ОСНОВЫ ОБРАБОТКИ ИСКЛЮЧЕНИЙ

Нередко в процессе выполнения программы могут возникать ошибки, при том необязательно по вине разработчика.

Некоторые из них трудно предусмотреть или предвидеть, а иногда и вовсе невозможно. Так, например, может неожиданно оборваться сетевое подключение при передаче файла. Подобные ситуации называются **исключениями**.

В языке Java предусмотрены специальные средства для обработки подобных ситуаций. Одним из таких средств является конструкция **try...catch...finally**.

Пример 1.
Возникновение исключения – выход за границы массива

Так как массив numbers может содержать только 3 элемента, то при выполнении инструкции numbers[4]=45 консоль отобразит исключение, и выполнение программы будет завершено.

```
public class Main {
    public static void main(String[] args) {

        int[] numbers = new int[3];
        numbers[4]=45;
        System.out.println(numbers[4]);

        }
    }
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
      at Main.main(Main.java:6)

Process finished with exit code 1
```

При использовании блока **try...catch** вначале выполняются все инструкции между операторами **try** и **catch**. Если в блоке **try** вдруг возникает исключение, то обычный порядок выполнения останавливается и переходит к инструкции **catch**. Поэтому когда выполнение программы дойдет до строки numbers[4]=45;, программа остановится и перейдет к блоку **catch**.

```
public class Main {
    public static void main(String[] args) {

        try{

            int[] numbers = new int[3];
            numbers[4]=45;
            System.out.println(numbers[4]);
        }

       catch(Exception ex){

            ex.printStackTrace();
        }

        System.out.println("Программа завершена");
        }
    }
java.lang.ArrayIndexOutOfBoundsException: 4
      at Main.main(Main.java:6)
Программа завершена

Process finished with exit code 0
```

Выражение catch имеет следующий синтаксис:

catch (тип_исключения имя_переменной).

В данном случае объявляется переменная **ex**, которая имеет тип **Exception**. Но если возникшее исключение не является исключением типа, указанного в

инструкции **catch**, то оно не обрабатывается, а программа просто зависает или выбрасывает сообщение об ошибке.

Но так как тип **Exception** является базовым классом для всех исключений, то выражение

catch(Exception ex)

будет обрабатывать практически все исключения.

Обработка же исключения в данном случае сводится к выводу на консоль стека трассировки ошибки с помощью метода

printStackTrace(),

определенного в классе **Exception**.

После завершения выполнения блока **catch** программа продолжает свою работу, выполняя все остальные инструкции после блока **catch**.

Конструкция **try..catch** также может иметь блок **finally**. Однако этот блок необязательный, и его можно при обработке исключений опускать. Блок **finally** выполняется в любом случае, возникло ли исключение в блоке **try** или нет.

Пример 2.
Блок finally выполняется в любом случае, то есть всегда

```
public class Main {
    public static void main(String[] args) {

        try{

            int[] numbers = new int[3];
            numbers[4]=45;
            System.out.println(numbers[4]);

        }
        catch(Exception ex){

            ex.printStackTrace();
        }

        finally{
            System.out.println("Блок finally");
        }

        System.out.println("Программа завершена");

        }
    }
```

```
java.lang.ArrayIndexOutOfBoundsException: 4
       at Main.main(Main.java:6)
Блок finally
Программа завершена

Process finished with exit code 0
```

Оператор throw

Чтобы сообщить о выполнении исключительных ситуаций в программе, можно использовать оператор **throw**.

То есть с помощью этого оператора мы сами можем создать исключение и вызвать его в процессе выполнения. Например, в нашей программе происходит ввод числа, и мы хотим, чтобы, если число больше 30, то возникало исключение

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args)  {
            try{
                Scanner in = new Scanner(System.in);


                System.out.println("Введите целое число ");
                int x = in.nextInt();


                if(x>=30){
                    throw new Exception("Число x должно быть меньше 30");
                }
            }
            catch(Exception ex){

                System.out.println(ex.getMessage());
            }
            System.out.println("Программа завершена");
        }
    }


31
Число x должно быть меньше 30
Программа завершена
```

Здесь для создания объекта исключения используется конструктор класса **Exception**, в который передается сообщение об исключении. И если число x окажется больше 29, то будет выброшено исключение и управление перейдет к блоку **catch**. В блоке **catch** мы можем получить сообщение об исключении с помощью метода **getMessage().**

Оператор throws

Иногда метод, в котором может генерироваться исключение, сам не обрабатывает это исключение. В этом случае в объявлении метода используется оператор **throws**, который надо обработать при вызове этого метода.

Например, у нас имеется метод вычисления факториала, и нам надо обработать ситуацию, если в метод передается число меньше 1:

```
import java.util.Scanner;

public class Main {

public static int getFactorial(int num) throws Exception{

    if(num<1) throw new Exception("The number is less than 1");
    int result=1;
    for(int i=1; i<=num;i++){

        result*=i;
    }
    return result;
}
```

С помощью оператора **throw** по условию выбрасывается исключение. В то же время метод сам это исключение не обрабатывает с помощью **try..catch**, поэтому в определении метода используется выражение **throws Exception**.

Теперь при вызове этого метода нам обязательно надо обработать выбрасываемое исключение:

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

    try {
        int result = getFactorial(-6);

        System.out.println(result);
```

```
    } catch (Exception ex) {

        System.out.println(ex.getMessage());
    }
}
}
```

```
The number is less than 1
```

Без обработки исключение у нас возникнет ошибка компиляции, и мы не сможем скомпилировать программу.

