

## ЗАНЯТИЕ 1.10

### ПРАКТИЧЕСКОЕ ЗАНЯТИЕ

Тема: Работа с файлами. Обработка исключений.

#### Упражнение №1

В данном упражнении предстоит изучить особенности обработки исключений. Создайте новое java-приложение. В методе main с использованием класса Scanner реализуйте ввод с клавиатуры имени человека и его возраст.

```
public static void main(String[] args){
    System.out.println( x:"Hello!");
    Scanner scan = new Scanner( in:System.in);
    System.out.println( x:"Please, enter your name:");
    String name = scan.next();
    System.out.println( x:"Please, enter your age:");
    int age = scan.nextInt();
    System.out.println( x:"Thank you!");
}
```

Рисунок 1

Убедитесь, что программа работает.

```
Building Practical5 1.0
-----[ jar ]-----

--- exec-maven-plugin:3.0.0:exec (default-cli) @ Practical5 ---
Hello!
Please, enter your name:
Ivan
Please, enter your age:
18
Thank you!
```

Рисунок 2

Как отреагирует программа, если при вводе возраста пользователь случайно (или специально) нажал на клавишу с каким-нибудь символом? Программа завершит свою работу с неконтролируемым исключением **InputMismatchException**.

```

3 | -- exec-maven-plugin:3.0.0:exec (default-cli) @ Practical5 ---
Hello!
Please, enter your name:
Ivan
Please, enter your age:
18r
3 | Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Scanner.java:864)
    at java.util.Scanner.next(Scanner.java:1485)
    at java.util.Scanner.nextInt(Scanner.java:2117)
    at java.util.Scanner.nextInt(Scanner.java:2076)
    at com.mirea.kt.practical5.Practical5.main(Practical5.java:39)
Command execution failed.
3 | org.apache.commons.exec.ExecuteException: Process exited with an error: 1 (Exit value: 1)
    at org.apache.commons.exec.DefaultExecutor.executeInternal (DefaultExecutor.java:404)
    at org.apache.commons.exec.DefaultExecutor.execute (DefaultExecutor.java:166)
    at org.codehaus.mojo.exec.ExecMojo.executeCommandLine (ExecMojo.java:982)
    at org.codehaus.mojo.exec.ExecMojo.executeCommandLine (ExecMojo.java:929)
    at org.codehaus.mojo.exec.ExecMojo.execute (ExecMojo.java:457)

```

Рисунок 3

Реализуем обработку исключения с использованием конструкции **try-catch**.

```

public static void main(String[] args) {
    System.out.println(x: "Hello!");
    Scanner scan = new Scanner(in: System.in);
    System.out.println(x: "Please, enter your name:");
    String name = scan.next();
    int age = 0;
    while (age == 0) {
        try {
            System.out.println(x: "Please, enter your age:");
            age = scan.nextInt();
        } catch (InputMismatchException ime) {
            scan.next();
            System.out.println(x: "Age error! Try again!");
        }
    }
    System.out.println(x: "Thank you!");
}

```

Рисунок 4

## Упражнение №2

В данном упражнении предстоит изучить особенности обработки исключений. Создайте новое java-приложение, добавьте в него класс Person, содержащий в качестве параметров фамилию, имя и возраст. В классе необходимо реализовать конструкторы (один из которых без параметров), сеттеры и геттеры. Другие методы добавлять не обязательно.

```
public class Person {
    private String firstName;
    private String lastName;
    private int age;

    public Person(String firstName, String lastName, int age) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
    }

    public Person() {
    }

    public void doSleep(int hours){
        System.out.println("Person sleep " + hours + " hours");
    }

    public void talk(){
        System.out.println("Hello! I am an ordinary person!");
    }

    public void doEat(String[] foods){
        //здесь какой-то код
    }

    public int getAge() {
        return age;
    }
}
```

Рисунок 5

В главном классе программы, в методе main создадим экземпляр класса Person с использованием конструктора без параметров. С помощью сеттеров инициализируем переменные *first\_name* и *age*.

```
public static void main(String[] args) {
    Person prs = new Person();
    prs.setFirstName( firstName: "Ivan");
    prs.setAge( age: 17);
}
```

