МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

Національний університет «Одеська юридична академія»

Факультет кібербезпеки та інформаційних технологій

Кафедра інформаційних технологій

Протокол практичної роботи №5

з дисципліни об'єктно-орієнтоване програмування

на тему: «Поліморфізм (пізнє зв’язування). Висхідне перетворення»

Виконав студент групи

ІПЗ-212

Корнійчук М. М.

Прийняв

Рудніченко М. Д.

Одеса, 2022

ЗМІСТ

[ВСТУП](#_ki7i591yqske) 3

[ТЕОРЕТИЧНА ЧАСТИНА](#_335513nxkbjl) 4

[ПРАКТИЧНА ЧАСТИНА](#_9cm35ikas1ko) 7

[ВИСНОВОК](#_xlxs31hqwy0n) 12

[ЛІТЕРАТУРА](#_5m79jhxx931e) 13

# 

# ВСТУП

Мета роботи – ознайомитися з механізмом поліморфізму в ООП, розібратися зі статичним та динамічним зв’язуванням, навчитися використовувати висхідне перетворення.

# ТЕОРЕТИЧНА ЧАСТИНА

Полиморфизм является третьей важной особенностью объектно-ориентированных языков, вместе с абстракцией и наследованием.

Он представляет еще одну степень отделения интерфейса от реализации, разъединения ***что*** от ***как***.  Полиморфизм улучшает организацию кода и его читаемость, а также способствует созданию **расширяемых** программ, которые могут «расти» не только в процессе начальной разработки проекта, но и при добавлении новых возможностей.

Для начала, вернемся к наследованию. Самая важная особенность наследования заключается не в том, что оно позволяет новому классу использовать поля и методы базового, а в том, что наследование выражает отношение между новым и базовым классом. Это отношение можно выразить как «Новый класс **является разновидностью** базового класса», это же справедливо и для объектов этих классов. Например, класс **Student** является разновидностью **Person**, класс **Bear** является разновидностью **Animal** и т.д.

Данное отношение поддерживается языком программирования. Например, рассмотрим базовый класс **Instrument**, который представляет музыкальные инструменты, и класс **Guitar**, который представляет музыкальный инструмент гитару. Так как наследование означает, что все методы базового класса также доступны в производном классе, любое сообщение, которое мы можем отправить базовому классу, можно отправить и производному классу. Если в классе **Instrument** есть метод **play()**, то он будет присутствовать и в классе **Guitar**.

Преобразование ссылки на объект Guitar в ссылку на объект Instrument называется восходящим преобразованием типов (upcasting). Восходящее преобразование всегда безопасно, так как это переход от конкретного типа к более общему. Иначе говоря, производный класс является надстройкой базового класса. Он может содержать больше методов, чем базовый класс, но обязан включать в себя все методы базового класса. Присоединение вызова метода к телу метода называется связыванием. Если связывание производится перед запуском программы (на этапе компиляции или компоновки), оно называется ранним (early) или статическим (static) связыванием (binding).

В Java существует механизм для фактического определения типа объекта во время работы программы для вызова подходящего метода. Иначе говоря, компилятор не знает типа объекта, но механизм вызова методов определяет фактический тип объекта и вызывает соответствующее тело метода.

Для всех методов Java используется механизм позднего связывания, если только метод не был объявлен как final (приватные методы являются final по умолчанию).

Итак, подведем итоги:

1) Статическое связывание в Java происходит на этапе компиляции, тогда как динамическое связывание происходит во время выполнения программы (в runtime (рантайме));

2) для private, final и static методов, а также для всех полей используется статическое связывание, тогда как для остальных методов (такие методы в некоторых языках программирования называются виртуальными (virtual)) используется динамическое связывание;

3) в статическом связывании используется тип ссылки, тогда как в динамическом связывании используется фактический тип объекта;

4) перегруженные методы используют статическое связывание, тогда как переопределенные методы используют статическое связывание.

Важно понимать, что если вы используете восходящее преобразование, то вы теряете информацию о полях и методах, которых нет в базовом классе.

# ПРАКТИЧНА ЧАСТИНА

Завдання 1.

1. Создайте класс Person, который должен содержать следующие поля и геттеры\сеттеры для этих полей:

Фамилия;

Имя;

Возраст.

Создайте метод printInfo(), который бы возвращал строку следующего формата: Человек <фамилия> <имя>, возраст: <возраст>

2. Создайте класс Student, который должен наследоваться от класса Person. Добавьте дополнительные поля, и геттеры\сеттеры для этих полей:

Группа;

Номер студенческого билета.

Переопределите метод printInfo(), который должен выводить строку следующего формата:

Студент группы <группа> <фамилия> <имя>, возраст: <возраст>. Номер студенческого билета: <номер>

3. Создайте класс Lecturer, который должен наследоваться от класса Person. Добавьте дополнительные поля, и геттеры\сеттеры для этих полей:

Кафедра;

Зарплата.

Переопределите метод printInfo(), который должен выводить строку следующего формата:

Преподаватель кафедры <кафедра> <фамилия> <имя>, возраст: <возраст>. Зарплата: <зарплата>

4. Используя восходящее преобразование, создайте в классе Main несколько объектов классов Student и Lecturer, после чего создайте массив, который бы мог включать объекты классов Person, Student, Lecturer. Заполните массив объектами этих классов.

5. Используя цикл, обратитесь к элементам массива и выведите в консоль, с помощью метода printInfo(), информацию от каждого объекта.

Код програми (рис. 1):



Рисунок . Код програми

Результат роботи програми наведено на рис. 2-5.

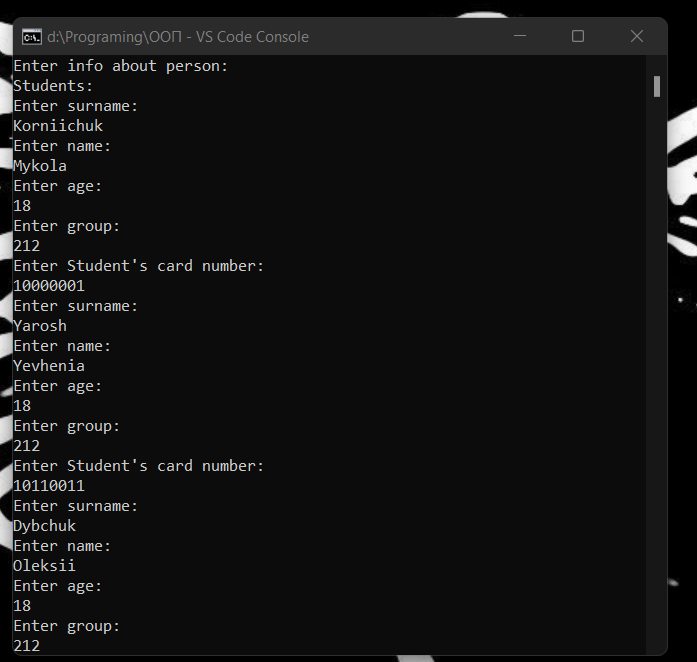


Рисунок . Приклад роботи програми до завдання 1 (введення вихідних даних)

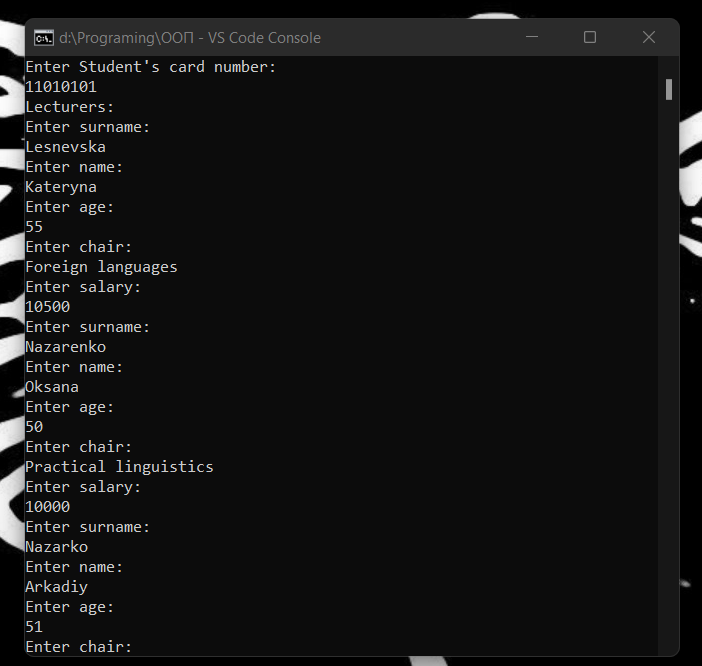


Рисунок . Приклад роботи програми до завдання 1 (введення вихідних даних)

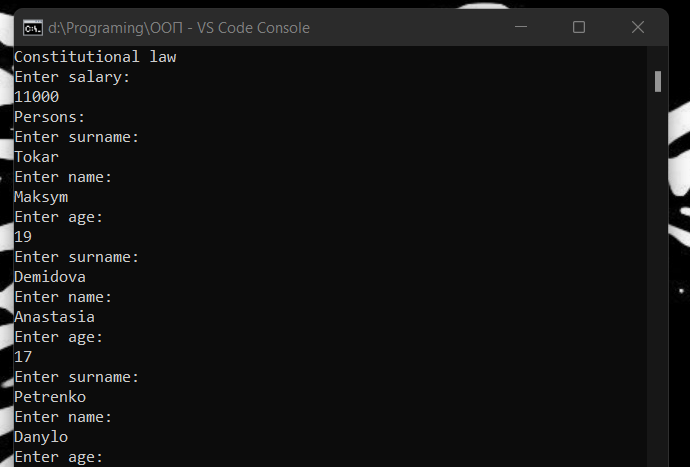


Рисунок . Приклад роботи програми до завдання 1 (введення вихідних даних)

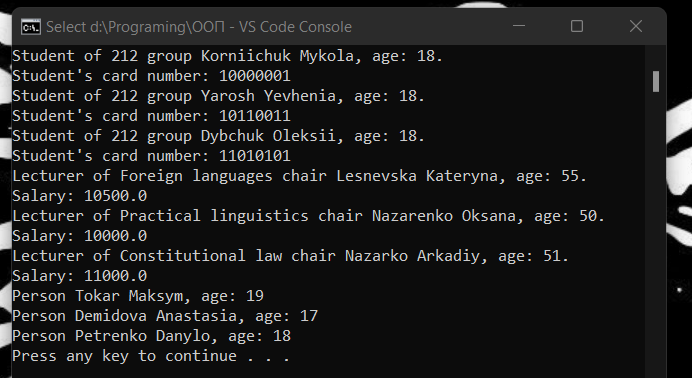


Рисунок . Приклад роботи програми до завдання 1 (вивід масиву)

# ВИСНОВОК

В ході роботи я ознайомився з механізмом поліморфізму в ООП, розібратися зі статичним та динамічним зв’язуванням, навчитися використовувати висхідне та низхідне перетворення для вирішення прикладних задач.

# 

# ЛІТЕРАТУРА

1. Васильев А. Н. Самоучитель Java с примерами и программами. 3-е издание. — СПб.: Наука и Техника, 2016. — 368 с.: ил.

Посилання на github: <https://github.com/chuguystyr/University.OOP/tree/%D0%9F%D0%A0-1>