­­МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

­­­

Лабораторна робота №2

з курсу «Об’єктно-орієнтоване програмування»

для студентів базового напрямку 6.08.04 "Комп’ютерні науки"

(заочна форма навчання)

Варіант 14

Виконав студент гр. КНз-2

Чалий Михайло

­­

Львів 2014

# Класи та об’єкти в Java

## Мета роботи

Ознайомитися із сучасною парадигмою програмування — об’єктно-орієнтованим програмуванням і реалізацією цієї парадигми в мові Java. Розібратися із правилами запису класів і вивчити їхні додаткові можливості.

## Теоретичні відомості

Об’єктно-орієнтоване програмування розвивається уже більше двадцати років. Існує декілька шкіл, кожна з яких пропонує свій набір принципів роботи з об’єктами і по-своєму викладає ці принципи. Але є декілька загальноприйнятих понять. На даній лабораторній роботі ми їх вивчемо.

### Абстракція

Описуючи поведінку будь-якого об’єкта, наприклад автомобіля, ми будуємо його модель. Модель, як правило, не може описати об’єкт повністю оскільки реальні об’єкти досить складні. Приходиться відбирати лише ті характеристики об’єкта, які є важливими для розв’язання поставленої перед нами задачі. Для опису вантажоперевезень важливою характеристикою буде вантажопідйомність автомобіля, а для опису автомобільних перегонів вона є несуттєвою. Для моделювання перегонів обов’язково необхідно описати метод набирання швидкості даною автівкою, а для вантажоперевезеннь це не важливо.

Ми повинні *абстрагуватися* від деяких конкретних деталей об’єкта. Дуже важливо вибрати правильну степінь абстракції. Поверхнева степінь дасть лише приблизний опис об’єкта, не дозволить правильно моделювати його поведінку. Глибока степінь абстракції зробить модель надто складною, перевантаженою деталями, і тому непридатною.

Опис кожної моделі робиться у вигляді одного або декількох *класів* (classes). Клас можна вважати проектом, зліпком, кресленням, за яким потім будуть створюватися конкретні об’єкти. Клас містить опис змінних і констант, які характеризують об’єкт. Вони називаються *полями класу* (class fields). Процедури, які описують поведінку об’єкта, називаються *методами класу* (class methods). Всередині класу можна описати і *вкладені класи* (nested classes) і *вкладені інтерфейси.* Поля, методи і вкладені класи першого рівня є *членами класу* (class members). Різні школи об’єктно-орієнтованного програмування пропонують різні терміни.

### Ієрархія

Ієрархія об’єктів давно використовується для їх класифікації.

Часто використовують генеалогічну термінологію: батьківський клас, дочірний клас, клас-нащадок, клас-предок.

У цій термінології говорять про *наслідування* (inheritance) класів.

### Відповідальність

В англомовній літературі звернення одного об’екта до іншого описується словом message *"повідомлення"*. Чому не використовується словосполучення "виклик методу", адже говорять: "Виклик процедури"? Тому що між цими поняттями існує, принаймні, три відмінності.

Повідомлення іде до конкретного об’єкта, що знає метод розв’язання задачі, у прикладі цей об’єкт — біжуче значення змінної person. У кожного об’єкта є свій біжучий стан, свої значення полів класу, і це може вплинути на виконання метод.

Спосіб виконання доручення, що міститься в повідомленні, залежить від об’єкта, якому воно адресоване. Один госодар поставить миску з "Chappi", другий кине кістку, третій вижене собаку на вулицю. Цю цікаву властивість називають *поліморфізмом* (polymorphism).

Звернення до методу відбудеться лише на етапі виконання програми, компілятор нічого не знає про метод. Це називається *"пізнім зв’язуванням"* напротивагу *"ранньому зв’язуванню",* при якому процедура приєднується до програми на етапі компонування.

Для того, щоб правильно реалізувати принцип відповідальності, застосовується четвертий принцип об’єктно-орієнтованого програмування — *модульність* (modularity).

### Модульність

Цей принцип стверджує — кожний клас повинен складати окремий модуль. Члени класу, до яких не планується звертання зовні, повинні бути інкапсульовані. В мові Java інкапсуляция досягається додаванням модифікатора private до опису члена класу.

## Завдання

1. Ознайомитися з технологією програмування та оформлення програм на мові Java.
2. Створити новий проект в середовищі Eclipse.
3. Створити клас у новоствореному проекті
4. Побудуваті ієрархію класів: автомобіль, поїзд, транспортний засіб, експрес. Розробити методи і властивості для кожного із розроблюваних класів. Написати тестовий застосунок для демонстрації властивостей класів.
5. Виконати компіляцію програми та запустити на виконання.

