Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Кафедра экономической информатики

ОТЧЕТ

ЛАБОРАТОРНАЯ РАБОТА №1

«Объектно-ориентированный анализ и моделирование, разработка статических моделей»

Выполнил:

Студент ФКП, гр.814301

Мацаль

Н.Ю.

Проверил:

Лыщик

А.П.

Минск 2020

**Цель:** Разработать диаграмму вариантов использования по предметной области «Онлайн система автомобильный дилер». Ознакомление с основными элементами определения, представления, проектирования и моделирования программных систем с помощью языка UML. Реализовать модель в программном коде JAVA.

**Теоретические сведения**

**Диаграмма прецедентов** (диаграмма вариантов использования) в UML – диаграмма, отражающая отношения между актёрами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

Основное назначение диаграммы — описание функциональности и поведения, позволяющее заказчику, конечному пользователю и разработчику совместно обсуждать проектируемую или существующую систему.

*Для отражения модели прецедентов на диаграмме используются:*

**· рамки системы** — прямоугольник с названием в верхней части и эллипсами (прецедентами) внутри. Часто может быть опущен без потери полезной информации.

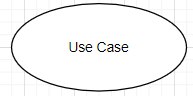
**· актёр** — стилизованный человечек, обозначающий набор ролей пользователя (понимается в широком смысле: человек, внешняя сущность, класс, другая система), взаимодействующего с некоторой сущностью (системой, подсистемой, классом). Актёры не могут быть связаны друг с другом (за исключением отношений обобщения/наследования).

*Стандартное графическое изображение актера:*



**· прецедент** — эллипс с надписью, обозначающий выполняемые системой действия (могут включать возможные варианты), приводящие к наблюдаемым актёрами результатам.

*Стандартное графическое изображение прецедента:*



Надпись может быть именем или описанием (с точки зрения актёров) того, «что» делает система (а не «как»). Имя прецедента связано с непрерываемым (атомарным) сценарием — конкретной последовательностью действий, иллюстрирующей поведение. В ходе сценария актёры обмениваются с системой сообщениями. Сценарий может быть приведён на диаграмме прецедентов в виде UML-комментария. С одним прецедентом может быть связано несколько различных сценариев.

Виды отношений:

· **ассоциативное отношение.**

Отношение между вариантом использования и актером, отражающее связь между ними. Оно устанавливает, какую конкретную роль играет актер при взаимодействии с экземпляром варианта использования.

**· отношение обобщения** — стрелка с не закрашенным треугольником (треугольник ставится у более общего прецедента).

Служит для указания того факта, что некоторый вариант использования А может быть обобщен до варианта использования Б (или актер А может быть обобщен до актера Б).

**· отношение включения** — пунктирная стрелка со стереотипом «include».

Указывает, что некоторое заданное поведение для одного варианта использования включается в качестве составного компонента в последовательность поведения другого варианта использования.

**· отношение расширения** — пунктирная стрелка со стереотипом «extend» (стрелка входит в расширяемый прецедент, в дополнительном разделе которого может быть указана точка расширения и, возможно в виде комментария, условие расширения).

Определяет взаимосвязь базового варианта использования с некоторым другим вариантом использования, функциональное поведение которого задействуется базовым не всегда, а только при выполнении некоторых дополнительных условий.

**Результат выполнения работы:**

***Описание диаграммы вариантов использования «Онлайн система автомобильный дилер»***

В системе задействованы два актера: персонал и покупатель.

*Персоналу доступны следующие use cases:*

1. «Авторизация»
2. «Просмотр списка машин»
3. «Подтверждение покупки»

*При просмотре списка машин персонал может: добавить, изменить статус, просмотреть и удалить машину из списка.*

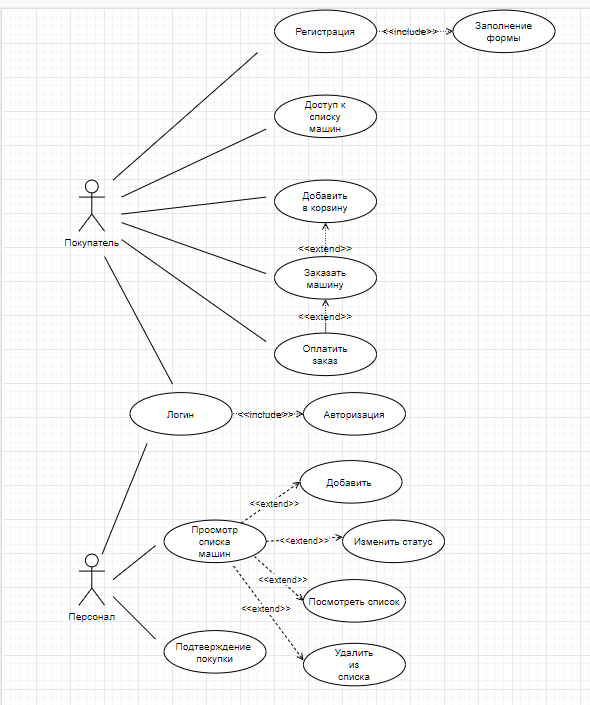
*Покупателю доступны следующие use cases:*

1. Логин
2. Регистрация
3. Доступ к списку машин.
4. Покупка машины.

