**Класс банковского счета с инкапсуляцией: все ключевые поля (номер, баланс, PIN)**

**закрыты двойным подчеркиванием**

class Account:

def \_\_init\_\_(self, account\_number, balance, pin):

# Инициализация приватных атрибутов

self.\_\_account\_number = account\_number

self.\_\_balance = balance

self.\_\_pin = pin

# Метод для пополнения счета

def deposit(self, amount, pin):

if pin == self.\_\_pin:

if amount > 0:

self.\_\_balance += amount

print(f"Счет пополнен на {amount}")

else:

print("Сумма должна быть положительной")

else:

print("Неверный PIN")

# Метод для снятия средств

def withdraw(self, amount, pin):

if pin == self.\_\_pin:

if 0 < amount <= self.\_\_balance:

self.\_\_balance -= amount

print(f"Снято со счета {amount}")

else:

print("Недостаточно средств или некорректная сумма")

else:

print("Неверный PIN")

# Метод для запроса баланса

def get\_balance(self, pin):

if pin == self.\_\_pin:

print(f'Баланс: {self.\_\_balance}')

return self.\_\_balance

else:

print("Неверный PIN")

return None

# Создание объекта счета

acc = Account('6000', 2000, 45678)

# Пополнение, снятие, проверка баланса

acc.deposit(2000, 45678)

acc.withdraw(200, 45678)

acc.get\_balance(45678)

**# Класс товара, демонстрирует инкапсуляцию цены**

class Product:

def \_\_init\_\_(self, name, price):

self.name = name

self.\_\_price = price

# Метод для установки скидки

def set\_discount(self, percent):

if percent < 0:

print("Скидка не может быть отрицательной.")

return

discount\_amount = self.\_\_price \* (percent / 100)

new\_price = self.\_\_price - discount\_amount

if new\_price < 0:

self.\_\_price = 0

else:

self.\_\_price = new\_price

print(f"Для {self.name} применена скидка {percent}%. Новая цена: {self.\_\_price}")

# Метод для получения финальной цены

def final\_price(self):

return self.\_\_price

# Примеры товаров и скидки

product1 = Product("Телефон", 13500)

product2 = Product("Книга", 870)

product3 = Product("Чашка", 230)

product4 = Product("Лампа", 950)

product1.set\_discount(30)

product2.set\_discount(80)

product3.set\_discount(60)

product4.set\_discount(50)

print("Финальная цена на телефон:", product1.final\_price())

print("Финальная цена на книгу:", product2.final\_price())

print("Финальная цена на чашку:", product3.final\_price())

print("Финальная цена на лампу:", product4.final\_price())

**# Класс курса, реализует добавление и удаление студентов, ограничение по количеству мест**

class Course:

def \_\_init\_\_(self, name, max\_seats):

self.\_\_name = name

self.\_\_students = []

self.\_\_max\_seats = max\_seats

# Метод для добавления студента

def add\_student(self, name):

if len(self.\_\_students) < self.\_\_max\_seats:

self.\_\_students.append(name)

print(f"{name} добавлен в курс.")

else:

print("Нет свободных мест!")

# Метод для удаления студента

def remove\_student(self, name):

if name in self.\_\_students:

self.\_\_students.remove(name)

print(f"{name} удалён из курса.")

else:

print(f"{name} нет в списке курса.")

# Получение списка всех студентов

def get\_students(self):

return self.\_\_students.copy()

# Использование курса

my\_course = Course("Программирование на Python", 2)

my\_course.add\_student("Василий")

my\_course.add\_student("Милана")

my\_course.add\_student("Нурс")

print("Студенты курса:", my\_course.get\_students())

my\_course.remove\_student("Камилла")

my\_course.remove\_student("Алтынай")

print("Обновленный список:", my\_course.get\_students())

**# Класс смарт-часов — показывает, как работает изменение приватного поля**

class SmartWatch:

def \_\_init\_\_(self, battery):

self.\_\_battery = battery

# Использование аккумулятора

def use(self, minutes):

used\_percent = minutes // 10

if used\_percent > self.\_\_battery:

used\_percent = self.\_\_battery

self.\_\_battery -= used\_percent

print(f"Использовано {minutes} минут. Заряд уменьшился на {used\_percent}%. Осталось: {self.\_\_battery}%.")

