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. 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.

```
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
```

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**.

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);
            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**().

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.

```
import java.io.*;

public class Main {

    public static void main(String[] args) throws IOException {


         FileOutputStream fileOutputStream = new FileOutputStream("f:\\test_01.txt");

        String greetings = "Привет!!!";

        fileOutputStream.write(greetings.getBytes());

        fileOutputStream.close();
    }
}
```

```
Exception in thread "main" java.io.FileNotFoundException: f:\test_01.txt
(Системе не удается найти указанный путь)
        at java.io.FileOutputStream.open0(Native Method)
        at java.io.FileOutputStream.open(FileOutputStream.java:270)
        at java.io.FileOutputStream.<init>(FileOutputStream.java:213)
        at java.io.FileOutputStream.<init>(FileOutputStream.java:101)
        at Main.main(Main.java:8)

Process finished with exit code 1
```

Пример обработки 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. 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
```

Использование исключений в Java позволяет повысить отказоустойчивость программы за счет использования «запасных» путей, отделить логику основного кода от кода обработки исключительных ситуаций за счет использования блоков **catch**, а также дает нам возможность переложить обработку исключений на пользователя нашего кода с помощью **throws**.

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.java: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
```

Введение в классы

Класс определяет новый тип данных.

Java является объектно-ориентированным языком, поэтому такие понятия как "класс" и "объект" играют в нем ключевую роль. Любую программу на Java можно представить как набор взаимодействующих между собой объектов.

Шаблоном или описанием объекта является класс, а объект представляет экземпляр этого класса.

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

        Person tom = new Person(); // создание объекта
        tom.displayInfo();

        // изменяем имя и возраст
        tom.name = "Tom";
        tom.age = 34;
        tom.displayInfo();
    }
}
class Person{

    String name;    // имя
    int age;        // возраст
    void displayInfo(){
        System.out.printf("Name: %s \tAge: %d\n", name, age);
    }
}
```

```
Name: null      Age: 0
Name: Tom       Age: 34
```

Для создания объекта Person используется выражение new Person(). Оператор new выделяет память для объекта Person. И затем вызывается конструктор по умолчанию, который не принимает никаких параметров. В итоге после выполнения данного выражения в памяти будет выделен участок, где будут храниться все данные объекта Person. А переменная tom получит ссылку на созданный объект.

Если конструктор не инициализирует значения переменных объекта, то они получают значения по умолчанию. Для переменных числовых типов это число 0, а для типа string и классов - это значение null (то есть фактически отсутствие значения).

Можно определить в классе несколько конструкторов

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

        Person bob = new Person();
// вызов первого конструктора без параметров
        bob.displayInfo();

        Person tom = new Person("Tom");
// вызов второго конструктора с одним параметром
        tom.displayInfo();

        Person sam = new Person("Sam", 25);
// вызов третьего конструктора с двумя параметрами
        sam.displayInfo();
    }
}
class Person{

    String name;    // имя
    int age;        // возраст
    Person()
    {
        name = "Undefined";
        age = 18;
    }
    Person(String n)
    {
        name = n;
        age = 18;
    }
    Person(String n, int a)
    {
        name = n;
        age = a;
    }
    void displayInfo(){
        System.out.printf("Name: %s \tAge: %d\n", name, age);
    }
}
```

```
Name: Undefined        Age: 18
Name: Tom      Age: 18
Name: Sam      Age: 25

Process finished with exit code 0
```

Пример. Класс Box

```
class Box {
    double width;
    double height;
    double depth;

    // This is the constructor for Box
    Box(double width, double height, double depth) {
        this.width = width;
```

```
        this.height = height;
        this.depth = depth;
    }

    double volume() {
        return width * height * depth;
    }
}
class Main {
    public static void main(String args[]) {

        Box mybox = new Box(10,20,15);
        System.out.println("volume = "+mybox.volume());
    }}
```

Задания

1.Создать класс прямоугольного треугольника. Поля – основание и высота. Методы – площадь, гипотенуза, периметр.

2.Создать класс окружность. Поля – координаты центра и радиус. Предусмотреть конструктор для вырожденной окружности, когда радиус равен нулю. Методы – площадь, длина окружности.

3.Создать класс студент. Поля – фамилия, имя, курс, группа, средний балл. Методы – перевести на следующий курс. Изменить средний балл. Стипендия в зависимости от среднего балла.

4.Создать класс книга. Поля – название, автор, год, цена, количество. Методы – изменить количество. Изменить цену в зависимости от количества. Стоимость всего количества книг.