*Если говорить об покупке машины покупатель может:*

Добавить в корзину машину, заказать нужную машину, оплатить заказ.

**Диаграмма вариантов использования**



**Реализация** **схемы 1,в программном коде JAVA**

class Cl6 implements I2{  
 public int f2(int a, int b) {  
 return a-b;  
 }  
 public int f1(int a, int b) {  
 return a\*b;  
 }  
 Cl6() {  
 System.*out*.println("f2: " + f2(3,1) + " f1: " + f1(1,1));  
 }  
}

package lab;  
  
abstract public class Cl7 implements I2 {  
 abstract int f7(int a, int b);  
}

abstract public class Cl9 implements I3 {  
 abstract String f9();  
}

abstract public class Cl11 {  
 abstract String f11();  
}

public class Cl12 extends Cl7{  
 @Override  
 int f7(int a, int b) {  
 return a + b;  
 }  
  
 @Override  
 public int f2(int a, int b) {  
 return a\*b;  
 }  
  
 @Override  
 public int f1(int a, int b) {  
 return a-b;  
 }  
 Cl12() {  
 System.*out*.println("f7: " + f7(3,4) + " f2: "+f2(1,2) + " f1: "+f1(2,1));  
 }  
}

public class Cl13 extends Cl7 {  
  
 @Override  
 int f7(int a, int b) {  
 return a + b;  
 }  
  
 @Override  
 public int f2(int a, int b) {  
 return a \*b;  
 }  
  
 @Override  
 public int f1(int a, int b) {  
 return a-b;  
 }  
 Cl13() {  
 System.*out*.println("f7: " + f7(3,4) + " f2: "+f2(1,2) + " f1: "+f1(2,1));  
 }  
}

public class Cl14 extends Cl11 implements I10 {  
 @Override  
 String f11() {  
 return "f11";  
 }  
  
 @Override  
 public String f10() {  
 return "f10";  
 }  
  
 @Override  
 public String f3(String a) {  
 return a;  
 }  
  
 @Override  
 public void f4(String s1) {  
 System.*out*.print("f4");  
 }  
  
 @Override  
 public int f5(int a, int b) {  
 return a-b;  
 }  
  
 @Override  
 public int f1(int a, int b) {  
 return a+b;  
 }  
  
 Cl14() {  
 System.*out*.println("class cl14");  
 }  
}

public class Cl15 extends Cl9 implements I8,I10 {  
  
 @Override  
 String f9() {  
 return "f9";  
 }  
  
 @Override  
 public String f10() {  
 return "f10";  
 }  
  
 @Override  
 public String f8() {  
 return "f8";  
 }  
  
 @Override  
 public int f2(int a, int b) {  
 return 2;  
 }  
  
 @Override  
 public String f3(String a) {  
 return null;  
 }  
  
 @Override  
 public void f4(String s1) {  
 System.*out*.print("f4 ");  
 }  
  
 @Override  
 public int f5(int a, int b) {  
 return a+b;  
 }  
  
 @Override  
 public int f1(int a, int b) {  
 return a-b;  
 }  
  
 Cl15() {  
 System.*out*.println("Class 15 ");  
 }  
}

interface I1 {  
 int f1(int a, int b);  
}

interface I2 extends I1{  
 int f2(int a, int b);  
}

public interface I3 extends I1{  
 String f3(String a);  
}

public interface I4 extends I1{  
 void f4(String s1);  
}

public interface I5 extends I1 {  
 int f5(int a, int b);  
}

public interface I8 extends I2, I3 {  
 String f8();  
}

public interface I10 extends I3, I4, I5 {  
 String f10();  
}

public class Main {  
 public static void main(String[] args) {  
 Cl6 cl6 = new Cl6();  
 Cl12 cl12 = new Cl12();  
 Cl13 cl13 = new Cl13();  
 Cl15 cl15 = new Cl15();  
 Cl14 cl14 = new Cl14();  
 }  
}

**Результат выполнения программы:**

f2: 2 f1: 1

f7: 7 f2: 2 f1: 1

f7: 7 f2: 2 f1: 1

Class 15

class cl14

**Вывод:** в данной лабораторной работе была диаграмма вариантов использования. Изучены основные элементы определения, представления, проектирования и моделирования программных систем с помощью языка UML.