# Зарядка аккумулятора

def charge(self, percent):

if percent < 0:

print("Нельзя добавить отрицательный заряд.")

return

self.\_\_battery += percent

if self.\_\_battery > 100:

self.\_\_battery = 100

print(f"Заряд увеличен на {percent}%. Теперь: {self.\_\_battery}%.")

# Получение текущего заряда

def get\_battery(self):

print(f"Текущий заряд: {self.\_\_battery}%")

return self.\_\_battery

watch = SmartWatch(100)

watch.use(35)

watch.charge(40)

watch.use(160)

watch.charge(-20)

watch.get\_battery()

**# Классы транспорта, пример наследования и переопределения метода**

class Transport:

def \_\_init\_\_(self, speed, capacity):

self.speed = speed

self.capacity = capacity

# Метод расчёта времени поездки

def travel\_time(self, distance):

time = distance / self.speed

print(f"Поездка на {type(self).\_\_name\_\_}: {time:.2f} часов")

return time

class Bus(Transport):

pass

class Train(Transport):

pass

class Airplane(Transport):

# Переопределённый метод для самолёта (ускорение полёта)

def travel\_time(self, distance):

time = super().travel\_time(distance)

air\_time = time \* 0.8

print(f"Поездка на Airplane с учётом ускорения: {air\_time:.2f} часов")

return air\_time

bus = Bus(speed=40, capacity=50)

train = Train(speed=100, capacity=300)

plane = Airplane(speed=600, capacity=200)

distance = 600

bus.travel\_time(distance)

train.travel\_time(distance)

plane.travel\_time(distance)

**# Классы персонажей — каждый имеет индивидуальный способ атаки**

class Character:

def \_\_init\_\_(self, name, health, attack):

self.name = name

self.health = health

self.attack = attack

def attack\_method(self):

pass

class Warrior(Character):

def attack\_method(self):

print(f"{self.name} брызгает яд {self.attack} -1умер!")

class Mage(Character):

def attack\_method(self):

print(f"{self.name} создает ураган {self.attack} почти смерть!")

class Archer(Character):

def attack\_method(self):

print(f"{self.name} создает гигантов {self.attack} умрешь не умрешь!")

w = Warrior("Баба Яга", 120, 30)

m = Mage("Кикимора", 80, 50)

a = Archer("Шамаханская царица", 100, 40)

w.attack\_method()

m.attack\_method()

a.attack\_method()

**# Абстрактный класс MediaFile — механизм для наследования типовых файлов**

from abc import ABC, abstractmethod

class MediaFile(ABC):

def \_\_init\_\_(self, title, duration):

self.title = title

self.\_\_duration = duration

@property

def duration(self):

return self.\_\_duration

@abstractmethod

def play(self):

pass

class AudioFile(MediaFile):

def play(self):

print(f"Воспроизводится аудиофайл: {self.title} ({self.duration} мин)")

class VideoFile(MediaFile):

def play(self):

print(f"Воспроизводится с изображением: {self.title} ({self.duration} мин)")

class Podcast(MediaFile):

def play(self):

print(f"Воспроизводится эпизод подкаста: {self.title} ({self.duration} мин)")

media\_files = [

AudioFile("Любимая песня", 2),

VideoFile("Увлекательный фильм", 90),

Podcast("Научный подкаст", 50),

]

for file in media\_files:

file.play()

**# Абстрактный класс платёжной системы с разными способами оплаты**

from abc import ABC, abstractmethod

class PaymentSystem(ABC):

@abstractmethod

def process\_payment(self, amount):

pass

class CreditCardPayment(PaymentSystem):

def process\_payment(self, amount):

print(f"Оплата {amount} сом. с кредитной карты проведена успешно")

class CryptoPayment(PaymentSystem):

def process\_payment(self, amount):

print(f"Переведено {amount} в криптовалюте $")

class BankTransfer(PaymentSystem):

def process\_payment(self, amount):

print(f"Банковский перевод на сумму {amount} сом. выполнен")

payments = [

CreditCardPayment(),

CryptoPayment(),

BankTransfer()

]

amounts = [1000, 50.0, 35000]

for p, a in zip(payments, amounts):

p.process\_payment(a)

**# Абстрактный класс Animal — каждый подкласс реализует свои методы eat и sleep**

from abc import ABC, abstractmethod

class Animal(ABC):

