**Lesson 19. Encapsulation**

На минулому занятті ми опрацювали два основоположних поняття для об’єктно-орієнтованого програмування – об’єкт і клас. Крім цих двох понять, а також конструкцій, які їх реалізовують, маємо також розглянути важливі принципи, які роблять ООП надпотужним підходом.

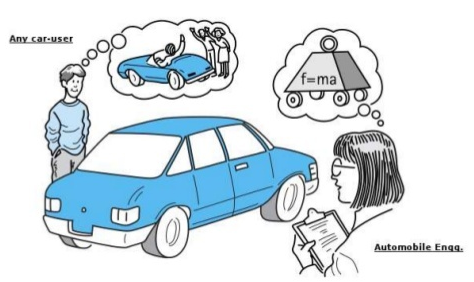
Ці принципи часто визначають трьома «страшними» словами: інкапсуляція, наслідування, поліморфізм.

На цьому занятті ми торкнемось одного з них – **інкапсуляції**. Певною мірою ми вже говорили про інкапсуляцію під час розгляду понять об’єкта і класу, однак тепер розберемось глибше.

**Abstraction**

Невід’ємним елементом об'єктно-орієнтованого програмування є **абстракція**. Загалом, абстракція це те, за допомогою чого люди спрощують сприйняття і розуміння речей довкола себе.

Наприклад, люди не думають про автомобіль, як сукупність десятків тисяч окремих частин. Вони сприймають автомобіль як транспортний засіб, що має набір певних функцій. Таке сприйняття дозволяє користуватись автомобілем так, щоб доїхати до продуктового магазину, не будучи приголомшеним, тим якою насправді складною штукою він є насправді. Зазвичай користувачі автівок ігнорують деталі того, як двигун, коробка передач і гальма працюють, натомість просто їх використовують для досягнення певного результату.

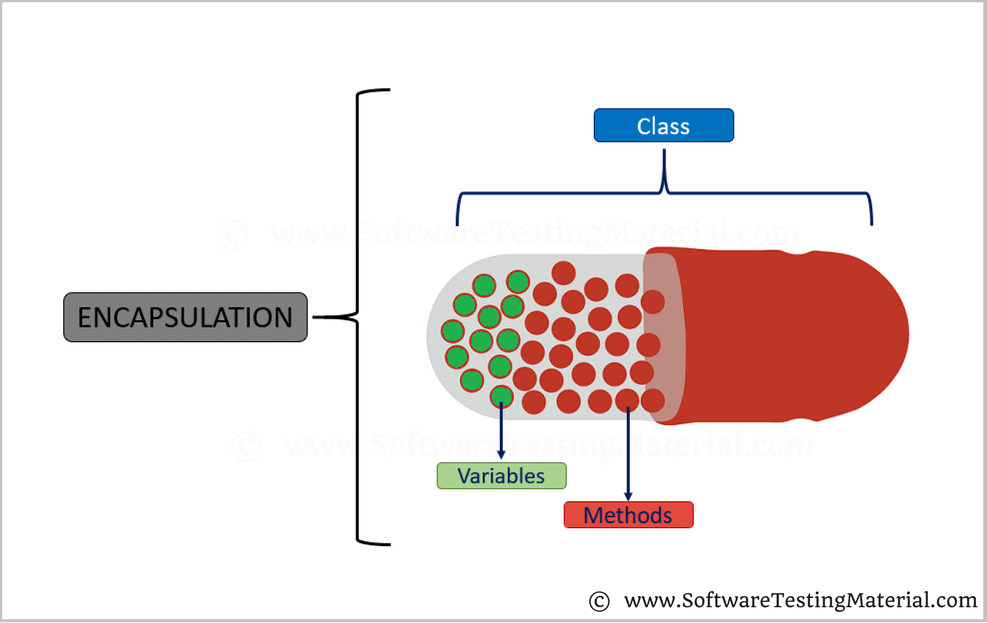


Аби слідувати абстракції в ООП, необхідно у процесі створення класів завжди думати про те, які елементи об’єкта (поведінка чи метод) повинні бути доступні користувачеві, для його використання, а які варто приховати тим самим спростивши його.

