Міністерство освіти і науки України

Національний університет «Львівська політехніка»



**ЗВІТ**

з лабораторної роботи №6

з дисципліни: “Кросплатформні засоби програмування”

на тему: “Параметризоване програмування”

Виконав:

ст. гр. КІ-305

Ніколенко О.В.

Прийняв:

Іванов Ю.С.

Львів – 2023

**Мета роботи:** оволодіти навиками параметризованого програмування мовою Java.

**Завдання:**

1. Створити параметризований клас, що реалізує предметну область задану варіантом. Клас має містити мінімум 4 методи опрацювання даних включаючи розміщення та виймання елементів. Парні варіанти реалізують пошук мінімального елементу, непарні – максимального. Написати на мові Java та налагодити програму-драйвер для розробленого класу, яка мстить мінімум 2 різні класи екземпляри яких розмішуються у екземплярі розробленого класу-контейнеру. Програма має розміщуватися в пакеті Група.Прізвище.Lab6 та володіти коментарями, які дозволять автоматично згенерувати документацію до розробленого пакету.
2. Автоматично згенерувати документацію до розробленого пакету.
3. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагмент згенерованої документації.
4. Дати відповідь на контрольні запитання.

**Варіант №20**

|  |
| --- |
| **Стек** |

**Лістинг програми:**

**Файл StackApp.java**

**package** KI305.Nikolenko.Lab6;

/\*\*

\* Class StackApp implements main method for Stack

\* class possibilities demonstration

\* **@author** Oleksandr Nikolenko

\* **@version** 1.0

\*/

**public** **class** StackApp {

/\*\*

\* **@param** args

\*/

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

{

Stack <? **super** Data> Stack1 = **new** Stack <Data>(10);

Stack1.Push(**new** text("Tania",5));

Stack1.Push(**new** Variable("Integer",8));

Stack1.Push(**new** Variable("Char",1));

Stack1.Push(**new** text("Vitalii",7));

Stack1.Push(**new** Variable("Float",4));

System.***out***.print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \n");

Stack1.getMin();

Stack1.Pop();

System.***out***.print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \n");

System.***out***.println("Stack after delete element ");

System.***out***.print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \n");

Stack1.Print();

}

}

**Файл Stack.java**

**package** KI305.Nikolenko.Lab6;

**import** java.util.ArrayList;

/\*\*

\* **@author** Oleksandr Nikolenko

\* Class Stack

\* **@version** 1.0

\*/

**class** Stack<T **extends** Data>

{

**private** ArrayList<T> arr;

**private** **int** top;

**private** **int** capacity;

**int** minEle;

/\*\*

\* Constructor

\*/

**public** Stack(**int** size)

{

arr = **new** ArrayList<T>(size);

capacity = size;

top = 0;

}

/\*\*

\* Method simulates finding the MinElement in Stack

\*/

**void** getMin()

{

// Get the minimum number in the entire stack

**if** (arr.isEmpty())

System.***out***.println("Stack is empty");

// variable minEle stores the minimum element in the stack.

**else**

System.***out***.println("Minimum Element in the stack is: " + minEle);

}

/\*\*

\* Method simulates push data

\*/

**public** **void** Push(T Data)

{

**if** (IsFull())

{

System.***out***.println("Stack is FULL!!! \n");

System.*exit*(-1);

}

**int** x =Data.getsize();

**if**(x<minEle)

minEle =x;

**if**(top==0)

minEle=x;

arr.add(Data);

++top;

Data.print();

}

/\*\*

\* Method simulates deleting data

\*/

**public** T Pop() {

// if stack is empty no element to pop

**if** (IsEmpty()) {

System.***out***.println("STACK EMPTY!");

// terminates the program

System.*exit*(1);

}

T t = arr.get(--top);

**if** (t.getsize() < minEle)

{

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

minEle = minEle - t.getsize();

}

// pop element from top of stack

System.***out***.println("Removing " + Peek().getTextName());

**return** t;

}

**public** T Peek()

{

**if** (!IsEmpty()) {

T t = arr.get(top);

**if** (t.getsize() < minEle)

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

**return** t;

}

**else** {

System.*exit*(-1);

}

**return** **null**;

}

**public** **int** GetSize() {

**return** top + 1;

}

**public** **boolean** IsEmpty() {

**return** top == -1;

}

**public** **boolean** IsFull() {

**return** top == capacity - 1;

}

**public** **void** Print() {

**for** (**int** i =0 ; i<top; i++) {

arr.get(i).print();