Рисунок 6

Переменная *last\_name* осталась не инициализированной. Так как эта переменная ссылочная, то по умолчанию она будет равна `null`. Как в таком случае будут выполняться методы у такой ссылочной переменной? Выведем в консоль длину имени и фамилии созданного экземпляра класса `Person`. Для этого необходимо использовать геттеры и метод класса `String` – `length()`.

```
30 public static void main(String[] args){
31     Person prs = new Person();
32     prs.setFirstName( firstName:"Ivan");
33     prs.setAge( age:17);
34     System.out.println("Length of first name: " + prs.getFirstName().length());
35     System.out.println("Length of last name: " + prs.getLastName().length());
36 }
```

Рисунок 7

```
Person prs = new Person();
prs.setFirstName("Ivan");
prs.setAge(17);
System.out.println("Length of first name: " + prs.getFirstName().length());
System.out.println("Length of last name: " + prs.getLastName().length());
```

Возникнет неконтролируемое исключение – **`NullPointerException`**. IDE пишет в каком файле и в какой строке произошел сбой (в данном случае на 35 строке кода).

```
Length of first name: 4
Exception in thread "main" java.lang.NullPointerException
    at com.mirea.kt.practical5.Practical5.main(Practical5.java:35)
Command execution failed.
org.apache.commons.exec.ExecuteException: Process exited with an error: 1 (Exit value: 1)
    at org.apache.commons.exec.DefaultExecutor.executeInternal (DefaultExecutor.java:404)
    at org.apache.commons.exec.DefaultExecutor.execute (DefaultExecutor.java:166)
    at org.codehaus.mojo.exec.ExecMojo.executeCommandLine (ExecMojo.java:982)
    at org.codehaus.mojo.exec.ExecMojo.executeCommandLine (ExecMojo.java:929)
    at org.codehaus.mojo.exec.ExecMojo.execute (ExecMojo.java:457)
    at org.apache.maven.plugin.DefaultBuildPluginManager.executeMojo (DefaultBuildPluginManager.java:124)
    at org.apache.maven.lifecycle.internal.MojoExecutor.doExecute2 (MojoExecutor.java:370)
    at org.apache.maven.lifecycle.internal.MojoExecutor.doExecute (MojoExecutor.java:351)
    at org.apache.maven.lifecycle.internal.MojoExecutor.execute (MojoExecutor.java:215)
```

Рисунок 8

Выполняя метод `prs.getLastName()`, получаем текущее значение переменной *last\_name*, которая в свою очередь равна **`null`**. При попытке вызвать любой метод у неинициализированной ссылочной переменной будет возникать **`NullPointerException`**. Реализуем обработку исключения с использованием конструкции **`try-catch`**.

```

public static void main(String[] args){
    Person prs = new Person();
    prs.setFirstName( firstName:"Ivan");
    prs.setAge( age:17);
    try{
        System.out.println("Length of first name: " + prs.getFirstName().length());
        System.out.println("Length of last name: " + prs.getLastName().length());
    }catch(NullPointerException npe){
        System.out.println("Кажется, мы имеем дело с "
            + "нулевой неинициализированной ссылкой");
    }
}

```

Рисунок 9

Теперь программа не «падает» при возникновении ошибки, а выводит соответствующее информационное сообщение в консоль. Изменим конструкцию **try-catch** на **try-catch-finally**.

```

public static void main(String[] args){
    Person prs = new Person();
    prs.setFirstName( firstName:"Ivan");
    prs.setAge( age:17);
    try{
        System.out.println("Length of first name: " + prs.getFirstName().length());
        System.out.println("Length of last name: " + prs.getLastName().length());
    }catch(NullPointerException npe){
        System.out.println("Кажется, мы имеем дело с "
            + "нулевой неинициализированной ссылкой");
    }finally{
        System.out.println( "Вычисление завершено");
    }
}

```

Рисунок 10

Код, указанный в блоке **finally** выполнится независимо от того возникло исключение или нет.

