МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

Національний університет «Одеська юридична академія»

Факультет кібербезпеки та інформаційних технологій

Кафедра інформаційних технологій

Протокол практичної роботи №6

з дисципліни об'єктно-орієнтоване програмування

на тему: «Абстрактні класи та інтерфейси. Механізм зворотного виклику»

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) 6

[ВИСНОВОК](#_xlxs31hqwy0n) 13

[ЛІТЕРАТУРА](#_5m79jhxx931e) 14

# 

# ВСТУП

Мета роботи – ознайомитися з механізмом абстрактних класів та інтерфейсів; розібратися з тим, коли й навіщо необхідно використовувати абстрактні класи й інтерфейси; вивчити використання механізму зворотного виклику.

# ТЕОРЕТИЧНА ЧАСТИНА

Интерфейсы и абстрактные классы улучшают структуру кода и способствуют отделению интерфейса от реализации. В первую очередь необходимо рассмотреть понятие абстрактного класса, который является промежуточной ступенью между обычным классом и интерфейсом.

В примере с классами фигур, методы базового класса Shape всегда были «фиктивными». Попытка вызова метода из класса Shape привела бы к ошибке в программе. Это было связано с тем, что класс Shape был нужен лишь для того, чтобы определить общий интерфейс всех классов, производных от него, а уже производные классы переопределяли эти методы и реализовывали их по-своему.

Класс Shape определял базовую форму, общность всех производных классов. Такие классы как Shape называют абстрактными базовыми классами или просто абстрактными классами.

Если в программе определяется такой абстрактный базовый класс вроде Shape, создание объектов такого класса практически всегда бессмысленно. Абстрактный класс создается для работы с набором классов через общий интерфейс. А если Shape только выражает интерфейс, а создание объектов такого класса не имеет смысла, лучше всего запретить пользователю создавать такие объекты, т.к. он может ненароком создать объект этого класса и попытаться с ним работать, что приведет к ошибкам в программе.

В языке Java для решения подобных задач применяются абстрактные методы. Абстрактный метод является незавершенным; он состоит только из объявления и не имеет тела.

Класс, содержащий абстрактные методы, называется абстрактным классом. Такие класс тоже должны помечаться ключевым словом abstract (в противном случае, компилятор выдает сообщение об ошибке).

Если вы объявите класс, производный от абстрактного класса, но хотите иметь возможность создания новых объектов нового типа, то вы должны переопределить все абстрактные методы базового класса. Если это не будет сделано, то производный класс тоже останется абстрактным, и компилятор заставит пометить новый класс ключевым словом abstract.

Можно обозначить класс ключевым словом abstract даже тогда, когда в нем нет ни одного абстрактного метода. Это бывает полезно, когда необходимо просто запретить создание экземпляров этого класса. Объявление класса как abstract не подразумевает, что все его методы должны быть абстрактными.

Создавать абстрактные классы и методы необходимо, т.к. они подчеркивают абстрактность класса, а также сообщают и пользователю класса, и компилятору, как следует с ним обходиться. Кроме того, абстрактные классы играют полезную роль при переработке программ, потому что они позволяют легко перемещать общие методы вверх по иерархии наследования.

Ключевое слово interface становится следующим шагом на пути к абстракции. Ключевое слово abstract позволяет создать в классе один или несколько неопределенных методов – разработчик предоставляет часть интерфейса без реализации, которая должна предоставляться производными классами. Ключевое слово interface используется для создания полностью абстрактных классов, вообще не имеющих реализации. Создатель интерфейса определяет имена методов, списки аргументов и типы возвращаемых значений, но не тела методов. Интерфейс описывает форму, но не реализацию.

Ключевое слово interface фактически означает: «Именно так должны выглядеть все классы, которые реализуют данный интерфейс». Поэтому любой код, использующий конкретный интерфейс, знает только то, какие методы вызываются для этого интерфейса, но не более того. Интерфейс определяет своего рода «протокол взаимодействия» между классами.

Кроме этого, в отличие от абстрактного класса, интерфейс позволяет реализовать, своего рода, множественное наследование.

Чтобы создать интерфейс, используйте ключевое слово interface вместо class. Как и в случае с классами, перед словом interface указывается модификатор доступа (public, protected и т.д.). Интерфейс также может содержать поля, они автоматически являются статическими (static) и неизменяемыми (final).

