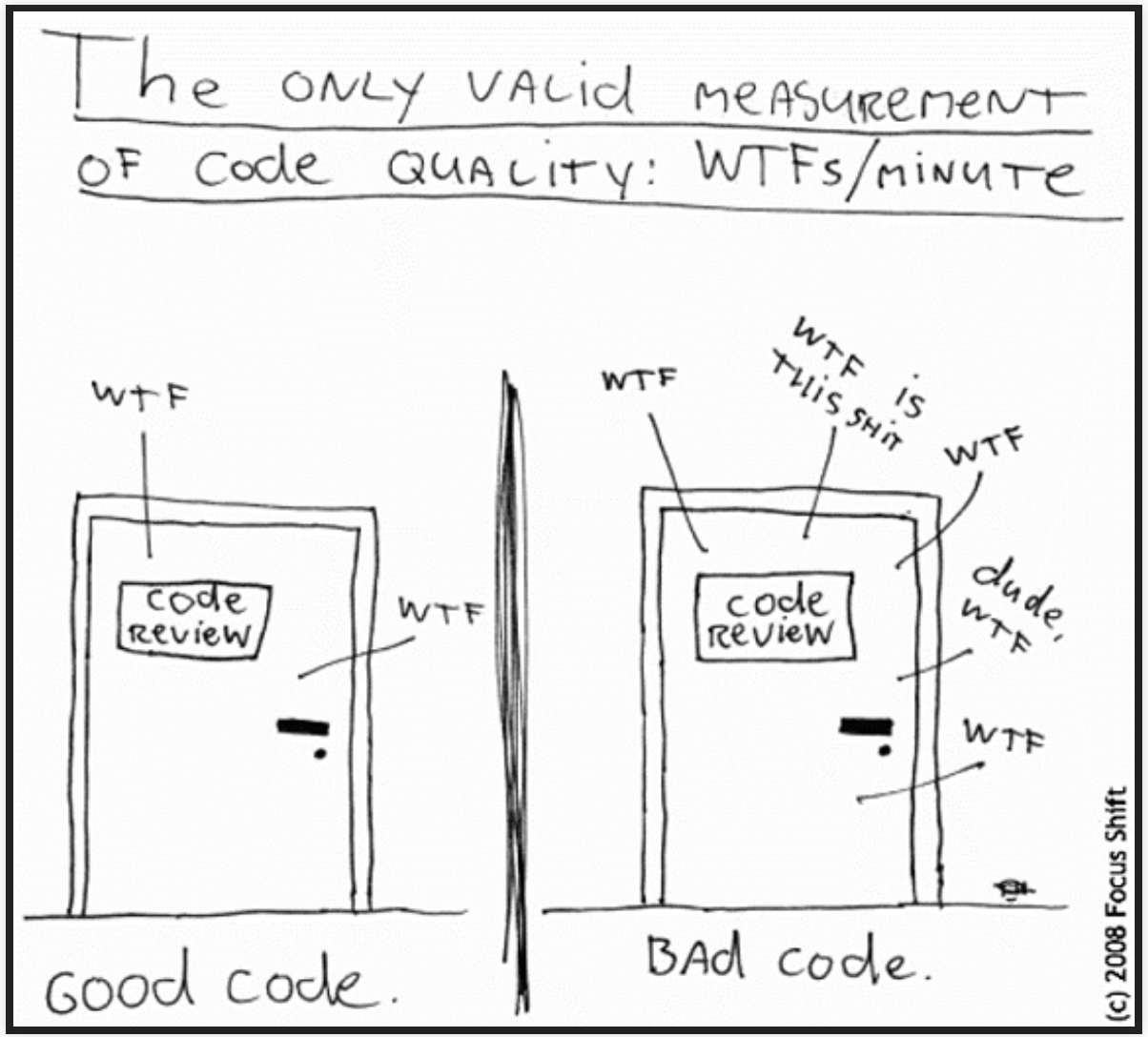
КАЧЕСТВЕН (CLEAN) CODE

13.11.2018

КАК МЕРИМ КАЧЕСТВОТО НА КОДА?



