Міністерство освіти і науки України
Національний технічний університет України „КПІ”
Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки
інформації та управління

ЗВІТ

до лабораторної роботи № 5
з предмету:

„ОСНОВИ ТЕХНОЛОГІЙ ПРОГРАМУВАННЯ”

Виконав студент *ІП-61 Кушка Михайло Олександрович, 2-й курс, ІП-6116*

(№ групи, прізвище, ім’я, по батькові, курс, номер залікової книжки)

Прийняв *Подрубайло О.О.*

(посада, прізвище, ім’я, по батькові)

Київ 2018

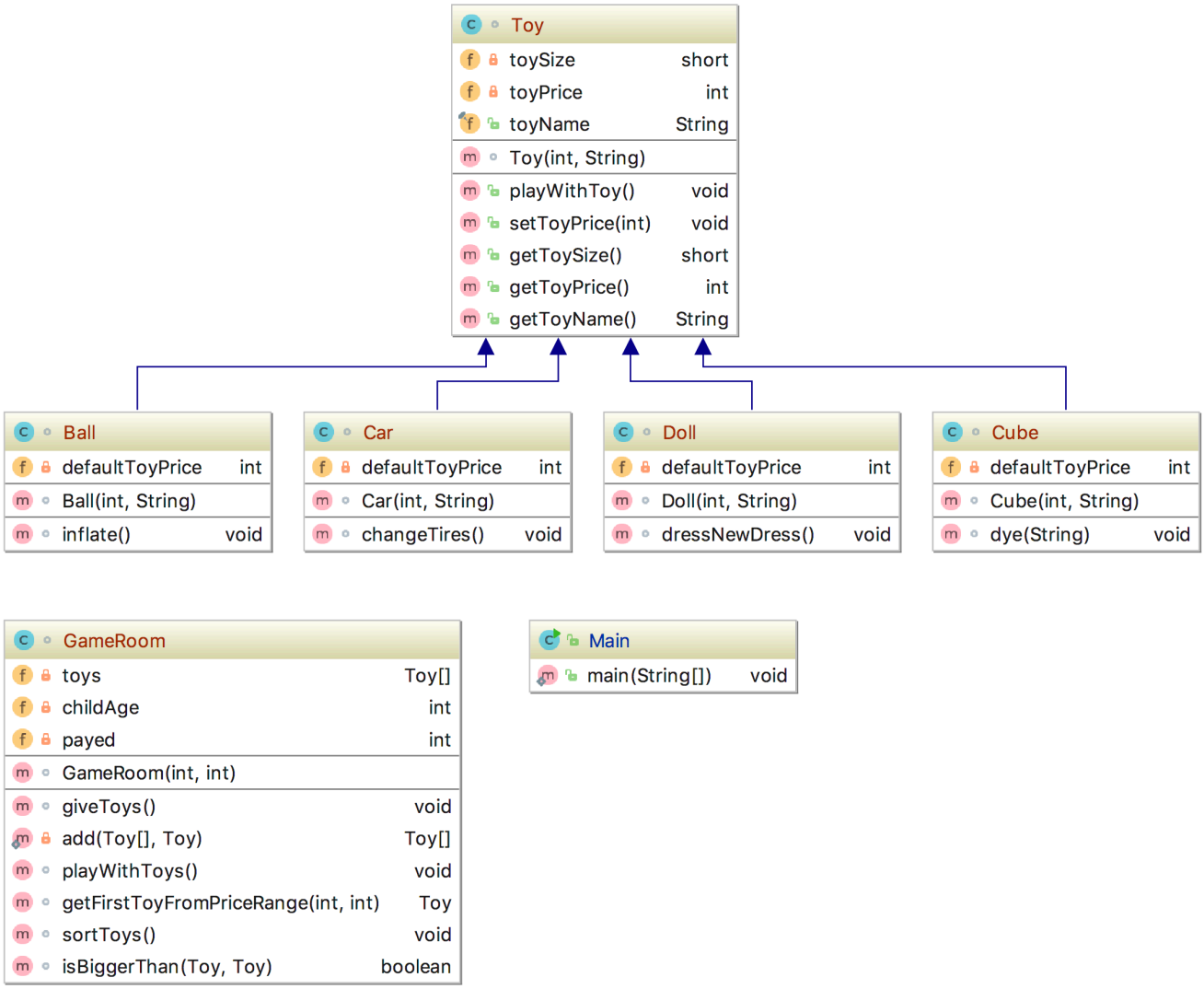
ЗМІСТ

1. ПОСТАНОВКА ЗАДАЧІ

Підготувати ігрову кімнату для дітей різних вікових груп. Іграшок повинно бути фіксована кількість у межах виділеної суми грошей. Повинні зустрічатися іграшки для різних вікових груп: маленькі, середні та великі машинки, ляльки, м'ячі, кубики. Провести сортування іграшок у кімнаті за будь-яким параметром. Знайти іграшку в кімнаті, що відповідає заданому діапазону вартості.

Створити узагальнений клас та не менше 3 класів-нащадків, що описують задану варіантом (п.2) область знань. Створити клас, що складається з масиву об’єктів, з яким можна виконати вказані варіантом дії. Необхідно обробити всі виключні ситуації, що можуть виникнути під час виконання програмного коду. Всі змінні повинні бути описані та значення їх задані у виконавчому методі. Код повинен відповідати стандартам JCC та бути детально задокументований.

2. ДІАГРАМА КЛАСІВ



3. ВИСНОВОК

Для реалізації необхідної системи класів відчувалася гостра необхідність в реалізації списків для можливості коректно, без зайняття великої кількості зайвої пам’яті додавати нові елементи до лінійної структури даних.

4. КОД ПРОГРАМИ

```java
/**
 * Java labs - Lab5
 * @version 1.5 2018-03-23
 * @author Misha Kushka
 */


/**
 * Implementation of the real-world toy with it's
 * properties such as size, type, color and such things
 * to do with it as playing with toy.
 */
class Toy {

    private short toySize; // size of the toy
    private int toyPrice; // price of the toy
    public final String toyName; // name of the toy

    /**
     * Toy's constructor, which sets toy's name
     * depending on the child age.
     *
     * @param childAge Age of the child.
     * @param newToyName Part of the toy name without
     * appendix of it's size.
     */
    Toy(int childAge, String newToyName) {
        if (childAge <= 5) {
            toySize = 1;
            newToyName = "small " + newToyName;
        } else if (childAge > 5 && childAge <= 10) {
            toySize = 2;
            newToyName = "medium " + newToyName;
        } else {
            toySize = 3;
            newToyName = "big " + newToyName;
        }

        toyName = newToyName;
    }

    /**
     * Immitates process of playing with toy.
     */
    public void playWithToy() {
        System.out.println("Now child is playing with the " + toyName + ".");
    }

    /**
     * Setter for the toy price.
     * @param newPrice New price of the toy to set.
     */
    public void setToyPrice(int newPrice) {
        toyPrice = newPrice;
    }

    /**
     * Getter for the toy size.
     * @return Size of the toy.
     */
    public short getToySize() {
        return toySize;
```

```
    }

    /**
     * Getter for the toy prise.
     * @return Price of the toy.
     */
    public int getToyPrice() {
        return toyPrice;
    }

    /**
     * Getter for the toy name.
     * @return Toy's name.
     */
    public String getToyName() {
        return toyName;
    }

}


/**
 * Ball toy for girls & boys of different ages.
 */
class Ball extends Toy {

    private int defaultToyPrice = 1; // price of the toy, not considering child age

    /**
     * Call's the constructor of the parent's Toy class,
     * sets toy's price, which isn't depends on the age
     * of the child.
     *
     * @param childAge Age of the child.
     * @param newToyName Name of the toy.
     */
    Ball(int childAge, String newToyName) {
        super(childAge, newToyName);
        setToyPrice(defaultToyPrice);
    }

    /**
     * Some another method for this class.
     */
    void inflate() {
        System.out.println("Inflate the ball.");
    }

}


/**
 * Car toy for boys of different ages.
 */
class Car extends Toy {

    private int defaultToyPrice = 3; // price of the toy, not considering child age

    /**
     * Call's the constructor of the parent's Toy class,
     * sets toy's price depends of the default toy price
     * and age of the child.
     *
     * @param childAge Age of the child.
     * @param newToyName Name of the toy.
```

```
     */
    Car(int childAge, String newToyName) {
        super(childAge, newToyName);
        setToyPrice(defaultToyPrice * getToySize());
    }

    /**
     * Some another method for this class.
     */
    void changeTires() {
        System.out.println("Now your car is equiped with the new tires.");
    }

}


/**
 * Cube toy for girls & boys of different ages.
 */
class Cube extends Toy {

    private int defaultToyPrice = 4; // price of the toy, not considering child age

    /**
     * Call's the constructor of the parent's Toy class,
     * sets toy's price depends of the default toy price
     * and age of the child.
     *
     * @param childAge Age of the child.
     * @param newToyName Name of the toy.
     */
    Cube(int childAge, String newToyName) {
        super(childAge, newToyName);
        setToyPrice(defaultToyPrice * getToySize());
    }

    /**
     * Some another method for this class.
     */
    void dye(String color) {
        System.out.println("Now color of the your cube is " + color + ".");
    }

}


/**
 * Doll toy for girls of different ages.
 */
class Doll extends Toy {

    private int defaultToyPrice = 5; // price of the toy, not considering child age

    /**
     * Call's the constructor of the parent's Toy class,
     * sets toy's price depends of the default toy price
     * and age of the child.
     *
     * @param childAge Age of the child.
     * @param newToyName Name of the toy.
     */
    Doll(int childAge, String newToyName) {
        super(childAge, newToyName);
        setToyPrice(defaultToyPrice * getToySize());
    }
```

```
    /**
     * Some another method for this class.
     */
    void dressNewDress() {
        System.out.println("Now your doll wears in the new dress.");
    }

}


/**
 * Implementation of the gaming room for children
 * of different ages.
 */
class GameRoom {

    private Toy[] toys = new Toy[0]; // array of toys in the game room
    private int childAge; // age of the child
    private int payed; // how much was payed for the room

    /**
     * Allow to pay for playing in the game room.
     * Depending on the amount of money child can
     * play with different number of toys.
     *
     * @param amount Amount of money to pay for playing.
     * @param age Age of the child in the room.
     */
    GameRoom(int amount, int age) {
        // Too low payment checker.
        if (amount < 1) {
            System.err.println("Sorry, but the cheapest toy costs $1.");
            System.exit(1);
        }
        childAge = age;
        payed = amount;

        // Fill toys array with toys.
        giveToys();

        // Show how many toys are available depends of the payed amount.
        try {
            System.out.println("Now you can play with " + toys.length + " toys.");
        } catch (NullPointerException e) {
            System.err.println("Add elements to the toys array first.");
            System.exit(2);
        }
    }

    /**
     * Fill the toys array with different toys object's
     * depends of the payed amount for the room.
     */
    void giveToys() {

        // Toys, which are in the room.
        Toy[] defaultToys = {new Car(childAge, "super car"),
                             new Doll(childAge, "cool doll"),
                             new Ball(childAge, "amazing ball"),
                             new Cube(childAge, "crazy cube")};

        int[] defaultToyPrices = new int[defaultToys.length]; // prices of toys in the room

        // Set prices for all toys in the room depending on the child age.
```

```
    for (int i = 0; i < defaultToyPrices.length; i++) {
        defaultToyPrices[i] = defaultToys[i].getToyPrice();
    }

    int totalPrice = 0; // total price of all toys for the current child
    int iteration = 0; // number of iterations of adding toys

    // A little bit randomly choose toys for the particular child
    // depending on the child age and payed amount.
    while (totalPrice < payed) {
        switch (iteration%4) {
            case 0:
                if (totalPrice + defaultToyPrices[0] <= payed) {
                    toys = add(toys, defaultToys[0]);
                    totalPrice += defaultToyPrices[0];
                }
                break;
            case 1:
                if (totalPrice + defaultToyPrices[1] <= payed) {
                    toys = add(toys, defaultToys[1]);
                    totalPrice += defaultToyPrices[1];
                }
                break;
            case 2:
                if (totalPrice + defaultToyPrices[2] <= payed) {
                    toys = add(toys, defaultToys[2]);
                    totalPrice += defaultToyPrices[2];
                }
                break;
            default:
                if (totalPrice + defaultToyPrices[3] <= payed) {
                    toys = add(toys, defaultToys[3]);
                    totalPrice += defaultToyPrices[3];
                }
                break;
        }
        iteration++;
    }

    System.out.println("Total price: $" + totalPrice);

}

/**
 * Add element to the Toy's array.
 *
 * @param originalArray Array to put element into.
 * @param newItem Element to put.
 * @return New array with added element.
 */
private static Toy[] add(Toy[] originalArray, Toy newItem) {
    int currentSize = originalArray.length;
    int newSize = currentSize + 1;
    Toy[] tempArray = new Toy[ newSize ];
    for (int i = 0; i < currentSize; i++) {
        tempArray[i] = originalArray [i];
    }
    tempArray[newSize-1] = newItem;
    return tempArray;
}

/**
 * Execute method of playing with all toys
 * of the particular child.
 */
```

```
void playWithToys() {
    for (Toy toy : toys) {
        toy.playWithToy();
    }
}

/**
 * Get first toy from the setted range by toy's price.
 *
 * @param min Minimum price of the toy to find.
 * @param max Maximum price of the toy to find.
 * @return First toy with price from range if found,
 * or null otherwise.
 */
Toy getFirstToyFromPriceRange(int min, int max) {

    // Check is min < max.
    if (min > max) {
        System.out.println("Attention! min value is bigger, than max value!");
    }

    // Iteratively find toy from the given range.
    for (int i = 0; i < toys.length; i++) {
        if (toys[i].getToyPrice() >= min && toys[i].getToyPrice() <= max) {
            return toys[i];
        }
    }

    return null;
}

/**
 * Sort toys by the name of their classes alphabetically.
 */
void sortToys() {
    int i, j; // iterators
    int n = toys.length; // length of the toys array
    Toy temp; // temporary Toy object to swap elements

    // Bubble sort for the array of toys.
    for (i = 0; i < n-1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (isBiggerThan(toys[j], toys[j + 1])) {
                temp = toys[j];
                toys[j] = toys[j+1];
                toys[j+1] = temp;
            }
        }
    }
}

boolean isBiggerThan(Toy first, Toy second) {
    int firstIndex, secondIndex;

    // Align class names with indexes.
    // First object.
    switch (first.getClass().getName()) {
        case ("Ball"):
            firstIndex = 0;
            break;
        case ("Car"):
            firstIndex = 1;
            break;
        case ("Cube"):
            firstIndex = 2;
```

```
                break;
            default:
                firstIndex = 3;
        }

        // Second object.
        switch (second.getClass().getName()) {
            case ("Ball"):
                secondIndex = 0;
                break;
            case ("Car"):
                secondIndex = 1;
                break;
            case ("Cube"):
                secondIndex = 2;
                break;
            default:
                secondIndex = 3;
        }

        if (firstIndex > secondIndex)
            return true;

        return false;
    }

}
```