Java Pro 4.1

TEL-RAN
by Starta Institute

Потоки ввода-вывода

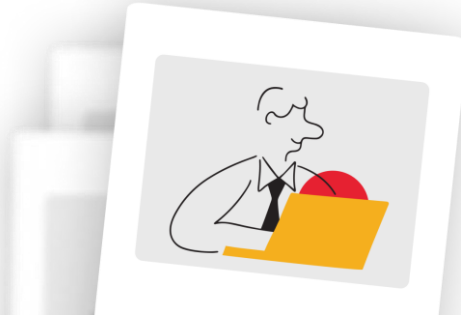
ПРЕПОДАВАТЕЛЬ

TEL-RAN
by Starta Institute

Юрий Костяной

Java/Kotlin backend-разработчик

- 3+ года опыта в коммерческой разработке
- 2+ года опыта в преподавании
- Проекты по интеграции сторонних платформ, CRM
- Проблемно-ориентированный подход в преподавании



ВАЖНО:

- Камера должна быть включена на протяжении всего занятия.
- Если у Вас возник вопрос в процессе занятия, пожалуйста, поднимите руку и дождитесь, пока преподаватель закончит мысль и спросит Вас, также можно задать вопрос в чате или когда преподаватель скажет, что начался блок вопросов.
- Организационные вопросы по обучению решаются с кураторами, а не на тематических занятиях.
- Вести себя уважительно и этично по отношению к остальным участникам занятия.
- Во время занятия будут интерактивные задания, будьте готовы включить камеру или демонстрацию экрана по просьбе преподавателя.

TEL-RAN
by Starta Institute

ПЛАН ЗАНЯТИЯ

1. Повторение
2. Основной блок
3. Вопросы по основному блоку
4. Домашняя работа

TEL-RAN
by Starta Institute

1 ПОВТОРЕНИЕ ИЗУЧЕННОГО

Повторение

TEL-RAN
by Starta Institute

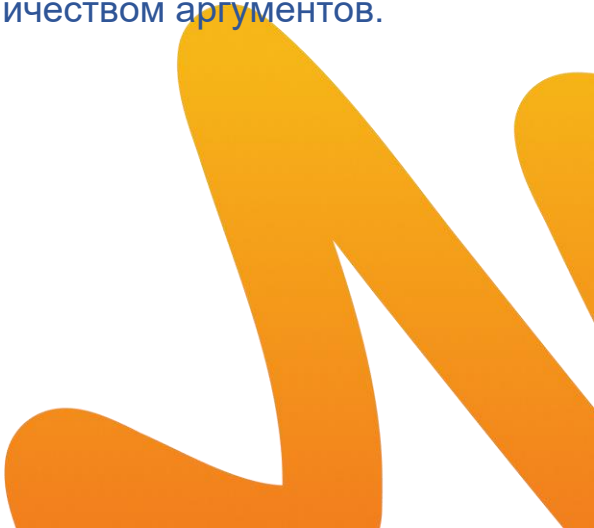
Для чего применяется паттерн *Строитель*?

Повторение

TEL-RAN
by Starta Institute

Для чего применяется паттерн *Строитель*?

Паттерн *Строитель* даёт возможность пошагового создания сложного объекта. Это упрощает код, позволяя избежать использования конструкторов с большим количеством аргументов.



Повторение

TEL-RAN
by Starta Institute

Исправьте ошибку в коде

```
import java.time.*;

public class Main1 {
    public static void main(String[] args) {
        LocalDate date = new LocalDate(2023, 11, 15);
        System.out.println("Дата: " + date);
    }
}
```

Повторение

TEL-RAN
by Starta Institute

Исправьте ошибку в коде

```
import java.time.*;

public class Main1 {
    public static void main(String[] args) {
        LocalDate date = new LocalDate(2023, 11, 15);
        System.out.println("Дата: " + date);
    }
}
```

У классов пакета java.time конструкторы объявлены как приватные. Для создания объектов используются статические методы *of*

Повторение

TEL-RAN
by Starta Institute

Что будет выведено в консоль?

```
public class Main {
    public static void main(String[] args) {
        LocalDate birthday = LocalDate.of(1976, 6, 4);
        getNextBirthday(birthday);
        System.out.println("Следующий день рождения будет " + birthday);
    }

    private static void getNextBirthday(LocalDate birthday) {
        birthday = birthday.plusYears(1);
    }
}
```

Повторение

TEL-RAN
by Starta Institute

Что будет выведено в консоль?

Вывод: *Следующий день рождения будет 1976-06-04*
Класс LocalDate является неизменяемым. При вызове метода plusYears появляется новый объект LocalDate, но ссылку на него мы не возвращаем из метода.

```
public class Main {
    public static void main(String[] args) {
        LocalDate birthday = LocalDate.of(1976, 6, 4);
        getNextBirthday(birthday);
        System.out.println("Следующий день рождения будет " + birthday);
    }

    private static void getNextBirthday(LocalDate birthday) {
        birthday = birthday.plusYears(1);
    }
}
```

Повторение

TEL-RAN
by Starta Institute

Что будет выведено в консоль?

```
import java.time.LocalDate;
import java.time.Period;

public class Main2 {
    public static void main(String[] args) {
        LocalDate startDate = LocalDate.of(2023, 11, 1);
        LocalDate endDate = LocalDate.of(2023, 10, 31);

        Period period = Period.between(startDate, endDate);
        System.out.println("Период: " + period.getYears() + " лет, " +
                period.getMonths() + " месяцев, " + period.getDays() + " дней");
    }
}
```

Повторение

TEL-RAN
by Starta Institute

Что будет выведено в консоль?

startDate позже, чем endDate. Период будет рассчитан, но он будет отрицательным:
Период: 0 лет, 0 месяцев, -1 дней

```
import java.time.LocalDate;
import java.time.Period;

public class Main2 {
    public static void main(String[] args) {
        LocalDate startDate = LocalDate.of(2023, 11, 1);
        LocalDate endDate = LocalDate.of(2023, 10, 31);

        Period period = Period.between(startDate, endDate);
        System.out.println("Период: " + period.getYears() + " лет, " +
                period.getMonths() + " месяцев, " + period.getDays() + " дней");
    }
}
```

Повторение

TEL-RAN
by Starta Institute

Найдите ошибку в коде.

```
import java.time.Duration;
import java.time.LocalTime;

public class Main3 {
    public static void main(String[] args) {
        LocalTime startTime = LocalTime.of(10, 30);
        LocalTime endTime = LocalTime.of(12, 45);

        Duration duration = Duration.between(startTime, endTime);
        System.out.printf("Продолжительность: %d часа", duration.toHours());
    }
}
```

Повторение

TEL-RAN
by Starta Institute

Найдите ошибку в коде.

Вывод: *Продолжительность: 2 часов*
Минуты были потеряны. Ниже приведён правильный вариант.

```
import java.time.Duration;
import java.time.LocalTime;

public class Main3 {
    public static void main(String[] args) {
        LocalTime startTime = LocalTime.of(10, 30);
        LocalTime endTime = LocalTime.of(12, 45);

        Duration duration = Duration.between(startTime, endTime);
        System.out.printf("Продолжительность: %d часа %d минут", duration.toHours(), duration.toMinutesPart());
    }
}
```

Повторение

TEL-RAN
by Starta Institute

Найдите ошибку в коде.

```
import java.time.LocalDateTime;
import java.time.ZoneId;
import java.time.ZonedDateTime;

public class Main4 {
    public static void main(String[] args) {
        ZonedDateTime zonedDateTime = ZonedDateTime.of(LocalDateTime.now());
        System.out.println("Дата и время в текущем часовом поясе: " + zonedDateTime);
    }
}
```

Повторение

TEL-RAN
by Starta Institute

Найдите ошибку в коде.

Для создания ZonedDateTime необходимо указать часовой пояс. Ниже приведён пример с указанием часового пояса, установленного в операционной системе, на которой запущена программа.

```
import java.time.LocalDateTime;
import java.time.ZoneId;
import java.time.ZonedDateTime;

public class Main4 {
    public static void main(String[] args) {
        ZonedDateTime zonedDateTime = ZonedDateTime.of(LocalDateTime.now(), ZoneId.systemDefault());
        System.out.println("Дата и время в текущем часовом поясе: " + zonedDateTime);
    }
}
```

TEL-RAN
by Starta Institute

2 ОСНОВНОЙ БЛОК

Введение

TEL-RAN
by Starta Institute

- Входит и выходит
- Файлики и папочки
- Перед отправкой

Проблема

TEL-RAN
by Starta Institute

Предположим, что мы хотим сохранить данные из оперативной памяти для постоянного хранения на жёстком диске. Нам потребуется такой класс или набор классов, который бы мог:

- создавать файл;
- записывать данные в файл;
- читать данные из файла;
- удалять файл;
- делать проверку, что файл с заданным именем уже существует, что он не защищён от записи, что он не занят другим процессом и т.д.

