IT-Academy

Типы данных и операторы

Автор: Юлий Слабко

Содержание

1. Ключевые и зарезервированные слова
2. Идентификаторы
3. Литералы
4. Разделители
5. Базовые типы данных
6. Операторы
7. Операторы управления
8. Документирование кода

Структура кода

1. **Пробелы**
2. **Комментарии**
3. **Лексемы**

Пробелы

1. **Пробел код ASCII 32**
2. **CR ASCII 10**
3. **LF (NL) ASCII 13**
4. **TAB ASCII 9**

Комментарии

1. **Однострочный**

```
// java 4 U
```

2. **Многострочный**

```
i++/* increment comment*/;
```

3. **Документирования**

```
/**
* @Author Yuli
*/
```

Комментарии. Документирование кода

В языке Java используются блочные и однострочные комментарии /* */ и //, аналогичные комментариям, применяемым в C++.

Введен также новый вид комментария /** */, который может содержать дескрипторы вида:

- **@author** – задает сведения об авторе;
- **@version** – задает номер версии класса;
- **@exception** – задает имя класса исключения;
- **@param** – описывает параметры, передаваемые методу;
- **@return** – описывает тип, возвращаемый методом;
- **@deprecated** – указывает, что метод устаревший и у него есть более совершенный аналог;
- **@since** – с какой версии метод (член класса) присутствует;
- **@throws** – описывает исключение, генерируемое методом;
- **@see** – что следует посмотреть дополнительно.

Пример

```java
public class User {
    /**
     * personal user's code
     */
    private int numericCode;
    /**
     * user's password
     */
    private String password;
    /**
     * see also chapter #3 "Classes"
     */
    public User() {
     password = "default";
    }
    /**
     * @return the numericCode
     * return the numericCode
     */
    public int getNumericCode() {
     return numericCode;
    }
```

```java
/**
 * @param numericCode the numericCode
to set
 * parameter numericCode to set
 */
public void setNumericCode(int
numericCode) {
     this.numericCode = numericCode;
}
/**
 * @return the password
 * return the password
 */
public String getPassword() {
     return password;
}
/**
 * @param password the password to set
 * parameter password to set
 */
public void setPassword(String
password) {
     this.password = password;
}
}
```

Пример

chapt02

class User

java.lang.Object
|__chapt02.User

public class **User**
extends java.lang.Object

Filed Summary

private int	numerucCode personal user's code
private java.lang.String	password user's password

Constructor Summary

User() see also chapter #3 "Classes"	

Method Summary

int	getNumericCode ()
java.lang.String	getPassword ()

Лексемы

- ключевые слова (key words);
- *идентификаторы* (identifiers);
- *литералы* (literals);
- разделители (separators);
- операторы (operators).

Ключевые и зарезервированные слова языка Java

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Идентификаторы Java

Идентификаторы – это имена, которые даются различным элементам языка для упрощения доступа к ним.

В именах переменных используются символы:

- **A-Z**
- **a-z**
- **0-9**
- **'$' и '_'**

Запрещено применение арифметических и логических операторов, а также символ '#'.

Правильно:

- **my$money**
- **_flag**
- **new_string**

Неправильно:

- field#
- open^flag
- 1searchIndex

Литералы

- **Целочисленные литералы:**
  - Тип **int**:
    - ✔ 0x51 - шестнадцатеричное значение
    - ✔ 015 - восьмеричное значение
    - ✔ 0b10011100 – двоичное значение;
    - ✔ 20 - десятичное значение
  - Тип **long**:
    - ✔ 0xffffL
- **Литералы с плавающей точкой:**
  - Тип **double**:
    - ✔ 1.918
    - ✔ .5 или 5.
    - ✔ 0.112E-05 или 0.52E25- экспоненциальная форма
  - Тип **float**:
    - ✔ 3.45F
  - **NAN**
  - **POSITIVE_INFINITY, NEGATIVE_INFINITY**

Литералы

- **Строки:**
  - ✔ "sample string "
- **Логические литералы:**
  - ✔ true и false
- **null - литерал**

Литералы

- **Символьные литералы:**
  - ✔ 'a', '\n', '\141', '\u005a'

\b \u0008 backspace BS – забой
\t \u0009 horizontal tab HT – табуляция
\n \u000a linefeed LF – конец строки
\f \u000c form feed FF – конец страницы
\r \u000d carriage return CR – возврат каретки
\" \u0022 double quote " – двойная кавычка
\' \u0027 single quote ' – одинарная кавычка
\\ \u005c backslash \ – обратная косая черта

\шестнадцатеричный код от \u0000 до \u00ff символа
в шестнадцатеричном формате.

