МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем и технологий

**Отчет**

**по лабораторной работе № 2-4**

**«**Объектно-Ориентированное Программирование. Инициализация состояния объекта. Статические компоненты класса. Инкапсуляция. Повторное использование кода. Наследование**»**

по дисциплине: «КПО»

Выполнили: студенты гр. 10701121

Журович К.С.

Гвоздь В.И.

Асанов Н.К.

Принял: преподаватель Станкевич С.Н.

Минск 2023

**Цель работы:**

**1)**Научиться грамотно использовать соответствующие средства, предоставляемые языком Java, для первоначальной инициализации состояния объекта, а также изучить истинное предназначение статических компонентов класса в языке Java.

**2)** Углубить свои фундаментальные знания в использовании методологии ООП, а также научиться практически применять инкапсуляцию с использованием средств, которые предоставляет язык Java.

**3)** Углубить свои фундаментальные знания в использовании методологии ООП, а также научиться практически применять инструменты языка Java для повторного использования кода в виде ассоциации, наследования, агрегации, композиции и делегирования.

**Задание**

**1)** Необходимо в проект, который был спроектирован и разработан в предыдущей лабораторной работе, внести следующие дополнения:

− для грамотной инициализации состояния объектов соответствующей предметной области добавить всевозможные средства инициализации, которые предоставляет язык Java (блоки инициализации, конструктор по умолчанию, конструкторы с параметрами, конструктор-копирования и т.д.);

− проанализировав соответствующую предметную область добавить в проект статические компоненты класса и возможность их первоначальной инициализации с помощью средств, который предоставляет язык Java. Дополнительно необходимо проанализировать стадии и способы инициализации как состояния объектов, так и состояния соответствующих объектов классов (объектов класса Class), а также их очередность вызова JVM. Привести анализ результатов и соответствующие выводы в отчёте.

**2)** Необходимо в проект, который был спроектирован и разработан в предыдущей лабораторной работе, внести следующие изменения и дополнения:

− скрыть реализацию всех компонентов и структур данных проекта, т.е. инкапсулировать все поля классов и методы, которые предназначены для внутреннего использования, с использованием модификаторов доступа языка Java, и предоставить только интерфейсную часть для внешнего взаимодействия;

− ввести, где это необходимо, высокоуровневые объекты-контейнеры, которые инкапсулируют структуру хранения множества объектов предметной области;   
− убрать из класса-контроллера код по инициализации объектов предметной области и ввести соответствующие программные компоненты, которые и будут предназначены для создания и инициализации объектов предметной области, т.е. использовать компоненты в виде фабрик ил строителей («креаторов»).

**3)** Необходимо произвести рефакторинг программной системы, созданной в предыдущей лабораторной работе, следующим образом:

− классы, описывающие объекты соответствующей предметной области (бизнес-объекты), должны быть сведены в иерархическую структуру (произвести, где это необходимо, классификацию типов); к примеру, в предыдущей лабораторной работе была только сущность автомобиль/автотранспорт, а теперь должна быть иерархия автотранспорта с соответствующими характеристиками: легковой автомобиль (седан, универсал, хэтчбэк, …), грузовой автомобиль (фура, самосвал, бетономешалка ,…), пассажирский автомобиль (автобус, микроавтобус, минивэн, …) и т.д., никто никого не ограничивает в фантазиях;

− логика системы должна быть реализована внутри соответствующих функциональных классов;

− логика системы и большинство других компонентов должны зависеть преимущественно только от абстракции, а не от реализаций;

− необходимо дополнительно для безопасности выполнения кода добавить по возможности в методы бизнес логики проверку входящих объектов на соответствие типа, с которым должна взаимодействовать данная логика.

