

Міністерство освіти і науки України
Національний університет “Львівська політехніка”

Кафедра ЕОМ



Звіт

з лабораторної роботи №7

з дисципліни: «Кросплатформенні засоби програмування»

на тему: «Параметризоване програмування»

Виконав: ст.гр. КІ-34

Скалій Т.В.

Прийняв:

викл. каф. ЕОМ

Іванов Ю. С.

Львів 2022

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 розміщуються у екземплярі розробленого класу-контейнеру. Програма має розміщуватися в пакеті Група.Прізвище.Lab7 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 KI34.Skalii.Lab7;
/**
 * Class StackApp implements main method for Stack
 * class possibilities demonstration
 * @author Tetiana Skalii
 * @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 KI34.Skalii.Lab7;
import java.util.ArrayList;
/**
 * @author Tetiana Skalii
 * 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 KI34.Skalii.Lab7;
/**
 * Class <code>text</code> implements Data
 * @author Tetiana Skalii
 * @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 KI34.Skalii.Lab7;

/**
 * Class <code>Thing</code> implements Data
 * @author Tetiana Skalii
 * @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 getsizes()
    {
        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

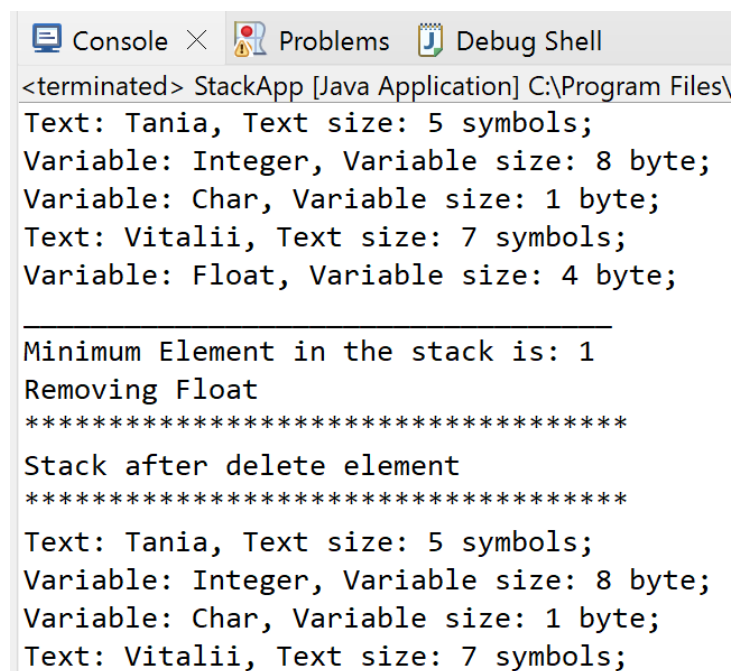
```

package KI34.Skalii.Lab7;

/**
 * Interface <code>Data</code> extends Comparable
 * @author Tetiana Skalii
 * @version 1.0
 */
interface Data extends Comparable<Data>
{
    public int getSize();
    public void print();
    public String getTextName();
    public int getsizes();
}

```

Результат виконання програми:



```

<terminated> StackApp [Java Application] C:\Program Files\
Text: Tania, Text size: 5 symbols;
Variable: Integer, Variable size: 8 byte;
Variable: Char, Variable size: 1 byte;
Text: Vitalii, Text size: 7 symbols;
Variable: Float, Variable size: 4 byte;

Minimum Element in the stack is: 1
Removing Float
*****
Stack after delete element
*****
Text: Tania, Text size: 5 symbols;
Variable: Integer, Variable size: 8 byte;
Variable: Char, Variable size: 1 byte;
Text: Vitalii, Text size: 7 symbols;

```

Рис.1.Результат виконання програми:

Згенерована документація

Package KI34.Skalii.Lab7

package KI34.Skalii.Lab7

All Classes and Interfaces	Interfaces	Classes
Class	Description	
Data	Interface Data extends Comparable	
Stack <T extends Data >	Class Stack implements Stack	
StackApp	Class StackApp implements main method for Stack class possibilities demonstration	
text	Class text implements Data	
Variable	Class Thing implements Data	

Рис.2.Вмістиме вкладки Package

Package KI34.Skalii.Lab7

Interface Data

All Superinterfaces:

Comparable[↗]<Data>

All Known Implementing Classes:

text, Variable

```
interface Data
```

```
extends Comparable↗<Data>
```

Interface Data extends Comparable

Version:

1.0

Author:

Tetiana Skalii

Method Summary

All Methods	Instance Methods	Abstract Methods
Modifier and Type	Method	Description
int	getsize()	
int	getSize()	
String [↗]	getTextName()	
void	print()	
Methods inherited from interface java.lang.Comparable [↗]		
	compareTo [↗]	

Рис.3.Вмістиме вкладки Class (interface Data)

Package

KI34.Skalii.Lab7

Class Stack<T extends Data>

java.lang.Object[Ⓔ]
KI34.Skalii.Lab7.Stack<T>

class Stack<T extends Data>
extends Object[Ⓔ]