Для создания класса, реализующего определенный интерфейс (или группу интерфейсов), используется ключевое слово implements. Фактически это означает «Интерфейс определяет форму, а здесь будет показано, как это работает». В остальном, это обычное наследование. В классе, который реализует интерфейс, реализуемые методы должны быть объявлены как public.

Когда метод работает с классом вместо интерфейса, мы ограничены использованием базового класса и его подклассами. Это исключает возможность использовать метод для класса, который не входит в эту иерархию. Интерфейс, в значительной степени ослабляет это ограничение. В результате код становится более универсальным.

В Java допускается объявлять переменные ссылочного интерфейсного типа, т.е. переменные, хранящие ссылки на интерфейс. Такая переменная может ссылаться на любой объект, реализующий ее тип интерфейса. При вызове метода для объекта по интерфейсной ссылке выполняется вариант этого метода, реализованный в классе данного объекта. Этот процесс аналогичен применению ссылки на суперкласс для доступа к объекту подкласса.

Так как интерфейс по определению не имеет реализации, нет ничего, что могло бы помешать совмещению нескольких интерфейсов. При объявлении класса, который совмещает несколько интерфейсов, имена интерфейсов перечисляются вслед за ключевым словом implements и разделяются запятыми.

Наследование позволяет легко добавить в интерфейс объявления новых методов, а также совместить несколько интерфейсов в одном.

Механизм обратного вызова широко распространен в программировании. При обратном вызове, программист задает действия, которые должны выполняться всякий раз, когда происходит некоторое событие. Например, можно задать действие, которое должно быть выполнено после клика на кнопку или при выборе определенного пункта меню.

Слушатель – это объект, который, как бы, «слушает» события, которые происходят с другим объектом. И когда это «слушаемое» событие происходит, вызывается указанный в интерфейсе метод этого объекта.

# ПРАКТИЧНА ЧАСТИНА

Завдання 1.

1. Переделайте лабораторную работу 5 с использованием механизма абстрактных классов.

Код програми до завдання 1.1 (рис. 1)

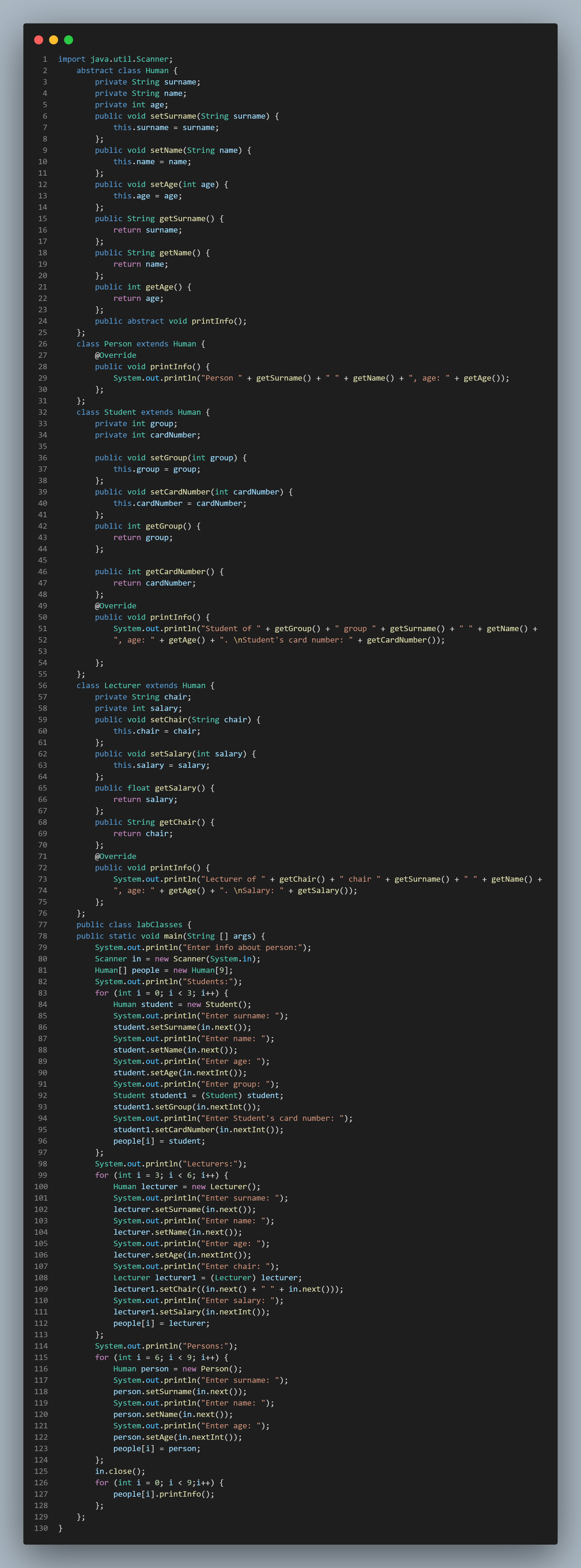


Рисунок 1. Код програми до завдання 1.1

Результат роботи програми наведено на рис. 2-5.