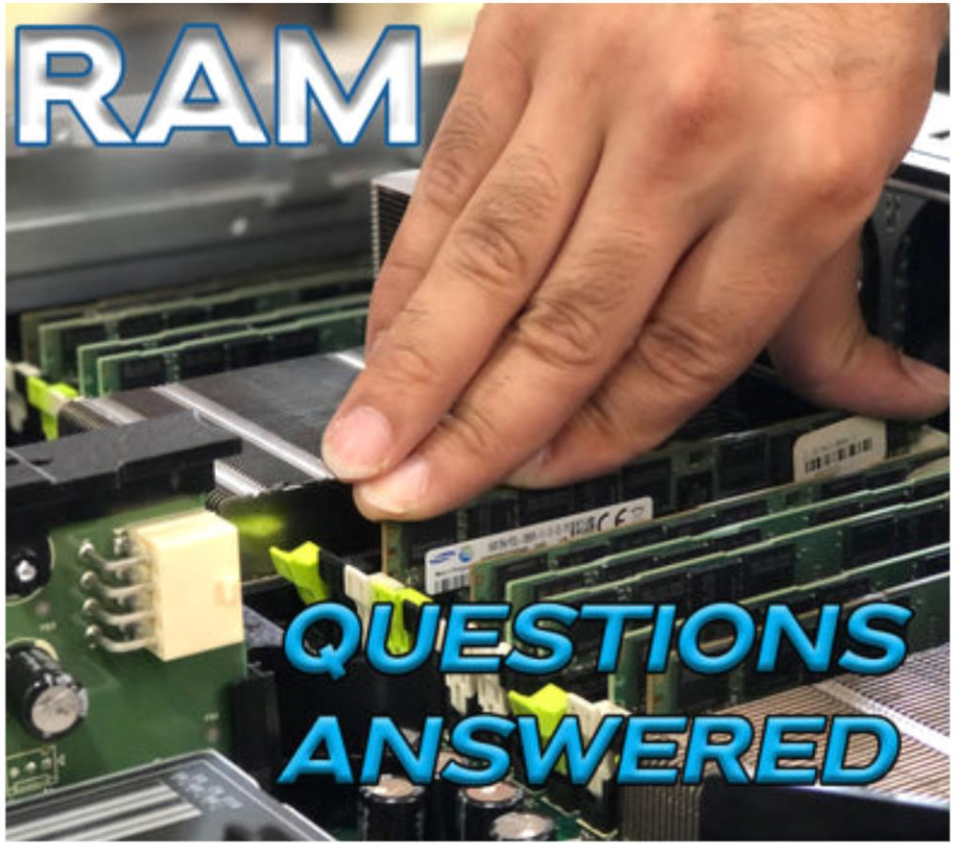
Большинство команд можно транслировать в ОС, т.к. ОС умеет всё вышеперечисленное.

Но как прочитать данные, если у нас 16 Гб ОЗУ, а файл занимает на диске, например, 20 Гб или 100 Гб?

Ответ

TEL-RAN
by Starta Institute

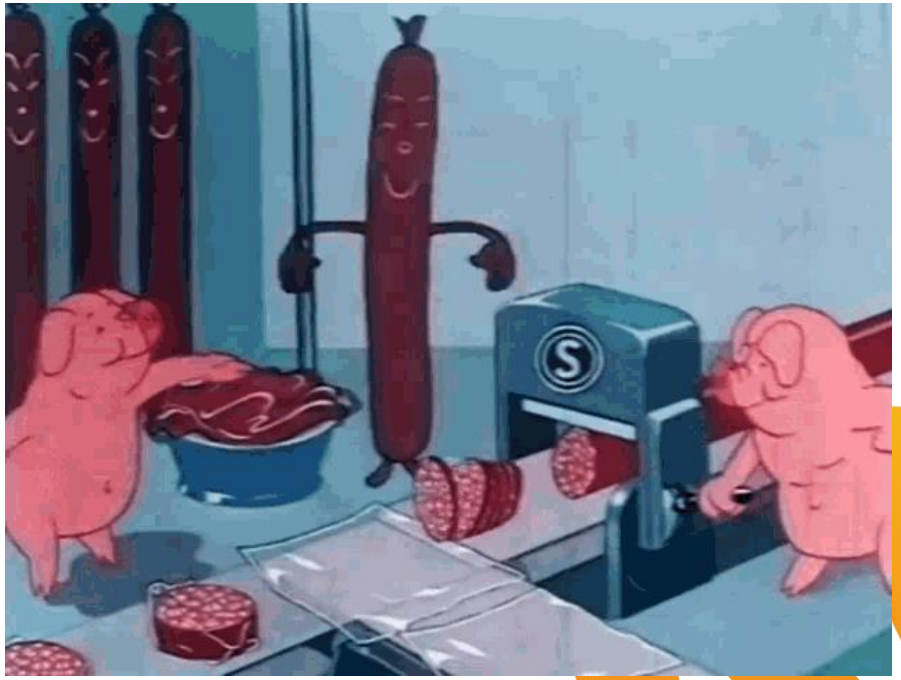
Нужно добавить больше оперативки! ☺



Проблема

TEL-RAN
by Starta Institute

Мы знаем, что все данные в компьютере в конечном итоге хранятся в виде примитивных типов, а те – в виде бит.



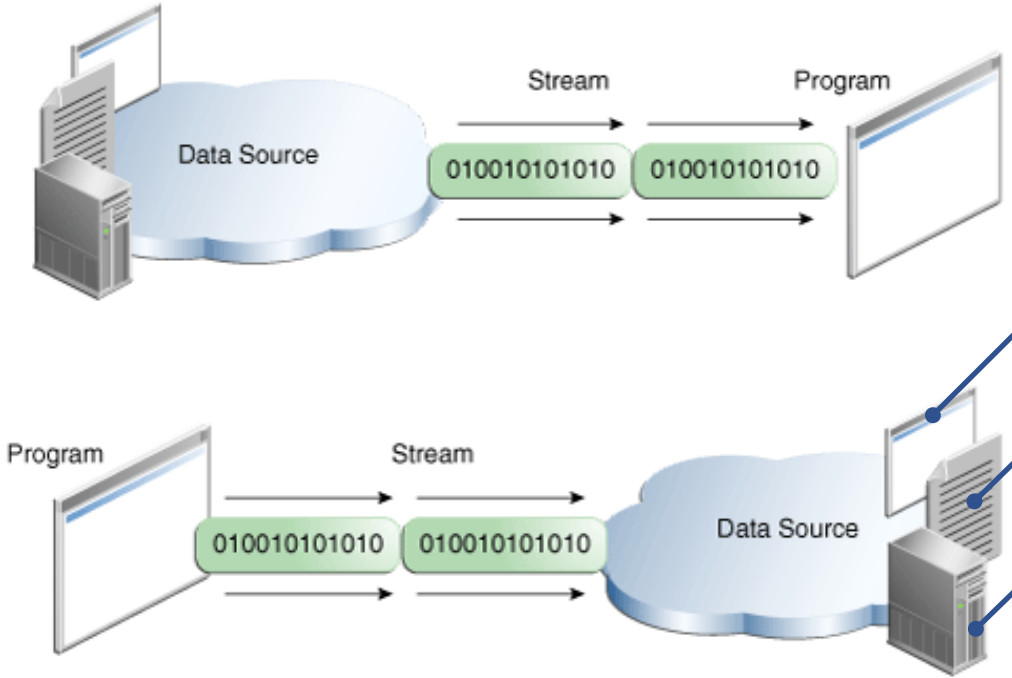
Минимальным примитивным типом является *byte* (8 бит). Ничего меньше в Java быть не может, значит, мы можем использовать универсальный способ записи любых данных, разбивая их на байты.

В таком случае нам нужна «машина для нарезки» данных на байты и передачи их в файл.

Входит и выходит

Потоки

Поток (stream) – это класс, позволяющих передавать данные порциями (потоки вывода) или получать данные порциями (потоки ввода).



Входит и выходит

Перегрузка слов «поток» и «stream»

Потоком в русском языке также называют *thread* (нить, подпроцесс) в многопоточном программировании. Термин stream также применяется в названии технологии *StreamAPI*, появившейся в Java 8 и позволяющей обрабатывать коллекции с помощью конвейерных (потоковых) методов.

```
public static void main(String[] args) {
    new Thread("Another thread").start(); // используем потоки (нити) в программе (многопоточная программа)

    try (InputStream input = new FileInputStream("notes.txt")) { // используем поток для чтения файла
        byte[] bytes = input.readAllBytes();
    } catch (IOException e) {
        System.err.println("Ошибка чтения файла: " + e.getLocalizedMessage());
    }

    List<String> startedWithJ = Stream.of("Bill", "Jim", "Jonh", "Anna", "Jane")
            .filter(s -> s.startsWith("J"))
            .toList(); // Отфильтровали коллекцию, используя Stream API (синтаксис языка)
}
```

Входит и выходит

Мы уже работали с потоками

Три стандартных потока для работы в консоли:

System.out – стандартный поток вывода

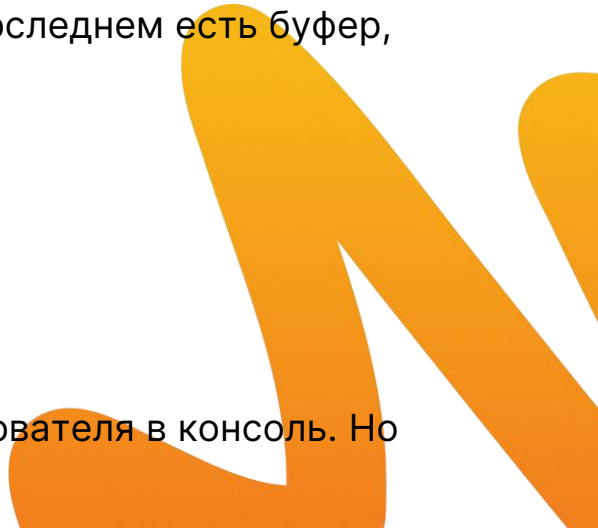
System.in – стандартный поток ввода

System.err – стандартный поток ошибок. Данный поток предназначен для вывода ошибок в консоль, обычно красным текстом, чтобы было более очевидно, что это текст ошибки. Этот поток выводит данные быстрее, чем *System.out,* потому что в последнем есть буфер, который «накапливает» данные до вывода в консоль.