ПРИНЦИПИ НА ЧИСТИЯ КОД - СПАЗВАЙ ООП ПРИНЦИПИТЕ

- Енкапсулация
  - Свеждай публичните части до минимум
- Наследяване
  - Не допускай code duplication

ПРИНЦИПИ НА ЧИСТИЯ КОД - СПАЗВАЙ ООП ПРИНЦИПИТЕ

- Полиморфизъм
  - Ползвай полиморфизъм винаги, когато е възможно
  - Използвай интерфейс за декларация, конкретна имплементация за инициализация
- Абстракция

ПРИНЦИПИ НА ЧИСТИЯ КОД - СТРЕМИ СЕ КЪМ ХУБАВ ОО ДИЗАЙН

- Използвай само по изключение пакета по подразбиране
- Един клас трябва да прави едно нещо
- Ако имаш "and" в името, вероятно не е така
- Ако имаш "Helper", "Manager", "Utility" в името, *може би* има по-добър дизайн
- Един метод трябва да прави едно нещо
- Методът трябва да е кратък: < 20 реда код

ПРИНЦИПИ НА ЧИСТИЯ КОД - СТРЕМИ СЕ КЪМ ХУБАВ ОО ДИЗАЙН

- Ако имаш "and" в името на метод, може би той прави повечко неща: раздели го на няколко
- Ако имаш много параметри на метод:
  - може би не му е мястото в този клас (а там, откъдето се взимат стойностите за тези параметри)
  - или трябва да направиш data object, който да групира семантично свързаните от тях
- Не злоупотребявай със static

ПРИНЦИПИ НА ЧИСТИЯ КОД

- Форматирай си кода
- Нямаш никакво оправдание да не го правиш (IDE shortcut?)
- Вместо "магически числа" в кода, ползвай подходящо именувани константи

ПРИНЦИПИ НА ЧИСТИЯ КОД - СМИСЛЕНИ ИМЕНА

- Имената на пакети, класове, интерфейси, член-променливи, методи, локални променливи трябва да са говорящи и да спазват установените конвенции

ПРИНЦИПИ НА ЧИСТИЯ КОД - СМИСЛЕНИ ИМЕНА

- пакети
  - само малки букви, със смислена йерархия
  - `bg.sofia.uni.fmi.mjt`
- класове, абстрактни класове, интерфейси, enums
  - съществителни, започващи с главна буква (upper camel case)
  - `Student`, `GameBoard`

ПРИНЦИПИ НА ЧИСТИЯ КОД - СМИСЛЕНИ ИМЕНА

- методи
  - глаголи, започващи с малка буква (camel case)
  - `reverseString()`, `calculateSalary()`
- константи
  - all-caps, с подчертавки между думите
  - `MAX_NAME_LENGTH`

ПРИНЦИПИ НА ЧИСТИЯ КОД

```
// Bad
if (x % 2 == 0)
    return x / 2;
```

```
// Good
if (x % 2 == 0) {
    return x / 2;
}
```

- Винаги ограждай в блок телата на if-else и цикли, дори да са с по един statement

ПРИНЦИПИ НА ЧИСТИЯ КОД

```
// Bad
if (x % 2 == 0) {
    return true;
} else {
    return false;
}
```

```
// Good
return x % 2 == 0;
```

- Изразявай се кратко. Малко код == малко бъгове

ПРИНЦИПИ НА ЧИСТИЯ КОД

- Слагай коментари, без да прекаляваш
- Хубавият код е self-explanatory, и все пак, на места си трябва коментар
- Разделяй нормалната логика от exception логиката
- Ползвай изключения вместо error codes и печатане на съобщения в конзолата
- Не suppress-вай / swallow-вай exceptions
- Никога не оставяй празен catch, или catch само `e.printStackTrace()`

ПРИНЦИПИ НА ЧИСТИЯ КОД

- Разделяй I/O логиката от бизнес логиката

JAVA КОД КОНВЕНЦИИ

- Запознай се с цялостна Java код конвенция и се придържай към нея.
- Две от най-популярните:
- Oracle Code Conventions for the Java Programming Language
- Google Java Style Guide