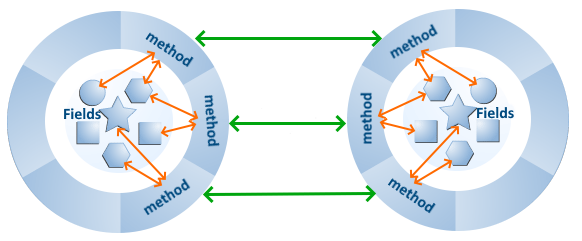
Абстракція в ООП забезпечується рядом інструментів через реалізацію головних принципів ООП: ***наслідування***, ***поліморфізм***, ***інкапсуляція***.

**What is encapsulation?**

Як ми знаємо, об’єкти поєднують у собі методи та змінні, і це вже є першим кроком до інкапсуляції, адже таким чином у об’єкті під одною оболонкою поєднуються дані, та код, який може маніпулювати цими даними.



Якщо брати в цілому, то **інкапсуляція** – це принцип, згідно з яким інформація про внутрішній стан об’єкта приховується (тобто до неї не повинно бути прямого доступу), а будь-яка взаємодія із полями, що відображають цей стан, здійснюється за допомогою методів класу.



Якщо перенести цей принцип у реальний світ, то його можна відслідкувати у всіх реальних об’ктах. Якщо ми хочемо змінити якийсь об’єкт (тобто його стан), то для цього повинні виконати специфічну дію (використати метод).

Для того, щоб змінити стан кота із голодного на ситий, його необхідно нагодувати, а для того, щоб збільшити температуру повітря в пічці, необхідно повернути регулятор газу. Без активної дії стан об’єкта не може змінитись.

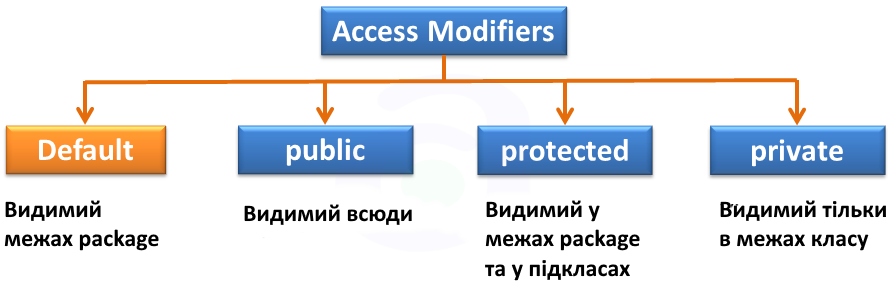
Серед **переваг**, які дозволяє отримати використання інкапсуляції, можна виділити наступні:

* Кращий контроль над полями і методами класу;
* Використання методів для доступу до даних дає можливість налаштувати деякі із полів у режим «тільки для зчитування», або «тільки для запису»;
* Підвищення гнучкості коду, адже можна змінити одну частину коду, не впливаючи на інші його частини;
* Підвищена безпека роботи із даними.

**Access modifiers**

Для визначення доступності до даних (як до полів, так і до методів) у Java є спеціальні модифікатори доступу, якими ми вже неодноразово користувались, але при цьому, не знали що воно таке. Серед них:

1. **default** (не потребує ключового слова)
2. **private**
3. **protected**
4. **public**



На даний момент нас найбільше будуть цікавити *public* і *private*, адже саме вони будуть використовуватись нами найчастіше. Крім того нам ще не відоме поняття package і підклас, тому *default* і *protected* ми поки що залишимо у категорії незвіданого.

Для реалізації приховування даних за принципом інкапсуляції використовується модифікатор *private*. Усі дані помічені таким модифікатором, будуть доступні тільки в середині класу, а отже для доступу до них використовуються методи із позначкою *public*.

Щоб краще зрозуміти як це працює, розглянемо прості приклади у наступному розділі.

**Приклад 1.** Створимо клас у якому матимемо поля і методи із доступом private і public:

**Example**



Тепер у іншому класі розглянемо, буде реалізовано доступ до даних у об’єкті цього класу:



Зверніть увагу на рядочки 13 і 14 де і показано, як діє інкапсуляція. Дані не доступні напряму з об’єкту, але при цьому їх можна отримувати і змінювати звертаючись до відповідних методів.