Можно перенастроить потоки так, чтобы, например:

System.out выводил данные на консоль, а *System.err* – в файл.

Класс *java.util.Scanner* используется для получения ввода пользователя в консоль. Но может быть применён для чтения любого входного потока.



Входит и выходит

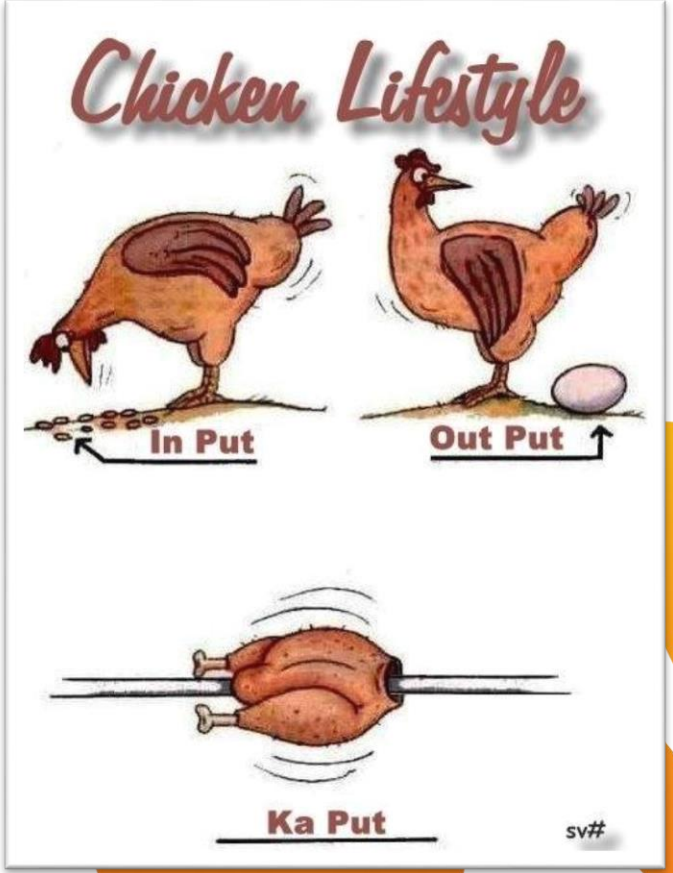
Java I/O

Java I/O (*Input and Output*) используется для обработки ввода и получения вывода.

Пакет *java.io* содержит все классы, необходимые для операций ввода и вывода:

File и *Path* – взаимозаменяемые классы для «привязки» программной сущности к реальному файлу или папке в ОС. Классы позволяют проверить наличие и доступность файла, находить соседние файлы и подниматься по иерархии каталогов.

InputStream и *OutputStream* – абстрактные классы, которые определяют основные методы потоков ввода-вывода.



Входит и выходит

Основные методы InputStream

read() – читает следующий байт данных;

readAll() – читает все доступные байты данных до конца потока;

readNBytes() – читает заданное количество байтов;

skip() – пропускает один байт;

skipNBytes() – пропускает заданное количество байтов;

mark() – используются для запоминания позиции в потоке;

reset() – используется для возвращения к ранее промаркерованной позиции в потоке. Это позволяет выполнить повторное прочтение байт.

mark() и *reset()* поддерживаются не во всех имплементациях *InputStream.*

Входит и выходит

Основные методы OutputStream

write() – записывает следующий байт данных или массив байт из потока;

flush() – если в потоке есть буфер, то все накопленные в нём данные будут принудительно записаны в целевого получателя (файл, БД и т.д.) до накопления полного буфера и до выхода таймаута.

Входит и выходит

Закрываем потоки

InputStream и *OutputStream* имплементируют интерфейс *Closable* и обладают методом *close()*. Поток будет находится в памяти и занимать ресурсы до вызова этого метода.

Потоки всегда нужно закрывать после их использования

```
try (InputStream input = new FileInputStream("notes.txt")) { // try-with-resources
    byte[] bytes = input.readAllBytes();
} catch (IOException e) {
    System.err.println("Ошибка чтения файла: " + e.getLocalizedMessage());
}
```

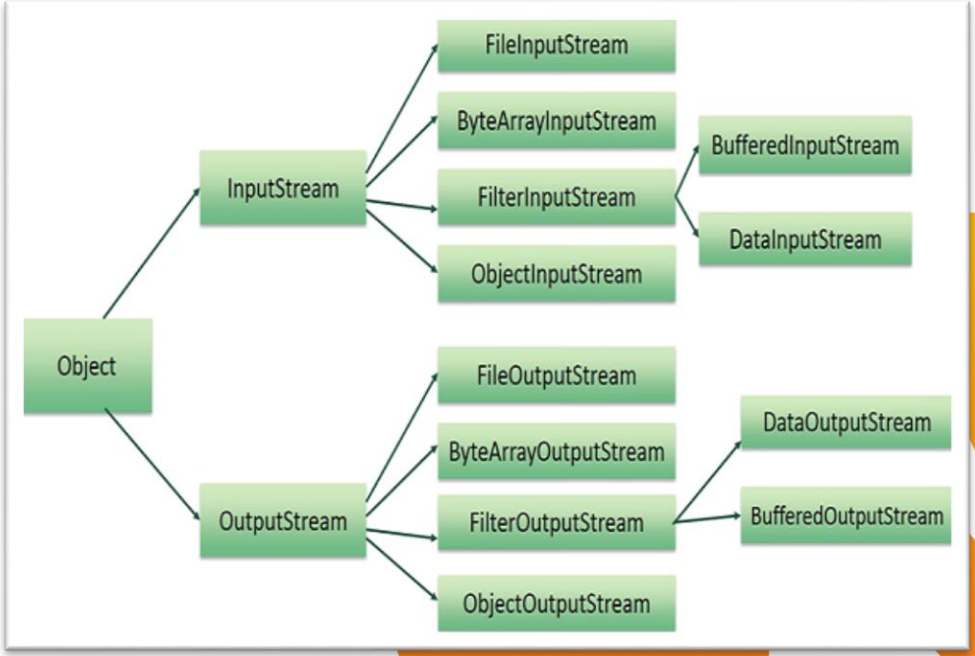
```
InputStream input = null;
try { // закрытие потока в секции finally
    input = new FileInputStream("notes.txt");
    byte[] bytes = input.readAllBytes();
} catch (IOException e) {
    System.err.println("Ошибка чтения файла: " + e.getLocalizedMessage());
} finally {
    if (input != null) input.close();
}
```

Входит и выходит

Закрываем потоки

Для побайтовой обработки данных из файлов применяются *FileInputStream* и *FileOutputStream.* Существуют и другие имплементации *InputStream* и *OutputStream*.

Кроме того, в Java символы хранятся с использованием соглашений Unicode, поэтому потоки могут сразу преобразовывать байты в символы, что позволяет нам читать/записывать данные посимвольно. Наиболее популярные имплементации – *FileReader* и *FileWriter*.



Задание

TEL-RAN
by Starta Institute

1 Создайте файл test.txt на компьютере. Внутри файла test.txt напишите несколько слов – каждое слово на новой строке. С помощью класса Scanner прочитайте содержимое файла, добавляя каждое прочитанное слово в список.