ИНСТРУМЕНТИ ЗА СТАТИЧЕН КОД АНАЛИЗ

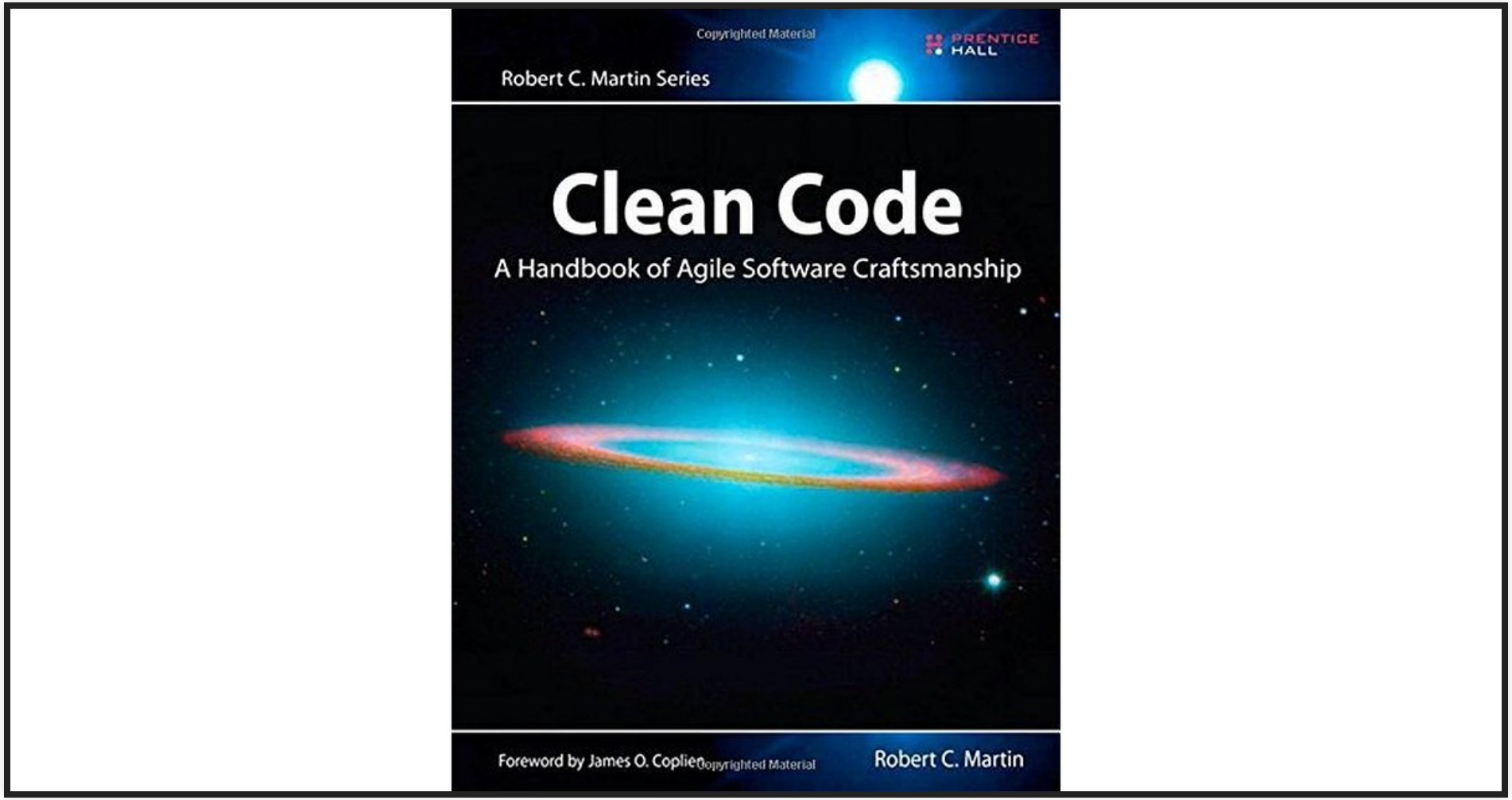
- Има инструменти за статичен код анализ, които
  - автоматизират придържането към код конвенции
  - намират и бъгове

ИНСТРУМЕНТИ ЗА СТАТИЧЕН КОД АНАЛИЗ

Някои от най-популярните open-source инструменти:

- checkstyle
- PMD
- FindBugs

БИБЛИЯТА


Copyrighted Material
PRENTICE HALL
Robert C. Martin Series
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Copyrighted Material
Robert C. Martin