\000 - \377 – восмеричное представление символа

Разделители

```
()[]{};.,
```

Базовые типы данных

В языке Java определено восемь базовых типов данных. Для каждого базового типа данных отводится конкретный размер памяти.

Тип	Размер (бит)	По умолчанию	Значения (диапазон или максимум)
boolean	8 (32)	false	true , false
byte	8	0	−128..127
char	16	'\u0000'	0..65535
short	16	0	−32768..32767
int	32	0	−2147483648..2147483647
long	64	0	9223372036854775807L
float	32	0.0	1.40239846e-45f 3.40282347E+38
double	64	0.0	4.94065645841246544e-324 1.79769313486231570E+308

Формат представления вещественного числа

Виды операторов

Арифметические операторы

+	Сложение	/	Деление
+=	Сложение (с присваиванием)	/=	Деление (с присваиванием)
–	Бинарное вычитание и унарное изменение знака	%	Деление по модулю
–=	Вычитание (с присваиванием)	%=	Деление по модулю (с присваиванием)
*	Умножение	++	Инкремент
*=	Умножение (с присваиванием)	--	Декремент

Битовые операторы

|	Или	>>	Сдвиг вправо
|=	Или (с присваиванием)	>>=	Сдвиг вправо (с присваиванием)
&	И	>>>	Сдвиг вправо с появлением нулей
&=	И (с присваиванием)	>>>=	Сдвиг вправо с появлением нулей и присваиванием
^	Исключающее или	<<	Сдвиг влево
^=	Исключающее или (с присваиванием)	<<=	Сдвиг влево с присваиванием
~	Унарное отрицание		

Пример

```
public class TypeByte {
    private static int j;
    public static void main(String[] args){
        byte b = 1,
         b1 = 1 + 2;
        final byte b2 = 1 + 2;
       //b = b1 + 1; //ошибка приведения типов
        b = (byte)(b1 + 1);//0
        show(b);
       //b = -b; //ошибка приведения типов
        b = (byte)-b;//1
        show(b);
        //b = +b1; //ошибка приведения типов
        b = (byte)+b1; //2
        show(b);
       b1 *= 2; //3
        show(b1);
        b1++; //4
        show(b1);
```

```
= 3;
 i; //ошибка приведения типов
yte) i; //5
);
1++; //работает!!!  //6
);
 f = 1.1f;
 f; //работает!!!  //7
);
```

В результате будет выведено:

0 res=4
1 res=-4
2 res=3
3 res=6
4 res=7
5 res=3
6 res=6
7 res=5

Пример. Оператор отношения и остатка

Для целых чисел:

7/5 возвращает 1
7/(-5) возвращает -1
(-7)/5 возвращает -1
(-7)/(-5) возвращает 1

7%5 возвращает 2
7%(-5) возвращает 2
(-7)%5 возвращает -2
(-7)%(-5) возвращает -2

Для вещественных чисел:

7.0/5.0 возвращает 1.4
7.0/(-5.0) возвращает -1.4
(-7.0)/5.0 возвращает -1.4
(-7.0)/(-5.0) возвращает 1.4

7.0/5.3 возвращает 1.3207547169811322

7.0%5.0 возвращает 2.0
7.0%(-5.0) возвращает 2.0
(-7.0)%5.0 возвращает -2.0
(-7.0)%(-5.0) возвращает -2.0

7.0%5.3 возвращает 1.7000000000000002

Пример

```
public class Operators {
  public static void main(String[] args) {
    System.out.println("5%1=" + 5%1 + "  5%2=" + 5%2);
    int b1 = 0xe;//14 или 1110
    int b2 = 0x9;//9  или 1001
    int i = 0;
    System.out.println(b1 + "|" + b2 + " = " + (b1|b2));
    System.out.println(b1 + "&" + b2 + " = " + (b1&b2));
    System.out.println(b1 + "^" + b2 + " = " + (b1^b2));
    System.out.println(     "~" + b2 + " = " + ~b2);
    System.out.println(b1 + ">>" + ++i + " = " + (b1>>i));
    System.out.println(b1 + "<<" + i + " = " + (b1<<i++));
    System.out.println(b1 + ">>>" + i +" = " + (b1>>>i));
  }
}
```

Результат выполнения:

5%1=0
5%2=1
14|9 = 15
14&9 = 8
14^9 = 7
~9 = -10
14>>1 = 7
14<<1 = 28
14>>>2 = 3

Пример сдвига

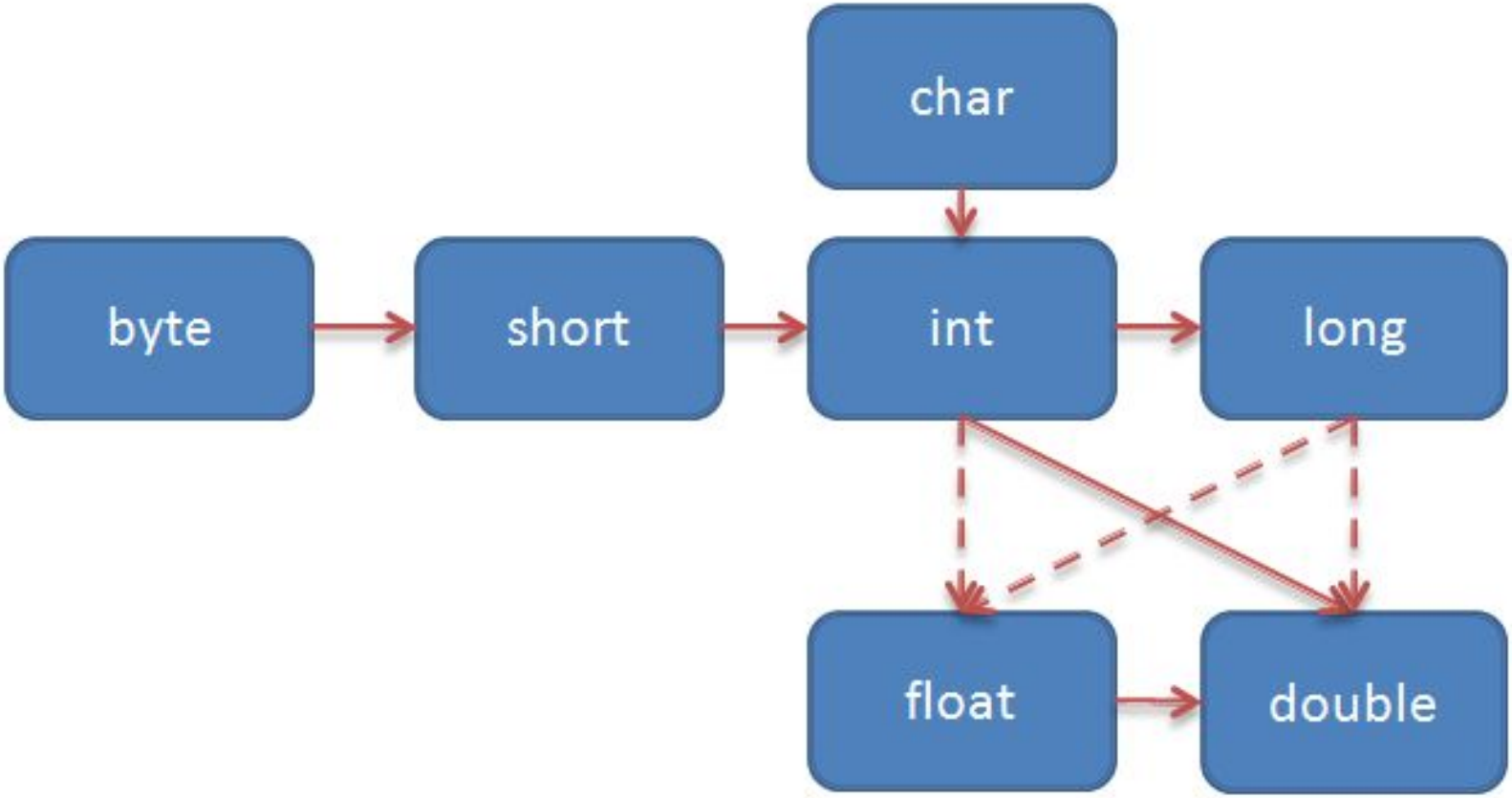
```
// Сдвиг влево для положительного числа 20
20 << 00 = 00000000000000000000000000010100 = 20
20 << 01 = 00000000000000000000000000101000 = 40
20 << 02 = 00000000000000000000000001010000 = 80
20 << 03 = 00000000000000000000000010100000 = 160
20 << 04 = 00000000000000000000000101000000 = 320 ...
20 << 25 = 00101000000000000000000000000000 = 671088640
20 << 26 = 01010000000000000000000000000000 = 1342177280
20 << 27 = 10100000000000000000000000000000 = -1610612736
20 << 28 = 01000000000000000000000000000000 = 1073741824
20 << 29 = 10000000000000000000000000000000 = -2147483648
20 << 30 = 00000000000000000000000000000000 = 0
20 << 31 = 00000000000000000000000000000000 = 0
// Сдвиг влево для отрицательного числа -21
-21 << 00 = 11111111111111111111111111101011 = -21
-21 << 01 = 11111111111111111111111111010110 = -42
-21 << 02 = 11111111111111111111111110101100 = -84
-21 << 03 = 11111111111111111111111101011000 = -168
-21 << 04 = 11111111111111111111111010110000 = -336
-21 << 05 = 11111111111111111111110101100000 = -672 ...
-21 << 25 = 11010110000000000000000000000000 = -704643072
-21 << 26 = 10101100000000000000000000000000 = -1409286144
-21 << 27 = 01011000000000000000000000000000 = 1476395008
-21 << 28 = 10110000000000000000000000000000 = -1342177280
-21 << 29 = 01100000000000000000000000000000 = 1610612736
-21 << 30 = 11000000000000000000000000000000 = -1073741824
-21 << 31 = 10000000000000000000000000000000 = -2147483648
```

Преобразования типов

В арифметических выражениях автоматически выполняются расширяющие преобразования:

- byte
- short
- int
- long
- float
- double

Преобразования типов



Виды операторов

Операторы отношения

Применяются для сравнения символов, целых и вещественных чисел, а также для сравнения ссылок при работе с объектами.

<	Меньше	>	Больше
<=	Меньше либо равно	>=	Больше либо равно
==	Равно	!=	Не равно

Логические операторы

||	Или	&&	И
!	Унарное отрицание		

К операторам относится также оператор определения принадлежности типу **instanceof,** оператор **[ ]** и тернарный оператор **?:** (if-then-else).
Логические операции выполняются над значениями типа **boolean** (**true** или **false**).
Оператор **instanceof** возвращает значение **true**, если объект является экземпляром данного класса.

Операторы управления

- **Оператор if:**

  Позволяет условное выполнение оператора или условный выбор двух операторов, выполняя один или другой, но не оба сразу.

```
if (boolexp) { /*операторы*/}
else { /*операторы*/ }//может отсутствовать
```

- **Оператор switch:**

```
switch(exp) {
    case exp1:/*операторы
}
```

Условное выражение

В качестве условия в операторе ***if*** может использоваться любое выражение, имеющее результат логического типа. Это может быть выражение с операторами сравнения или логическими операторами:

```
if (a < b) {
   x = b - a;
} else {
   x = a - b;
}
```

В данном примере вычисляется модуль разности двух чисел a и b.

Условное выражение

Пример:

```
boolean y = false;
int x = 96;
if (x % 16 == 0) {
   y = true;
} else {
   y = false;
}
System.out.println(y);
```

Какое значение получит переменная **y**?

Вложенные операторы if

В операторе ***if*** в качестве оператора, выполняемого по условию, может использоваться другой оператор ***if***. В таком случае говорят о вложенных условных операторах. Такая конструкция имеет вид:

```
if (условие1) {
if (условие2) {
      оператор1;
} else {
   оператор2;
}
} else {
оператор3;
}
```

Пример вложенных операторов if

```
if (a > b) {
  if (a > c) {
    System.out.println("максимальное число:" +
a);
  } else {
    System.out.println("максимальное число:" +
c);
  }
} else {
  if (b > c) {
    System.out.println("максимальное число:" +
b);
  } else {
    System.out.println("максимальное число:" +
c);
  }
```

Конструкция if else if

Встречаются задачи, в которых следует сделать выбор между более чем двумя возможными вариантами. Тогда применяется конструкция ***if else if***.

```
if (условие1) {
   оператор1;
} else if (условие2) {
   оператор2;
} else if (условие3) {
   оператор3;
} else {
   оператор4;
}
```

Пример использования if else if

```
if (n == 1) {
        System.out.println("Понедельник");
} else if (n == 2) {
        System.out.println("Вторник");
} else if (n == 3) {
        System.out.println("Среда");
} else if (n == 4) {
        System.out.println("Четверг");
} else if (n == 5) {
        System.out.println("Пятница");
} else if (n == 6) {
        System.out.println("Суббота");
} else if (n == 7) {
    System.out.println("Воскресенье");
} else {
    System.out.println("Дня с таким номером не
существует");
}
```

Пример №1

Чему равны x и y на выходе?