## Упражнение №3

Далее рассмотрим механизм создания собственного исключения и проброса его по стеку вызовов. Создайте новый проект, добавьте в него класс **Person** (рисунок 1) или скопируйте класс из предыдущего упражнения.

В классе **Person** добавьте новую переменную – *budget* (размер бюджета человека). Переменная будет иметь тип **int**. Реализуйте для нее сеттер и геттер.

```

public class Person {
    private String firstName;
    private String lastName;
    private int age;
    private int budget;

    public int getBudget() {
        return budget;
    }

    public void setBudget(int budget) {
        this.budget = budget;
    }
}

```

Рисунок 11

Создайте новый класс – **Market** (магазин). Переменные класса: *title* (название), *catalog* (набор товаров в продаже), *openingHour* и *closingHour* (часы открытия и закрытия магазина). Реализуйте конструктор (с 1 параметром), сеттеры и геттеры.

```

public class Market {

    private String title; // название магазина
    private HashMap<String, Integer> catalog; // каталог товаров магазина с ценами
    private int openingHour; // час открытия магазина
    private int closingHour; // час закрытия магазина

    public Market(String title) {
        this.title = title;
    }

    public HashMap<String, Integer> getCatalog() {
        return catalog;
    }

    public void setCatalog(HashMap<String, Integer> catalog) {
        this.catalog = catalog;
    }

    public String getTitle() {
        return title;
    }
}

```

Рисунок 12

Добавьте в созданный класс метод **sale** (продавать), который в качестве аргументов будет принимать: ссылку на объект покупателя (в данном случае Person), коллекцию (набор товаров из корзины покупателя). Метод в

результате работы будет возвращать набор купленных продуктов. В теле метода пока реализуйте возврат пустой коллекции (это временно, позже мы заполним ее продуктами).

```
private String title; // название магазина
private HashMap<String, Integer> catalog; // каталог товаров магазина с ценами
private int openingHour; // час открытия магазина
private int closingHour; // час закрытия магазина

public Market(String title) {
    this.title = title;
}

public HashMap<String, Integer> getCatalog() {
    return catalog;
}

public void setCatalog(HashMap<String, Integer> catalog) {
    this.catalog = catalog;
}

public ArrayList<String> sale(Person p, ArrayList<String> bucket) {
    ArrayList<String> purchases = new ArrayList<>();
    return purchases;
}

public String getTitle() {
    return title;
}
```

Рисунок 13

Создайте новый класс – `MarketException`. Класс будет наследоваться от класса **Exception** и реализовывать два его конструктора.

```
public class MarketException extends Exception{

    public MarketException() {
    }

    public MarketException(String string) {
        super(string);
    }
}
```

Рисунок 14

В главном классе в методе `main` создадим объект класса `Person`.

```
public static void main(String[] args){
    // task 2
    Person person = new Person( firstName:"Ivan",  lastName:"Ivanov",  age:17);
    person.setBudget( budget:1000);
}
```

Рисунок 15

Создайте экземпляр класса `Market`, а также добавьте для него каталог некоторых товаров с ценами. Названия товаров в магазине уникальные, поэтому используется `HashMap`.

```
public static void main(String[] args){
    // task 2
    Person person = new Person( firstName:"Ivan",  lastName:"Ivanov",  age:17);
    person.setBudget( budget:1000);
    Market myMarket = new Market( title:"4etverochka"); // это просто название магазина
    myMarket.setOpeningHour( openingHour:11);
    myMarket.setClosingHour( closingHour:23);
    HashMap<String, Integer> ctlg = new HashMap<>();
    //добавим несколько товаров для примера
    ctlg.put( key:"Хлеб бородинский",  value:50);
    ctlg.put( key:"Колбаса московская",  value:200);
    ctlg.put( key:"Конфеты студенческие",  value:20);
    ctlg.put( key:"Сок яблочный Добрый",  value:50);
    ctlg.put( key:"Шампанское Российское",  value:250);

    myMarket.setCatalog( catalog:ctlg);
}
```