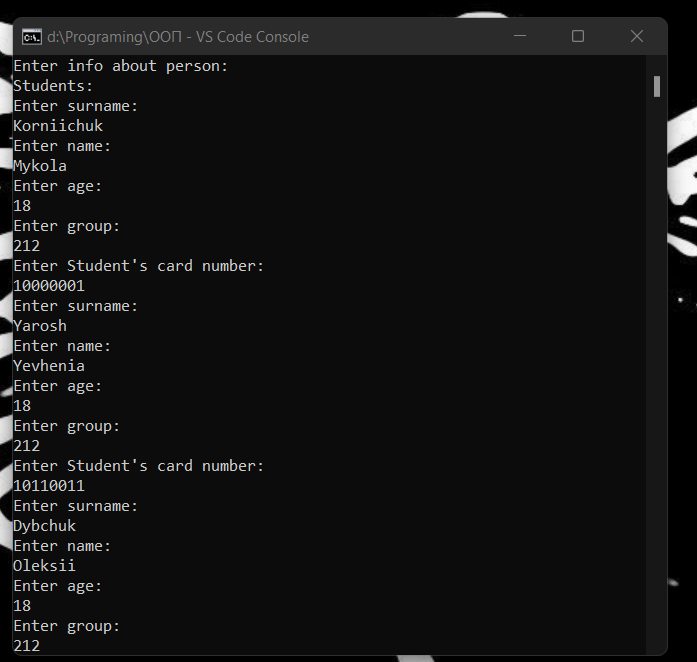


Рисунок 2. Приклад роботи програми до завдання 1.1 (введення вихідних даних)

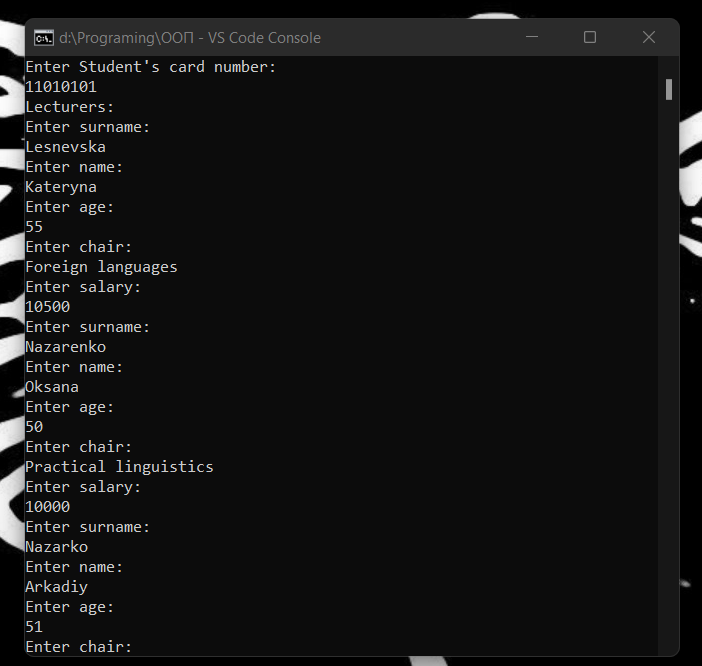


Рисунок 3. Приклад роботи програми до завдання 1.1 (введення вихідних даних)

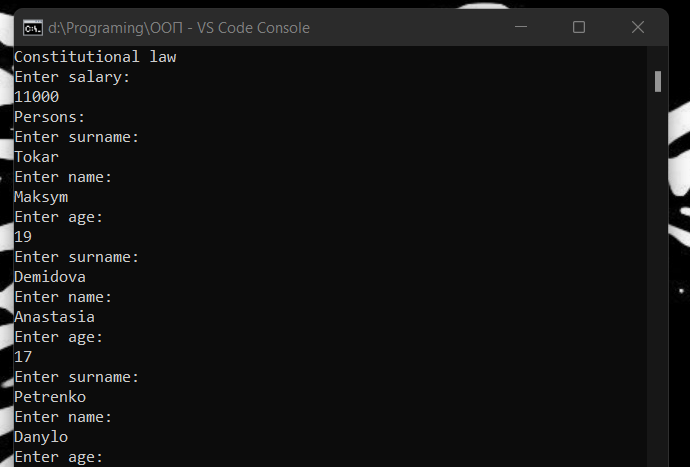


Рисунок 4. Приклад роботи програми до завдання 1.1 (введення вихідних даних)

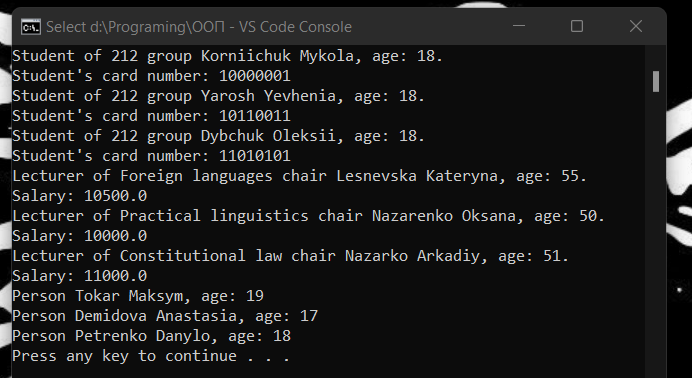


Рисунок 5. Приклад роботи програми до завдання 1.1 (вивід масиву)

2. Переделайте лабораторную работу 5 с использованием механизма интерфейсов;

Код програми (рис. 6):



Рисунок 6. Код програми до завдання 1.2

Результат роботи програми наведено на рис. 7-10.

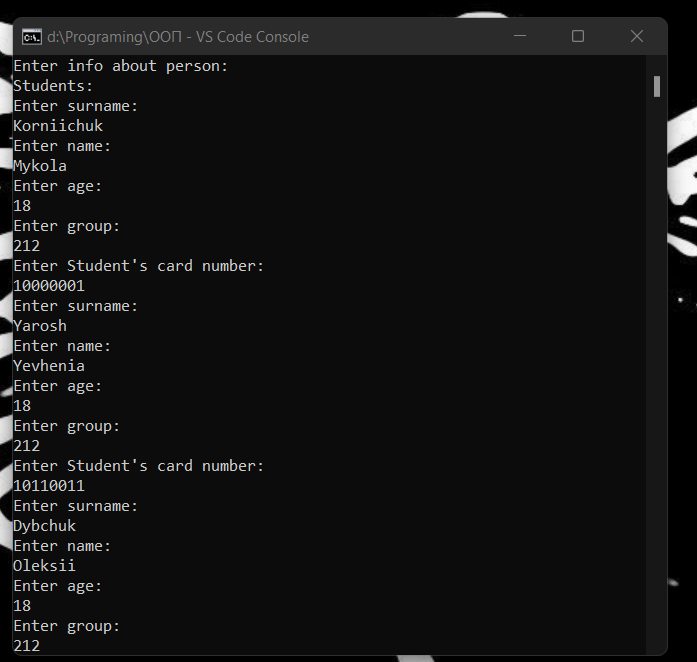


Рисунок 7. Приклад роботи програми до завдання 1.1 (введення вихідних даних)

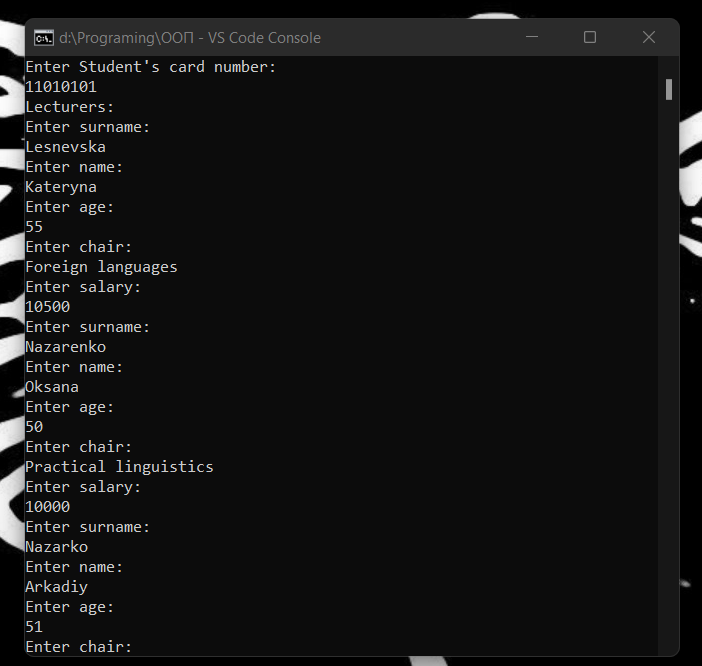


Рисунок 8. Приклад роботи програми до завдання 1.1 (введення вихідних даних)

