|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |
| Институт кибербезопасности и цифровых технологий |
| Кафедра КБ-2 «Прикладные информационные технологии» |

**ПРАКТИЧЕСКАЯ РАБОТА №4 по дисциплине**

**«Технологии и методы программирования»**

***(наименование дисциплины)***

**Тема практической работы   
«Итератор. Посетитель»**

Студент группы: БИСО-01-20 Давыдова Д.А.

*(шифр учебной группы) (Фамилия И.О)*

Преподаватель: Лесько С.А.

*(Фамилия И.О)*

Москва, 2023

Практическая работа №4

[**Стратегия** 3](#_Toc134720515)

[**Шаблонный метод** 6](#_Toc134720516)

**Итератор**

Листинг программы:

from abc import ABC, abstractmethod

from typing import List

class applesItem:

    def \_\_init\_\_(self, number):

        self.number = number

    def \_\_str\_\_(self):

        return f"яблочко имеет номер: {self.number}"

class Iterator(ABC):

    @abstractmethod

    def next(self) -> applesItem:

        ...

    @abstractmethod

    def has\_next(self) -> bool:

        ...

class applesnumIterator(Iterator):

    def \_\_init\_\_(self, apples: List[applesItem]):

        self.\_apples = apples

        self.\_index = 0

    def next(self) -> applesItem:

        apples\_item = self.\_apples[self.\_index]

        self.\_index += 1

        return apples\_item

    def has\_next(self) -> bool:

        return False if self.\_index >= len(self.\_apples) else True

class applesAggregate:

    def \_\_init\_\_(self, amount\_num: int = 10):

        self.num = [applesItem(it+1) for it in range(amount\_num)]

        print(f"В корзинке для фруктов "

              f"{amount\_num} яблок")

    def amount\_num(self) -> int:

        return len(self.num)

    def iterator(self) -> Iterator:

        return applesnumIterator(self.num)

if \_\_name\_\_ == "\_\_main\_\_":

    apples = applesAggregate(10)

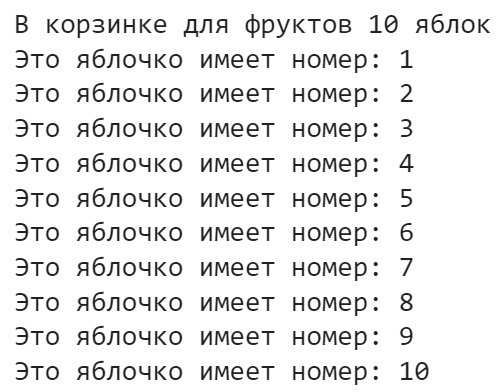
    iterator = apples.iterator()

    while iterator.has\_next():

        item = iterator.next()

        print("Это " + str(item))

Результат работы алгоритма:



**Посетитель**

Листинг программы:

from abc import ABC, abstractmethod

from typing import List

class OrderItemVisitor(ABC):

    @abstractmethod

    def visit(self, item) -> float:

        ...

class ItemElement(ABC):

    @abstractmethod

    def accept(self, visitor: OrderItemVisitor) -> float:

        ...

class Cake(ItemElement):

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

        self.name = name

        self.price = price

    def get\_price(self) -> float:

        return self.price

    def accept(self, visitor: OrderItemVisitor) -> float:

        return visitor.visit(self)

class Drink(ItemElement):

    def \_\_init\_\_(self, name: str, price: float, capacity: float):

        self.name = name

        self.price = price

        self.capacity = capacity

    def get\_price(self) -> float:

        return self.price

    def get\_capacity(self) -> float:

        return self.capacity

    def accept(self, visitor: OrderItemVisitor) -> float:

        return visitor.visit(self)

class WithOutDiscountVisitor(OrderItemVisitor):

    def visit(self, item: ItemElement) -> float:

        cost = 0

        if isinstance(item, Cake):

            cost = item.get\_price()

        elif isinstance(item, Drink):

            cost = item.get\_capacity() \* item.get\_price()

        return cost

class OnlyTortDiscountVisitor(OrderItemVisitor):

    def visit(self, item: ItemElement) -> float:

        cost = 0

        if isinstance(item, Cake):

            cost = item.get\_price()

            cost -= cost \* 0.20 #Скидка 20% на торт

        elif isinstance(item, Drink):

            cost = item.get\_capacity() \* item.get\_price()

        return cost

class OnlyDrinkDiscountVisitor(OrderItemVisitor):

    def visit(self, item: ItemElement) -> float:

        cost = 0

        if isinstance(item, Cake):

            cost = item.get\_price()

        elif isinstance(item, Drink):

            cost = item.get\_capacity() \* item.get\_price()

            cost -= cost \* 0.40 #Скидка 40% на кофе

        return cost

class AllDiscountVisitor(OrderItemVisitor):

    def visit(self, item: ItemElement) -> float:

        cost = 0

        if isinstance(item, Cake):

            cost = item.get\_price()

        elif isinstance(item, Drink):

            cost = item.get\_capacity() \* item.get\_price()

        cost -= cost \* 0.50 #Скидка 50% на все товары

        return cost

class Waiter:

    def \_\_init\_\_(self, discount: OrderItemVisitor):

        self.order: List[ItemElement] = []

        self.discount\_calculator = discount

    def set\_order(self, order: List[ItemElement]) -> None:

        self.order = order

    def set\_discount(self, discount: OrderItemVisitor) -> None:

        self.discount\_calculator = discount

    def calculate\_finish\_price(self) -> float:

        price = 0

        if self.order:

            for item in self.order:

                price += item.accept(self.discount\_calculator)

        return price

order: List[ItemElement] = [Cake("Napoleon cake", 10.1),

                            Drink("Latte", 5, 0.3),

                            Cake("Medovik", 9.5),

                            Drink("Cappuccino", 4, 0.5),

                            Cake("Cheesecake", 11.9),

                            Cake("Tiramisu", 15.2)]

discount = WithOutDiscountVisitor()

waiter = Waiter(discount)

waiter.set\_order(order)

n=1

while n==1:

    print("Choose the item: Without, Cake, Drink, All")

    choice=input()

    while choice not in ("Without", "Cake", "Drink", "All"):

        print("Re-enter")

        choice=input()

    if choice=="Without":

        discount = WithOutDiscountVisitor()

        waiter = Waiter(discount)

        waiter.set\_order(order)

        print(f"Order amount excluding discounts: "

              f"{round(waiter.calculate\_finish\_price(),2)}")

    elif choice=="Cake":

        discount = OnlyTortDiscountVisitor()

        waiter.set\_discount(discount)

        print(f"The amount of the order including the discount on the cake: "

              f"{round(waiter.calculate\_finish\_price(),2)}")

    elif choice=="Drink":

        discount = OnlyDrinkDiscountVisitor()

        waiter.set\_discount(discount)

        print(f"The amount of the order including the discount on the drink: "

              f"{round(waiter.calculate\_finish\_price(),2)}")

    elif choice=="All":

        discount = AllDiscountVisitor()

        waiter.set\_discount(discount)

        print(f"The amount of the order with a discount on everything: "

              f"{round(waiter.calculate\_finish\_price(),2)}")

    print("Do you want to repeat? 1-Да 2-Нет")

    n = int(input())

if n == 2:

  print("Thanks for the order!")

Результат работы алгоритма:

