Министерство образования и науки Российской Федерации Федеральное агентство по образованию

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Вятский государственный университет»

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

**Использование средств автоматической генерации документации и рефакторинга программного кода**

Лабораторная работа №2

по курсу «Разработка программных систем»

Вариант 3

Выполнил студент группы ИВТб-31 /Категов А. Д./ Проверил преподаватель /Чистяков Г.А./

Киров 2024

Задание:

Подготовить комплект технической документации на разработанный ранее набор классов. Для выполнения лабораторной работы необходимо решить следующие задачи:

− Провести ряд преобразований программного кода, полученного в ходе выполнения предыдущей работы, с использованием встроенных средств рефакторинга;

− Сопроводить код комментариями с использованием Javadoc;

− Сгенерировать документацию к разработанным классам.

1. Рефракторинг

Было проведено форматирование кода с помощью встроенных средств автоформатирования IDE Eclipse.

1. Исходный код с комментариями Javadoc

Листинг приведен в приложении А.

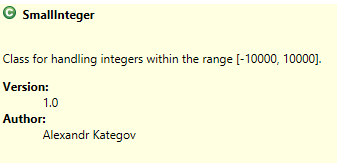
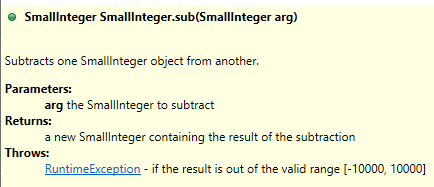


Рисунок 1 – Техническая документация

**Вывод**: в ходе данной лабораторной работы были изучены встроенные в среду разработки Eclipse средства для автоматической генерации документации и рефакторинга кода и стандарт документирования классов Javadoc. На основе полученных знаний был сгенерирован HTML-файл технической документации Javadoc. На основании всего сказанного выше можно сделать вывод о том, что цель достигнута, необходимые знания получены, задание выполнено верно, что доказывается полученными результатами.

Приложение А

(обязательное)

Листинг программы

**import** java.util.\*; // for Scanner

/\*\*

\* Main class of the application, containing the entry point.

\* **@author** Alexandr Kategov

\* **@version** 1.0

\*/

**public** **class** Main {

/\*\*

\* Prints the menu with available operations.

\*/

**public** **static** **void** printMenu() {

System.***out***.println("1) a + b");

System.***out***.println("2) a - b");

System.***out***.println("3) a \* b");

System.***out***.println("4) a / b");

System.***out***.println("5) a % b");

System.***out***.println("0) Выход");

}

/\*\*

\* Prompts the user for input and validates to ensure an integer is entered.

\*

\* **@param** s the Scanner object used for reading input

\* **@return** the integer entered by the user

\*/

**public** **static** **int** input(Scanner s) {

**while** (s.hasNextInt() == **false**) { // Continues to prompt the user until a valid integer is entered

System.***out***.print("Введите целое число: ");

s.next(); // Skip the invalid input

}

**return** s.nextInt(); // Return the entered integer

}

/\*\*

\* Entry point of the application.

\*

\* **@param** args the command line arguments

\*/

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***); // Create a Scanner object for reading input

**int** key; // Variable to store the user's choice of operation

**do** {

System.***out***.print("Введите число a: ");

SmallInteger a = **new** SmallInteger(*input*(scan));

System.***out***.print("Введите число b: ");

SmallInteger b = **new** SmallInteger(*input*(scan));

System.***out***.println();

*printMenu*();

key = *input*(scan);

**switch** (key) {

**case** 1: // Perform addition and display the result

System.***out***.println(a.getValue() + " + " + b.getValue() + " = " + a.add(b) + "\n");

**break**;

**case** 2: // Perform subtraction and display the result

System.***out***.println(a.getValue() + " - " + b.getValue() + " = " + a.sub(b) + "\n");

**break**;

**case** 3: // Perform multiplication and display the result

System.***out***.println(a.getValue() + " \* " + b.getValue() + " = " + a.mul(b) + "\n");

**break**;

**case** 4: // Perform integer division and display the result

System.***out***.println(a.getValue() + " / " + b.getValue() + " = " + a.integerDivision(b) + "\n");

**break**;

**case** 5: // Perform remainder division and display the result

System.***out***.println(a.getValue() + " % " + b.getValue() + " = " + a.remainderDivision(b) + "\n");

**break**;

**default**:

**break**;

}

} **while** (key != 0); // Continue until the user chooses to exit (0)

scan.close();

}

}

/\*\*

\* Class for handling integers within the range [-10000, 10000].