@abstractmethod

def eat(self):

pass

@abstractmethod

def sleep(self):

pass

class Lion(Animal):

def eat(self):

print("Лев ловит и ест мясо")

def sleep(self):

print("Лев спит в тени")

class Elephant(Animal):

def eat(self):

print("Слон ест листья и траву")

def sleep(self):

print("Слон спит стоя")

class Snake(Animal):

def eat(self):

print("Змея проглатывает добычу целиком")

def sleep(self):

print("Змея сворачивается в кольцо и спит")

animals = [

Lion(),

Elephant(),

Snake()

]

for animal in animals:

animal.eat()

animal.sleep()

**# Абстрактный класс Document — имитируем работу с разными типами документов**

from abc import ABC, abstractmethod

class Document(ABC):

@abstractmethod

def open(self):

pass

@abstractmethod

def edit(self):

pass

@abstractmethod

def save(self):

pass

class WordDocument(Document):

def open(self):

print("Word-документ открыт")

def edit(self):

print("Word-документ: редактирование невозможно")

def save(self):

print("Word-документ сохранён")

class PdfDocument(Document):

def open(self):

print("PDF-документ открыт")

def edit(self):

print("PDF-документ: редактирование невозможно")

def save(self):

print("PDF-документ сохранён")

class SpreadsheetDocument(Document):

def open(self):

print("Таблица Excel открыта")

def edit(self):

print("Таблица Excel редактируется")

def save(self):

print("Таблица Excel сохранена")

documents = [

WordDocument(),

PdfDocument(),

SpreadsheetDocument()

]

for doc in documents:

doc.open()

doc.edit()

doc.save()

**# Абстрактный класс Lesson — уникальная реализация метода start(**)

from abc import ABC, abstractmethod

class Lesson(ABC):

@abstractmethod

def start(self):

pass

class VideoLesson(Lesson):

def start(self):

print("Видеоурок начинается. Включите видео.")

class QuizLesson(Lesson):

def start(self):

print("Тест начался. Ответьте на вопросы.")

class TextLesson(Lesson):

def start(self):

print("Практическая часть урока начинается. Читайте внимательно.")

lessons = [

VideoLesson(),

QuizLesson(),

TextLesson()

]

for lesson in lessons:

lesson.start()

**# Абстрактный класс Notification — разные типы уведомлений реализуют метод send**

from abc import ABC, abstractmethod

class Notification(ABC):

@abstractmethod

def send(self, message):

pass

class EmailNotification(Notification):

def send(self, message):

print(f"Отправка письма: {message}")

class SMSNotification(Notification):

def send(self, message):

print(f"Отправка SMS: {message}")

class PushNotification(Notification):

def send(self, message):

print(f"Push-уведомление: {message}")

notifications = [

EmailNotification(),

SMSNotification(),

PushNotification()

]

for n in notifications:

n.send("Ваша отправка одобрена!")

**# Классы геометрических фигур — у каждой свой метод вычисления периметра**

class Square:

def \_\_init\_\_(self, side):

self.side = side

def perimeter(self):

return 4 \* self.side

class Circle:

def \_\_init\_\_(self, radius):

self.radius = radius

def perimeter(self):

return 2 \* 3.14 \* self.radius

class Triangle:

def \_\_init\_\_(self, a, b, c):

self.a = a

self.b = b

self.c = c

def perimeter(self):

return self.a + self.b + self.c

shapes = [

Square(5),

Circle(3),

Triangle(3, 4, 5)

]

for shape in shapes:

print("Периметр:", shape.perimeter())

**# Абстрактный класс сотрудника — разные роль через метод work()**

from abc import ABC, abstractmethod

class Employee(ABC):

@abstractmethod

def work(self):

pass

class Manager(Employee):

def work(self):

print("Менеджер организует работу команды")

class Developer(Employee):

def work(self):

print("Разработчик пишет код")

class Designer(Employee):

def work(self):

print("Дизайнер создает дизайн-проекты")

employees = [

Manager(),

Developer(),

Designer()

]

for emp in employees:

emp.work()

**# Абстрактный класс заклинания — метод cast выводит эффект на цель**

from abc import ABC, abstractmethod

class Spell(ABC):

@abstractmethod

def cast(self, target):

pass

class FireSpell(Spell):

def cast(self, target):

print(f"{target}: наносит урон с помощью посоха!")

class IceSpell(Spell):

def cast(self, target):

print(f"{target}: замораживает!")

class HealingSpell(Spell):

def cast(self, target):

print(f"{target}: благодаря ему живешь!")

spells = [

FireSpell(),

IceSpell(),

HealingSpell()

]

for spell in spells:

spell.cast("Герой")