**Результаты выполнения работы**

**1.1. Разработка предметной области**

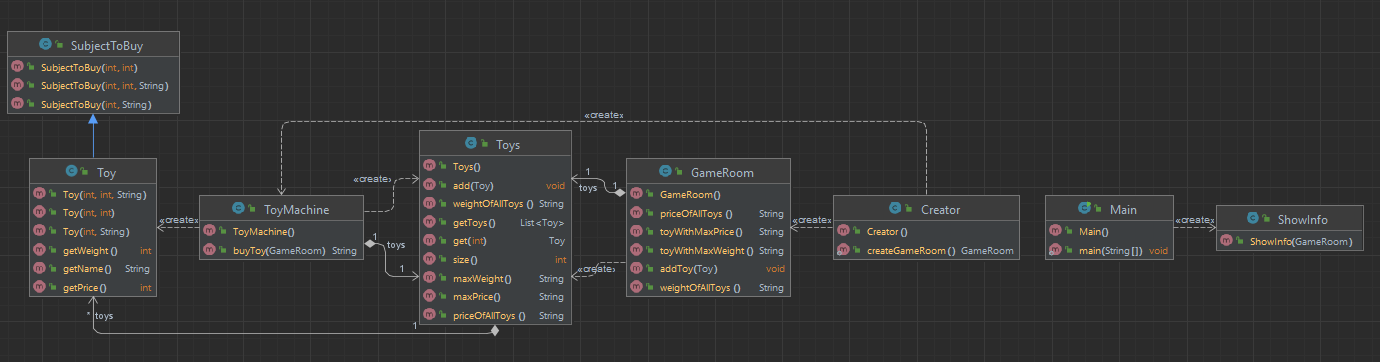


Рис. 1 – Диаграмма классов, созданная в среде InteligeIdea

**2.2. Листинг программы**

**/Controller**

**//Creator**

package controller;  
  
import model.GameRoom;  
  
public class Creator {  
 public static GameRoom createGameRoom(){  
 GameRoom gameRoom = new GameRoom();  
 ToyMachine toyMachine = new ToyMachine();  
 for (int i = 0; i < 30; i++)  
 System.*out*.println(toyMachine.buyToy(gameRoom));  
 return gameRoom;  
 }  
}

**//ToyMachine**

package controller;  
  
import model.Toys;  
import model.GameRoom;  
import model.Toy;  
  
import java.util.Random;  
  
public class ToyMachine {  
 private Toys toys;  
 public ToyMachine(){  
 toys = new Toys();  
 toys.add(new Toy(1,200,"Стикер"));  
 toys.add(new Toy(2,100,"Брелок"));  
 toys.add(new Toy(2,1000,"Матрёшка"));  
 toys.add(new Toy(3,400,"Автомобиль"));  
 toys.add(new Toy(4,400,"Поезд"));  
 toys.add(new Toy(5,200,"Солдатик"));  
 }  
 public String buyToy(GameRoom gameRoom){  
 Random random = new Random();  
 Toy toy =toys.get(random.nextInt(toys.size()));  
 gameRoom.addToy(toy);  
 return "Игрушка "+toy.getName()+" добалена и игровую комнату";  
 }  
}

**/Model**

**//GameRoom**

package model;  
  
public class GameRoom {  
 private final Toys toys;  
  
 public GameRoom(){  
 toys=new Toys();  
 }  
  
 public void addToy(Toy toy){  
 toys.add(toy);  
 }  
 public String priceOfAllToys(){  
 return toys.priceOfAllToys();  
 }  
  
 public String weightOfAllToys(){  
 return toys.weightOfAllToys();  
 }  
 public String toyWithMaxPrice(){  
 return toys.maxPrice();  
 }  
 public String toyWithMaxWeight(){  
 return toys.maxWeight();  
 }  
  
}

**//SubjectToBuy**

package model;  
  
public class SubjectToBuy {  
 protected final int price;  
 protected final int weight;  
 protected final String name;  
  
 public SubjectToBuy(int price, int weight, String name) {  
 this.price = price;  
 this.weight = weight;  
 this.name = name;  
 }  
  
 public SubjectToBuy(int weight, String name) {  
 this.weight = weight;  
 this.name = name;  
 this.price = 2;  
 }  
  
 public SubjectToBuy(int price, int weight) {  
 this.price = price;  
 this.weight = weight;  
 this.name = "Kostya";  
  
 }  
}

**///Toy**

package model;  
  
public class Toy extends SubjectToBuy{  
 public Toy(int price, int weight, String name) {  
 super(price, weight, name);  
 }  
  
 public Toy(int weight, String name) {  
 super(weight, name);  
 }  
  
 public Toy(int price, int weight) {  
 super(price, weight);  
 }  
  
 public int getPrice() {  
 return price;  
 }  
  
 public int getWeight() {  
 return weight;  
 }  
  
 public String getName() {  
 return name;  
 }

}

**///Toys**

package model;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class Toys {  
 private final List<Toy> toys ;  
  