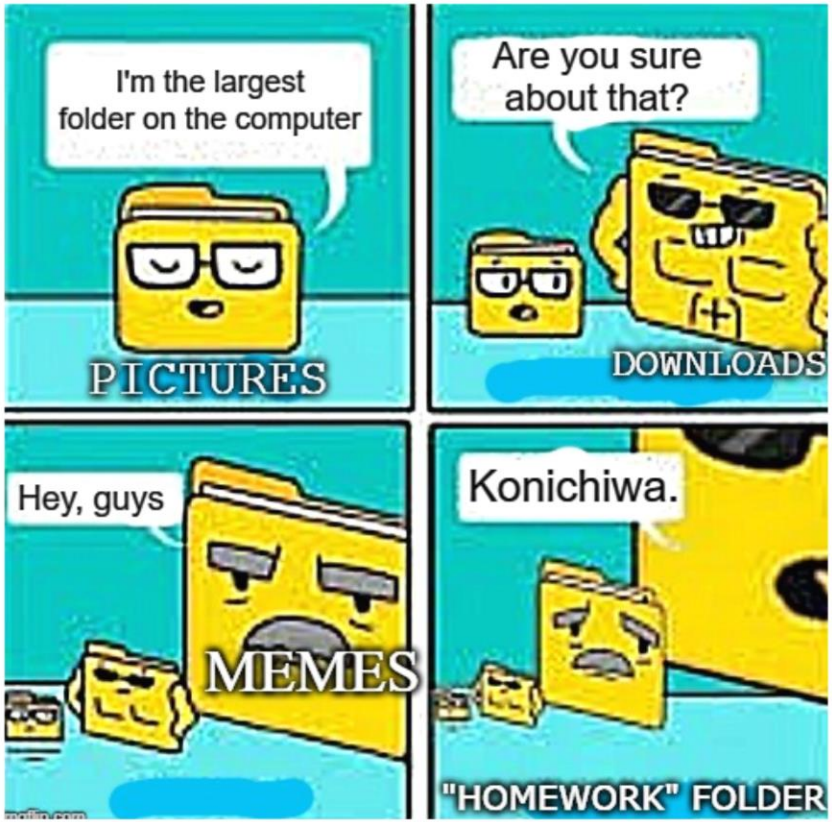
2 Создайте метод, который принимает список слов и выходной поток в который нужно выводить данные. Вызовите метод, передав в него поток вывода в консоль, затем повторите вызов, передав в метод поток записи в файл.

Проблема

TEL-RAN
by Starta Institute

Работа с содержимым файлов – полезный навык, но очень часто нужно производить операции на уровне файлов и папок.

Можно ли организовать такую работу без создания потоков?



Файлики и папочки

Класс Paths

TEL-RAN
by Starta Institute

Paths – класс с единственным статическим методом *get()*. Его создали исключительно для того, чтобы из переданной строки или URI получить объект типа *Path*.

```
import java.nio.file.Path;
import java.nio.file.Paths;

public class Main {

    public static void main(String[] args) {

        Path testFilePath = Paths.get("C:\\Users\\Username\\Desktop\\testFile.txt");
    }
}
```

Файлики и папочки

Класс Path

TEL-RAN
by Starta Institute

Path – это переработанный аналог класса *File*. Работать с ним значительно проще, чем с *File*: из него убрали многие утилитные (статические) методы, и перенесли их в класс *Files*; в *Path* были упорядочены возвращаемые значения методов. В классе *File* методы возвращали то *String*, то *boolean*, то *File* – разобраться было непросто.

Файлики и папочки

Методы Path

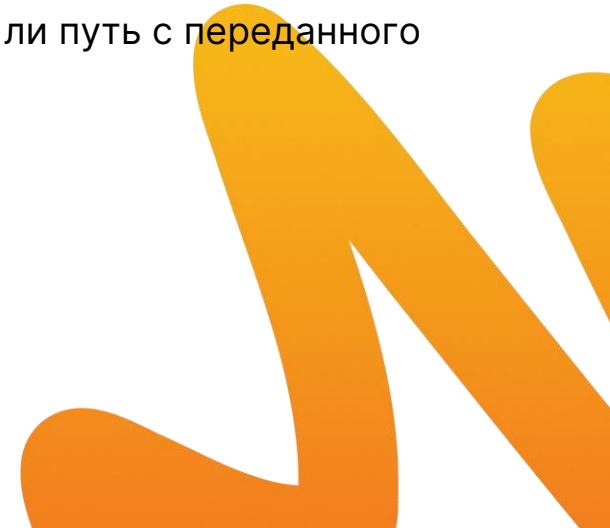
TEL-RAN
by Starta Institute

getFileName() – возвращает имя файла из пути;

getParent() – возвращает «родительскую» директорию по отношению к текущему пути (то есть ту директорию, которая находится выше по дереву каталогов);

getRoot() – возвращает «корневую» директорию; то есть ту, которая находится на вершине дерева каталогов;

startsWith(), endsWith() – проверяют, начинается/заканчивается ли путь с переданного пути.



Файлики и папочки

Пример использования Path

TEL-RAN
by Starta Institute

```
import java.nio.file.Path;
import java.nio.file.Paths;

public class Main {

    public static void main(String[] args) {
        Path testFilePath = Paths.get("C:\\Users\\Username\\Desktop\\testFile.txt");
        Path fileName = testFilePath.getFileName();
        System.out.println(fileName);
        Path parent = testFilePath.getParent();
        System.out.println(parent);
        Path root = testFilePath.getRoot();
        System.out.println(root);
        boolean endWithTxt = testFilePath.endsWith("Desktop\\testFile.txt");
        System.out.println(endWithTxt);
        boolean startsWithLalala = testFilePath.startsWith("lalalala");
        System.out.println(startsWithLalala);
    }
}
```

Файлики и папочки

Пример использования Path

TEL-RAN
by Starta Institute

В *Path* есть группа методов, которая упрощает работу с абсолютными (полными) и относительными путями:

boolean isAbsolute() – возвращает *true*, если текущий путь является абсолютным;

Path normalize() – «нормализует» текущий путь, удаляя из него ненужные “.” или “..”, которые используются в некоторых ОС при обозначении путей: “.” – текущая директория “..” – родительская директория.

Например: “./Pictures/dog.jpg” обозначает, что в той директории, в которой мы сейчас находимся, есть папка Pictures, а в ней – файл “dog.jpg”

Path relativize() – вычисляет относительный путь между текущим и переданным путем.

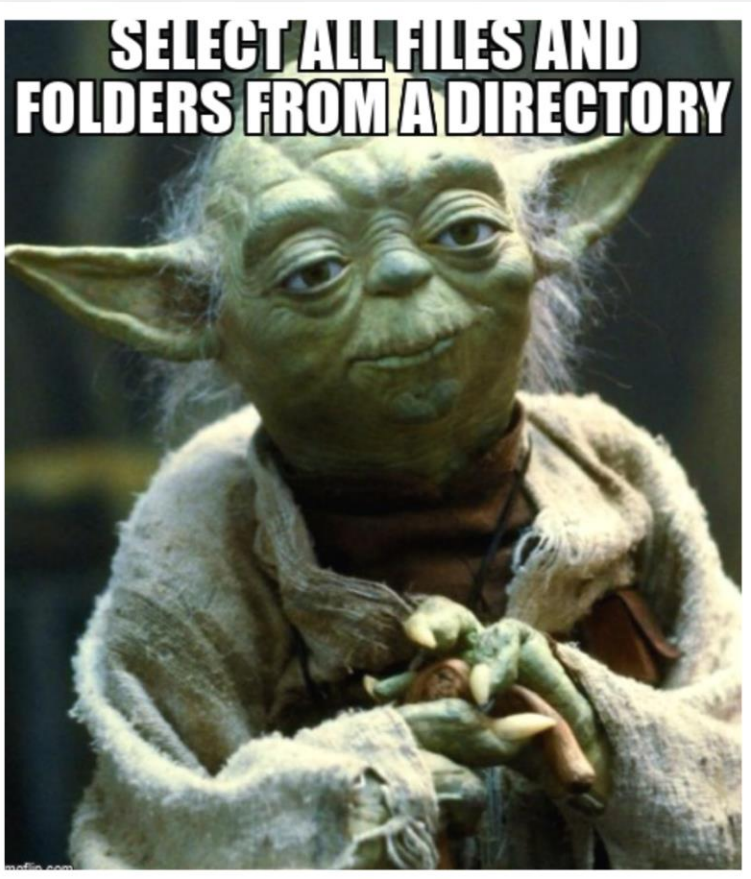
Файлики и папочки

Класс Files

TEL-RAN
by Starta Institute

Files – утилитный класс, куда были вынесены статические методы из класса *File*; позволяет создавать, удалять и перемещать файлы и директории.

Files похож на *Arrays* или *Collections*, только работает он с файлами, а не с массивами и коллекциями.



Файлики и папочки

Методы Files

TEL-RAN
by Starta Institute

createFile() – создание файла;

createDirectory() – создание каталога;

copy() – копирование файла/каталога;

move() – перемещением файла/каталога;

delete() – удаление файла/каталога.

Files также работает с содержимым файлов:

write() – запись данных в файл;

read() – прочитать байт;

readAllBytes() – прочитать все байты;

readAllLines() – прочитать все строки из файла, возвоащает List<String>.

