (така команда називається викликом функції). Під час виклику функції виконуються команди, з яких вона складається. Виклик функції може повертати значення (аналогічно операції) і тому може використовуватися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

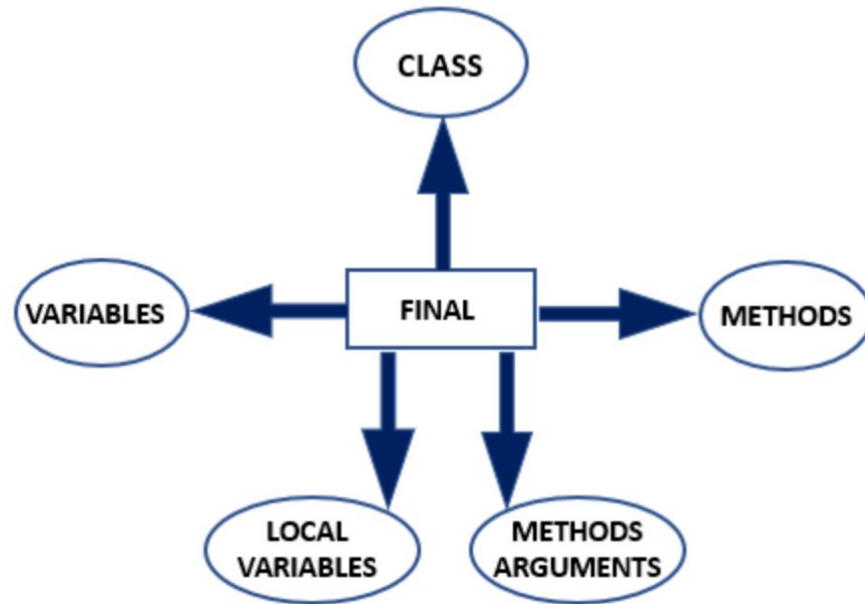




Access Modifiers	Non-Access Modifiers
<p>private default or No Modifier protected public</p>	<p>static final abstract synchronized transient volatile strictfp</p>



# 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.

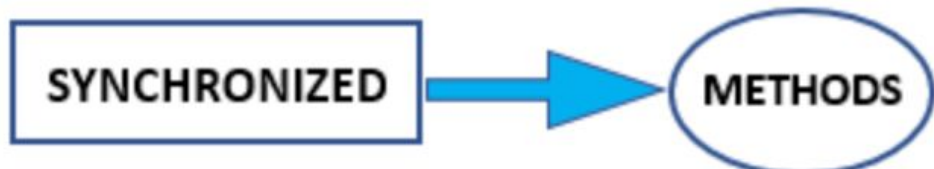


## 2. Abstract Non-Access Modifier



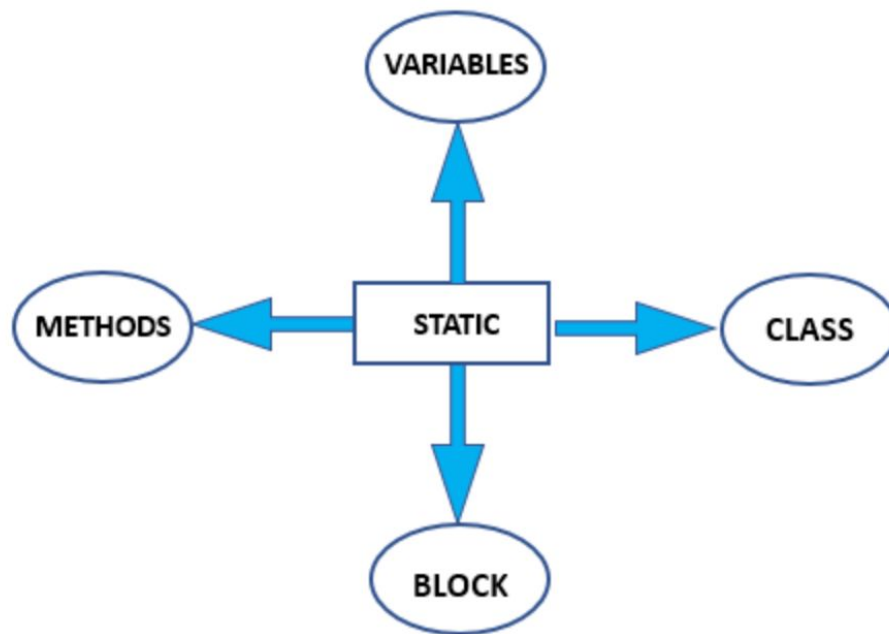
Клас оголошується як абстрактний, щоб вказати, що цей клас не може бути створений, що означає, що жодні об'єкти не можуть бути сформовані для цього класу, але можуть бути успадковані. Тим не менш, цей клас має конструктор, який буде викликатися всередині конструктора його підкласу. Він може містити як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