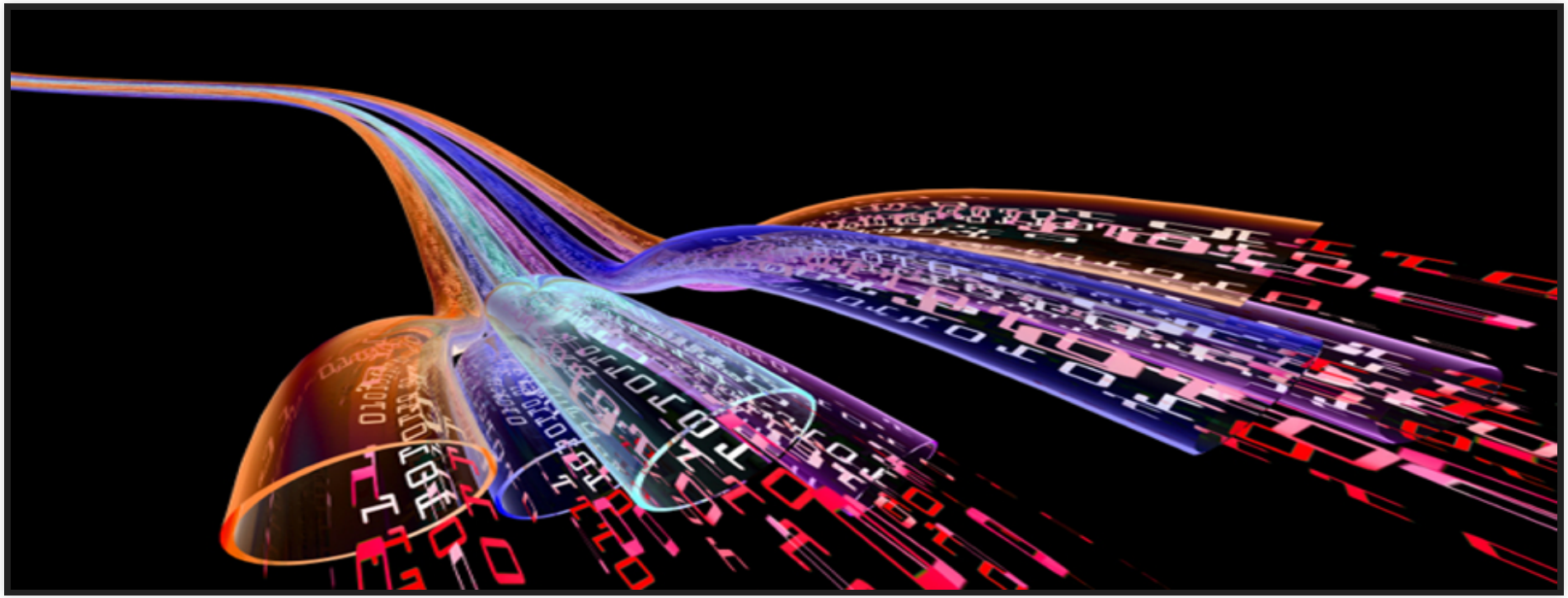
PLANETGEEK

Clean Code Cheat Sheet

ВЪПРОСИ

УПРАВЛЕНИЕ НА ВХОД И ИЗХОД

РАБОТА С ПОТОЦИ



ЩЕ РАЗГЛЕДАМЕ

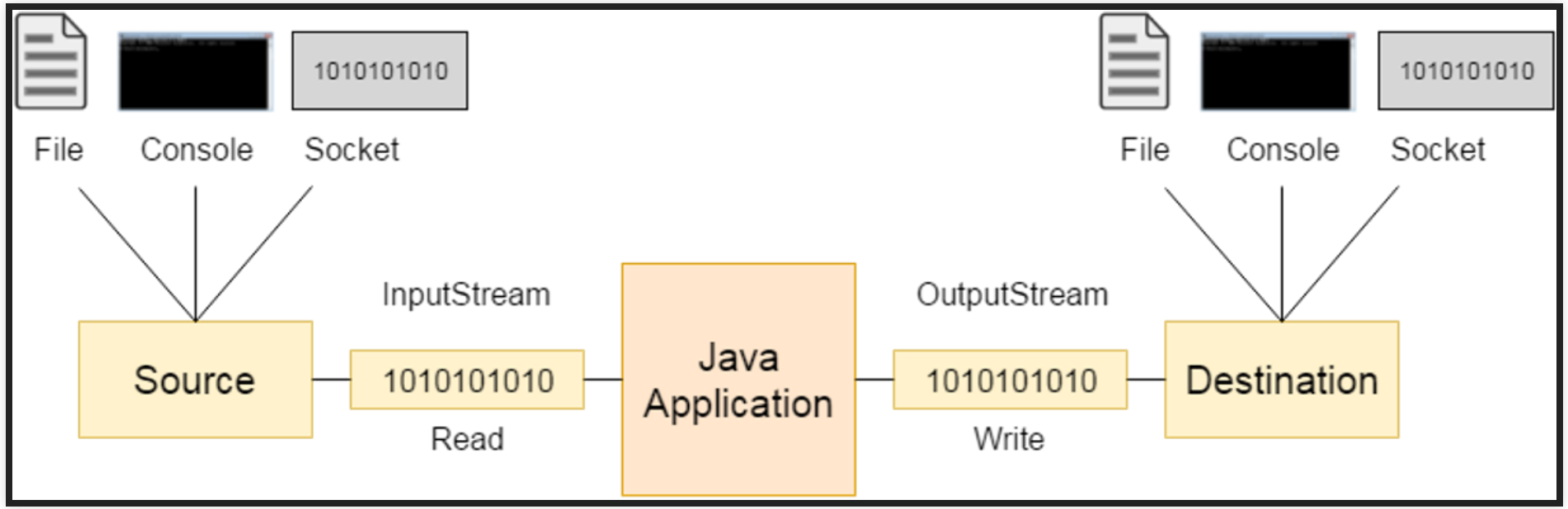
- Концепцията за поток
- Входно-изходните потоци в Java

КАКВО Е ПОТОЧНА ЛИНИЯ?

