1.Реализовать простой класс.
Требования к выполнению
• Реализовать пользовательский класс по варианту.
• Создать другой класс с методом main, в котором будут находится примеры использования
пользовательского класса.
Для каждого класса
• Создать поля классов
• Создать методы классов
• Добавьте необходимые get и set методы (по необходимости)
• Укажите соответствующие модификаторы видимости
• Добавьте конструкторы
• Переопределить методы toString() и equals()

1) Равнобедренный треугольник, заданный длинами сторон – Предусмотреть возмож-
ность определения площади и периметра, а так же логический метод, определяющий суще-
ствует или такой треугольник. Конструктор должен позволить создавать объекты с начальной

инициализацией. Реализовать метод equals, выполняющий сравнение объектов данного типа.

Код программы

```
package com.company;

class IsoscelesTriangle
{
   private double basis; //основание
   private double side; //две боковые равные стороны

   public IsoscelesTriangle()
   {
      basis = 0;
      side = 0;
   }

   public IsoscelesTriangle(double basis, double side)
   {
```

```
    this.basis = basis;
    this.side = side;
}

public double getBasis()
{
    return basis;
}

public double getSide()
{
    return side;
}

public void setBasis(double basis)
{
    this.basis = basis;
}

public void setSide(double side)
{
    this.side = side;
}

public boolean isExist()
{
    if(2*side>basis && basis+side>side)
        return true;
    else
        return false;
}

public double getSquare()
{
    double h = Math.sqrt( Math.pow(side, 2) - Math.pow(basis/2, 2) );

    return basis*h/2;
}

public double getPerimeter()
{
    return basis + 2*side;
}

public boolean equals(IsoscelesTriangle otherIsoscelesTriangle)
{
```

```
        if(basis == otherIsoscelesTriangle.basis && side == otherIsoscelesTriangle.side &&
basis>0 && side>0)
            return true;
        else
            return false;
    }

    public String toString()
    {
        StringBuilder builder = new StringBuilder();

        builder.append("IsoscelesTriangle { basis: ");
        builder.append(basis);
        builder.append(", side: ");
        builder.append(side);
        builder.append(" }");

        return builder.toString();
    }
}

public class Main {

    public static void main(String[] args) {
        IsoscelesTriangle triangleNotExist = new IsoscelesTriangle(5, 2);

        System.out.println("triangleNotExist exists? "+triangleNotExist.isExist());

        IsoscelesTriangle triangleEmpty = new IsoscelesTriangle();
        System.out.println(triangleEmpty.toString());

        System.out.println("triangleEmpty exists? "+triangleEmpty.isExist());

        IsoscelesTriangle triangle = new IsoscelesTriangle(4, 3);

        triangle.setBasis(7);
        triangle.setSide(5);
        System.out.println("triangle basis: "+triangle.getBasis());
        System.out.println("triangle side: "+triangle.getSide());

        System.out.println("triangle exists? "+triangle.isExist());

        System.out.println("triangle square: "+triangle.getSquare());
        System.out.println("triangle perimeter: "+triangle.getPerimeter());

        IsoscelesTriangle triangleEqual = new IsoscelesTriangle(7, 5);
```

```
        System.out.println("triangle = triangleEqual ? "+triangle.equals(triangleEqual));
        System.out.println("triangle = triangleNotExist ? "+triangle.equals(triangleNotExist));
    }
}
```

Результат

```
triangleNotExist exists? false
IsoscelesTriangle { basis: 0.0, side: 0.0 }
triangleEmpty exists? false
triangle basis: 7.0
triangle side: 5.0
triangle exists? true
triangle square: 12.497499749949988
triangle perimeter: 17.0
triangle = triangleEqual ? true
triangle = triangleNotExist ? false
```

2.Разработать автоматизированную систему на основе некоторой структуры данных, манипулирующей объектами пользовательского класса. Реализовать требуемые функции обработки данных

Требования к выполнению

• Задание посвящено написанию классов, решающих определенную задачу автоматизации;

• Данные для программы загружаются из файла (формат произволен). Файл создать и написать вручную.