Первый:

```
int x = 5, y = 7;
if (x == 5) {
y = 9;
} else if (y == 9) {
x = 3;
}
```

x = 5, y = 9

Пример №2

Чему равны x и y на выходе?

Второй:

```
int x = 5, y = 7;
if (x == 5) {
   y = 9;
}
if (y == 9) {
   x = 3;
}
```

```
x = 3, y = 9
```

Операторы управления

- **Оператор switch**

  Оператор switch передает управление одному из нескольких операторов в зависимости от значения выражения.

```
switch(exp) {
   case exp1:/*операторы, если exp==exp1*/
      break;
   case exp2:/*операторы, если exp==exp2*/
      break;
   default: /* операторы Java */
}
```

Значения exp1,…, expN должны быть константами и могут иметь значения типа int, byte, short, char или enum.

С Java 7 тип String, Character, Byte, Short и Integer

Пример оператора switch

```
switch (dayNumber) {
  case 1:
       System.out.println("Понедельник");
       break;
  case 2:
       System.out.println("Вторник");
       break;
  case 3:
       System.out.println("Среда");
       break;
  ...
  default:
       System.out.println("Дня с таким номером не
существует");
}
```

Оператор switch

Надо учитывать, что если какой-либо из разделов ***switch*** не заканчивается оператором ***break***, начнут выполняться операторы из следующего раздела:

```
switch (dayNumber) {
  case 1:
       System.out.println("Понедельник");
  case 2:
       System.out.println("Вторник");
       break;
...
  default:
   System.out.println("Дня с таким номером не существует");
}
```

В данном фрагменте если *dayNumber* имеет значение 1, будут выведены на консоль

Понедельник
Дня с таким номером не существует.

Метки

Любой оператор, или блок, может иметь метку. Метку можно указывать в качестве параметра для операторов ***break*** и ***continue***. Область видимости метки ограничивается оператором, или блоком, к которому она относится.

```
public class Labels {
    static int x = 5;
    static {    }
    public Labels() {    }
    public static void main(String[] args) {
        Labels t = new Labels();
        int x = 1;
        Lbl1: {
            if(x == 0) break Lbl1;
        }
        Lbl2:{
            if(x > 0) break Lbl2; // break Lbl1 - ERROR
        }
    }
}
```

Вопросы

Функция

Пример исходного кода.

```java
public UserDataConfigVO parseExcel(Long organizationId, Long agentId, List<UserDataFieldVO> fields, Map<String, String> mappingData,
        InputStream inputStream, BusinessRuleImportStatisticsVO statistics) {

    try (Workbook wb = WorkbookFactory.create(inputStream)) {

        Sheet sheet = wb.getSheetAt(0);

        Row headerRow = sheet.getRow(HEADER_ROW_INDEX);
        if (headerRow == null) {
            throw new TPMException("Unable to parse header. Header row (index = 0) is not find in the file");
        }

        /* Process and build header */

        Map<UserDataFieldVO, Integer> fieldToCellMapping = new HashMap<>();
        int statusCellIndex = -1;
        int lastCellIndex = 0;

        for (int cellNum = START_CELL_INDEX; cellNum <= headerRow.getLastCellNum(); cellNum++) {
            Cell cell = headerRow.getCell(cellNum);
            if (isEmptyCell(cell)) {
                break;
            }
            String headerValue = getCellValue(cell);
            UserDataFieldVO mappedField = findMappedField(fields, mappingData, headerValue);

            if (mappedField != null && !fieldToCellMapping.containsKey(mappedField)) {
                fieldToCellMapping.put(mappedField, cellNum);
                statistics.getColumnsImported().add(headerValue);

            } else if (statusCellIndex == -1 && isStatusColumn(headerValue)) {
                // note: process Status column and ignore (don't put into stats)
                statusCellIndex = cellNum;

            } else {
                statistics.getNotImportedColumns().add(headerValue);
            }
            lastCellIndex = cellNum;        }

        List<UserDataConfigItemVO> configItems = new ArrayList<>();

        if (fieldToCellMapping.isEmpty()) {
            // no need parse body/lines when there is not matched fields
            statistics.addMessage(LOCALIZATOR.getMessage("excel.parsers.BusinessRuleConfigParser.no.matched.fields.skip.lines"));
            return buildParseExcelResult(Collections.emptyList(), configItems);
        }

        AgentDateParams agentDateParams = getAgentDataParams(agentId, organizationId);
        List<DateFormat> dateFormats = getDateFormats(agentDateParams);

        Set<String> failedEidFieldNames = new HashSet<>();
        List<String> partnerEids = new ArrayList<>();

        FieldReplacementConfig replacementConfig = loadFieldReplacementConfig();

        boolean containsEidFields = findAnyOrNull(fieldToCellMapping.keySet(), eq(UserDataFieldVO::getType, PARTNER)) != null;
        if (containsEidFields) {
            partnerEids = organizationService.getOrganizationPartnerEids(organizationId);
        }

 return buildParseExcelResult(new ArrayList<>(fieldToCellMapping.keySet()), configItems);
```