## Реалізація

Лістінг 1. Vehicle.java

**package** transport;

**public** **abstract** **class** Vehicle {

**protected** **boolean** beforeStart(){

**return** **true**;

}

**public** **void** start(){

System.*out*.printf("Start engines of the %s...%n", getName());

**if** (beforeStart()){

System.*out*.printf("Engines of the %s started!%n", getName());

}

}

**protected** **void** afterStop(){

}

**public** **void** stop(){

System.*out*.printf("Stop engines of the %s...%n", getName());

afterStop();

}

**abstract** **int** getWheelsNum();

**abstract** String getName();

}

Лістінг 2. Car.java

**package** transport;

**public** **class** Car **extends** Vehicle {

**private** **float** gassLevel = 10.0f;

@Override

**int** getWheelsNum() {

**return** 4;

}

@Override

**protected** **boolean** beforeStart(){

**if** (!enoughGass()){

System.*out*.printf("Not enough gass. Currently have %f%n", gassLevel);

**return** **false**;

}

**return** **super**.beforeStart();

}

**private** **boolean** enoughGass(){

**return** gassLevel > 100.0;

}

**public** **void** refuel(**float** gass){

gassLevel += gass;

System.*out*.printf("Reguel car with %f, now in backs %f%n", gass, gassLevel);

}

@Override

String getName() {

**return** "car";

}

}

Лістінг 3. Train.java

**package** transport;

**public** **abstract** **class** Train **extends** Vehicle {

**private** **int** carNum;

**protected** Train(**int** carNum){

**this**.carNum = carNum;

}

@Override

**int** getWheelsNum() {

**return** carNum \* 8;

}

**int** getCarNum(){

**return** carNum;

}

}

Лістінг 4. Express.java

**package** transport;

**public** **class** Express **extends** Train {

**public** Express(){

**super**(4);

}

@Override

**protected** **boolean** beforeStart(){

infoPassengers("Express is about to start, be carefull!");

**return** **super**.beforeStart();

}

@Override

**public** **void** afterStop(){

infoPassengers("Express arrived. You can leave train.");

}

**private** **void** infoPassengers(String msg){

System.*out*.printf("Inform Passengers: %s%n", msg);

}

@Override

String getName() {

**return** "express train";

}

}

Лістінг 5. Transport.java

**package** transport;

**public** **class** Transport {

**public** **static** **void** main(String[] args) {

Car car1 = **new** Car();

car1.start();

car1.refuel(150.0f);

car1.start();

car1.stop();

Express express1 = **new** Express();

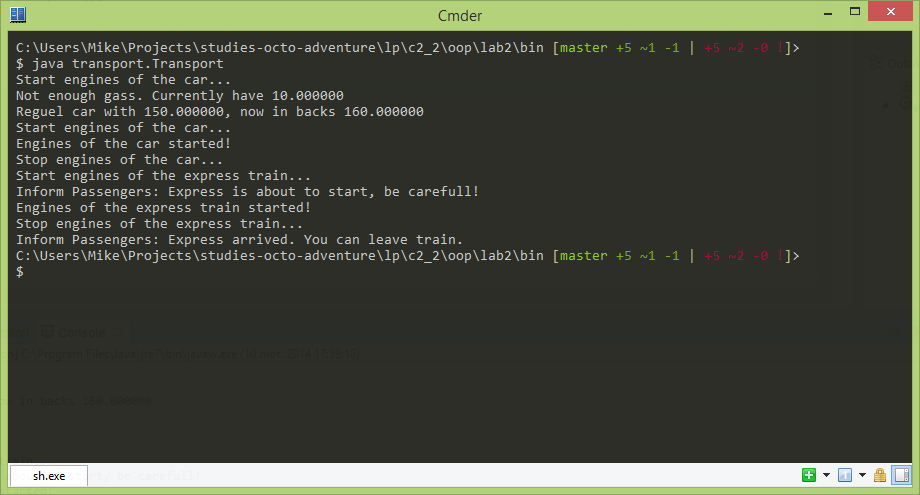
express1.start();

express1.stop();

}

}

## Результат



## Висновки

Ознайомився із сучасною парадигмою програмування — об’єктно-орієнтованим програмуванням і реалізацією цієї парадигми в мові Java. Розібратився із правилами запису класів і вивчив їхні додаткові можливості.