Файлики и папочки

Пример применения Files

TEL-RAN
by Starta Institute

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

import static java.nio.file.StandardCopyOption.REPLACE_EXISTING;

public class Main {

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

        //создание файла
        Path testFile1 = Files.createFile(Paths.get("C:\\Users\\Username\\Desktop\\testFile111.txt"));
        System.out.println("Был ли файл успешно создан?");
        System.out.println(Files.exists(Paths.get("C:\\Users\\Username\\Desktop\\testFile111.txt")));
```

Файлики и папочки

Пример применения Files

TEL-RAN
by Starta Institute

```
        //создание директории
        Path testDirectory = Files.createDirectory(Paths.get("C:\\Users\\Username\\Desktop\\testDirectory"));
        System.out.println("Была ли директория успешно создана?");
        System.out.println(Files.exists(Paths.get("C:\\Users\\Username\\Desktop\\testDirectory")));

        //перемещаем файл с рабочего стола в директорию testDirectory. Перемещать надо с указанием имени файла в папке!
        testFile1 = Files.move(testFile1, Paths.get("C:\\Users\\Username\\Desktop\\testDirectory\\testFile111.txt"),
REPLACE_EXISTING);

        System.out.println("Остался ли наш файл на рабочем столе?");
        System.out.println(Files.exists(Paths.get("C:\\Users\\Username\\Desktop\\testFile111.txt")));

        System.out.println("Был ли наш файл перемещен в testDirectory?");
        System.out.println(Files.exists(Paths.get("C:\\Users\\Username\\Desktop\\testDirectory\\testFile111.txt")));

        //удаление файла
        Files.delete(testFile1);
        System.out.println("Файл все еще существует?");
        System.out.println(Files.exists(Paths.get("C:\\Users\\Username\\Desktop\\testDirectory\\testFile111.txt")));
    }
}
```

Задание

Ранее, при изучении Map, мы создали программу-переводчик для английского языка. Программа выдавала перевод введённого слова в соответствии с имеющимся у неё словарём. Если слова не было в словаре, то программа просила пользователя ввести перевод и добавляла слово в словарь. Теперь нужно добавить в программу возможность хранить свой словарь в виде файла. Словарь загружается при старте программы. Также нужно добавить возможность импортировать слова из других файлов словарей (пример - Словарь1.vocab).

Словарь1.vocab

Проблема

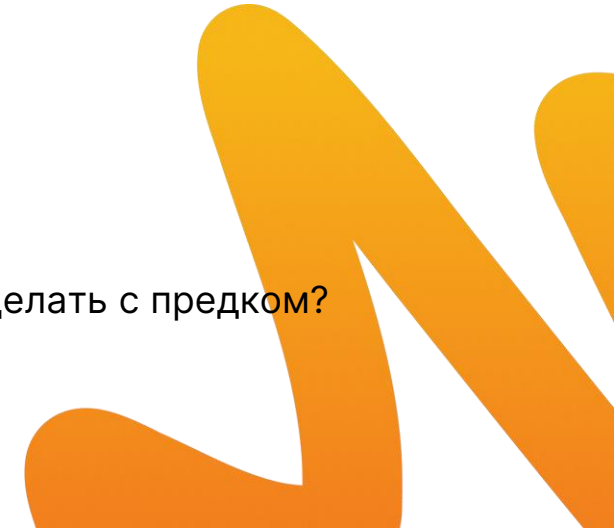
TEL-RAN
by Starta Institute

Записать в файл примитивные типы и даже строки достаточно просто.

Но каким образом записывать более сложные структуры данных? Структуры данных в ООП, будь то массив, коллекция, мапа, дерево или что-нибудь ещё, выражаются соответствующими экземпляры классов. *Значит, нужен механизм, который умеет сохранять любые объекты в виде последовательности байт.*

А что если,

- объект будет содержать ссылки на другие объекты?
- экземпляр будет принадлежать классу-наследнику? Что делать с предком?
- не все поля нужно сохранить?



Перед отправкой

Ответ - сериализация

TEL-RAN
by Starta Institute

Перед отправкой

Сериализация и десериализация

TEL-RAN
by Starta Institute

Сериализация – это механизм записи состояния объекта в поток. По сути, этот механизм может сохранить структуру объекта, т.е. набор его полей, включая типы полей и их значения.

Десериализация – обратный процесс – процесс извлечения или восстановления состояния объекта из потока.

Перед отправкой

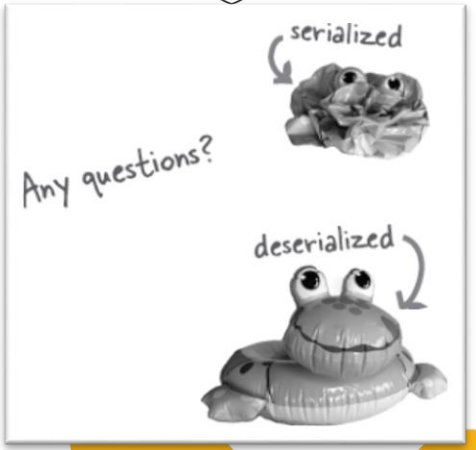
Механизм сериализации

TEL-RAN
by Starta Institute

Исходный код класса, объекты которого можно сериализовать:

```
import java.io.Serializable;

class TestSerial implements Serializable {
    public byte version = 100;
    public byte count = 0;
}
```



Объект класса TestSerial, записанный в файл, в HEX выглядит так:

```
AC ED 00 05 73 72 00 0A 53 65 72 69 61 6C 54 65 73 74 A0 0C 34 00 FE B1 DD F9 02 00 02 42 00 05
63 6F 75 6E 74 42 00 07 76 65 72 73 69 6F 6E 78 70 00 64
```

У *TestSerial* всего два поля типа byte, т.е. объект должен состоять из двух байт в сериализованном виде. Но размер сериализованного объекта составляет 51 байт из-за оверхеда, который добавляет механизм сериализации. Такой оверхед нужен для того, чтобы алгоритм был универсален для любой структуры данных.

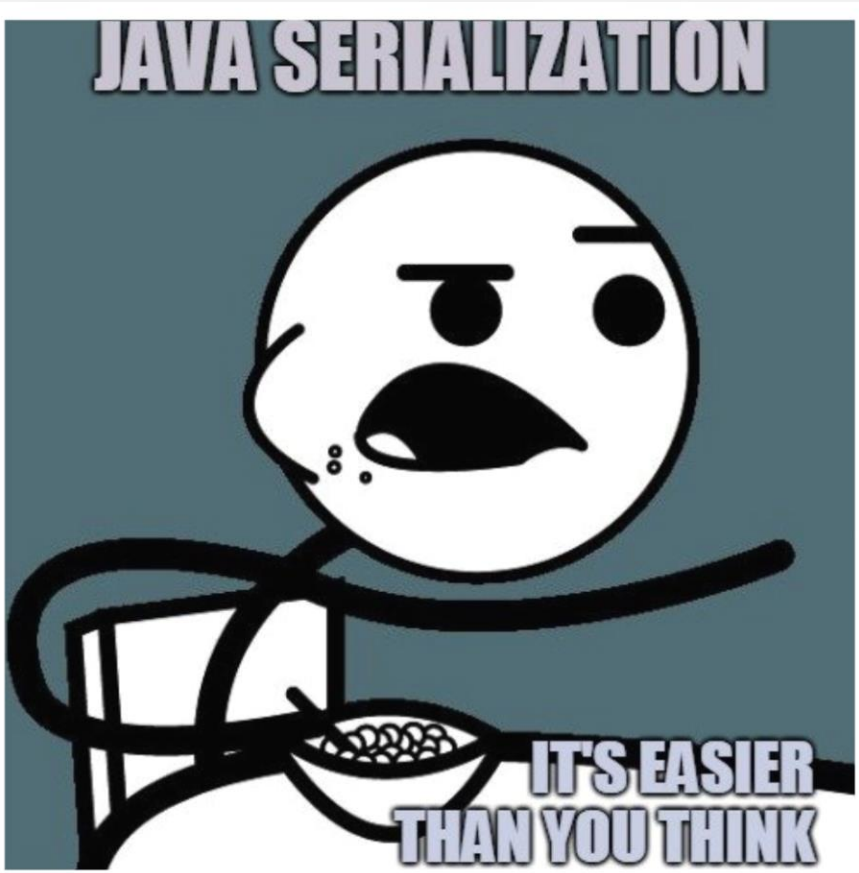
Перед отправкой

Механизм сериализации

TEL-RAN
by Starta Institute

Алгоритм сериализации:

- запись метаданных о классе, ассоциированном с объектом (структура класса)
- рекурсивная запись описания суперклассов, выполняющаяся до тех пор пока не будет достигнут java.lang.Object
- запись фактических данных экземпляра, начиная с самого верхнего суперкласса
- если у класса есть поля ссылочного типа, то далее записываются сериализованные объекты этих полей.



Перед отправкой

Механизм сериализации

TEL-RAN
by Starta Institute

Дан класс из иерархии наследования *SerialTest.* После сериализации:

```
class Parent implements Serializable {
    int parentVersion = 10;
}

class Contain implements Serializable{
    int containVersion = 11;
}
public class SerialTest extends Parent implements Serializable {
    int version = 66;
    Contain con = new Contain();

    public int getVersion() {
        return version;
    }
}
```

```
AC ED 00 05 73 72 00 0A 53 65 72 69 61 6C
54 65 73 74 05 52 81 5A AC 66 02 F6 02 00 02
49 00 07 76 65 72 73 69 6F 6E 4C 00 03 63 6F
6E 74 00 09 4C 63 6F 6E 74 61 69 6E 3B 78
72 00 06 70 61 72 65 6E 74 0E DB D2 BD 85
EE 63 7A 02 00 01 49 00 0D 70 61 72 65 6E
74 56 65 72 73 69 6F 6E 78 70 00 00 00 0A 00
00 00 42 73 72 00 07 63 6F 6E 74 61 69 6E
FC BB E6 0E FB CB 60 C7 02 00 01 49 00 0E
63 6F 6E 74 61 69 6E 56 65 72 73 69 6F 6E 78
70 00 00 00 0B
```

Перед отправкой

Механизм сериализации

TEL-RAN by Starta Institute

Дан класс из иерархии наследования *SerialTest.* После сериализации:

```
class Parent implements Serializable {
    int parentVersion = 10;
}

class Contain implements Serializable{
    int containVersion = 11;
}
public class SerialTest extends Parent implements Serializable {
    int version = 66;
    Contain con = new Contain();

    public int getVersion() {
        return version;
    }
}
```

```
AC ED 00 05 73 72 00 0A 53 65 72 69 61 6C
54 65 73 74 05 52 81 5A AC 66 02 F6 02 00 02
49 00 07 76 65 72 73 69 6F 6E 4C 00 03 63 6F
6E 74 00 09 4C 63 6F 6E 74 61 69 6E 3B 78
72 00 06 70 61 72 65 6E 74 0E DB D2 BD 85
EE 63 7A 02 00 01 49 00 0D 70 61 72 65 6E
74 56 65 72 73 69 6F 6E 78 70 00 00 00 0A 00
00 00 42 73 72 00 07 63 6F 6E 74 61 69 6E
FC BB E6 0E FB CB 60 C7 02 00 01 49 00 0E
63 6F 6E 74 61 69 6E 56 65 72 73 69 6F 6E 78
70 00 00 00 0B
```

AC ED: STREAM_MAGIC. Говорит о том, что используется протокол сериазизации.
00 05: STREAM_VERSION. Версия сериализации.
0x73: TC_OBJECT. Обозначение нового объекта.

Перед отправкой

Механизм сериализации

TEL-RAN
by Starta Institute

Теперь алгоритм записывает поле int version = 66;

```
class Parent implements Serializable {
    int parentVersion = 10;
}

class Contain implements Serializable{
    int containVersion = 11;
}
public class SerialTest extends Parent implements Serializable {
    int version = 66;
    Contain con = new Contain();

    public int getVersion() {
        return version;
    }
}
```

```
AC ED 00 05 73 72 00 0A 53 65 72 69 61 6C
54 65 73 74 05 52 81 5A AC 66 02 F6 02 00 02
49 00 07 76 65 72 73 69 6F 6E 4C 00 03 63 6F
6E 74 00 09 4C 63 6F 6E 74 61 69 6E 3B 78
72 00 06 70 61 72 65 6E 74 0E DB D2 BD 85
EE 63 7A 02 00 01 49 00 0D 70 61 72 65 6E
74 56 65 72 73 69 6F 6E 78 70 00 00 00 0A 00
00 00 42 73 72 00 07 63 6F 6E 74 61 69 6E
FC BB E6 0E FB CB 60 C7 02 00 01 49 00 0E
63 6F 6E 74 61 69 6E 56 65 72 73 69 6F 6E 78
70 00 00 00 0B
```

0x72: TC_CLASSDESC. Обозначение нового класса.
00 0A: Длина имени класса.
53 65 72 69 61 6c 54 65 73 74: *SerialTest*, имя класса.
05 52 81 5A AC 66 02 F6: *SerialVersionUID*, идентификатор класса.
0x02: Различные флаги. Этот специфический флаг говорит о том, что объект поддерживает сериализацию.
00 02: Число полей в классе.

Перед отправкой

Механизм сериализации

TEL-RAN by Starta Institute

Теперь алгоритм записывает поле int version = 66;

```
class Parent implements Serializable {
    int parentVersion = 10;
}

class Contain implements Serializable{
    int containVersion = 11;
}
public class SerialTest extends Parent implements Serializable {
    int version = 66;
    Contain con = new Contain();

    public int getVersion() {
        return version;
    }
}
```

```
AC ED 00 05 73 72 00 0A 53 65 72 69 61 6C
54 65 73 74 05 52 81 5A AC 66 02 F6 02 00 02
49 00 07 76 65 72 73 69 6F 6E 4C 00 03 63 6F
6E 74 00 09 4C 63 6F 6E 74 61 69 6E 3B 78
72 00 06 70 61 72 65 6E 74 0E DB D2 BD 85
EE 63 7A 02 00 01 49 00 0D 70 61 72 65 6E
74 56 65 72 73 69 6F 6E 78 70 00 00 00 0A 00
00 00 42 73 72 00 07 63 6F 6E 74 61 69 6E
FC BB E6 0E FB CB 60 C7 02 00 01 49 00 0E
63 6F 6E 74 61 69 6E 56 65 72 73 69 6F 6E 78
70 00 00 00 0B
```

0x49: Код типа поля. 49 это «I», которое закреплено за *Int*.
00 07: Длина имени поля.
76 65 72 73 69 6F 6E: *version*, имя поля.
4C 00 03 63 6F 6E: некоторый оверхед.

Перед отправкой

Механизм сериализации

TEL-RAN
by Starta Institute

```
class Parent implements Serializable {
    int parentVersion = 10;
}

class Contain implements Serializable{
    int containVersion = 11;
}
public class SerialTest extends Parent implements Serializable {
    int version = 66;
    Contain con = new Contain();

    public int getVersion() {
        return version;
    }
}
```

```
AC ED 00 05 73 72 00 0A 53 65 72 69 61 6C
54 65 73 74 05 52 81 5A AC 66 02 F6 02 00 02
49 00 07 76 65 72 73 69 6F 6E 4C 00 03 63 6F
6E 74 00 09 4C 63 6F 6E 74 61 69 6E 3B 78
72 00 06 70 61 72 65 6E 74 0E DB D2 BD 85
EE 63 7A 02 00 01 49 00 0D 70 61 72 65 6E
74 56 65 72 73 69 6F 6E 78 70 00 00 00 0A 00
00 00 42 73 72 00 07 63 6F 6E 74 61 69 6E
FC BB E6 0E FB CB 60 C7 02 00 01 49 00 0E
63 6F 6E 74 61 69 6E 56 65 72 73 69 6F 6E 78
70 00 00 00 0B
```

Затем алгоритм записывает следующее поле, contain con = new contain(); Это объект следовательно будет записано каноническое JVM обозначение этого поля.

0x74: TC_STRING. Обозначает новую строку.
00 09: Длина строки.
4C 63 6F 6E 74 61 69 6E 3B: *Lcontain*; Каноническое JVM обозначение.
0x78: TC_ENDBLOCKDATA, Конец опционального блока данных для объекта.

Перед отправкой

Механизм сериализации

TEL-RAN by Starta Institute

```
class Parent implements Serializable {
    int parentVersion = 10;
}

class Contain implements Serializable{
    int containVersion = 11;
}
public class SerialTest extends Parent implements Serializable {
    int version = 66;
    Contain con = new Contain();

    public int getVersion() {
        return version;
    }
}
```

```
AC ED 00 05 73 72 00 0A 53 65 72 69 61 6C
54 65 73 74 05 52 81 5A AC 66 02 F6 02 00 02
49 00 07 76 65 72 73 69 6F 6E 4C 00 03 63 6F
6E 74 00 09 4C 63 6F 6E 74 61 69 6E 3B 78
72 00 06 70 61 72 65 6E 74 0E DB D2 BD 85
EE 63 7A 02 00 01 49 00 0D 70 61 72 65 6E
74 56 65 72 73 69 6F 6E 78 70 00 00 00 0A 00
00 00 42 73 72 00 07 63 6F 6E 74 61 69 6E
FC BB E6 0E FB CB 60 C7 02 00 01 49 00 0E
63 6F 6E 74 61 69 6E 56 65 72 73 69 6F 6E 78
70 00 00 00 0B
```

Следующим шагом алгоритма является запись описания класса *Parent*, который является непосредственным суперклассом для *SerialTest*.

0x72: TC_CLASSDESC. Обозначение нового класса.
00 06: Длина имени класса.
70 61 72 65 6E 74: *parent*, имя класса
0E DB D2 BD 85 EE 63 7A: *SerialVersionUID*, идентификатор класса.
0x02: Различные флаги. Этот флаг обозначает что класс поддерживает сериализацию.
00 01: Число полей в классе

Перед отправкой

Механизм сериализации

TEL-RAN by Starta Institute

```
class Parent implements Serializable {
    int parentVersion = 10;
}

class Contain implements Serializable{
    int containVersion = 11;
}
public class SerialTest extends Parent implements Serializable {
    int version = 66;
    Contain con = new Contain();

    public int getVersion() {
        return version;
    }
}
```

```
AC ED 00 05 73 72 00 0A 53 65 72 69 61 6C
54 65 73 74 05 52 81 5A AC 66 02 F6 02 00 02
49 00 07 76 65 72 73 69 6F 6E 4C 00 03 63 6F
6E 74 00 09 4C 63 6F 6E 74 61 69 6E 3B 78
72 00 06 70 61 72 65 6E 74 0E DB D2 BD 85
EE 63 7A 02 00 01 49 00 0D 70 61 72 65 6E
74 56 65 72 73 69 6F 6E 78 70 00 00 00 0A 00
00 00 42 73 72 00 07 63 6F 6E 74 61 69 6E
FC BB E6 0E FB CB 60 C7 02 00 01 49 00 0E
63 6F 6E 74 61 69 6E 56 65 72 73 69 6F 6E 78
70 00 00 00 0B
```

Теперь алгоритм записывает описание полей класса *Parent*, класс имеет одно поле, *int parentVersion = 100;*

0x49: Код типа поля. *49* обозначает «I», которое закреплено за *Int*.
00 0D: Длина имени поля.
70 61 72 65 6E 74 56 65 72 73 69 6F 6E: *parentVersion*, имя поля.
0x78: TC_ENDBLOCKDATA, конец опционального блока данных для объекта.
0x70: TC_NULL, обозначает то, что больше нет суперклассов, потому что мы достигли верха иерархии классов.

Перед отправкой

Механизм сериализации

TEL-RAN by Starta Institute

```
class Parent implements Serializable {
    int parentVersion = 10;
}

class Contain implements Serializable{
    int containVersion = 11;
}
public class SerialTest extends Parent implements Serializable {
    int version = 66;
    Contain con = new Contain();

    public int getVersion() {
        return version;
    }
}
```

До этого алгоритм сериализации записывал описание классов, ассоциированных с объектом, и всех его суперклассов.