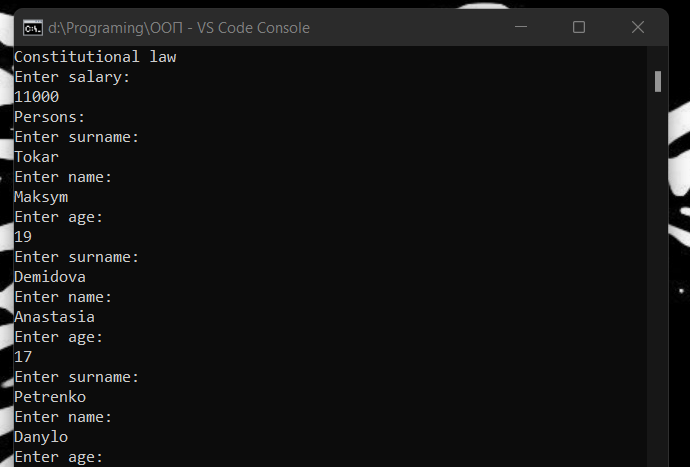


Рисунок 9. Приклад роботи програми до завдання 1.1 (введення вихідних даних)

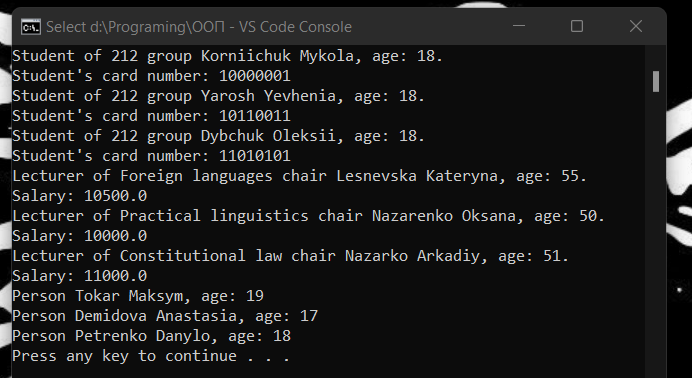


Рисунок 10. Приклад роботи програми до завдання 1.1 (вивід масиву)

Завдання 2.

Независимо от варианта, необходимо следующее:

• В пакете carshop.cars создать абстрактный класс Саг, содержащий следующие поля и методы:

int speed;

boolean isSellOut;

double regularPrice;

String color;

double getSalePrice() ;

• В пакете carshop.cars создать класс Truck, расширяющий класс Саг и содержащий следующие поля и методы:

int weight;

// если вес больше 2000, скидка 10%.

double getSalePrice() ;

• В пакете carshop.cars создать класс Ford, расширяющий класс Саг, содержащий следующие поля и методы:

int year;

int manufacturerDiscount;

// (вычисляется как цена минус скидка производителя)

double getSalePrice();

• В пакете carshop.cars создать класс Sedan, расширяющий класс Саг и содержащий следующие поля и методы:

int length;

// (если длина больше 20, скидка 5%)

double getSalePrice();

• В пакете carshop.interface создать два интерфейса Admin и Customer, содержащие следующие методы:

Admin:

getIncome() - суммарная цена проданных автомобилей

Customer

getCarsPrice()

getCarColors()

getCarPrice(int id)

getCarColor(int id)

purchaseCar(int id)

• В пакете carshop.impl создать класс MyOwnAutoShop реализующий интерфейсы Admin и Customer. Он содержит объект класса Sedan, 2 объекта класса Ford, 2 объекта класса Truck. Объекты хранить в виде массива. В этом случае id в методах будет индексом элемента в массиве.

• В пакете carshop, создать класс, содержащий статический метод main, который должен демонстрировать работу приложения.