}

}

**public** String toString()

{

String Ans = "";

**for** (**int** i = 0; i < top; i++) {

Ans += String.*valueOf*(arr.get(i)) ;

}

Ans += String.*valueOf*(arr.get(top));

**return** Ans;

}

}

**Файл text.java**

**package** KI305.Nikolenko.Lab6;

/\*\*

\* Class <code>text</code> implements Data

\* **@author** Oleksandr Nikolenko

\* **@version** 1.0

\*\*/

**public** **class** text **implements** Data

{

**private** String textName;

**private** **int** size;

/\*\*

\* Constructor

\* **@param** tName Name of text

\* **@param** tsize Text size

\*/

**public** text(String tName, **int** tsize)

{

textName = tName;

size = tsize;

}

/\*\*

\* Method returns text name

\* **@return** text name

\*/

**public** String getTextName()

{

**return** textName;

}

/\*\*

\* Method sets the new text name

\* **@param** name text name

\*/

**public** **void** setThingName(String name)

{

textName = name;

}

/\*\*

\* Method returns text size

\* **@return** text size

\*/

**public** **int** getsize()

{

**return** size;

}

/\*\*

\* Method simulates comparing text size

\*/

**public** **int** compareTo(Data p)

{

Integer s = size;

**return** s.compareTo(p.getSize());

}

/\*\*

\* Method simulates printing info about text

\*/

**public** **void** print()

{

System.***out***.print("Text: " + textName + ", Text size: " + size + " symbols;\n");

}

@Override

**public** **int** getSize() {

// **TODO** Auto-generated method stub

**return** 0;

}

}

**Файл Variable.java**

**package** KI305.Nikolenko.Lab6;

/\*\*

\* Class <code>Thing</code> implements Data

\* **@author** Oleksandr Nikolenko

\* **@version** 1.0

\*\*/

**public** **class** Variable **implements** Data

{

**private** String varName;

**private** **int** size;

/\*\*

\* Constructor

\* **@param** vName Name of variable

\* **@param** vsize Variable size

\*/

**public** Variable (String vName, **int** vsize)

{

varName = vName;

size = vsize;

}

/\*\*

\* Method returns variable name

\* **@return** variable name

\*/

**public** String getTextName()

{

**return** varName;

}

/\*\*

\* Method sets the new variable name

\* **@param** name variable name

\*/

**public** **void** setVarName(String name)

{

varName = name;

}

/\*\*

\* Method returns variable size

\* **@return** variable size

\*/

**public** **int** getsize()

{

**return** size;

}

/\*\*

\* Method simulates comparing variable size

\*/

**public** **int** compareTo(Data p)

{

Integer s = size;

**return** s.compareTo(p.getSize());

}

/\*\*

\* Method simulates printing info about variable

\*/

**public** **void** print()

{

System.***out***.print("Variable: " + varName + ", Variable size: " + size + " byte;\n");

}

//@Override

**public** **int** getSize() {

// **TODO** Auto-generated method stub

**return** 0;

}

}

**Файл Data.java**

**package** KI305.Nikolenko.Lab6;

/\*\*

\* Interface <code>Data</code> extends Comparable

\* **@author** Oleksandr Nikolenko

\* **@version** 1.0

\*\*/

**interface** Data **extends** Comparable<Data>

{

**public** **int** getSize();

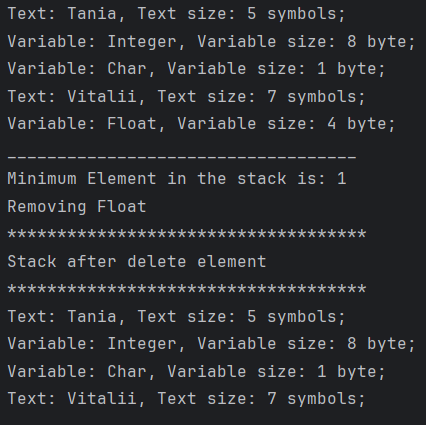
**public** **void** print();