Class Stack implements Stack

Version:
1.0

Author:
Tetiana Skalii

Field Summary

Fields

Modifier and Type	Field	Description
(package private) int	minEle	

Constructor Summary

Constructors

Constructor	Description
Stack(int size)	Constructor

Method Summary

All Methods

Instance Methods

Concrete Methods

Modifier and Type	Method	Description
(package private) void	getMin()	Method simulates finding the MinElement in Stack
int	GetSize()	
boolean	IsEmpty()	
boolean	IsFull()	
T	Peek()	
T	Pop()	Method simulates deleting data
void	Print()	
void	Push(T Data)	Method simulates push data
String [Ⓔ]	toString()	

Methods inherited from class java.lang.Object[Ⓔ]
clone[Ⓔ], equals[Ⓔ], finalize[Ⓔ], getClass[Ⓔ], hashCode[Ⓔ], notify[Ⓔ], notifyAll[Ⓔ], wait[Ⓔ], wait[Ⓔ], wait[Ⓔ]

Рис.4.Вмістиме вкладки Class (Stack)

Package

KI34.Skalii.Lab7

Class StackApp

java.lang.Object[Ⓔ]
KI34.Skalii.Lab7.StackApp

public class StackApp
extends Object[Ⓔ]

Class StackApp implements main method for Stack class possibilities demonstration

Version:
1.0

Author:
Tetiana Skalii

Constructor Summary

Constructors

Constructor	Description
StackApp()	

Method Summary

All Methods

Static Methods

Concrete Methods

Modifier and Type	Method	Description
static void	main(String [Ⓔ] [] args)	

Methods inherited from class java.lang.Object[Ⓔ]
clone[Ⓔ], equals[Ⓔ], finalize[Ⓔ], getClass[Ⓔ], hashCode[Ⓔ], notify[Ⓔ], notifyAll[Ⓔ], toString[Ⓔ], wait[Ⓔ], wait[Ⓔ], wait[Ⓔ]

Рис.5.Вмістиме вкладки Class (StackApp)

Package [Kl34.Skalii.Lab7](#)

Class text

[java.lang.Object](#)[Ⓔ]
[Kl34.Skalii.Lab7.text](#)

All Implemented Interfaces:
[Comparable](#)[Ⓔ]<[Data](#)>, [Data](#)

```
public class text
extends ObjectⒺ
implements Data
```

Class text implements Data

Version:

1.0

Author:

Tetiana Skalii

Constructor Summary

Constructors

Constructor	Description
text(String[Ⓔ] tName, int tsize)	Constructor

Method Summary

All Methods

Instance Methods

Concrete Methods

Modifier and Type	Method	Description
int	compareTo(Data p)	Method simulates comparing text size
int	getsize()	Method returns text size
int	getSize()	
String [Ⓔ]	getTextName()	Method returns text name
void	print()	Method simulates printing info about text
void	setThingName(String[Ⓔ] name)	Method sets the new text name

Methods inherited from class [java.lang.Object](#)[Ⓔ]

[clone](#)[Ⓔ], [equals](#)[Ⓔ], [finalize](#)[Ⓔ], [getClass](#)[Ⓔ], [hashCode](#)[Ⓔ], [notify](#)[Ⓔ], [notifyAll](#)[Ⓔ], [toString](#)[Ⓔ], [wait](#)[Ⓔ], [wait](#)[Ⓔ], [wait](#)[Ⓔ]

Рис.6.Вмістиме вкладки Class (text)

Package [Kl34.Skalii.Lab7](#)

Class Variable

[java.lang.Object](#)[Ⓔ]
[Kl34.Skalii.Lab7.Variable](#)

All Implemented Interfaces:
[Comparable](#)[Ⓔ]<[Data](#)>, [Data](#)

```
public class Variable
extends ObjectⒺ
implements Data
```

Class Variable implements Data

Version:

1.0

Author:

Tetiana Skalii

Constructor Summary

Constructors

Constructor	Description
Variable(String[Ⓔ] vName, int vsize)	Constructor

Method Summary

All Methods

Instance Methods

Concrete Methods

Modifier and Type	Method	Description
int	compareTo(Data p)	Method simulates comparing variable size
int	getsize()	Method returns variable size
int	getSize()	
String [Ⓔ]	getTextName()	Method returns variable name
void	print()	Method simulates printing info about variable
void	setVarName(String[Ⓔ] name)	Method sets the new variable name

Methods inherited from class [java.lang.Object](#)[Ⓔ]

[clone](#)[Ⓔ], [equals](#)[Ⓔ], [finalize](#)[Ⓔ], [getClass](#)[Ⓔ], [hashCode](#)[Ⓔ], [notify](#)[Ⓔ], [notifyAll](#)[Ⓔ], [toString](#)[Ⓔ], [wait](#)[Ⓔ], [wait](#)[Ⓔ], [wait](#)[Ⓔ]

Рис.7.Вмістиме вкладки Class (Variable)

Відповіді на контрольні запитання:

1. Дайте визначення терміну «параметризоване програмування».

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

2. 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}>
{...}
```

3. Розкрийте синтаксис створення об'єкту параметризованого класу.

Синтаксис створення об'єкту параметризованого класу:

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

4. Розкрийте синтаксис визначення параметризованого методу.

Синтаксис оголошення параметризованого методу:

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

5. Розкрийте синтаксис виклику параметризованого методу.

Синтаксис виклику параметризованого методу:

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

6. 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.

7. 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.

8. 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 «сирій» 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.

9. 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.

10. Застосування підстановочних типів.

Підстановочні типи дозволяють реалізувати:

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

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