Завдання 3. Необходимо описать интерфейс, содержащий одну функцию: int count(int number);

Данный интерфейс позволяет вв1числятв характеристики чисел. Необходимо реализовать два класса для этого интерфейса. Первый класс должен вычислить количество символов в десятичной записи для этого числа. Второй класс должен вычислить количество различных простых чисел, являющихся множителями данного числа. Необходимо, чтобы приложение запросило у пользователя число и выдало статистические сведения для обоих классов.

Код програми до завдання 2 (рис. 11-18):



Рисунок 11. Код абстрактного класу Car

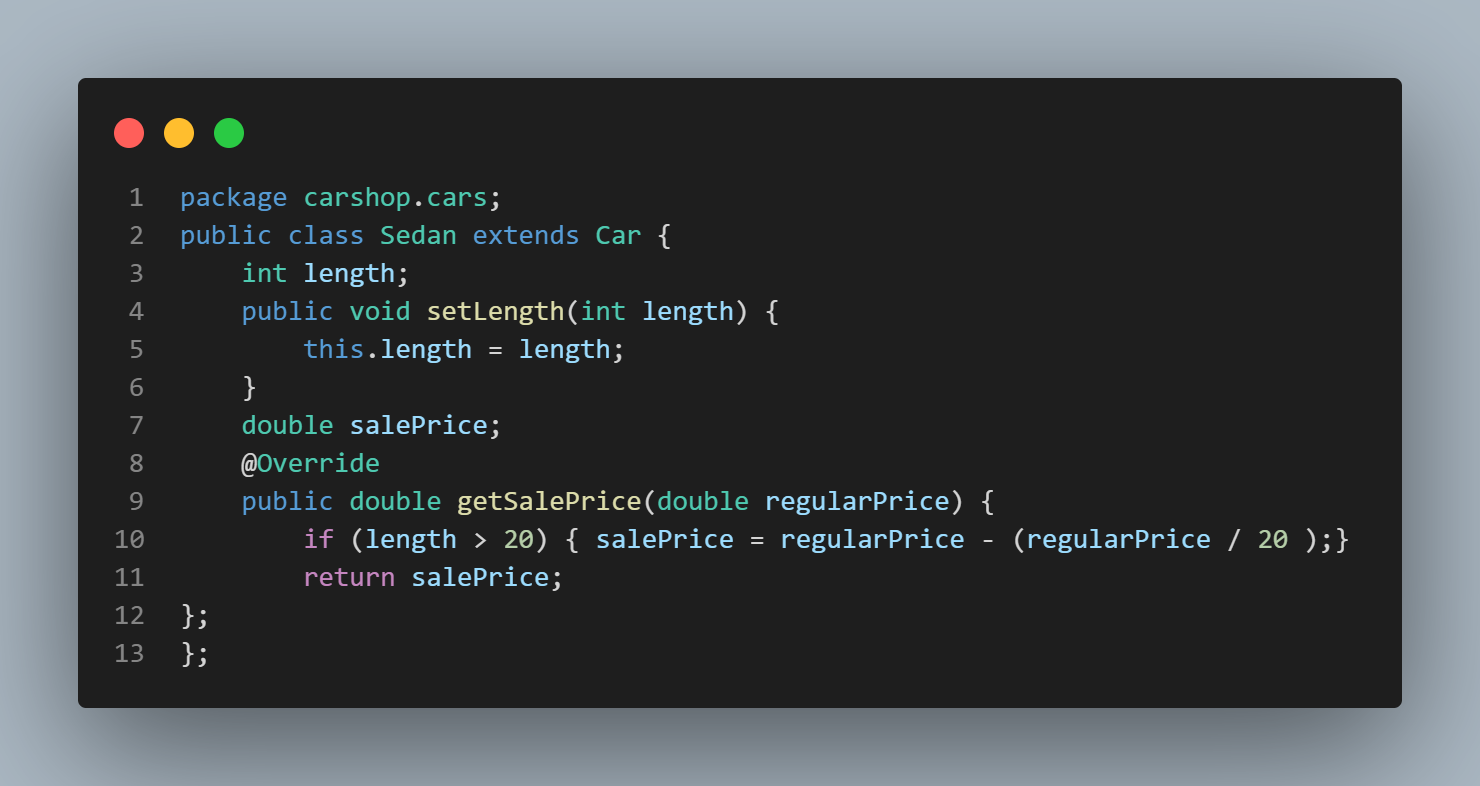


Рисунок 12. Код класу Sedan



Рисунок 13. Код класу Ford



Рисунок 14. Код класу Truck

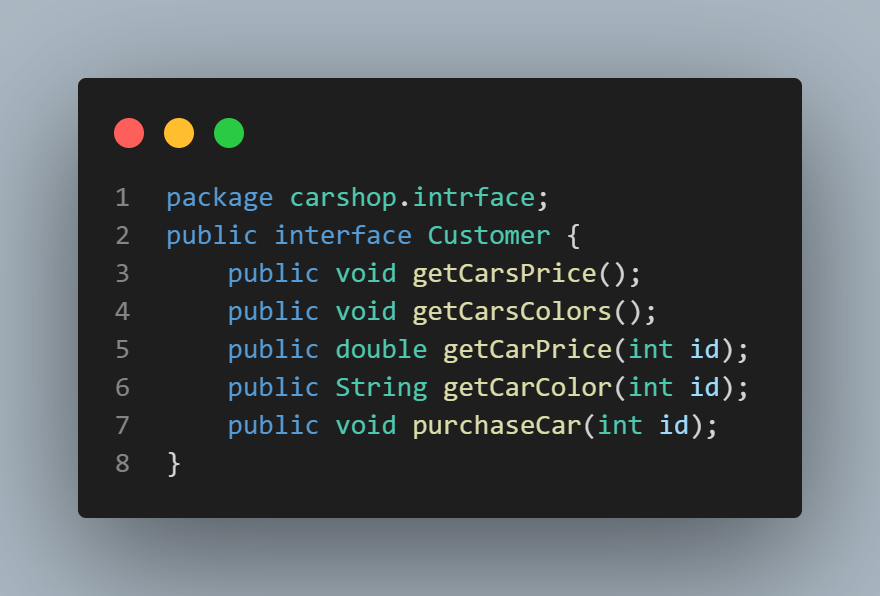


Рисунок 15. Код інтерфейсу Customer

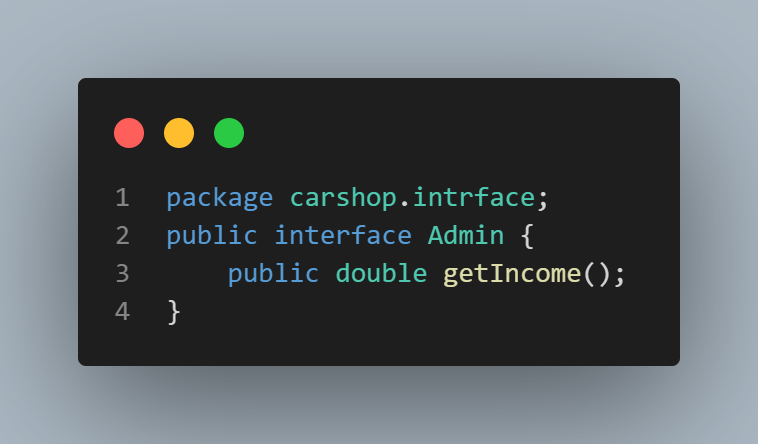


Рисунок 16. Код інтерфейсу Admin

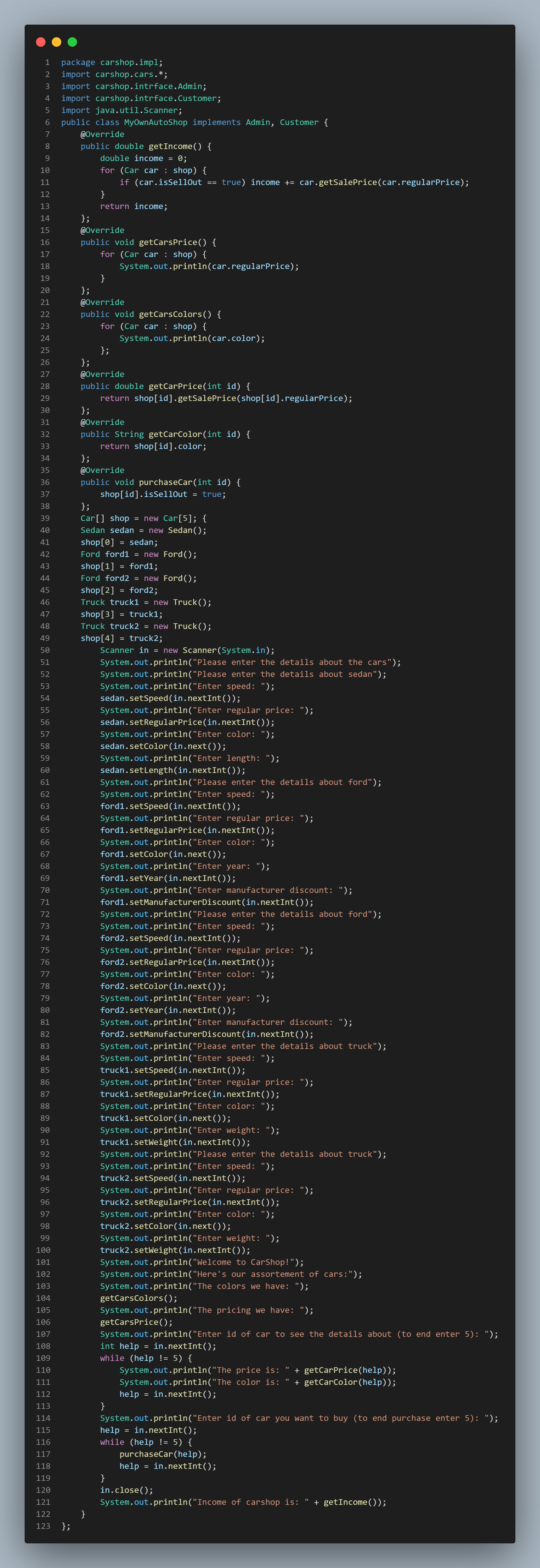


Рисунок 17. Код, що імплементує класи та інтерфейси