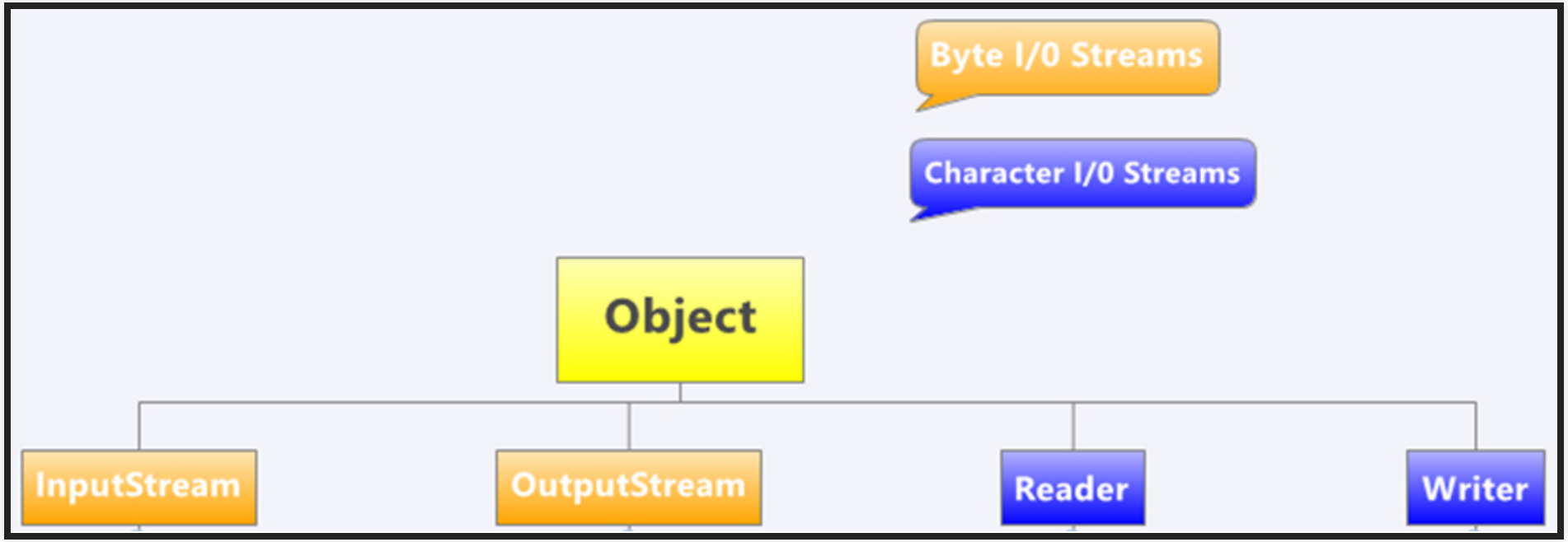
ВХОДНО-ИЗХОДНИ ПОТОЦИ

- Концепцията за вход-изход в Java се основава на *потоци* (*streams*)
- *Потокът* е абстракция за безкраен поток от данни
- Може да се четат данни от поток или да се пишат данни в поток
- В Java потоците може да се основават на байтове или на символи

ИЗТОЧНИК НА ДАННИ И ДЕСТИНАЦИЯ ЗА ДАННИ



java.io.*


Byte I/0 Streams
Character I/0 Streams
Object
InputStream
OutputStream
Reader
Writer

ВХОДНО-ИЗХОДНИ ПОТОЦИ

`InputStream`, `OutputStream`, `Reader` и `Writer` имат много наследници, създадени за различни цели:

- Достъп до файлове
- Достъп до мрежи
- Достъп до буфери в паметта
- Междунишкова комуникация (Pipes)

ВХОДНО-ИЗХОДНИ ПОТОЦИ

- Буфериране
- Филтриране
- Парсване
- Четене и писане на текст
- Четене и писане на примитивни данни
- Четене и писане на обекти

ВИДОВЕ ПОТОЦИ СПОРЕД НУЖДАТА

	Byte Based		Character Based	
	Input	**Output**	**Input**	**Output**
Basic	`InputStream`	`OutputStream`	`Reader` `InputStreamReader`	`Writer` `OutputStreamWriter`
Arrays	`ByteArrayInputStream`	`ByteArrayOutputStream`	`CharArrayReader`	`CharArrayWriter`
Files	`FileInputStream`	`FileOutputStream`	`FileReader`	`FileWriter`
Buffering	`BufferedInputStream`	`BufferedOutputStream`	`BufferedReader`	`BufferedWriter`
Filtering	`FilterInputStream`	`FilterOutputStream`	`FilterReader`	`FilterWriter`
Parsing	`PushbackInputStream` `StreamTokenizer`		`PushbackReader` `LineNumberReader`	
Strings			`StringReader`	`StringWriter`
Data	`DataInputStream`	`DataOutputStream`		
Data - Formatted		`PrintStream`		`PrintWriter`
Objects	`ObjectInputStream`	`ObjectOutputStream`		