 public Toys() {  
 this.toys = new ArrayList<>();  
 }  
  
  
 public String priceOfAllToys(){  
 int sum = 0;  
 for (Toy toy:toys) {  
 sum+=toy.getPrice();  
 }  
 return "Цена всех игрушек равна "+sum;  
 }  
 public String weightOfAllToys(){  
 int sum = 0;  
 for (Toy toy:toys) {  
 sum+=toy.getWeight();  
 }  
 return "Вес всех игрушек равен "+sum;  
 }  
 public String maxWeight(){  
 Toy maxToy = toys.get(toys.size()-1);  
 for (Toy toy:toys) {  
 if(maxToy.getWeight()<toy.getWeight())  
 maxToy=toy;  
 }  
 return maxToy.getName()+" - игрушка с самым большим весом равным "+maxToy.getWeight();  
 }  
 public String maxPrice(){  
 Toy maxToy = toys.get(toys.size()-1);  
 for (Toy toy:toys) {  
 if(maxToy.getPrice()<toy.getPrice())  
 maxToy=toy;  
 }  
 return maxToy.getName()+" - игрушка с самой большой ценой равной "+maxToy.getPrice();  
 }  
 public List<Toy> getToys() {  
 return toys;  
 }  
 public Toy get(int number) {  
 return toys.get(number);  
 }  
 public void add(Toy toy) {  
 this.toys.add(toy);  
 }  
 public int size(){  
 return toys.size();  
 }  
}

**/View**

**//ShowInfo**

package view;  
  
import model.GameRoom;  
import controller.ToyMachine;  
  
public class ShowInfo {  
 public ShowInfo(GameRoom gameRoom){  
 System.*out*.println(gameRoom.priceOfAllToys());  
 System.*out*.println(gameRoom.weightOfAllToys());  
 System.*out*.println(gameRoom.toyWithMaxPrice());  
 System.*out*.println(gameRoom.toyWithMaxWeight());  
 }

}

**//Main**

import controller.Creator;  
import view.ShowInfo;  
  
public class Main {  
 public static void main(String[] args) {  
 new ShowInfo(Creator.*createGameRoom*());  
 }  
}

**3. Выполнение программы**

- Была реализована архитектура MVC. Добавлен класс «Creator.java» в модуле Controller для инициализации игрушек комнаты, а также класс «ShowInfo» в модуле View для вывода информации о комнаты в консоль.

- Были добавлены конструкторы по умолчанию, копирования для классов

- Были добавлены статические переменные и методы.

- Также была введена проверка некоторых входных параметров на корректность.

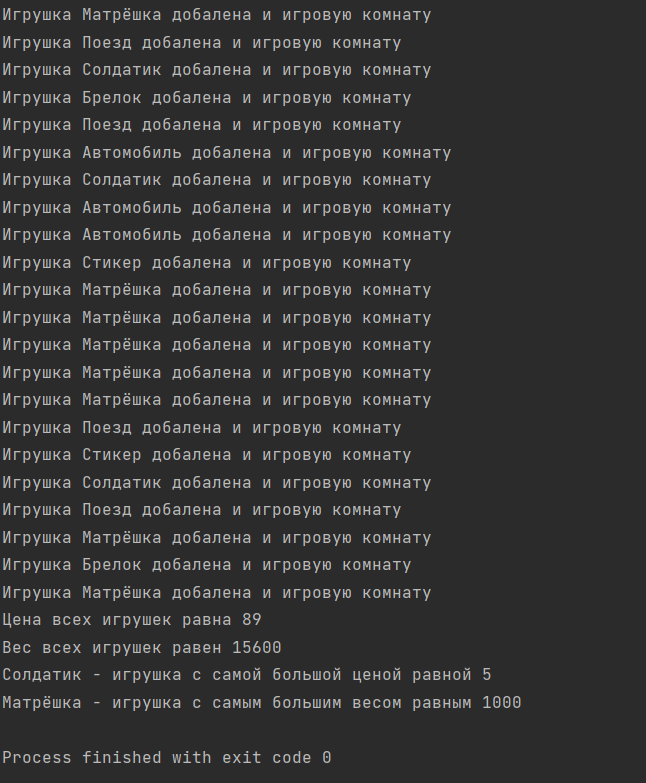


Рис. 4 – Результат выполнения метода Main

**4. Выводы**

Научились грамотно использовать соответствующие средства, предоставляемые языком Java, для первоначальной инициализации состояния объекта, а также изучили истинное предназначение статических компонентов класса в языке Java. Углубили свои фундаментальные знания в использовании методологии ООП, а также научились практически применять инкапсуляцию с использованием средств, которые предоставляет язык Java. Углубили свои фундаментальные знания в использовании методологии ООП, а также научились практически применять инструменты языка Java для повторного использования кода в виде ассоциации, наследования, агрегации, композиции и делегирования.