Министерство образования и науки Российской Федерации

Северо-Восточный Федеральный Университет им. М.К. Аммосова

Институт математики и информатики

Кафедра математической экономики и прикладной информатики

ОТЧЕТ

По дисциплине алгоритмизация и программирование

На тему наследование, обработка исключительных ситуаций.

**Студент:** Чин-фю Алина Васильевна

Группа: ПИ-19-2

Якутск 2019

# Задание 4.1

Пример 4.1: Выводится “0”, затем вызывается ошибка RuntimeException. Эту ошибку ловит catch и выводит “1” и “java.lang.RuntimeException: Непроверяемая ошибка”. В конце выводит “2”.

Пример 4.2: Выводится “0”, затем вызывается ошибка RuntimeException, пытается вывести “1”, но программа ломается, выводя ошибку java: unreachable statement.

Пример 4.3: Выводится “0”, затем вызывается ошибка RuntimeException, после этого она перехватывается перехватчиком RuntimeException e, выводя “2” и игнорируя NullPointerException и пропуская Error. В конце выводит “4”.

Пример 4.4: Выводится “0”, затем вызывается ошибка RuntimeException. После этого она перехватывается перехватчиком Exception e, выводя “2”, игнорируя NullPointerException и пропуская Error. В конце выводит “4”.

Пример 4.5: Выводится “0”, затем вызывается ошибка RuntimeException. Ошибка не перехватывается и вызывает ошибку RuntimeException: ошибка.

Пример 4.6: Выводится “0”, затем вызывается ошибка RuntimeException. Она не перехватывается, так как последовательность перехвата не соответствует иерархии исключений, программа ломается.

Пример 4.7: Выводится “0”, затем вызывается ошибка NullPointerException. Ошибка перехватывается, выводит “1” и вызывается ошибка ArithmeticException. Программа выдает ошибку, так как нельзя вызывать ошибки в catch.

Пример 4.8: Выводится “0”, затем вызывается ошибка RuntimeException. В любом случае выводится “1” так как он находится в finally. Программа выдает ошибку, так как RuntimeException не перехватывается.

Пример 4.9: Выводится “0”, затем возвращается 55, то есть выходит из метода выводя 55, но так как в программе имеется finally с sout(“1”) сначала выведется 1, а затем 55.

Пример 4.10: Выводится “0”, возвращается 15. Так как тут имеется finally, то выводится “1” и возвращается 20, в итоге выводится:

**0**

**1**

**20**

Пример 4.11: Выводится “0”, затем вызывается ошибка NullPointerException. Ошибка перехватывается и выводит “1”, потом выводится “2” из finally. После всего этого выводится “3”.

Пример 4.12: Из main в m передается 2 значения – «str = null» и «сhislo = 0.000001». Если str == null, то вызывается ошибка IllegalArgumentException.

Пример 4.13: Задается переменная l = длине массива args, выводится “Размер массива= ” длина массива args = 0

h/l – получается ошибка ArithmeticException. Присваивается String значение “10” элементу массива args с индексом l +1 (0 + 1). Перехватывается ArithmeticException и выводится “Деление на ноль”. Выходит из try.

Пример 4.14: Задается переменная l = длине массива args. Выводится “Размер массива= ” длина массива args = 0, вызывается метод m (x = l), там задается h = 10/x. Ошибка перехватывается и выводится “Ошибка: Деление на ноль”.

# Задание 4.2(11 вариант)

Задание 1

import java.util.InputMismatchException;  
import java.util.Scanner;  
  
public class ex4\_2\_1 {  
 public static void main(String[] args) {  
 try {  
 Scanner sc = new Scanner(System.*in*);  
 int x = sc.nextInt();  
 int[][] arr = new int[x][x];  
 int max = 1;  
 for(int i = 1; i <= x; i++){  
 for(int j = 1; j <= x; j++){  
 System.*out*.println("arr["+i+"]["+j+"]");  
 int k = sc.nextInt();  
 arr[i-1][j-1] = k;  
 }  
 if(arr[i-1][i-1] % 2 == 0){  
 max = arr[i-1][i-1];  
 }  
 }  
 System.*out*.println("Массив:");  
 for(int i = 0; i < x; i++){  
 for(int j = 0; j < x; j++){  
 System.*out*.print("["+arr[i][j]+"] ");  
 }  
 System.*out*.println();  
 }  
 for(int i = 0; (i < x)&&(i < x); i++){  
 if((arr[i][i] % 2 == 0) && (arr[i][i] > max)){  
 max = arr[i][i];  
 }  
 }  
 if(max % 2 == 1){  
 throw new RuntimeException();  
 }else{  
 System.*out*.println("Максимум среди четных чисел на главной диагонали");  
 System.*out*.println("["+max+"]");  
 }  
 }catch (InputMismatchException e) {  
 System.*out*.println("Ввод строки вместо числа");  
 }catch (RuntimeException e) {  
 System.*out*.println("Hет четных чисел");  
 }  
 }  
}

Словесное описание алгоритма:

1. Сначала пытаемся перехватить ошибки, вписываем длину квадратного массива и инициализируем переменную max = 1;
2. Потом заполняем массив;
3. Выводим массив;
4. Ищем максимальное четное число на главной диагонали
5. Если max = 1 то тогда кидаем ошибку RuntimeException(), если нет то тогда выводим «Максимум среди четных чисел на главной диагонали» и само число;
6. Если ошибка RuntimeException() перехватывается, то выводится «Нет четных чисел» и если InputMismatchException() перехватывается, то выводится «Ввод строки вместо числа»

Задание 2

import java.util.InputMismatchException;  
import java.util.Scanner;  
  
public class ex4\_2\_2 {  
 public static void createArr(){  
 Scanner sc = new Scanner(System.*in*);  
 int x = sc.nextInt();  
 int[][] arr = new int[x][x];  
 int max = 1;  
 for(int i = 1; i <= x; i++){  
 for(int j = 1; j <= x; j++){  
 System.*out*.println("arr["+i+"]["+j+"]");  
 int k = sc.nextInt();  
 arr[i-1][j-1] = k;  
 }  
 if(arr[i-1][i-1] % 2 == 0){  
 max = arr[i-1][i-1];  
 }  
 }  
 *writeArr*(arr);  
 *diagonal*(arr, max);  
 }  
  
 public static void writeArr(int[][] arr){  
 System.*out*.println("Массив:");  
 for(int i = 0; i < arr.length; i++){  
 for(int j = 0; j < arr[0].length; j++){  
 System.*out*.print("["+arr[i][j]+"] ");  
 }  
 System.*out*.println();  
 }  
 }  
  
 public static void diagonal(int[][] arr, int max){  
 for(int i = 0; (i < arr.length)&&(i < arr[0].length); i++){  
 if((arr[i][i] % 2 == 0) && (arr[i][i] > max)){  
 max = arr[i][i];  
 }  
 }  
 if(max % 2 == 1){  
 throw new RuntimeException();  
 }else{  
 System.*out*.println("Максимум среди четных чисел на главной диагонали");  
 System.*out*.println("["+max+"]");  
 }  
 }  
  
 public static void main(String[] args) {  
 try{  
 *createArr*();  
 }catch (InputMismatchException e) {  
 System.*out*.println("Ввод строки вместо числа");  
 }catch (RuntimeException e) {  
 System.*out*.println("Hет четных чисел");  
 }  
 }  
}

1. В методе main пытаемся перехватить ошибки, вызываем метод createArr;
2. В методе createArr, в котором инициализируется переменная max = 1, затем вписываем длину массива и заполняем его, вызываем методы writeArr и diagonal (в первую отсылаем наш массив, во вторую отсылаем массив с переменной max);
3. writeArr выводит наш квадратный массив;
4. diagonal находит максимальное четное число в главной диагонали и если max не изменился, то вызывается ошибка RuntimeException();
5. Возвращаемся к методу main и если перехватилась InputMismatchException(), то выводится «Ввод строки вместо числа», если перехватилась RuntimeException(), то выводится «Нет четных чисел»

# КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое наследование?

Наследование - это механизм, позволяющий создавать классы на основе другого класса.

2. Иерархия классов исключений, базовые классы и их статус.

Исключение – это ошибка, возникающая во время исполнения программы (но не в процессе компиляции). В JVM предусмотрена реакция на любую ошибку – автоматическое срабатывание обработчика исключений по умолчанию. В результате этого программа завершает свою работу, а пользователь видит на экране сформированную трассу стека (Stack Trace), где указывается класс, соответствующий перехваченному исключению, место расположения ошибки и последовательность вызываемых методов, через которые эта ошибка передается («летит»).

На вершине иерархии исключений стоит класс Throwable, являющийся наследником класса Object. Каждый из типов исключений является подклассом класса Throwable. Два непосредственных наследника класса Throwable – Error и Exception делят иерархию подклассов исключений на две различные ветви.

3. Типовая структура обработчика исключительной ситуации.

К механизму обработки исключений в Java имеют отношение 5 клю­чевых слов: — **try**, **catch**, **throw**, **throws** и **finally**. Схема работы этого механизма следующая. Вы пытаетесь (try) выполнить блок кода, и если при этом возникает ошибка, система возбуждает (throw) исключение, ко­торое в зависимости от его типа вы можете перехватить (catch) или пере­дать умалчиваемому (finally) обработчику.

**try** – попытаться выполнить;

**catch** – перехватить и обработать ошибку;

**finally** – окончательно (финальный блок, выполняемый всегда);

**throw** – генерация («бросание») исключения;

**throws** – пометка метода, «бросающего» исключение.

4. Последовательность перехвата ошибок родительскими и дочерними классами.

Последовательность перехвата должна соответствовать иерархии классов исключений. Предок не должен перехватывать исключения раньше потомков.

5. Как помечается метод, генерирующий исключительную ситуацию, но не обрабатывающий ее?

Если метод способен к порождению исключений, которые он не обрабатывает, он должен быть определен так, чтобы вызывающие методы могли сами предохранять от данного исключения. Для этого используется ключевое слово throws в сигнатуре метода. Это необходимо для всех исключений, кроме исключений типа Error и RuntimeException, и, соответственно, для любых их подклассов.