Рисунок 16

Предположим, человек хочет приобрести в магазине конфеты и шампанское. Создайте новую коллекцию (аналог корзины или тележки) и добавьте туда товары для покупки.

```
myMarket.setCatalog( catalog:ctlg);
ArrayList<String> bucket = new ArrayList<>();
bucket.add( e:"Конфеты студенческие");
bucket.add( e:"Шампанское Российское");
```

Рисунок 17

Но метод `sale` пока ничего не умеет. Добавьте код в метод `sale` класса `Market` так, чтобы при попытке купить алкоголь несовершеннолетним метод «выбрасывал» исключение. Также необходимо проверять достаточно ли у человека средств на покупки (если не хватает, то также вызывать исключение).



```

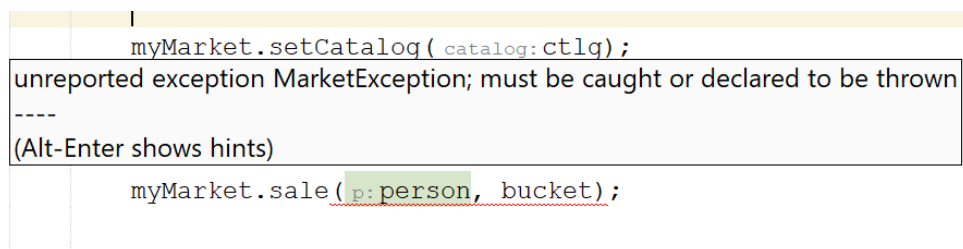
public ArrayList<String> sale(Person p, ArrayList<String> bucket) throws MarketException{
    ArrayList<String> purchases = new ArrayList<>();
    int wallet = p.getBudget();
    for(String product : bucket){
        wallet = wallet - this.catalog.get(key:product);
        if(wallet < 0){
            throw new MarketException(string:"Недостаточно средств для покупки");
        }
        if(p.getAge() < 18){
            if(product.contains(s:"Пиво") || product.contains(s:"Вино") || product.contains(s:"Шампанское")){
                throw new MarketException(string:"Недостаточно прав для покупки");
            }
        }
        purchases.add(e:product); // сложили покупки в пакет
        p.setBudget(budget:wallet); // Обновили бюджет - он стал меньше
    }
    return purchases;
}

```

Рисунок 18

Необходимо обратить внимание на ключевые слова **throw** и **throws**. С помощью первого генерируется исключение **MarketException**, с помощью второго – «пробрасывается» далее по стеку вызовов. То есть исключение **MarketException** необходимо обработать в том месте кода, где будет вызван метод **sale**!

При попытке вызвать метод **sale** из главного класса в методе **main** возникает ошибка:



```

myMarket.setCatalog(catalog:ctlq);
unreported exception MarketException; must be caught or declared to be thrown
----
(Alt-Enter shows hints)
myMarket.sale(p:person, bucket);

```

Рисунок 19

IDE подсказывает, что необходимо реализовать обработку исключения **MarketException**. То есть необходимо либо добавить в код конструкцию **try-catch**, либо «пробросить» исключение дальше по стеку. Попробуйте оба варианта. На рисунке ниже представлен первый способ. При возникновении ошибки реализуем вывод информации в консоль.

```

ArrayList<String> bucket = new ArrayList<>();
bucket.add(e:"Конфеты студенческие");
bucket.add(e:"Шампанское Российское");
try {
    myMarket.sale(p:person, bucket);
} catch (MarketException ex) {
    System.out.println(x:"Ошибка при покупке:");
    System.out.println(x:ex.getMessage());
}

```

Рисунок 20

Попробуйте самостоятельно реализовать проверку текущего времени в методе **sale**, чтобы «выбросить» исключение в случае, если покупатель пытается совершить покупку в кассе в нерабочее время.

Подсказка: для получения информации о текущем времени необходимо использовать методы класса `LocalTime`.

```
LocalTime local = LocalTime.now();  
String nowTime = local.toString();
```

Рисунок 21

## Упражнение №4

В данном упражнении предстоит изучить работу с файлами – создание, открытие файла, модификация.

Создайте новый проект, добавьте в него новый класс – **FileUtils**. В данном классе будут располагаться статические методы для различных операций с файлами.