В качестве альтернативы мы могли бы и не использовать оператор **throws**, а обработать исключение прямо в методе:

```
import java.util.Scanner;


public class Main {

    public static int getFactorial(int num){

        int result=1;
        try{
            if(num<1) throw new Exception("The number is less than 1");

            for(int i=1; i<=num;i++){

                result*=i;
            }
        }
        catch(Exception ex){

            System.out.println(ex.getMessage());
            result=num;
        }
        return result;
    }

    public static void main(String[] args) {



            int result = getFactorial(-6);

    }
}
```

Обработка нескольких исключений

В Java имеется множество различных типов исключений, и мы можем разграничить их обработку, включив дополнительные блоки **`catch:`**

```
public class Main {

    public static void main(String[] args) {

        int[] numbers = new int[3];

        try{
            //numbers[6]=45;
            numbers[6]=Integer.parseInt("gfd");
        }
        catch(ArrayIndexOutOfBoundsException ex){

            System.out.println("Выход за пределы массива");
        }
        catch(NumberFormatException ex){

            System.out.println("Ошибка преобразования из строки в
число");
        }
        }
    }

Ошибка преобразования из строки в число
```

Если у нас возникает исключение определенного типа, то оно переходит к соответствующему блоку **`catch`**.

Пять слов для работы с исключениями

В Java есть пять ключевых слов для работы с исключениями:

1. **`try`** - данное ключевое слово используется для отметки начала блока кода, который потенциально может привести к ошибке.
2. **`catch`** - ключевое слово для отметки начала блока кода, предназначенного для перехвата и обработки исключений.
3. **`finally`** - ключевое слово для отметки начала блока кода, которое является дополнительным. Этот блок помещается после последнего

блока 'catch'. Управление обычно передаётся в блок 'finally' в любом случае.

4. **`throw`** - служит для генерации исключений.
5. **`throws`** - ключевое слово, которое прописывается в сигнатуре метода, и обозначающее что метод потенциально может выбросить исключение с указанным типом.

Классы исключений

Базовым классом для всех исключений является класс **`Throwable`**. От него уже наследуются два класса: **`Error`** и **`Exception`**. Все остальные классы являются производными от этих двух классов.

Класс **`Error`** описывает внутренние ошибки в исполняющей среде Java. Программист имеет очень ограниченные возможности для обработки подобных ошибок.

Собственно исключения наследуются от класса **`Exception`**. Среди этих исключений следует выделить класс **`RuntimeException`**. **`RuntimeException`** является базовым классом для так называемой группы непроверяемых исключений (**`unchecked exceptions`**) - компилятор не проверяет факт обработки таких исключений и их можно не указывать вместе с оператором **`throws`** в объявлении метода. Такие исключения являются следствием ошибок разработчика, например, неверное преобразование типов или выход за пределы массива.

Некоторые из классов непроверяемых исключений:

- **`ArithmeticException`**: исключение, возникающее при делении на ноль
- **`IndexOutOfBoundException`**: индекс вне границ массива

- **`IllegalArgumentException`**: использование неверного аргумента при вызове метода
- **`NullPointerException`**: использование пустой ссылки
- **`NumberFormatException`**: ошибка преобразования строки в число

Некоторые Методы класса Exception

- Метод **`getMessage()`** возвращает сообщение об исключении
- Метод **`getStackTrace()`** возвращает массив, содержащий трассировку стека исключения
- Метод **`printStackTrace()`** отображает трассировку стека

```
try{
    int x = 4/0;
}
catch(Exception ex){

    ex.printStackTrace();
}
java.lang.ArithmeticException: / by zero
     at Main.main
Process finished with exit code 0
```

Пример. Обработка исключений

```
public class Main {

    void print(String s) {

        if (s == null) {
            throw new NullPointerException("Exception: s is null!");
        }

        System.out.println("Inside method print: " + s);
    }

    public static void main(String[] args) {

       Main print = new Main();

       String[] list = new String[3];
```

```
        list[0]="first step";
        list[1]=null;
        list[2]="second step";

        for (String s:list) {

            try {
                print.print(s);
            }
            catch (NullPointerException e) {

                System.out.println(e.getMessage());

                System.out.println("Exception was processed.");
                System.out.println("Program continues");

            }

            finally {

                System.out.println("Inside block finally");
            }

            System.out.println("Go program....");

            System.out.println("-----------------");
        }

    }
}
```

```
Inside method print: first step
Inside block finally
Go program....
-----------------
Exception: s is null!
Exception was processed. Program continues
Inside block finally
Go program....
-----------------
Inside method print: second step
Inside block finally
Go program....
-----------------

Process finished with exit code 0
```

