|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

**Факультет «Информатика и системы управления»**

**Кафедра ИУ5 «Системы обработки информации и управления»**

**Отчет по лабораторной работе № 4**

**Разработка интернет приложений**

Студент:

группы ИУ5-54Б

Ведьгун Е.А.

2020 г.

Москва

**Цель работы**

изучение реализации шаблонов проектирования и возможностей модульного тестирования в языке Python.

**Задание**

1. Необходимо для произвольной предметной области реализовать три шаблона проектирования: один порождающий, один структурный и один поведенческий. В качестве справочника шаблонов можно использовать [следующий каталог.](https://refactoring.guru/ru/design-patterns/catalog)
2. Для каждой реализации шаблона необходимо написать модульный тест. В модульных тестах необходимо применить следующие технологии:
   * TDD - фреймворк.
   * BDD - фреймворк.
   * Создание Mock-объектов.

**Текст программы**

Director.py

from Lab4.patterns.web\_element import WebElementBuilder, ConcreteWebElementBuilder  
  
  
# Класс директора  
class WebDirector():  
 def \_\_init\_\_(self, builder: WebElementBuilder):  
 self.\_builder = builder  
  
 def make\_label(self):  
 self.\_builder.setName("<label>")  
 self.\_builder.introduce\_text("I am <label>")  
 self.\_builder.introduce\_visibility(True)  
 self.\_builder.introduce\_action("updated", labelUpdate)  
  
 def make\_button(self):  
 self.\_builder.setName("<button>")  
 self.\_builder.introduce\_height(30)  
 self.\_builder.introduce\_width(120)  
 self.\_builder.introduce\_visibility(True)  
 self.\_builder.introduce\_text("I am <button>")  
 self.\_builder.introduce\_action("clicked", buttonClick)  
  
 def make\_link(self):  
 self.\_builder.setName("<a>")  
 self.\_builder.introduce\_text("I am <a>")  
 self.\_builder.introduce\_visibility(True)  
 self.\_builder.introduce\_link("https://smth.com")  
  
 def make\_radioButton(self):  
 self.\_builder.setName("<radio>")  
 self.\_builder.introduce\_text("I am <radio>")  
 self.\_builder.introduce\_connectivity("previousElem")  
 self.\_builder.introduce\_visibility(True)  
  
  
def labelUpdate():  
 print("\n\033[33mДополнительная логика для лейбла\033[33m\n")  
  
  
def buttonClick():  
 print("\n\033[33mДополнительная логика кнопки\033[33m\n")

Web\_element.py

from abc import ABC, abstractmethod, abstractproperty  
  
  
# Абстрактный класс посредника  
class Mediator(ABC):  
 def notify(self, sender: object, event: str) -> None:  
 pass  
  
  
# Абстрактный класс строителя  
class WebElementBuilder(ABC):  
  
 @abstractproperty  
 def product(self):  
 pass  
  
 @abstractmethod  
 def setName(self):  
 pass  
  
 @abstractmethod  
 def introduce\_height(self):  
 pass  
  
 @abstractmethod  
 def introduce\_width(self):  
 pass  
  
 @abstractmethod  
 def introduce\_visibility(self):  
 pass  
  
 @abstractmethod  
 def introduce\_connectivity(self):  
 pass  
  
 @abstractmethod  
 def introduce\_action(self):  
 pass  
  
 @abstractmethod  
 def introduce\_link(self):  
 pass  
  
 @abstractmethod  
 def introduce\_text(self):  
 pass  
  
  
# Конкретный класс строителя  
class ConcreteWebElementBuilder(WebElementBuilder):  
 def \_\_init\_\_(self):  
 self.reset()  
  
 def reset(self):  
 self.\_product = WebElement()  
 self.\_mediator = None  
  
 @property  
 def product(self):  
 product = self.\_product  
 self.reset()  
 return product  
  
 def setName(self, name):  
 self.\_product.nameOfElement = name  
  
 def introduce\_height(self, value):  
 self.\_product.add("height", value)  
  
 def introduce\_width(self, value):  
 self.\_product.add("width", value)  
  
 def introduce\_visibility(self, value):  
 self.\_product.add("visibility", value)  
  
 def introduce\_connectivity(self, value):  
 self.\_product.add("connectivity", True)  
 self.\_product.add("mainElement", value)  
  
 def introduce\_action(self, name, value):  
 self.\_product.add("action", (name, value))  
  
 def introduce\_link(self, value):  
 self.\_product.add("link", value)  
  
 def introduce\_text(self, value):  
 self.\_product.add("text", value)  
  
  
# Класс веб элемента  
class WebElement():  
 def \_\_init\_\_(self):  
 self.parts = {}  
 self.\_name = None  
  
 @property  
 def nameOfElement(self):  
 return self.\_name  
  
 @nameOfElement.setter  
 def nameOfElement(self, nameOfElement):  
 self.\_name = nameOfElement  
  
 @property  
 def mediator(self) -> Mediator:  
 return self.\_mediator  
  
 @mediator.setter  
 def mediator(self, mediator: Mediator):  
 self.\_mediator = mediator  
  
 @nameOfElement.getter  
 def nameOfElement(self):  
 return self.\_name  
  
 def add(self, partName, partValue):  
 self.parts.update({partName: partValue})  
  
 def printParts(self):  
 print("\033[33m\n {} \n\033[37m".format(self.\_name))  
 for i, j in self.parts.items():  
 print("{} = {}".format(i, j))  
 return self.parts  
  
 def makeAction(self):  
 self.\_mediator.notify(self, self.parts["action"][0])  
 func = self.parts["action"][1]  
 func()  
  
  
# Конкретная реализация посредника  
class ConcreateMediator(Mediator):  
 def \_\_init\_\_(self, webElement1: WebElement, webElement2: WebElement):  
 self.\_webElement1 = webElement1  
 self.\_webElement1.mediator = self  
 self.\_webElement2 = webElement2  
 self.\_webElement2.mediator = self  
  
 def notify(self, sender: object, event: str):  
 if (event == "clicked") and (sender.nameOfElement == "<button>"):  
 print("\033[35m\nНажата кнопка, запрошено обновление label\033[37m\n")  
 self.\_webElement2.makeAction()  
 elif (event == "updated") and (sender.nameOfElement == "<label>"):  
 print("\033[35m\nПосле нажатия кнопки label был обновлен\033[37m\n")