Для тренировки работы с файлами необходимо создать рабочий каталог, чтобы не «мусорить» в файловой системе. Сделаем это программно!

В классе `FileUtils` создайте статический метод **checkWorkDirectory**, который будет проверять существует ли такой каталог. Если его нет, то метод должен его создать. Метод будет принимать на вход в качестве аргументов: абсолютный путь, где располагается каталог для работы, а также имя рабочего каталога. Возвращаемое значение необходимо реализовать логического типа (`boolean`). Для проверки существования и создания каталога необходимо воспользоваться методами класса **File** – `exists` и `mkdir`.

```
public class FileUtils {  
  
    public static boolean checkWorkDirectory(String path, String dirName){  
        File workDir = new File(path + "/" + dirName);  
        if(workDir.exists()){ // проверка существования  
            return true;  
        }else{  
            // создание каталога методом mkdir  
            // mkdir - возвращает true, если каталог был успешно создан  
            return workDir.mkdir();  
        }  
    }  
}
```

Рисунок 22

Вызовите метод `checkWorkDirectory` из главного класса. Используйте в качестве имени рабочей директории фамилию студента.

```
public static void main(String[] args){
    // task 3
    boolean isExistWorkDir = FileUtils.checkWorkDirectory(path:"C:\\", dirName:"Ivanov");
    System.out.println("Exist work directory: " + isExistWorkDir);
}
```

Рисунок 23

Второй метод, который необходимо реализовать в классе `FileUtils` – **`writeStringToFile`**. Данный метод в качестве аргументов будет принимать на путь к файлу и строку, которую необходимо записать в файл.

Запись строковых данных в файл удобно реализовать с помощью наследника класса `Writer` – **`FileWriter`**.

```
/**
 *
 * @author User
 */
public class FileUtils {

    public static void writeStringToFile(String path, String str){
        FileWriter fw = new FileWriter(string:path);
        fw.write(str);
        fw.close();
    }

}
```

Рисунок 24

IDE подсвечивает ошибки – работа с потоками ввода/вывода могут вызывать различные контролируемые исключения. Поэтому необходимо реализовать конструкцию **`try-catch`** для обработки исключений или пробросить исключения дальше по стеку с помощью ключевого слова **`throws`**. Реализуем второй вариант:

```
public static void writeStringToFile(String path, String str) throws IOException{
    FileWriter fw = new FileWriter(string:path);
    fw.write(str);
    fw.close();
}
```

Рисунок 25

При вызове метода `writeStringToFile` из главного класса необходимо реализовать конструкцию `try-catch`.

```
public static void main(String[] args){
    // task 3
    boolean isExistWorkDir = FileUtils.checkWorkDirectory(path:"C:\\", dirName:"Ivanov");
    System.out.println("Exist work directory: " + isExistWorkDir);

    if(isExistWorkDir){
        String data = "Example string";
        try {
            FileUtils.writeStringToFile(path:"C:\\Ivanov\\test.txt", str:data);
        } catch (IOException ex) {
            System.out.println("При записи в файл возникла ошибка: " + ex.getMessage());
        }
    }
}
```

Рисунок 26

Убедитесь, что в рабочем каталоге появился текстовый файл с содержимым «Example string».

Добавьте в класс `FileUtils` статический метод для чтения файла – **`readBytesFromFile`**. Данный метод будет читать содержимое файла из потока ввода в байтовый массив. В результате работы метод должен вернуть полученный массив.

```
public static byte[] readBytesFromFile(String path) throws FileNotFoundException, IOException{
    File file = new File(string:path);
    if(file.exists()){ // проверка что файл существует
        FileInputStream fis = new FileInputStream(file);
        //available возвращает количество доступных для чтения байтов из входного потока
        byte[] data = new byte[fis.available()];
        fis.read(b:data); // чтение из потока в массив data
        fis.close(); //закрытие потока
        return data;
    }else{
        // если файл не существует, то метод возвращает пустой массив
        return new byte[]{};
    }
}
```

Рисунок 27

Сделайте вызов метода `readBytesFromFile` из главного класса и выведете в консоль размер полученного массива. При вызове необходимо использовать конструкцию `try-catch`.

```

public static void main(String[] args){
    // task 3
    boolean isExistWorkDir = FileUtils.checkWorkDirectory(path:"C:\\", dirName:"Ivanov");
    System.out.println("Exist work directory: " + isExistWorkDir);

    if(isExistWorkDir){
        String data = "Example string";
        try {
            FileUtils.writeStringToFile(path:"C:\\Ivanov\\test.txt", str:data);
        } catch (IOException ex) {
            System.out.println("При записи в файл возникла ошибка: " + ex.getMessage());
        }
    }

    try {
        byte[] array = FileUtils.readBytesFromFile(path:"C:\\Ivanov\\test.txt");
        System.out.println("Array len: " + array.length);
    } catch (IOException ex) {
        System.out.println("При чтении из файла возникла ошибка: " + ex.getMessage());
    }
}

```

Рисунок 28

В массиве **array** сейчас находятся байты строки из файла test.txt. Это можно увидеть, выполним отладку:

```

3 public static void main(String[] args){
4     // task 3
5     boolean isExistWorkDir = FileUtils.checkWorkDirectory(path:"C:\\", dirName:"Ivanov");
6     System.out.println("Exist work directory: " + isExistWorkDir);
7
8     if(isExistWorkDir){
9         String data = "Example string";
10        try {
11            FileUtils.writeStringToFile(path:"C:\\Ivanov\\test.txt", str:data);
12        } catch (IOException ex) {
13            System.out.println("При записи в файл возникла ошибка: " + ex.getMessage());
14        }
15    }
16
17    try {
18        byte[] array = FileUtils.readBytesFromFile(path:"C:\\Ivanov\\test.txt");
19        System.out.println("Array len: " + array.length);
20    } catch (IOException ex) {
21        System.out.println("При чтении из файла возникла ошибка: " + ex.getMessage());
22    }
23 }

```

Name	Type	Value
isExistWorkDir	boolean	true
array	byte[]	#86(length=14)
array[0]	byte	0x45
array[1]	byte	0x78
array[2]	byte	0x61
array[3]	byte	0x64
array[4]	byte	0x70

Рисунок 29

По ASCII-таблице можно проверить значения байтов и соответствующие им символы.

Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	\$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-~	63	3F	?	95	5F	_	127	7F	DEL

Рисунок 30

Выполним простейшее шифрование данных массива. Зашифруем строку! Необходимо в цикле «пробежать» по массиву и выполнить какую-нибудь операцию (например XOR) с каждым элементом.

```
try {
    byte[] array = FileUtils.readBytesFromFile(path: "C:\\Ivanov\\test.txt");
    System.out.println("Array len: " + array.length);
    for(int i = 0; i < array.length; i++){
        // с каждым байтом массива проводим операцию XOR с 0x1A (можете выбрать любой байт)
        array[i] = (byte) (array[i] ^ 0x1A);
    }
} catch (IOException ex) {
    System.out.println("При чтении из файла возникла ошибка: " + ex.getMessage());
}
```

Рисунок 31

Теперь массив `array` хранит в себе зашифрованную строку. Самостоятельно запишите новый массив в новый файл с помощью класса **FileOutputStream** (создайте для этого новый метод в классе `FileUtils`).

## Индивидуальное задание

**Задание:** разработать консольную Java-программу для работы с файлами и их содержимым.

### Требования:

- 1) На сдачу практического задания отводится **7 дней**.
- 2) Вариант определяется согласно порядковому номеру студента в журнале.
- 3) В случае дистанционного выполнения практического задания код программы необходимо выложить на Github и предоставить ссылку на него.
- 4) Итоговая программа должна компилироваться без ошибок, полностью выполнять требуемый функционал и при старте выводить в консоль номер варианта и ФИО студента.
- 5) Названия переменных и методов должны отражать суть и нести смысловую нагрузку.
- 6) Необходимо придерживаться стилистике по написанию Java-кода.
- 7) Необходимо предусмотреть защиту от ввода «аномальных» (ошибочных) значений и **реализовать обработку исключений!**