```
AC ED 00 05 73 72 00 0A 53 65 72 69 61 6C
54 65 73 74 05 52 81 5A AC 66 02 F6 02 00 02
49 00 07 76 65 72 73 69 6F 6E 4C 00 03 63 6F
6E 74 00 09 4C 63 6F 6E 74 61 69 6E 3B 78
72 00 06 70 61 72 65 6E 74 0E DB D2 BD 85
EE 63 7A 02 00 01 49 00 0D 70 61 72 65 6E
74 56 65 72 73 69 6F 6E 78 70 00 00 00 0A 00
00 00 42 73 72 00 07 63 6F 6E 74 61 69 6E
FC BB E6 0E FB CB 60 C7 02 00 01 49 00 0E
63 6F 6E 74 61 69 6E 56 65 72 73 69 6F 6E 78
70 00 00 00 0B
```

Теперь будут записаны фактические данные, ассоциированные с объектом. Сначала записываются члены класса *parent*:
00 00 00 0A: 10, Значение *parentVersion*.

Перед отправкой

Механизм сериализации

TEL-RAN by Starta Institute

```
class Parent implements Serializable {
    int parentVersion = 10;
}

class Contain implements Serializable{
    int containVersion = 11;
}
public class SerialTest extends Parent implements Serializable {
    int version = 66;
    Contain con = new Contain();

    public int getVersion() {
        return version;
    }
}
```

До этого алгоритм сериализации записывал описание классов, ассоциированных с объектом, и всех его суперклассов.

```
AC ED 00 05 73 72 00 0A 53 65 72 69 61 6C
54 65 73 74 05 52 81 5A AC 66 02 F6 02 00 02
49 00 07 76 65 72 73 69 6F 6E 4C 00 03 63 6F
6E 74 00 09 4C 63 6F 6E 74 61 69 6E 3B 78
72 00 06 70 61 72 65 6E 74 0E DB D2 BD 85
EE 63 7A 02 00 01 49 00 0D 70 61 72 65 6E
74 56 65 72 73 69 6F 6E 78 70 00 00 00 0A 00
00 00 42 73 72 00 07 63 6F 6E 74 61 69 6E
FC BB E6 0E FB CB 60 C7 02 00 01 49 00 0E
63 6F 6E 74 61 69 6E 56 65 72 73 69 6F 6E 78
70 00 00 00 0B
```

00 00 00 42: 66, Значение version

Перед отправкой

Механизм сериализации

TEL-RAN by Starta Institute

```
class Parent implements Serializable {
    int parentVersion = 10;
}

class Contain implements Serializable{
    int containVersion = 11;
}
public class SerialTest extends Parent implements Serializable {
    int version = 66;
    Contain con = new Contain();

    public int getVersion() {
        return version;
    }
}
```

```
AC ED 00 05 73 72 00 0A 53 65 72 69 61 6C
54 65 73 74 05 52 81 5A AC 66 02 F6 02 00 02
49 00 07 76 65 72 73 69 6F 6E 4C 00 03 63 6F
6E 74 00 09 4C 63 6F 6E 74 61 69 6E 3B 78
72 00 06 70 61 72 65 6E 74 0E DB D2 BD 85
EE 63 7A 02 00 01 49 00 0D 70 61 72 65 6E
74 56 65 72 73 69 6F 6E 78 70 00 00 00 0A 00
00 00 42 73 72 00 07 63 6F 6E 74 61 69 6E
FC BB E6 0E FB CB 60 C7 02 00 01 49 00 0E
63 6F 6E 74 61 69 6E 56 65 72 73 69 6F 6E 78
70 00 00 00 0B
```

Алгоритму необходимо записать информацию об объекте класса *Contain*.

0x73: TC_OBJECT, обозначает новый объект.
0x72: TC_CLASSDESC, обозначает новый класс.
00 07: Длина имени класса.
63 6F 6E 74 61 69 6E: *Contain*, имя класса.
FC BB E6 0E FB CB 60 C7: *SerialVersionUID*, идентификатор этого класса.
0x02: Различные флаги. Этот флаг обозначает что класс поддерживает сериализацию.
00 01: Число полей в классе.

Перед отправкой

Механизм сериализации

TEL-RAN by Starta Institute

```
class Parent implements Serializable {
    int parentVersion = 10;
}

class Contain implements Serializable{
    int containVersion = 11;
}
public class SerialTest extends Parent implements Serializable {
    int version = 66;
    Contain con = new Contain();

    public int getVersion() {
        return version;
    }
}
```

Алгоритм должен записать описание единственного поля класса Conatin, int containVersion = 11;

```
AC ED 00 05 73 72 00 0A 53 65 72 69 61 6C
54 65 73 74 05 52 81 5A AC 66 02 F6 02 00 02
49 00 07 76 65 72 73 69 6F 6E 4C 00 03 63 6F
6E 74 00 09 4C 63 6F 6E 74 61 69 6E 3B 78
72 00 06 70 61 72 65 6E 74 0E DB D2 BD 85
EE 63 7A 02 00 01 49 00 0D 70 61 72 65 6E
74 56 65 72 73 69 6F 6E 78 70 00 00 00 0A 00
00 00 42 73 72 00 07 63 6F 6E 74 61 69 6E
FC BB E6 0E FB CB 60 C7 02 00 01 49 00 0E
63 6F 6E 74 61 69 6E 56 65 72 73 69 6F 6E 78
70 00 00 00 0B
```

0x49: Код типа поля. 49 обозначает «I», которое закреплено за *Int*.
00 0E: Длина имени поля.
63 6F 6E 74 61 69 6E 56 65 72 73 69 6F 6E: containVersion, имя поля.
0x78: TC_ENDBLOCKDATA, конец опционального блока данных для объекта.

Перед отправкой

Механизм сериализации

TEL-RAN by Starta Institute

```
class Parent implements Serializable {
    int parentVersion = 10;
}

class Contain implements Serializable{
    int containVersion = 11;
}
public class SerialTest extends Parent implements Serializable {
    int version = 66;
    Contain con = new Contain();

    public int getVersion() {
        return version;
    }
}
```

```
AC ED 00 05 73 72 00 0A 53 65 72 69 61 6C
54 65 73 74 05 52 81 5A AC 66 02 F6 02 00 02
49 00 07 76 65 72 73 69 6F 6E 4C 00 03 63 6F
6E 74 00 09 4C 63 6F 6E 74 61 69 6E 3B 78
72 00 06 70 61 72 65 6E 74 0E DB D2 BD 85
EE 63 7A 02 00 01 49 00 0D 70 61 72 65 6E
74 56 65 72 73 69 6F 6E 78 70 00 00 00 0A 00
00 00 42 73 72 00 07 63 6F 6E 74 61 69 6E
FC BB E6 0E FB CB 60 C7 02 00 01 49 00 0E
63 6F 6E 74 61 69 6E 56 65 72 73 69 6F 6E 78
70 00 00 00 0B
```

Дальше алгоритм проверяет, имеет ли *Contain* родительский класс. Если имеет, то алгоритм начинает запись этого класса;

но в нашем случае суперкласса у contain нету, и алгоритм записывает TC_NULL.
0x70: TC_NULL

Перед отправкой

Механизм сериализации

TEL-RAN
by Starta Institute

```
class Parent implements Serializable {
    int parentVersion = 10;
}

class Contain implements Serializable{
    int containVersion = 11;
}
public class SerialTest extends Parent implements Serializable {
    int version = 66;
    Contain con = new Contain();

    public int getVersion() {
        return version;
    }
}
```

В конце алгоритм записывает фактические данные ассоциированные с объектом класса *Conatin*.