### 3. Synchronized Non-Access Modifier



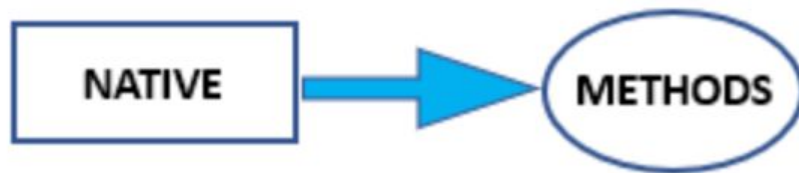
Це ключове слово допомагає запобігти одночасному доступу кількох потоків до одного методу, таким чином синхронізуючи потік програми та виводячи бажані результати за допомогою функції багатопоточності.

## 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.



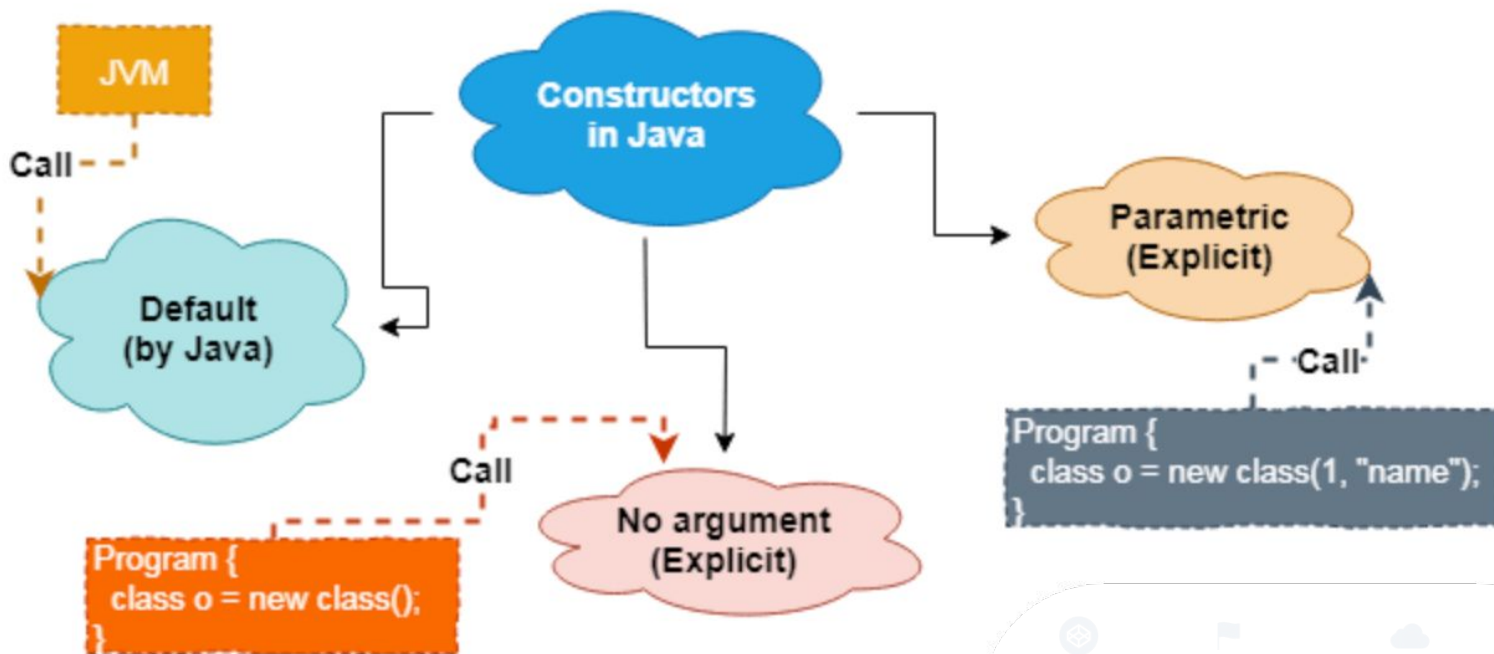
## 7. Transient Non-Access Modifier

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