### Дополнительная информация:

- 1) Перед выполнением задания рекомендуется ознакомиться с Лекцией №1.9.
- 2) Для конвертации введенных пользователем значений Hex-string (для 3 варианта) в байтовый массив можно использовать данный метод:

```
public static byte[] hexStringToByteArray(String s) throws IllegalArgumentException {
    int len = s.length();
    if (len % 2 == 1) {
        throw new IllegalArgumentException("Error hexstring");
    }
    byte[] data = new byte[len / 2];
    for (int i = 0; i < len; i += 2) {
        data[i / 2] = (byte) ((Character.digit(s.charAt(i), 16) << 4)
            + Character.digit(s.charAt(i+1), 16));
    }
    return data;
}
```

**Вариант 1.** Разработать программу для «склейки» файлов. Обязательные требования к программе:

- Количество фалов для «склейки» и пути к ним задает пользователь после старта программы.

- Результирующий файл должен быть сохранен рядом с первым указанным файлом для «склейки».

Пример содержимого 1-го входного файла (6 байт):

0x1A, 0x2B, 0x3C, 0x3C, 0x44, 0x44

Пример содержимого 2-го входного файла (14 байт):

0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF, 0xAA, 0xBB

Содержимое выходного файла (20 байт):

0x1A, 0x2B, 0x3C, 0x3C, 0x44, 0x44, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF, 0xAA, 0xBB

**Вариант 2.** Разработать программу для модификации текстового файла (расширение txt), содержащего строки. Обязательные требования к программе:

- Путь к исходному файлу задает пользователь после старта программы.
- Необходимо открыть файл, прочитать содержащиеся в нем строки, перевернуть каждую строку и записать их все в новый файл.
- Новый файл должен быть сохранен рядом с исходным.

Пример содержимого входного файла:

*4ewZ55*

*456346*

*VBhdb*

*QWERTY*

*12345678*

Пример содержимого выходного файла:

*55Zwe4*

*643654*

*bdhBV*

*YTREWQ*

*87654321*



**Вариант 3.** Разработать программу для модификации входного файла. Модификация заключается в последовательном наложении гаммы (операция XOR) на байты входного файла. Обязательные требования к программе:

- Путь к исходному файлу, а также байты гаммы задает пользователь после старта программы.
- Необходимо открыть файл, прочитать содержащиеся в нем байты, осуществить побайтовую операцию XOR с гаммой и записать выходной результат в новый файл.
- Новый файл должен быть сохранен рядом с исходным.

Пример содержимого входного файла (6 байт):

0x1A, 0x2B, 0x3C, 0x3C, 0x44, 0x44

Гамма: ABCD (2 байта)

Содержимое выходного файла (6 байт):

0xB1, 0xE6, 0x97, 0xF1, 0xEF, 0x89

**Вариант 4.** Разработать программу для поиска файлов заданного расширения. Обязательные требования к программе:

- Путь к каталогу для поиска файлов задает пользователь после старта программы.
- Поиск необходимо осуществить также во всех вложенных директориях.
- Для каждого файла необходимо определять размер в байтах и записывать его рядом с названием.
- Сортировка выходного результата должна начинаться с названий каталогов.

Пример выходного результата:

*Finding docx in d:\Work\MIREA\*

*d:\Work\MIREA\Tasks\1.docx 12874*

*d:\Work\MIREA\Tasks\2.docx 27943*

**Вариант 5.** Разработать программу для вывода списка всех файлов и каталогов для заданного пути. Обязательные требования к программе:

- Путь к каталогу для сканирования задает пользователь после старта программы.
- Программа также должна выводить названия всех **вложенных** файлов и каталогов.

- Для каждого файла необходимо определять размер в байтах и записывать его рядом с названием.
- Сортировка выходного результата должна начинаться с названий каталогов.

Пример выходного результата:

*Scanning d:\Work\MIREA\*

*d:\Work\MIREA\Java folder*

*d:\Work\MIREA\Android folder*

*d:\Work\MIREA\test.txt 5653*

*d:\Work\MIREA\Java\task.txt 800*

*d:\Work\MIREA\Android\run.bat 123*

*d:\Work\MIREA\Android\android-emulator.exe 57235*