**public** String getTextName();

**public** **int** getsize();

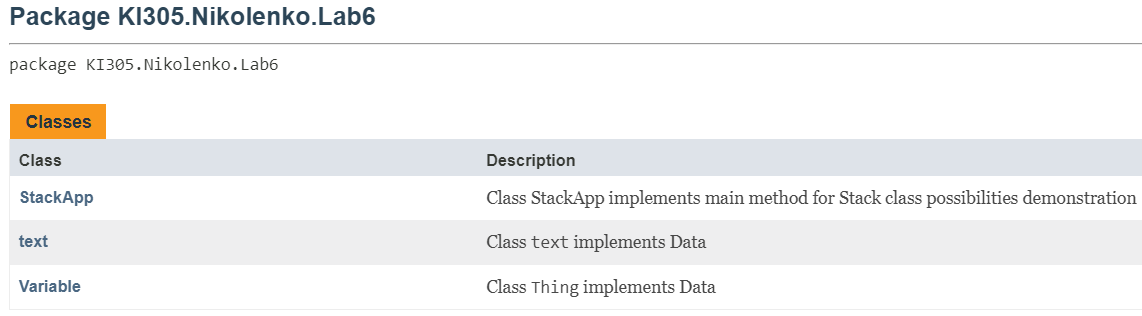
}

**Результат виконання програми:**

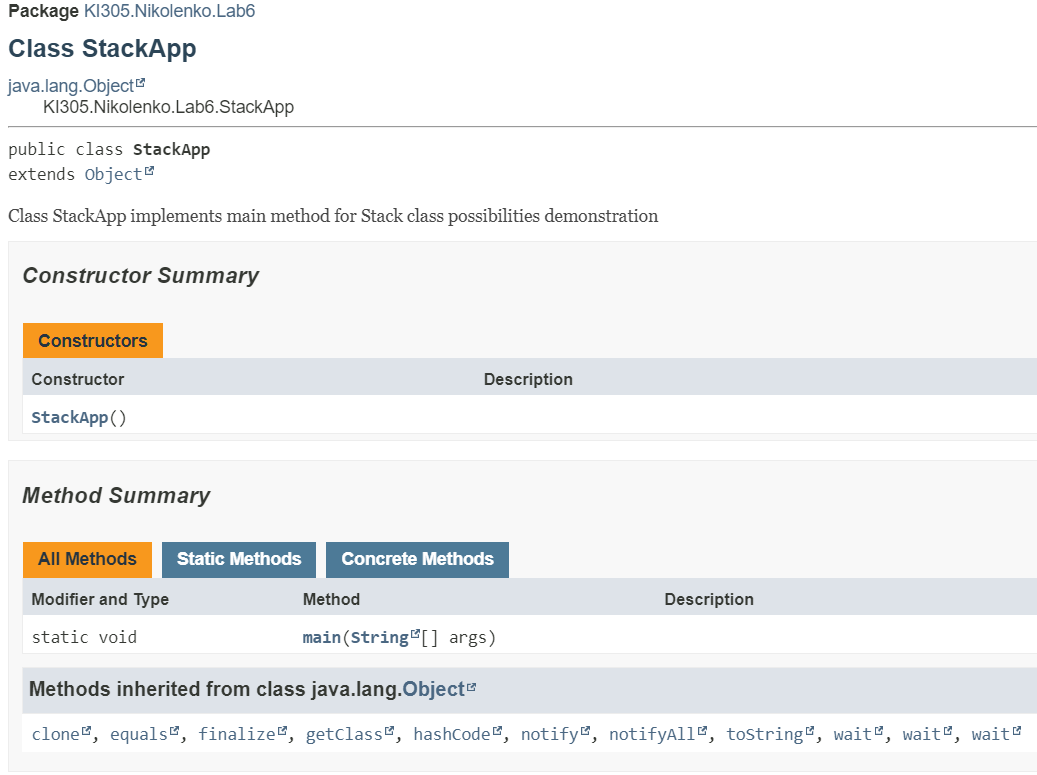
**

*Рис.1.Результат виконання програми*

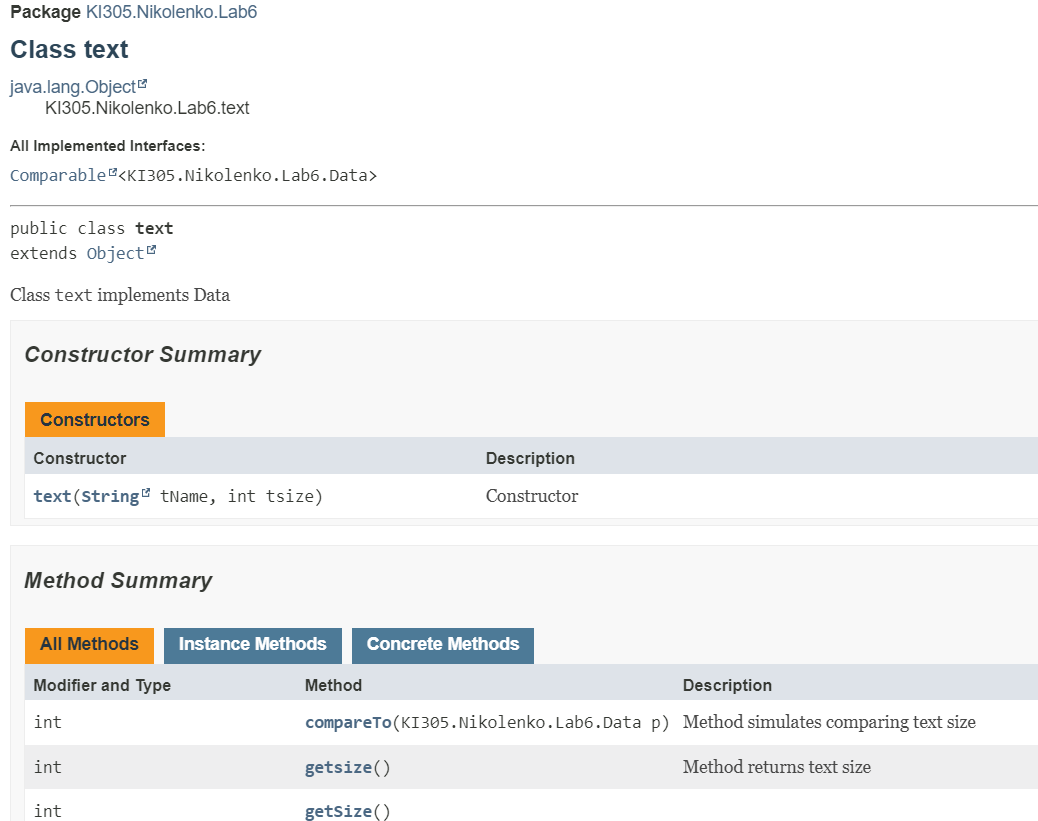
**Згенерована документація**

**

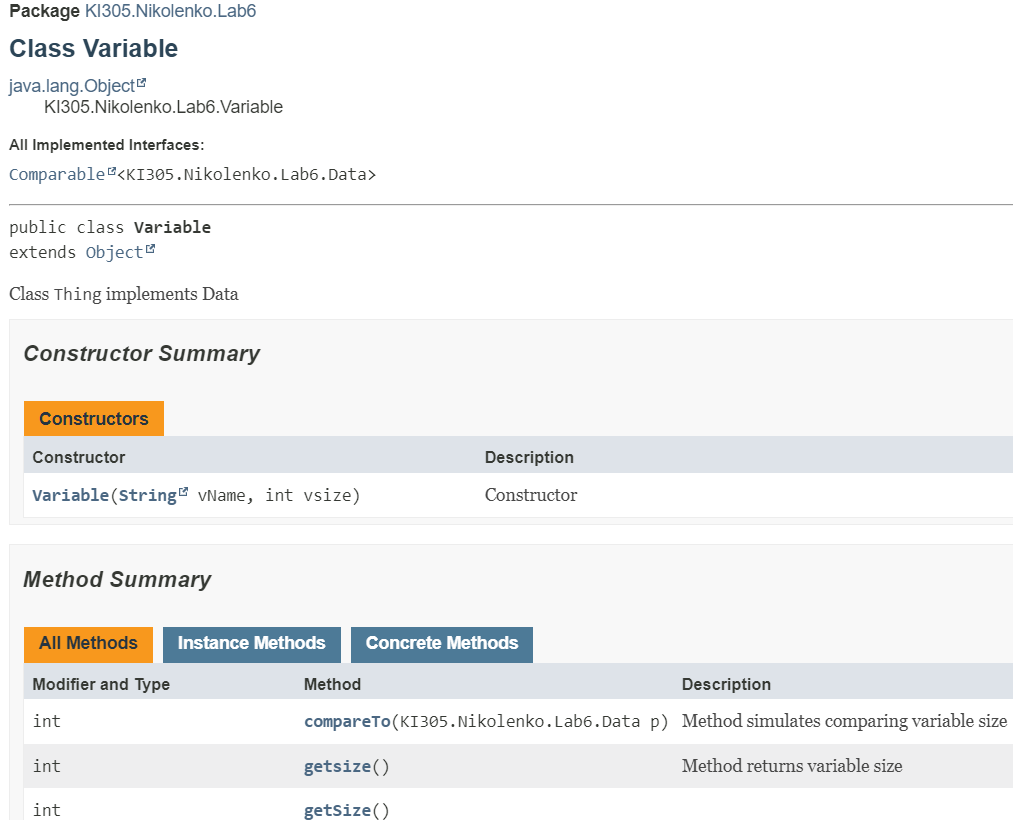
*Рис.2.Вмістиме вкладки Package*

**

*Рис.3.Вмістиме вкладки Class (StackApp)*

**

*Рис.4.Вмістиме вкладки Class (text)*

**

*Рис.5.Вмістиме вкладки Class (Variable)*

**Відповіді на контрольні запитання:**

1. **Дайте визначення терміну «параметризоване програмування».**

Параметризоване програмування є аналогом шаблонів у С++. Воно полягає у написанні коду, що можна багаторазово застосовувати з об’єктами різних класів.

1. **Розкрийте синтаксис визначення простого параметризованого класу.**

Параметризований клас – це клас з однією або більше змінними типу.

Синтаксис оголошення параметризованого класу:

[public] class НазваКласу <параметризованийТип{,параметризованийТип}>

{…}

1. **Розкрийте синтаксис створення об’єкту параметризованого класу.**

Синтаксис створення об’єкту параметризованого класу:

НазваКласу < перелікТипів > = new НазваКласу < перелікТипів > (параметри);

1. **Розкрийте синтаксис визначення параметризованого методу.**

Синтаксис оголошення параметризованого методу:

Модифікатори <параметризованийТип{,параметризованийТип}> типПовернення назваМетоду(параметри);