Пример исходного кода с функцией.

```
public UserDataConfigVO parseExcel(Long organizationId, Long agentId, Map<String, String> fieldsMappingData,
                                   BusinessRuleImportStatisticsVO statistics, List<UserDataFieldVO> fields,
                                   InputStream inputStream, ImportProgressHandler progressHandler) {
    try (Workbook wb = WorkbookFactory.create(inputStream)) {
        SheetData data = new SheetData(wb.getSheetAt(0));

        /* Process and build header */
        processColumnHeaders(data, fieldsMappingData, statistics, fields);

        /* Process and build body */
        List<UserDataConfigItemVO> configItems = new ArrayList<>();

        if (data.getFieldToCellMapping().isEmpty()) {
            // no need parse body/lines when there is not matched fields
            statistics.addMessage(LOCALIZATOR.getMessage("excel.parsers.BusinessRuleConfigParser.no.matched.fields.skip.lines"));
            return buildParseExcelResult(Collections.emptyList(), configItems);
        }

        Set<String> failedEidFieldNames = new HashSet<>();
        List<String> partnerEids = new ArrayList<>();

        FieldReplacementConfig replacementConfig = loadFieldReplacementConfig();

        boolean containsEidFields = findAnyOrNull(data.getFieldToCellMapping().keySet(), eq(UserDataFieldVO::getType, PARTNER)) != null;
        if (containsEidFields) {
            partnerEids = organizationService.getOrganizationPartnerEids(organizationId);
        }

        parseRows(data, organizationId, agentId, statistics, progressHandler, partnerEids, configItems, failedEidFieldNames,
replacementConfig);

        if (!failedEidFieldNames.isEmpty()) {
            collectFailedEidMessages(failedEidFieldNames, statistics);
        }

        return buildParseExcelResult(new ArrayList<>(data.getFieldToCellMapping().keySet()), configItems);

    } catch (IOException e) {
        LOG.error("Can not create Workbook Sheet from stream", e);
        throw new TPMException("Can not create workbook Sheet for stream");

    } catch (InvalidFormatException e) {
        LOG.error("Can not create Workbook Sheet from file. File has invalid format", e);
        throw new TPMException("Can not create Workbook Sheet from file. File has invalid format (valid xls or xlsx are only supported)");
    }
}
```

Зачем нужно вынесение функции?

1. **Улучшает читабельность кода**
2. **Убирает дублирование кода**
3. **Изолирует части кода, уменьшая вероятность ошибок**
4. **Скрывает сложность кода (инкапсуляция)**

Алгоритм

1. **Создать новый метод с названием, отражающим суть его работы.**
2. **Скопировать фрагмент кода в новый метод. Удалить этот код из старого места и заменить вызовом метода.**

Методы с переменным числом параметров

IT-Academy

!!! Возможность передачи в метод нефиксированного числа параметров позволяет отказаться от предварительного создания массива объектов для его последующей передачи в метод.

Чтобы передать несколько массивов в метод по ссылке, следует использовать следующее объявление:

```
void methodName(Тип[]... args){}
```

Методы с переменным числом аргументов могут быть перегружены:

```
void methodName(Integer...args) {}
void methodName(int x1, int x2) {}
void methodName(String...args) {}
```

Определение количества параметров. Пример


```
public class DemoVarArgs {
public static int getArgCount(Integer... args) {
    if (args.length == 0)
    System.out.print("No arg=");
    for (int i : args)
    System.out.print("arg:" + i + "  ");
    return args.length;
}
public static void main(String args[]) {
    System.out.println("N=" + getArgCount(7, 71, 555));
    Integer[] i = { 1, 2, 3, 4, 5, 6, 7 };
    System.out.println("N=" + getArgCount(i));
    System.out.println(getArgCount());
}
}
```

Вопросы

IT-Academy

Спасибо за внимание