1) Стековый калькулятор

Написать стековый калькулятор, который принимает в качестве аргумента командой строки имя файла, содержащего команды. Если аргумента нет, то использовать стандартный поток ввода для чтения команд. Для вычислений допускается использовать вещественные числа.

Реализовать следующий набор команд:

• # – строка с комментарием.

• POP , PUSH – снять/положить число со/на стек(а).

• + , - , * , /, SQRT – арифметические операции. Используют один или два верхних элемента

стека, изымают их из стека, помещая результат назад

• PRINT – печать верхнего элемента стека (без удаления).

• DEFINE – задать значение параметра. В дальнейшем везде использовать вместо параметра

это значение.

Содержимое стека и список определенных именованных параметров передавать команде в виде специального объекта – контекста исполнения. Разработать группу классов исключений, которые будут выбрасывать команды при исполнении. В случае возникновения исключения – выводить информацию об ошибке и продолжать исполнение программы (из файла или команд вводимых с консоли)

Код программы

Main:

```
package com.company;

import com.company.programexceptions.SqrtFromNegativeException;
import com.company.programexceptions.StackEmptyException;
import com.company.programexceptions.UnknownCommandException;
import com.company.programexceptions.ZeroDivisionException;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
import java.util.Stack;

public class Main {

    public static void main(String[] args) {


        ExecContext context = new ExecContext();
        Commands commands = new Commands();

        File file;
        Scanner scanner;

        if(args.length==0)
            scanner = new Scanner(System.in);
        else if(args.length==1) {
            try {
                file = new File(args[0]);
                scanner = new Scanner(file, "utf-8");
            }
            catch(FileNotFoundException e){
                System.out.println("Error: file not found");
                return;
```

```
    }
}
else{
    System.out.println("Error: unexpected params");
    return;
}

while(scanner.hasNextLine())
{
    String command = scanner.nextLine();

    if(command.indexOf("#")==0)
        continue;
    else if(command.indexOf("POP")==0)
    {
        try
        {
            context = commands.PopCommand(context, command);
        }
        catch(StackEmptyException e)
        {
            System.out.println("Error: stack is empty or too few values in stack");
        }
        catch(UnknownCommandException e)
        {
            System.out.println("Error: unknown command");
        }
        catch(Exception e)
        {
            System.out.println("Error: sorry, unexpected error");
        }
    }
    else if(command.indexOf("PUSH")==0)
    {
        try
        {
            context = commands.PushCommand(context, command);
        }
        catch(UnknownCommandException e)
        {
            System.out.println("Error: unknown command");
        }
        catch(Exception e)
        {
            System.out.println("Error: sorry, unexpected error");
            e.printStackTrace();
```

```
    }
}
else if(command.indexOf("+")==0)
{
    try
    {
        context = commands.PlusCommand(context, command);
    }