ВХОДНО-ИЗХОДНИ ПОТОЦИ: ЖИЗНЕН ЦИКЪЛ

Потоците се

- Създават
- Използват
- Затварят

ПРИМЕР С `InputStream`

```
InputStream inputStream =
    new FileInputStream("c:\\data\\input-text.txt");

int data = inputStream.read();

while (data != -1) {
    doSomethingWithData(data);
    data = inputStream.read();
}

inputStream.close();
```

java.io.InputStream

```
int             available();
void            close();
void            mark(int readlimit);
boolean         markSupported();
abstract int    read();
int             read(byte[] b);
int             read(byte[] b, int off, int len);
byte[]          readAllBytes();
int             readNBytes(byte[] b, int off, int len);
void            reset();
long            skip(long n);
long            transferTo(OutputStream out);
```

ПРИМЕР С `OutputStream`

```
OutputStream os = new FileOutputStream("test.txt"));
os.write("Hello FMI!".getBytes());
os.flush();
os.close();
```

`FileOutputStream`: APPEND OR OVERWRITE?

```
// appends to file
OutputStream output =
    new FileOutputStream("c:\\data\\output-text.txt", true);

// overwrites file
OutputStream output =
    new FileOutputStream("c:\\data\\output-text.txt", false);
```

java.io.OutputStream

```
void           close();
void           flush();
void           write(byte[] b);
void           write(byte[] b, int off, int len);
abstract void write(int b);
```

ПРИМЕР С `DataInputStream`

```
DataInputStream input =
    new DataInputStream(new FileInputStream("binary.data"));

int    aByte   = input.read();
int    anInt   = input.readInt();
float  aFloat  = input.readFloat();
double aDouble = input.readDouble();
//etc.

input.close();
```