\*

\* **@author** Alexandr Kategov

\* **@version** 1.0

\*/

**public** **class** SmallInteger {

**private** **int** value; // Stores the value of the number

**private** **static** **final** **int** ***MAX\_VALUE*** = 10000; // Maximum allowable value

**private** **static** **final** **int** ***MIN\_VALUE*** = -10000; // Minimum allowable value

/\*\*

\* Primary constructor that accepts an integer and checks if it is within the valid range.

\*

\* **@param** value the integer value to be stored

\* **@throws** RuntimeException if the value is out of the valid range [-10000, 10000]

\*/

**public** SmallInteger(**int** value) {

**if** (value > ***MAX\_VALUE*** || value < ***MIN\_VALUE***) {

**throw** **new** RuntimeException("Value out of range");

}

**this**.value = value;

}

/\*\*

\* Copy constructor that creates a new object based on an existing one.

\*

\* **@param** other the SmallInteger object to copy

\*/

**public** SmallInteger(SmallInteger other) {

**this**(other.value); // Calls the primary constructor

}

/\*\*

\* Adds two SmallInteger objects.

\*

\* **@param** arg the SmallInteger to add

\* **@return** a new SmallInteger containing the result of the addition

\* **@throws** RuntimeException if the result is out of the valid range [-10000, 10000]

\*/

**public** SmallInteger add(SmallInteger arg) {

**int** result = **this**.value + arg.value;

**if** (result > ***MAX\_VALUE*** || result < ***MIN\_VALUE***) {

**throw** **new** RuntimeException("Result out of range");

}

**return** **new** SmallInteger(result);

}

/\*\*

\* Subtracts one SmallInteger object from another.

\*

\* **@param** arg the SmallInteger to subtract

\* **@return** a new SmallInteger containing the result of the subtraction

\* **@throws** RuntimeException if the result is out of the valid range [-10000, 10000]

\*/

**public** SmallInteger sub(SmallInteger arg) {

**int** result = **this**.value - arg.value;

**if** (result > ***MAX\_VALUE*** || result < ***MIN\_VALUE***) {

**throw** **new** RuntimeException("Result out of range");

}

**return** **new** SmallInteger(result);

}

/\*\*

\* Multiplies two SmallInteger objects.

\*

\* **@param** arg the SmallInteger to multiply by

\* **@return** a new SmallInteger containing the result of the multiplication

\* **@throws** RuntimeException if the result is out of the valid range [-10000, 10000]

\*/

**public** SmallInteger mul(SmallInteger arg) {

**int** result = **this**.value \* arg.value;

**if** (result > ***MAX\_VALUE*** || result < ***MIN\_VALUE***) {

**throw** **new** RuntimeException("Result out of range");

}

**return** **new** SmallInteger(result);

}

/\*\*

\* Performs integer division of one SmallInteger object by another.

\*

\* **@param** arg the SmallInteger to divide by

\* **@return** a new SmallInteger containing the result of the division

\* **@throws** ArithmeticException if division by zero occurs

\*/

**public** SmallInteger integerDivision(SmallInteger arg) {

**if** (arg.value == 0) {

**throw** **new** ArithmeticException("Division by zero");

}

**int** result = **this**.value / arg.value;

**return** **new** SmallInteger(result);

}

/\*\*

\* Finds the remainder of the division of one SmallInteger object by another.

\*

\* **@param** arg the SmallInteger to divide by

\* **@return** a new SmallInteger containing the remainder of the division

\* **@throws** ArithmeticException if division by zero occurs

\*/

**public** SmallInteger remainderDivision(SmallInteger arg) {

**if** (arg.value == 0) {

**throw** **new** ArithmeticException("Division by zero");

}

**int** result = **this**.value % arg.value;

**return** **new** SmallInteger(result);

}

/\*\*

\* Gets the current value.

\*

\* **@return** the integer value stored in this SmallInteger

\*/

**public** **int** getValue() {

**return** **this**.value;

}

}