|  |
| --- |
| 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 №5 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 №5

[**Абстрактная фабрика** 3](#_Toc135815614)

[**Строитель** 5](#_Toc135815615)

[**Адаптер** 7](#_Toc135815616)

[**Посредник** 8](#_Toc135815617)

**Абстрактная фабрика**

Листинг программы:

from abc import ABC, abstractmethod

class Legs(ABC):

    def \_\_init\_\_(self, object: str):

        self.\_object = object

    @abstractmethod

    def create(self): pass

class Cap(ABC):

    def \_\_init\_\_(self, object: str):

        self.\_object = object

    @abstractmethod

    def create(self): pass

class ModernLegs(Legs):

    def \_\_init\_\_(self):

        super().\_\_init\_\_("Модерн")

    def create(self):

        print(f'Созданы ножки стула: {self.\_object}')

class ModernCap(Cap):

    def \_\_init\_\_(self):

        super().\_\_init\_\_("Модерн")

    def create(self):

        print(f'Созданы сиденье и спинка стула: {self.\_object}')

class VenLegs(Legs):

    def \_\_init\_\_(self):

        super().\_\_init\_\_("Викторианский")

    def create(self):

        print(f'Созданы ножки стула: {self.\_object}')

class VenCap(Cap):

    def \_\_init\_\_(self):

        super().\_\_init\_\_("Викторианский")

    def create(self):

        print(f'Созданы сиденье и спинка стула: {self.\_object}')

class GuiAbstractTable(ABC):

    @abstractmethod

    def getLegs(self) -> Legs: pass

    @abstractmethod

    def getCap(self) -> Cap: pass

class ModernGuiFactory(GuiAbstractTable):

    def getLegs(self) -> Legs:

        return ModernLegs()

    def getCap(self) -> Cap:

        return ModernCap()

class VenGuiFactory(GuiAbstractTable):

    def getLegs(self) -> Legs:

        return VenLegs()

    def getCap(self) -> Cap:

        return VenCap()

class Application:

    def \_\_init\_\_(self, table: GuiAbstractTable):

        self.\_gui\_table = table

    def create\_gui(self):

        legs = self.\_gui\_table.getLegs()

        cap = self.\_gui\_table.getCap()

        legs.create()

        cap.create()

def create\_factory(objectname: str) -> GuiAbstractTable:

    tabled = {

        "Модерн": ModernGuiFactory,

        "Викторианский": VenGuiFactory}

    return tabled[objectname]()

objectname = "Модерн"

cr = create\_factory(objectname)

app = Application(cr)

app.create\_gui()

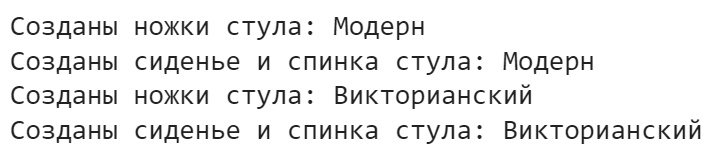
objectname = "Викторианский"

cr = create\_factory(objectname)

app = Application(cr)

app.create\_gui()

Результат работы алгоритма:



**Строитель**

Листинг программы:

from abc import ABC, abstractmethod

class Product:

    bread = ["Кунжутные", "Обычные"]

    meal = ['Двойная котлета', 'Котлета']

    nam = ['Помидоры', 'Огурцы']

    souse = ["Кетчуп", "Горчица"]

class burger:

    def \_\_init\_\_(self, name):

        self.name = name

        self.meal = None

        self.topping = []

        self.souse = None

        self.botbread = None

    def printer(self):

        print(f'Название:{self.name}\n' \

              f'Мясо:{self.meal}\n' \

              f'Топинги:{[it for it in self.topping]}\n' \

              f'Соус:{self.souse}\n' \

              f'Булочки:{self.botbread}\n')

class Builder(ABC):

    @abstractmethod

    def add\_sauce(self) -> None: pass

    @abstractmethod

    def add\_meal(self) -> None: pass

    @abstractmethod

    def add\_topping(self) -> None: pass

    @abstractmethod

    def prepare\_botbread(self) -> None: pass

    @abstractmethod

    def get\_bur(self) -> burger: pass

class Director:

    def \_\_init\_\_(self):

        self.builder = None

    def set\_builder(self, builder: Builder):

        self.builder = builder

    def make\_bur(self):

        if not self.builder:

            raise ValueError("Builder didn't set")

        self.builder.add\_sauce()

        self.builder.add\_meal()

        self.builder.add\_topping()

        self.builder.prepare\_botbread()

class RichBu(Builder):

    def \_\_init\_\_(self):

        self.bur = burger("Премиум")

    def add\_sauce(self) -> None:

        self.bur.souse = Product.souse[0]