Вложенные блоки try - catch

```
import java.util.Random;

class Main {
    public static void main(String args[]) {
        Random r = new Random();
        int a, b;
        int c[] = {-1, 1};
        for (int i = 1; i < 10; i++) {
            try {
                a = r.nextInt(3); // значение 0,1 или 2
                b = 100 / a; // возможно деление на ноль
                System.out.println("b=" + b);
                // Вложенные блоки
                try {
                    if (a == 1) a = a / (a - 1); // деление на ноль
                    else c[a] = 200; // выход за границы массива
                } catch (ArrayIndexOutOfBoundsException e) {
                    System.out.println("Выход за границы массива: " + e);
                }
            } catch (ArithmeticException e) {
                System.out.println("Деление на ноль: " + e);
            }
            System.out.println("*******************************");
        }
    }
}
```

- Блоки try-catch можно вкладывать один в другой
- Число вложений ограничено реализацией JVM

Задания

1. В таблице ниже приведена программа, при запуске которой возникает исключение деления на ноль. Перепишите код таким образом, чтобы это исключение обрабатывалось в программе.

```
public class Main {

    public static void main(String[] args) {

        int zero=0;
        int number = 1;
        int result = number / zero;
    }
}
Exception in thread "main" java.lang.ArithmeticException: / by zero
      at Main.main(Main.java:7)

Process finished with exit code 1
```

2. В таблице ниже приведена программа, при запуске которой возникает исключение **нулевого указателя**. Перепишите код таким образом, чтобы это исключение обрабатывалось в программе.

```
public class Main {

    public static void main(String[] args) {

        String a = null; //null value
        System.out.println(a.charAt(0));

    }
}
Exception in thread "main" java.lang.NullPointerException
      at Main.main(Main.java:6)

Process finished with exit code 1
```

3. В таблице ниже приведена программа, при запуске которой возникает исключение **выхода за границы массива**. Перепишите код таким образом, чтобы это исключение обрабатывалось в программе.

```
public class Main {

    public static void main(String[] args) {


            String a = "This is like chipping "; // length is 22
```

```
            char c = a.charAt(24); // accessing 25th element
            System.out.println(c);
        }
}
```

```
Exception in thread "main"
java.lang.StringIndexOutOfBoundsException: String index out of
range: 24
      at java.lang.String.charAt(String.java:658)
      at Main.main(Main.java:8)

Process finished with exit code 1
```

4. В таблице ниже приведена программа, при запуске которой возникает исключение **NumberFormatException**. Перепишите код таким образом, чтобы это исключение обрабатывалось в программе.

```
public class Main {
    public static void main(String[] args)  {

        int num = Integer.parseInt("akki");
        System.out.println(num);
    }
}
```

```
Exception in thread "main" java.lang.NumberFormatException: For input
string: "akki"
      at
java.lang.NumberFormatException.forInputString(NumberFormatException.jav
a:65)
      at java.lang.Integer.parseInt(Integer.java:580)
      at java.lang.Integer.parseInt(Integer.java:615)
      at Main.main(Main.java:7)

Process finished with exit code 1
```

5. В таблице ниже приведена программа, при запуске которой возникает исключение **ArrayIndexOutOfBoundsException.** Перепишите код таким образом, чтобы это исключение обрабатывалось в программе.

```
public class Main {

    public static void main(String[] args)  {

        int a[] = new int[5];
        a[6] = 9;

        System.out.println(2+2);
    }

}
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 6
       at Main.main(Main.java:6)

Process finished with exit code 1
```

6. Пользователь вводит целые числа. Выводите на экран предупреждение, что ноль вводить нельзя. Если пользователь ввел ноль, то программа считает это исключением, обрабатывает его и заканчивает свою работу.

7. Пользователь вводит число и строку. Нужно обработать исключения – строка на равна null, число не равно 0. Найти сумму строки и числа.

```
int x = 5;
String text = "X=" + x;
System.out.println(text);
```

8. Пользователь вводит строку. Если строка – палиндром, то нужно сгенерировать исключение и обработать его.

```
String s1 = "XAAX";
if (s1.equals((new StringBuilder(s1)).reverse().toString()))
{
    System.out.println(s1);
};
```

9. Программа генерирует случайные числа в диапазоне от 1 до 10. Создать и обрабатывать исключения, если числа простые.

10. Пользователь вводит строки. Выводите на экран предупреждение, что нельзя вводить строки, содержащие строку “qw”. Если пользователь ввел “qw”, то программа считает это исключением, обрабатывает его и заканчивает свою работу. Если пользователь ничего не ввел, или ввел пробел, то программа напоминает пользователю, что нужно обязательно ввести строку и предлагает это сделать.

```
String s = "www.mysite.com";

boolean isContain1 = s.contains("mysite");
System.out.println(isContain1);// нашел - выведет true

boolean isContain2 = s.contains("mysite.ru");
System.out.println(isContain2);// не нашел - выведет false
```