**ІІІ. ОБ’ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ МОВОЮ *JAVA***

Об’єктно-орієнтоване програмування (ОПП) настільки інтегроване в *Java*, що написання навіть найпростіших програм вимагає знань базових принципів ООП. Як відомо, всі програми складаються з двох елементів: коду і даних. Вони можуть взаємодіяти з зовнішнім світом за принципом "що відбувається" або "на що впливає". Існує два стилі побудови програм. Перший − процесноорієнтована модель. Цей підхід представляє програму як низку послідовно виконуваних операцій (такі мови програмування, як *Pascal*, *Fortran*, *C* успішно використовують цю модель). Однак внаслідок збільшення розмірів і складності програми такий підхід має значні недоліки. З метою розв’язання цих проблем розроблено другий підхід, який називають об’єктно-орієнтованим програмуванням. Програми, побудовані згідно принципу ООП, можна вважати такими, в яких дані керують доступом до коду. Перші об’єктно-орієнтовані мови виникли ще у 1967р. У 1983 р. Б’ярн Страуструп розробив першу версію *С* з класами. Спеціалісти − розробники *Java* також були програмістами на *С++*.

### 3.1. Теоретичні аспекти ОПП

Усі об’єктно-орієнтовані мови програмування забезпечують реалізацію базових концепцій: інкапсуляції, поліморфізму, наслідування.

#### 3.1.1. Абстракція (класи та об’єкти)

Центральним елементом об’єктно-орієнтованого програмування є абстракція. Завдяки можливості абстрагуватися людство справляється з проявами складних речей в оточуючому світі. Наприклад, ніхто не думає про автомобіль як про набір з тисячі деталей, коли треба переїхати з одного місця в інше. Автомобіль сприймається як об’єкт в цілому з певними характеристиками: швидкість руху, необхідна кількість пального і т.п.

Існує могутній механізм створення абстракції, який полягає в використанні ієрархічної класифікації. Такий підхід дозволяє проаналізувати семантику складних систем, розбивши їх на простіші фрагменти.

Ієрархічні абстракції складних систем можна застосувати і до комп’ютерних програм. Дані з традиційної процесно-орієнтованої програми за допомогою абстракції можна перетворити на об’єкти. А послідовність обробки цих даних може стати набором повідомлень, які передаються між об’єктами. Таким чином, кожний із об’єктів описує власну унікальну поведінку, відповідаючи на повідомлення виконати певну дію. В цьому і полягає суть ООП.

#### 3.1.2. Інкапсуляція та передавання повідомлень

Інкапсуляція (*encapsulation* − в перекладі "герметизація) - це механізм, який зв’язує код і дані, захищаючи у цьому випадку їх від зовнішнього пошкодження і некоректного використання. Це захисна оболонка, яка забороняє довільний доступ іншому кодові, зовнішньому щодо даного. Доступ до даних і коду суворо контролюється інтерфейсом. Кожний знає як доступитися до даних, не задумуючись над деталями реалізації коду.

У мові *Java* (як і в інших об’єктно-орієнтованих мовах) основою інкапсуляції є клас. Клас визначає структуру суті, створеної даними і кодом, і поведінку цієї суті, яка, використовуючи визначення класу, може втілюватися у багатьох об’єктах. Кожний об’єкт класу повторює структуру і поведінку, визначену класом, так ніби його відлили за допомогою форми у вигляді класу. Тому об’єкти називаються екземплярами класу. Клас є логічною конструкцією, а об’єкт − його фізичним втіленням. При створенні класу визначаються дані і код. Зокрема, дані називають змінними класу, а код, який виконується над цими даними, − методами класу. Поведінку та інтерфейс класу визначають методи, які опрацьовують дані екземпляру класу. Кожний метод або змінну визначають як внутрішні або загальнодоступні. Якщо доступ до власних членів класу збоку інших частин програми можна отримати тільки через загальнодоступні методи, то можна вважати, що принцип інкапсуляції цілковито реалізовано.

#### 3.1.3. Наслідування

Усі об’єкти реального світу можна впорядкувати в певний ієрархічний ряд (зверзу донизу). Наприклад, спанієль належить до класу собак, які є ссавцями, які належать до більш узагальненого поняття тварин. Кожний рівень ієрархічного ряду має свої специфічні характеристики. Якщо не застосовувати принцип наслідування, то для кожного об’єкта з найнижчого рівня треба було б визначати усі характеристики заново.

Цей аспект ООП дає змогу створювати нові класи, які успадковують функціональні можливості уже існуючих.

#### 3.1.4. Поліморфізм

Концепція поліморфізму полягає в тому, що за допомогою одного інтерфейсу реалізують декілька методів. Наприклад, під час створенняі різних об’єктів працює метод *create*, хоча для кожного з них його реалізовано по-різному. Вибір конкретної дії (тобто методу) стосовно кожної ситуації перекладається на компілятор або інтерпретатор. Програмістові необхідно запам’ятати, як застосувати загальний інтерфейс: *open, close* тощо.

***3.1.5. Спільна дія поліморфізму, інкапсуляції та наслідування***

Найповніше ілюструє силу об’єктно-орієнтованого підходу приклад з реального життя − об’єкт "автомобіль". Усі водії використовують наслідування для управління різними типами засобів руху: автобусом, вантажним автомобілем і та іншими засобами пересування. Постійно водії зустрічаються з інкапсульованими засобами в автомобілі. Педалі гальма, газу, кермо − це частини складного механізму, які пропонують простий інтерфейс, заховуючи при цьому конкретну реалізацію дій. Нарешті, поліморфізм, демонструє здатність виробників автомобілів пропонувати все нові варіанти одного і того ж, за змістом, транспортного засобу. Адже в усіх автомобілях необхідно натиснути на педаль гальма, щоб зупинитись, повернути кермо для зміни напряму руху і т.п.

Тобто трансформація окремих деталей в об’єкт, який названо "автомобілем", досягається шляхом застосування інкапсуляції, наслідування і поліморфізму. Те ж саме справедливо і для програм. Застосовуючи об’єктно-орієнтований підхід, різні частини складної програми можна зібрати докупи з метою створення зв’язаного, стійкого до помилок і працюючого цілого.