    def add\_meal(self) -> None:

        self.bur.meal = Product.meal[0]

    def add\_topping(self) -> None:

        self.bur.topping.append(Product.nam[0])

        self.bur.topping.append(Product.nam[1])

    def prepare\_botbread(self) -> None:

        self.bur.botbread = Product.bread[0]

    def get\_bur(self) -> burger:

        return self.bur

class DefaultBur(Builder):

    def \_\_init\_\_(self):

        self.bur = burger("Обычный")

    def add\_sauce(self) -> None:

        self.bur.souse = Product.souse[1]

    def add\_meal(self) -> None:

        self.bur.meal = Product.meal[1]

    def add\_topping(self) -> None:

        self.bur.topping.append(Product.nam[0])

    def prepare\_botbread(self) -> None:

        self.bur.botbread = Product.bread[1]

    def get\_bur(self) -> burger:

        return self.bur

director = Director()

print("Премиум-1, Обычный-2")

a=int(input())

if a==1:

    builder = RichBu()

else:

    builder = DefaultBur()

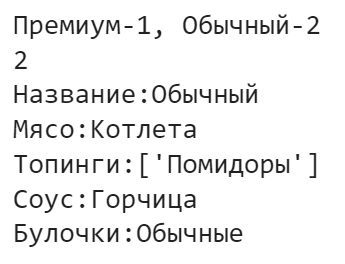
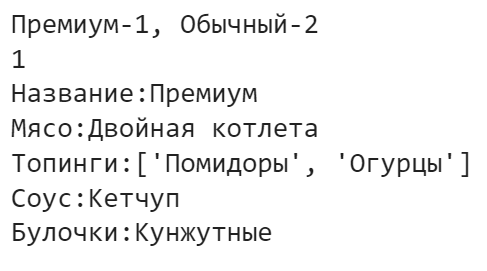
director.set\_builder(builder)

director.make\_bur()

burger = builder.get\_bur()

burger.printer()

Результат работы алгоритма:



**Адаптер**

Листинг программы:

class intTime:

    def geti(self):

        return 123456

class strTime:

    def gets(self):

        return "1111"

class Adapter(intTime,strTime):

    def get1(self):

        return str(self.geti())

    def get2(self):

        return int(self.gets())

work = Adapter()

intwork=intTime()

strwork=strTime()

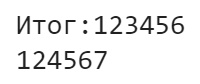
#print("Итог:" + intwork.geti()) Выдаст ошибку

#print(intwork.geti()+strwork.gets()) Выдаст ошибку

print("Итог:" + work.get1())

print(work.get2()+intwork.geti())

Результат работы алгоритма:



**Посредник**

Листинг программы:

from abc import ABC, abstractmethod

class User():

    def \_\_init\_\_(self, med, name, pol):

        self.mediator = med

        self.name = name

        self.pol = pol

    @abstractmethod

    def send(self, msg):

        pass

    @abstractmethod

    def sendM(self, msg):

        pass

    @abstractmethod

    def sendW(self, msg):

        pass

    @abstractmethod

    def receive(self, msg):

        pass

class ChatMediator:

    def \_\_init\_\_(self):

        self.users = []

    def add\_user(self, user):

        self.users.append(user)

    def send\_message(self, msg, user):

        for u in self.users:

            if u != user:

                u.receive(msg)

    def send\_messageM(self,msg,user):

        for u in self.users:

            if u != user and u.pol == "M":

                u.receive(msg)

    def send\_messageW(self,msg,user):

        for u in self.users:

            if u != user and u.pol == "W":

                u.receive(msg)

class ConcreteUser(User):

    def send(self, msg):

        print(self.name + ": Отправил сообщение: " + msg)

        self.mediator.send\_message(msg, self)

    def sendM(self, msg):

        print(self.name + ": Отправил парням: " + msg)

        self.mediator.send\_messageM(msg, self)

    def sendW(self, msg):

        print(self.name + ": Отправил девушкам: " + msg)

        self.mediator.send\_messageW(msg, self)

    def receive(self, msg):

        print(self.name + ": Получено сообщение: " + msg)

def printl():

    print("-" \* 50)

mediator = ChatMediator()

user1 = ConcreteUser(mediator, "Александр", "M")

user2 = ConcreteUser(mediator, "Михаил", "M")

user3 = ConcreteUser(mediator, "Даша", "W")

user4 = ConcreteUser(mediator, "Светлана", "W")

mediator.add\_user(user1)

mediator.add\_user(user2)

mediator.add\_user(user3)

mediator.add\_user(user4)

user1.send("Всем добрый день!")

printl()

user1.sendM("Привет, мальчики! Какие планы?")

printl()

user1.sendW("Привет, девушки! Как дела?")

printl()

Результат работы алгоритма:

