**Контрольная работа № 2**

**Тема: «Построение диаграммы классов UML»**

**ЦЕЛЬ КОНТРОЛЬНОЙ РАБОТЫ:**

* Освоить базовые принципы использования UML-диаграмм при разработке программного обеспечения;
* Построить диаграмму классов для разрабатываемой программы.

**ХОД РАБОТЫ:**

Унифицированный язык моделирования (Unified Modelling Language – UML) – это стандартный инструмент для разработки «чертежей» программного обеспечения.

Диаграмма UML – это графическое представление набора элементов, чаще всего изображенного в виде связного графа вершин (сущностей) и дуг (связей). Диаграмма служит для визуализации системы с различных точек зрения, поэтому отдельная диаграмма – это проекция системы. UML включает в себя 13 видов диаграмм.

Диаграмма классов (UML class diagram) – показывает набор классов, интерфейсов и коопераций, а также связи между ними. Диаграммы классов, включающие активные классы, представляют статическое представление процессов системы.

Класс (class) – это описание множества объектов с одинаковыми атрибутами, операциями, связями и семантикой. Изображается в виде прямоугольника.

Связь (relationship) – это соединение сущностей. В ОО моделировании есть три наиболее важные виды связей: зависимости, обобщения и ассоциации. Связь изображается в виде пути с использованием разнообразных типов линий, каждый из которых соответствует определенному виду связи.

**ЗАДАНИЕ НА КОНТРОЛЬНУЮ РАБОТУ:**

Построить диаграмму классов для программы, которая разрабатывается в рамках выполнения курсовой работы.

Для построения диаграмм рекомендуется использовать специальные приложения. В рамках данного курса рекомендуется использовать приложение StarUML – <http://staruml.io/>

**Памятка для создания диаграммы классов**

**Моделирование классов**

**1. Класс**

|  |  |
| --- | --- |
| **public class** Human {  **private** String **name**; *// private это "-"* Boolean **gender** = **true**; *// default это "~"* **protected long chromosome**; *// protected это "#"* **public int age**; *// public это "+"   // Статические атрибуты подчеркиваются* **public static long** *dna*;  *// Константы можно отобазить как readOnly* **final int SECRET** = 924;   */\* Как правило, конструкторы  \* изображаются как обычные методы \*/* **public** Human() {}  **public** Human (String name) {**this**.**name** = name;}   */\* Методы отображаются как  \* [-~#+]имя(тип\_аргументов): возвращаемый тип  \* Например: public String foo (int a, double b)  \* будет +foo(int, double): String \*/* **public void** breath() {}  **private void** sleep(**int** hours) {}  **protected boolean** sneeze() {**return true**;}  **int** run (**int** speed, String direction) {**return** 0;}  **public static int** calculateAge() {**return** 0;} } | C:\Users\NICKGODOV\OneDrive\Политех\ООП (2015)\Сайт\Другие_материалы\Main.png |

**2. Абстрактный класс**

|  |  |
| --- | --- |
| **public abstract class** Mammal {  *// Абстрактный класс и метод выделяются курсивом* **public abstract void** feedWithMilk(Mammal baby); } | **C:\Users\NICKGODOV\OneDrive\Политех\ООП (2015)\Сайт\Другие_материалы\Main.png** |

**3. Интерфейс**

|  |  |
| --- | --- |
| **public interface** Resizeable {  *// Интерфейс можно изображать  // в сокращенном (слева) и в  // полном виде (справа).  // Сокращенный вид не отображает члены  // интерфейса.   // Сокращенный вид используется в других  // диаграммах, в диаграмме классов лучше  // использовать полный вид* **int *MAX\_SCALE*** = 25;   **void** resize(**int** scale); } | C:\Users\NICKGODOV\OneDrive\Политех\ООП (2015)\Сайт\Другие_материалы\Main.png |

**Моделирование связей**

**1. Обобщение**

|  |  |
| --- | --- |
| *// В Java обобщение реализуется  // через наследование. // Обратите внимание, что стрелка идет // от под-класса к суперклассу // (как бы "восходит" к суперклассу) // Форма и вид стрелки и форма и вид линии // строго определен. Связи как раз // и различаются в диаграммах // по форме и виду // стрелок и линий* **class** SuperClass {} **class** SubClass **extends** SuperClass {} **class** AnotherSubClass **extends** SuperClass{} **class** SubSubClass **extends** SubClass {} | C:\Users\NICKGODOV\OneDrive\Политех\ООП (2015)\Сайт\Другие_материалы\Main.png |

**2. Реализация интерфейса**

|  |  |
| --- | --- |
| **public interface** Resizeable {  **int *MAX\_SCALE*** = 25;  **void** resize(**int** scale); }  **class** Shape **implements** Resizeable {  @Override  **public void** resize(**int** scale) {} } | C:\Users\NICKGODOV\OneDrive\Политех\ООП (2015)\Сайт\Другие_материалы\Main.png |

**3. Зависимость**

|  |  |
| --- | --- |
| *// Зависимость, это когда класс A (в нашем случае // Barista)использует класс B (CoffeeMachine)*  *// для реализации своего функционала // Таким образом, класс A зависит от класса B // Зависимость всегда направлена от класса зависимого // класса (A) к классу, от которого зависят (B) // Зависимость - связь типа "uses a" (использует), // Бариста использует кофе-машину для того, чтобы // приготовить кофе*  *// Как правило, если класс A в своем методе принимает в*  *// качестве аргумента, создает внутри новый или // возвращает объект класса B, то речь идет о // зависимости* **class** Barista {  **public** Espresso makeCoffee(CoffeeMachine machine) {  **return** machine.brewEspresso();  } } **class** Espresso {}  **class** CoffeeMachine {  **public** Espresso brewEspresso() {  **return new** Espresso();  } } | C:\Users\NICKGODOV\OneDrive\Политех\ООП (2015)\Лабораторные\лаб8\Main.png |