ПРИМЕР С BufferedReader

```
Reader reader = new FileReader("data/data.bin");

BufferedReader bufferedReader = new BufferedReader(reader)) {

String line = bufferedReader.readLine();

while (line != null) {
    line = bufferedReader.readLine();
}
```

ПРИМЕР С `ObjectOutputStream`

```
ObjectOutputStream objectOutputStream =
    new ObjectOutputStream(new FileOutputStream("object.data")

MyClass object = new MyClass();

output.writeObject(object);

output.close();
```

java.io.Serializable

```
import java.io.Serializable;

public class Person implements Serializable {

    private static final long serialVersionUID = 1234L;

    public String name = null;
    public int age = 0;
}
```

ИМА ЛИ ПРОБЛЕМ В ТОЗИ КОД?

```
InputStream input =
    new FileInputStream("c:\\data\\input-text.txt");
int data = input.read();

while (data != -1) {
    doSomethingWithData(data);
    data = input.read();
}

input.close();
```

А ТУК?

```
InputStream input = null;
try {
    input = new FileInputStream("c:\\data\\input-text.txt");
    int data = input.read();
    while (data != -1) {
        doSomethingWithData(data);
        data = input.read();
    }
} catch (IOException e) { //do something with e...
} finally {
    if (input != null) {
        input.close();
    }
}
```

TRY-WITH-RESOURCES

- `try` блок, който декларира един или повече ресурси и автоматично затваря всеки ресурс в края на блока
- Ресурс може да е обект от произволен клас, който имплементира интерфейса `java.lang.AutoCloseable` (което включва всички класове, които имплементират `java.io.Closeable`)

ПРИМЕР ЗА TRY-WITH-RESOURCES

```
static String readFirstLineFromFile(String path)
                                    throws IOException {
    try (BufferedReader br =
                 new BufferedReader(new FileReader(path))) {
        return br.readLine();
    }
}
```

- Каквото е декларирано в кръглите скоби след try, се `.close()`-ва автоматично при излизане от `try` блока

java.io.Reader

```
abstract void close();
void         mark(int readAheadLimit);
boolean      markSupported();
int          read();
int          read(char[] cbuf);
abstract int read(char[] cbuf, int off, int len);
int          read(CharBuffer target);
boolean      ready();
void         reset();
long         skip(long n);
```

java.io.Writer

```
Writer          append(char c);
Writer          append(CharSequence csq);
Writer          append(CharSequence csq, int start, int end);
abstract void   close();
abstract void   flush();
void            write(char[] cbuf);
abstract void   write(char[] cbuf, int off, int len);
void            write(int c);
void            write(String str);
void            write(String str, int off, int len);
```

ТРИТЕ СИСТЕМНИ ПОТОКА

- `System.in` (`InputStream`)
- `System.out` (`PrintStream`)
- `System.err` (`PrintStream`)

java.util.Scanner

```
Scanner scanner = new Scanner(System.in);
int i = scanner.nextInt();
String str = scanner.nextLine();
// ...
```

java.util.Formatter

```
StringBuilder sb = new StringBuilder();
// Send all output to the Appendable object sb
Formatter formatter = new Formatter(sb, Locale.US);

// Explicit argument indices may be used to re-order output.
formatter.format("%4$2s %3$2s %2$2s %1$2s", "a", "b", "c", "d"
// -> " d  c  b  a"

// Optional locale as the first argument can be used to get
// locale-specific formatting of numbers.  The precision and w
// can be given to round and align the value.
formatter.format(Locale.FRANCE, "e = %+10.4f", Math.E);
// -> "e =    +2,7183"
```

java.util.Formatter

```
// The '(' numeric flag may be used to format negative numbers
// with parentheses rather than a minus sign. Group separators
// are automatically inserted.
formatter.format("Amount diff since last statement: $ %(,.2f",
                 balanceDelta);
// -> "Amount diff since last statement: $ (6,217.58)"

// Writes a formatted string to System.out.
System.out.format("Local time: %tT", Calendar.getInstance());
// -> "Local time: 13:34:18"
// Writes formatted output to System.err.
System.err.printf("Unable to open file '%1$s': %2$s",
                  fileName, exception.getMessage());
// -> "Unable to open file 'food': No such file or directory"
```

ВЪПРОСИ