    catch(StackEmptyException e)
    {
        System.out.println("Error: stack is empty or too few values in stack");
    }
    catch(UnknownCommandException e)
    {
        System.out.println("Error: unknown command");
    }
    catch(Exception e)
    {
        System.out.println("Error: sorry, unexpected error");
    }
}
else if(command.indexOf("-")==0)
{
    try
    {
        context = commands.MinusCommand(context, command);
    }
    catch(StackEmptyException e)
    {
        System.out.println("Error: stack is empty or too few values in stack");
    }
    catch(UnknownCommandException e)
    {
        System.out.println("Error: unknown command");
    }
    catch(Exception e)
    {
        System.out.println("Error: sorry, unexpected error");
    }
}
else if(command.indexOf("*")==0)
{
    try
    {
        context = commands.MultiplyCommand(context, command);
    }
```

```
    catch(StackEmptyException e)
    {
        System.out.println("Error: stack is empty or too few values in stack");
    }
    catch(UnknownCommandException e)
    {
        System.out.println("Error: unknown command");
    }
    catch(Exception e)
    {
        System.out.println("Error: sorry, unexpected error");
    }
}
else if(command.indexOf("/")==0)
{
    try
    {
        context = commands.DivideCommand(context, command);
    }
    catch(StackEmptyException e)
    {
        System.out.println("Error: stack is empty or too few values in stack");
    }
    catch(UnknownCommandException e)
    {
        System.out.println("Error: unknown command");
    }
    catch(ZeroDivisionException e)
    {
        System.out.println("Error: zero division");
    }
    catch(Exception e)
    {
        System.out.println("Error: sorry, unexpected error");
    }
}
else if(command.indexOf("SQRT")==0)
{
    try
    {
        context = commands.SqrtCommand(context, command);
    }
    catch(StackEmptyException e)
    {
        System.out.println("Error: stack is empty or too few values in stack");
    }
```

```
    catch(UnknownCommandException e)
    {
        System.out.println("Error: unknown command");
    }
    catch(SqrtFromNegativeException e)
    {
        System.out.println("Error: sqrt from negative");
    }
    catch(Exception e)
    {
        System.out.println("Error: sorry, unexpected error");
    }
}
else if(command.indexOf("PRINT")==0)
{
    try
    {
        commands.PrintCommand(context, command);
    }
    catch(StackEmptyException e)
    {
        System.out.println("Error: stack is empty or too few values in stack");
    }
    catch(UnknownCommandException e)
    {
        System.out.println("Error: unknown command");
    }
    catch(Exception e)
    {
        System.out.println("Error: sorry, unexpected error");
    }
}
else if(command.indexOf("DEFINE")==0)
{
    try
    {
        commands.DefineCommand(context, command);
    }
    catch(UnknownCommandException e)
    {
        System.out.println("Error: unknown command");
    }
    catch(Exception e)
    {
        System.out.println("Error: sorry, unexpected error");
    }
```

```
            }
            else if(command.indexOf("EXIT")==0)
                break;
            else
            {
                System.out.println("Error: unknown command");
            }
        }
    }
}
```