**4. Ассоциация**

|  |  |
| --- | --- |
| *// Ассоциация – постоянная структурна связь между // классами. Это связь более сильная, чем // зависимость. Говорят что это "has a" (имеет) // связь. В нашем случае, "Owner has a car", // "Владелец имеет машину"* **class** Car {} **class** Owner {  private Car **car**;    **public void** setCar (Car c) {  **this**.**car** = c;  } } | C:\Users\NICKGODOV\OneDrive\Политех\ООП (2015)\Лабораторные\лаб8\Main.png |

**4.1 Односторонняя и бинарная ассоциация**

|  |  |
| --- | --- |
| *// Ассоциация бывает однонаправленная // как в этом случае // Она связывает один класс с другим в // определенном направлении. // То есть, у класса Owner структурной частью // является объект класса Car // А класс Car о существовании класса Owner // не знает* **class** Car {} **class** Owner {  private Car **car**;   **public void** setCar (Car c) {  **this**.**car** = c;  } } *// или двунаправленная, как // в этом случае // Она связывает между собой // два класса в двустороннем порядке // Структурной частью класса Parent // является объект класса Child, // и наоборот, структурной частью класса // Child является объект класса Parent* **class** Parent {  private Child **child**;   **public void** setChild (Child c) {  **this**.**child** = c;  } }  **class** Child {  private Parent **parent**;   **public void** setParent (Parent p) {  **this**.**parent** = p;  } } | C:\Users\NICKGODOV\OneDrive\Политех\ООП (2015)\Лабораторные\лаб8\Main.png |

**4.2 Множественность**

|  |  |
| --- | --- |
| *// Множественность указывает сколько объектов // класса принимают участие в ассоциации // Виды множественности: // 0..1 - 0 или 1 объект // 1 - ровно один объект // \* (вариант: 0..\*) - 0 или более // 1..\* - один или более // Можно указать точный диапазон // Например, 3..12 (минимум 3, максимум 12)* **class** Car {} **class** Owner {  Car **car**; *// Связь 0..1* **public void** setCar (Car c) {  **this**.**car** = c;  } } *// Обратите внимание, что если вы указываете // нижнюю границу 1 (например, 1 или 1..\* или // 1..5) то это значит, что ваш объект класса А не // может быть создан без обязательного*  *// создания объекта класса B.* **class** Child {  Parent **parent**; *// Связь 1 , т.к. мы не можем  // создать объект без указания родителя* **public** Child (Parent p) {  **if** (p == **null**) {  **this**.**parent** = **new** Parent();  } **else this**.**parent** = p;  } } **class** Parent {  Child[] **children**; *// А здесь связь 0..\*  // т.к. объект класса Parent  // может быть создан без объекта класса Child  // Т.к. у нас массив, то значит у нас верхняя  // граница не определена и это \** **public** Parent() {}  **public void** setChildren (Child[] c) {  **this**.**children** = c;  } } | C:\Users\NICKGODOV\OneDrive\Политех\ООП (2015)\Лабораторные\лаб8\Main.png |

**4.3 Агрегация и композиция. Агрегация**

|  |  |
| --- | --- |
| *// Если вам нужно уточнить отношение ассоциации // мы можем использовать такие виды ассоциации, как // агрегация и композиция. // Когда мы используем ассоциацию, мы подразумеваем // что взаимодействующие объекты классов являются // равноправными. Объекты классов Child и Parent // являются равноправными, также как и Owner и Car. // Но если мы хотим сказать что студент является // составляющей частью университета, то мы можем // использовать агрегацию* **class** University {  **private** Student[] **students**; } **class** Student {} *// Здесь важно учитывать, что объект класса Student // может быть составной частью еще какого-то класса // и если объект класса University будет уничтожен, // то не обязательно, что будут уничтожены объекты // класса Student // Помните, что мы в этом случае передаем ЛИШЬ // ССЫЛКУ НА ОБЪЕКТ, и такие ссылки на один и тот // же объект мы можем передать куда-то еще* | C:\Users\NICKGODOV\OneDrive\Политех\ООП (2015)\Лабораторные\лаб8\Main.png |

**4.4 Агрегация и композиция. Композиция**

|  |  |
| --- | --- |
| *// Композиция - еще более сильная // связь, чем агрегация. Мы можем // использовать эту связь, чтобы показать, что // составной объект НЕ МОЖЕТ СУЩЕСТВОВАТЬ БЕЗ // "ОСНОВНОГО" ОБЪЕКТА // Это значит, что если основной объект будет // уничтожен, то гарантированно будут // уничтожены его составные объекты* **class** Car {  private Roof **roof** = **new** Roof();  private Wheel[] **wheels**;    *// Обратите, что при композиции  // как правило мы создаем экземпляр составного  // объекта либо при объявлении переменной   // либо в конструкторе. То есть, не передаем  // ссылку извне, как это было при агрегации,  // а создаем "внутри" класса* **public** Car() {  **wheels** = **new** Wheel[4];  **wheels**[0] = **new** Wheel();  **wheels**[1] = **new** Wheel();  **wheels**[2] = **new** Wheel();  **wheels**[3] = **new** Wheel();  } } **class** Roof {} **class** Wheel {} | C:\Users\NICKGODOV\OneDrive\Политех\ООП (2015)\Лабораторные\лаб8\Main.png |