```
AC ED 00 05 73 72 00 0A 53 65 72 69 61 6C
54 65 73 74 05 52 81 5A AC 66 02 F6 02 00 02
49 00 07 76 65 72 73 69 6F 6E 4C 00 03 63 6F
6E 74 00 09 4C 63 6F 6E 74 61 69 6E 3B 78
72 00 06 70 61 72 65 6E 74 0E DB D2 BD 85
EE 63 7A 02 00 01 49 00 0D 70 61 72 65 6E
74 56 65 72 73 69 6F 6E 78 70 00 00 00 0A 00
00 00 42 73 72 00 07 63 6F 6E 74 61 69 6E
FC BB E6 0E FB CB 60 C7 02 00 01 49 00 0E
63 6F 6E 74 61 69 6E 56 65 72 73 69 6F 6E 78
70 00 00 00 0B
```

00 00 00 0B: 11, значение *containVersion*.

Перед отправкой

Идентификатор класса

При стандартной сериализации учитывается порядок объявления полей в классе. Во всяком случае, так было в ранних версиях, в JVM версии 1.6.

Состав же методов с очень большой вероятностью повлияет на стандартный механизм, при том, что поля могут вообще остаться теми же.

Чтобы этого избежать, есть следующий механизм. В каждый класс, реализующий интерфейс *Serializable*, на стадии компиляции добавляется еще одно поле – *private static final long serialVersionUID*.

Это поле содержит уникальный идентификатор версии сериализованного класса. Оно вычисляется по содержимому класса – полям и их порядку объявления, методам и их порядку объявления. Соответственно, при любом изменении в классе это поле поменяет свое значение.

При десериализации значение этого поля сравнивается с имеющимся у класса в виртуальной машине. Если значения не совпадают, выбрасывается *java.io.InvalidClassException*

Перед отправкой

Идентификатор класса

Способ обмануть проверку может оказаться полезным, если набор полей класса и их порядок уже определен, а методы класса могут меняться.

Вручную в классе определить поле *private static final long serialVersionUID*. В принципе, значение этого поля может быть абсолютно любым. Десериализация всегда будет разрешена.

В версии 5.0 в документации появилось приблизительно следующее: *крайне рекомендуется всем сериализуемым классам декларировать это поле в явном виде, ибо вычисление по умолчанию очень чувствительно к деталям структуры класса, которые могут различаться в зависимости от реализации компилятора, и вызывать таким образом неожиданные InvalidClassException при десериализации.*

Перед отправкой

Стандартные способы сериализации

	Наследовать интерфейс *java.io.Serializable*	Наследовать интерфейс *java.io.Externalizable*
Требуется переопределить методы	Нет (маркерный интерфейс*)	Да: writeExternal() и readExternal()
Есть возможность исключить поля из сериализации	Да, с помощью модификатора поля *transient*	Да, в переопределяемых методах
Требование к конструктору класса	Не нужен конструктор для создания восстановления объекта – он просто полностью восстановится из байтов	при восстановлении сначала будет создан объект с помощью конструктора в точке объявления, а затем в него будут записаны значения его полей из байтов, полученных при сериализации
Преимущества	Для стандартного решения код короче	Полноценный интерфейс с возможностью гибкой настройки сериализации
Минусы	Для изменения стандартного поведения используется «костыль»: добавление (не переопределение) в методов writeObject() и readObject()	Много кода

**Serializable* говорит JVM: «объекты этого класса можно сериализовать».

Перед отправкой

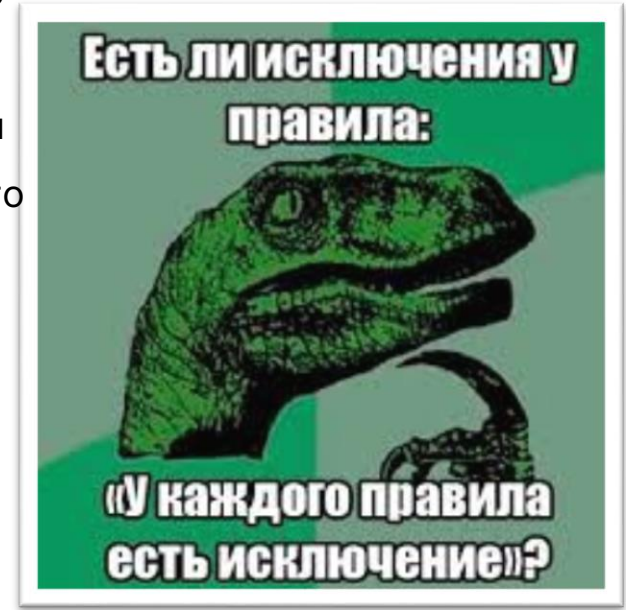
Исключение полей

TEL-RAN
by Starta Institute

1 Когда требуется исключить поле из процесса сериализации (например, если тип поля не поддерживает сериализацию или является чувствительной информацией, как, например, пароль), то в классическом подходе используется модификатор поля *transient*.

2 При стандартной сериализации поля, имеющие модификатор *static*, не сериализуются. Соответственно, после десериализации это поле значения не меняет. Разумеется, при реализации *Externalizable* сериализовать и десериализовать это поле никто не мешает, однако я крайне не рекомендую этого делать, т.к. это может привести к трудноуловимым ошибкам.

3 Поля с модификатором *final* сериализуются как обычные, но их невозможно десериализовать при использовании *Externalizable*.



Перед отправкой

Гибкая сериализация – жёсткие костыли

Указанные ниже методы не входят в интерфейс *java.io.Serializable.* Они были добавлены позже работают непосредственно с JVM.

```
private void writeObject(ObjectOutputStream stream) throws IOException {
    stream.defaultWriteObject(); // Стандартный метод сериализации
    System.out.println("Our writeObject");
}

private void readObject(ObjectInputStream stream) throws IOException, ClassNotFoundException {
    stream.defaultReadObject(); // Стандартный метод десериализации
    System.out.println("Our readObject");
}
```

Т. к. методы не переопределяются, а добавляются в класс, то компилятор не отслеживает правильность их сигнатур. При нарушении сигнатуры метода сериализация не будет работать для данного класса.

Перед отправкой

Сериализация. Класс ObjectOutputStream

Для сериализации объектов и записи в поток используется класс *ObjectOutputStream*. В конструктор *ObjectOutputStream* передается поток, в который производится запись:

```
new ObjectOutputStream(OutputStream out);
```

Помимо общих методов, присущих потокам *ObjectOutputStream* содержит:

write(byte[] buf), writeBoolean(boolean val), writeChar(int val), writeDouble(double val, writeFloat(float val), writeInt(int val), writeLong(long val), writeShort(int val) – для записи примитивов;

writeByte(int val) – записывает в поток один младший байт из val;

writeUTF(String str) – записывает в поток строку в кодировке UTF-8;

writeObject(Object obj) – записывает в поток отдельный объект.

Перед отправкой

Пример применения ObjectOutputStream

```
import lombok.Data;
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;

public class Program {
    public static void main(String[] args) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("person.dat"))) {
            Person p = new Person("Sam", 33, 178, true);
            oos.writeObject(p);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
@Data
class Person implements Serializable {
    private final String name;
    private final int age;
    private final double height;
    private final boolean isMarried;
}
```

Перед отправкой

Десериализация. Класс ObjectInputStream

Класс *ObjectInputStream* отвечает за обратный процесс – чтение ранее сериализованных данных из потока. В конструкторе он принимает ссылку на поток ввода:

```
new ObjectInputStream(InputStream in);
```

По составу методов *ObjectInputStream* аналогичен *ObjectOutputStream,* но использует методы чтения, а не записи. Приведу основные:

String readUTF() – считывает строку из потока;

Object readObject() – считывает объект из потока.

Перед отправкой

Пример применения ObjectInputStream

```
import java.io.FileInputStream;
import java.io.ObjectInputStream;

public class Program2 {

    public static void main(String[] args) {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream("person.dat"))) {
            Person p = (Person) ois.readObject();
            System.out.printf("Name: %s \t Age: %d \n", p.getName(), p.getAge());
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

Перед отправкой

Новый способ клонирования

TEL-RAN
by Starta Institute

Сериализация и десериализация позволяют создать клон объекта. Это дорогостоящий способ клнирования, поэтому он почти не применяется. Зато позволяет обойти некоторые правила и, например, создать клона для класса, реализующего паттерн Одиночка.

Задание

TEL-RAN
by Starta Institute

Студент 1: сделать класс учётки пользователя для сериализации. С 2-3 полями. Передать класс студентам 2 и 3.

Студент 2: сериализовать объект полученного класса в файл. Передать файл студенту 3.

Студент 3: из полученного файла десериализовать объект и вывести его в консоль.

TEL-RAN
by Starta Institute

3 Домашнее задание

Домашнее задание

TEL-RAN
by Starta Institute

1 Есть архив VttFiles.zip с набором vtt-файлов (субтитры к аудиодорожкам). Из субтитров нужно собрать единый файл txt для последующего создания документа со скриптами аудиодорожек. В txt поместите только тексты из субтитров. Все лишние записи (время, WEBVTT) должны быть удалены.

VttFiles.zip

2 *Files* обладает способом обхода дерева каталогов. Более подробно можно об этом почитать в статье https://habr.com/ru/articles/437694/

Создайте программу, которая будет принимать путь к существующей папке на Вашем компьютере. После этого программа должна вывести имена файлов и папок на каждом уровне, начиная с текущей папки до корня дерева (диска C или другого диска).

3 Создайте класс Сотрудник с полями имя, фамилия, дата рождения, должность, заработная плата. Сериализуйте объект и сохраните в файл. Поле заработной платы сериализовать не нужно, т.к. это коммерческая тайна организации.

Дополнительная практика

Изучите статью https://habr.com/ru/articles/47109/

Самостоятельно реализуйте одновременную запись в консоль и в файл. В консоль нужно выводить все данные, в первый файл – обычные сообщения, во второй – сообщения исключений.

ЗАКЛЮЧЕНИЕ

TEL-RAN
by Starta Institute