ExecContent:

```
package com.company;

import java.util.HashMap;
import java.util.Stack;

public class ExecContext {

    private Stack<Double> stack;
    private HashMap<String, Double> params;

    public ExecContext() {
        stack = new Stack<Double>();
        params = new HashMap<String,Double>();
    }

    public void pushInStack(double value){
        stack.push(value);
    }

    public double popFromStack(){
        return stack.pop();
    }

    public double peekFromStack(){
        return stack.peek();
    }

    public boolean stackIsEmpty(){
        return stack.empty();
    }

    public void setParam(String name, double value){
```

```
        params.put(name, value);
    }

    public Double getParam(String name){
        return params.get(name);
    }
}
```

Commands:

```
package com.company;

import com.company.programexceptions.SqrtFromNegativeException;
import com.company.programexceptions.StackEmptyException;
import com.company.programexceptions.UnknownCommandException;
import com.company.programexceptions.ZeroDivisionException;

public class Commands {

    public ExecContext PopCommand(ExecContext context, String command) throws Exception
    {
        if(!command.equals("POP"))
            throw new UnknownCommandException();

        if(context.stackIsEmpty())
            throw new StackEmptyException();

        context.popFromStack();

        return context;
    }

    public ExecContext PushCommand(ExecContext context, String command) throws Exception
    {
        String[] args = command.split(" ");

        if(args.length!=2)
            throw new UnknownCommandException();

        if(!args[0].equals("PUSH"))
            throw new UnknownCommandException();

        Double value = null;
        try{
            value = Double.parseDouble(args[1]);
```

```
        }
        catch(NumberFormatException e)
        {
            value = context.getParam(args[1]);
        }

        if(value==null)
            throw new UnknownCommandException();

        context.pushInStack(value);

        return context;
    }

    public ExecContext PlusCommand(ExecContext context, String command) throws Exception
    {
        if(!command.equals("+"))
            throw new UnknownCommandException();

        double value1, value2;

        if(context.stackIsEmpty())
            throw new StackEmptyException();

        value2 = context.popFromStack();

        if(context.stackIsEmpty()) {
            context.pushInStack(value2);
            throw new StackEmptyException();
        }

        value1 = context.popFromStack();

        context.pushInStack(value1+value2);

        return context;
    }

    public ExecContext MinusCommand(ExecContext context, String command) throws
Exception
    {
        if(!command.equals("-"))
            throw new UnknownCommandException();

        double value1, value2;
```

```
        if(context.stackIsEmpty())
            throw new StackEmptyException();

        value2 = context.popFromStack();

        if(context.stackIsEmpty()) {
            context.pushInStack(value2);
            throw new StackEmptyException();
        }

        value1 = context.popFromStack();

        context.pushInStack(value1-value2);

        return context;
    }

    public ExecContext MultiplyCommand(ExecContext context, String command) throws
Exception
    {
        if(!command.equals("*"))
            throw new UnknownCommandException();

        double value1, value2;

        if(context.stackIsEmpty())
            throw new StackEmptyException();

        value2 = context.popFromStack();

        if(context.stackIsEmpty()) {
            context.pushInStack(value2);
            throw new StackEmptyException();
        }

        value1 = context.popFromStack();

        context.pushInStack(value1*value2);

        return context;
    }

    public ExecContext DivideCommand(ExecContext context, String command) throws
Exception
    {
        if(!command.equals("/"))
```

```
        throw new UnknownCommandException();

    double value1, value2;

    if(context.stackIsEmpty())
        throw new StackEmptyException();

    value2 = context.popFromStack();

    if(value2==0) {
        context.pushInStack(value2);
        throw new ZeroDivisionException();
    }

    if(context.stackIsEmpty()) {
        context.pushInStack(value2);
        throw new StackEmptyException();
    }

    value1 = context.popFromStack();

    context.pushInStack(value1/value2);

    return context;
}

public ExecContext SqrtCommand(ExecContext context, String command) throws Exception
{
    if(!command.equals("SQRT"))
        throw new UnknownCommandException();

    double value;

    if(context.stackIsEmpty())
        throw new StackEmptyException();

    value = context.popFromStack();

    if(value<0) {
        context.pushInStack(value);
        throw new SqrtFromNegativeException();
    }

    context.pushInStack(Math.sqrt(value));

    return context;
```

```
    }

    public void PrintCommand(ExecContext context, String command) throws Exception
    {
        if(!command.equals("PRINT"))
            throw new UnknownCommandException();

        if(context.stackIsEmpty())
            throw new StackEmptyException();

        System.out.println(context.peekFromStack());
    }

    public ExecContext DefineCommand(ExecContext context, String command) throws
Exception
    {
        String[] args = command.split(" ");

        if(args.length!=3)
            throw new UnknownCommandException();

        if(!args[0].equals("DEFINE"))
            throw new UnknownCommandException();

        Double value = null;
        try{
            value = Double.parseDouble(args[2]);
        }
        catch(NumberFormatException e)
        {
            throw new UnknownCommandException();
        }

        context.setParam(args[1], value);

        return context;

    }
}
```

Файл команд

```
#it's comment
DEFINE param1 25
DEFINE param2 15
PUSH param1
PUSH param2
+
PRINT
PUSH 10
-
PRINT
PUSH 2
*
PRINT
PUSH 0
/
POP
PUSH 5
/
PRINT
PUSH 13
+
SQRT
PRINT
```

Результат

```
PUSH 13
PUSH 12
-
PRINT
1.0
PUSH 144
SQRT
PRINT
12.0
```