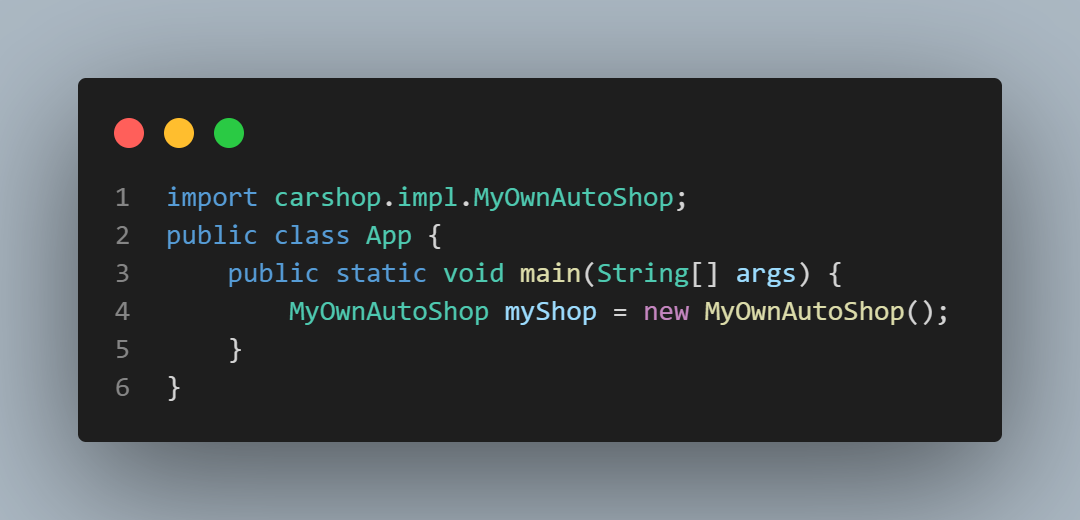


Рисунок 18. Код головного класу застосунка

Результат виконання програми наведено на рис. 19:

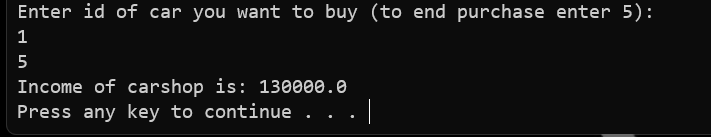


Рисунок 19. Приклад роботи програми до завдання 2

Код програми до завдання 3 (рис. 20):



Рисунок 20. Код програми до завдання 3

Результат роботи програми наведено на рис. 21:

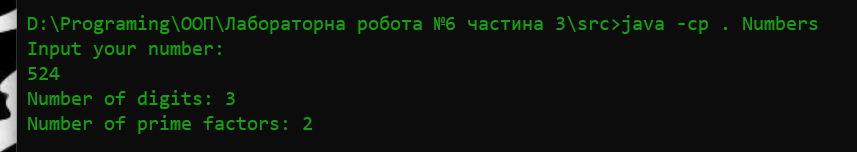


Рисунок 21. Приклад роботи програми до завдання 3

# 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>