**Завдання 1**. Створіть клас ***HomePlant***, що буде відображати у програмі кімнатну рослину. Стан рослинки повинен відображатись наступною інформацією:

**Tasks**

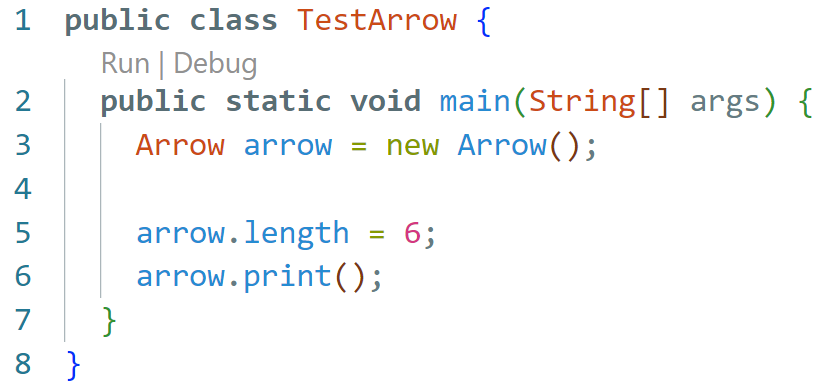
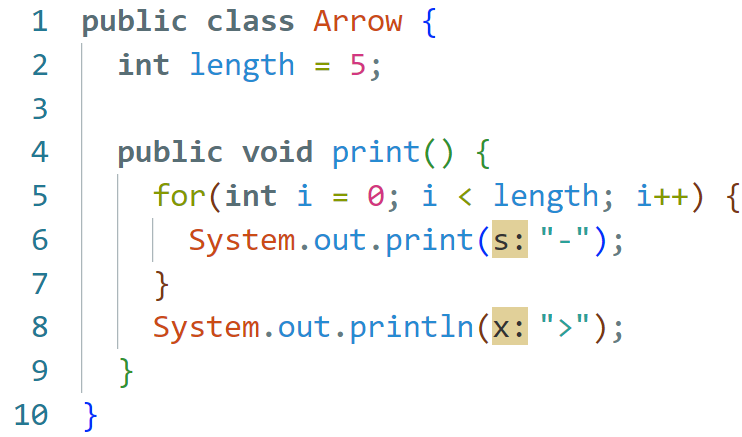
* максимальний об’єм води, що може бути залитий в горщик (об’єм);
* залишок води в горщику;
* кількість гілочок, які має рослина;

Поведінка рослинки повинна визначатись наступними пунктами:

* рослинку можна поливати (збільшувати об’єм води у горщику, але не більше ніж максимальний об’єм);
* можна змінити об’єм горщика (пересадити);
* можна порахувати кількість гілочок, що має рослинка;
* рослинка може рости (метод ***grow***() у якості аргументу приймає кількість годин світла, які отримує рослинка на день, а на виході повинен повертати об’єм кисню, який виробляє рослина)
* під час росту рослинка споживає воду: кожна гілочка потребує 10 мл води;
* якщо під час росту рослинка має достатню кількість води на кожну гілочку і також має не менше 4 годин сонячного світла на добу, то у неї виростає додаткова гілочка;
* якщо під час росту рослинка не має ані достатньо води, ані світла, то у неї засихає одна гілочка;
* кожна гілочка рослинки за умови достатньої кількості світла виробляє 100 мл кисню, але якщо годин сонячного світла менше 4, то об’єм кисню зменшується в двічі.

При виконанні завдання врахуйте усі необхідні доступи до даних та методів за допомогою модифікаторів доступу. Додатково створіть клас ***Task1***, де протестуйте роботу ***HomePlant*** у методі *main*.

**Homework**

**Завдання 1.** Розгляньте класи Arrow та TestArrow на рисунку нижче. Чи буде працювати програма при запуску методу **main**? Якщо ні, то як це виправити? Якщо так, то як можна покращити код (враховуючи принцип інкапсуляції)?