1. **Розкрийте синтаксис виклику параметризованого методу.**

Синтаксис виклику параметризованого методу:

(НазваКласу|НазваОб’єкту).[<перелікТипів>] НазваМетоду(параметри);

1. **Яку роль відіграє встановлення обмежень для змінних типів?**

Бувають ситуації, коли клас або метод потребують накладення обмежень на змінні типів. Наприклад, може бути ситуація, коли метод у процесі роботи викликає з-під об’єкта параметризованого типу метод, що визначається у деякому інтерфейсі. У такому випадку немає ніякої гарантії, що цей метод буде реалізований у кожному класі, що передається через змінну типу. Щоб вирішити цю проблему у мові Java можна задати обмеження на множину можливих типів, що можуть бути підставлені замість параметризованого типу.

1. **Як встановити обмеження для змінних типів?**

У мові Java можна задати обмеження на множину можливих типів, що можуть бути підставлені замість параметризованого типу. Для цього після змінної типу слід використати ключове слово extends і вказати один суперклас, або довільну кількість інтерфейсів (через знак &), від яких має походити реальний тип, що підставляється замість параметризованого типу. Якщо одночасно вказуються інтерфейси і суперклас, то суперклас має стояти першим у списку типів після ключового слова extends.Класи DataOutputStream і DataInputStream дозволяють записувати і зчитувати дані примітивних типів.

1. **Розкрийте правила спадкування параметризованих типів.**

Правила спадкування параметризованих типів:

1. Всі класи, що утворені з одного і того ж параметризованого класу з використанням різних значень змінних типів є незалежними навіть якщо між цими типами є залежність спадкування.
2. Завжди можна перетворити параметризований клас у «сирий» клас, при роботі з яким захист від некоректного коду є значно слабшим, що дозволяє здійснювати небезпечні присвоєння об’єктів параметризованого класу об’єктам «сирого» класу.
3. Параметризовані класи можуть розширювати або реалізовувати інші параметризовані класи. В цьому відношенні вони не відрізняються від звичайних класів.
4. **Яке призначення підстановочних типів?**

Підстановочні типи були введені у мову Java для збільшення гнучкості жорсткої існуючої системи параметризованих типів. На відміну від неї підстановочні типи дозволяють враховувати залежності між типами, що виступають параметрами для параметризованих типів. Це в свою чергу дозволяє застосовувати обмеження для параметрів, що підставляються замість параметризованих типів. Завдяки цьому підвищується надійність параметризованого коду, полегшується робота з ним та розділяється використання безпечних методів доступу і небезпечних модифікуючих методів.

1. **Застосування підстановочних типів.**

Підстановочні типи дозволяють реалізувати:

1. обмеження підтипу;
2. обмеження супертипу;
3. необмежені підстановки.

**Висновок:**На даній лабораторній роботі я оволодів навиками параметризованого програмування мовою Java.