ЛЕКЦИЯ I

JAVA OOP: Text Files, Generics

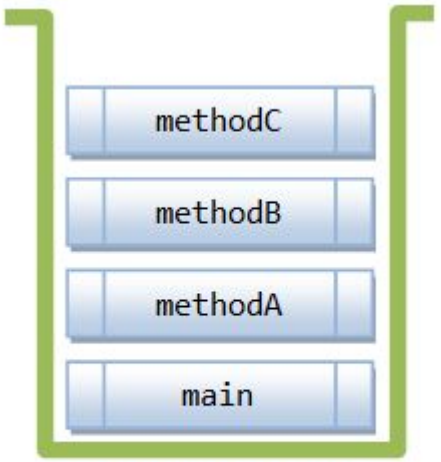
Теодор Костадинов

Съдържание

Try / Catch

Кодът хвърля грешки

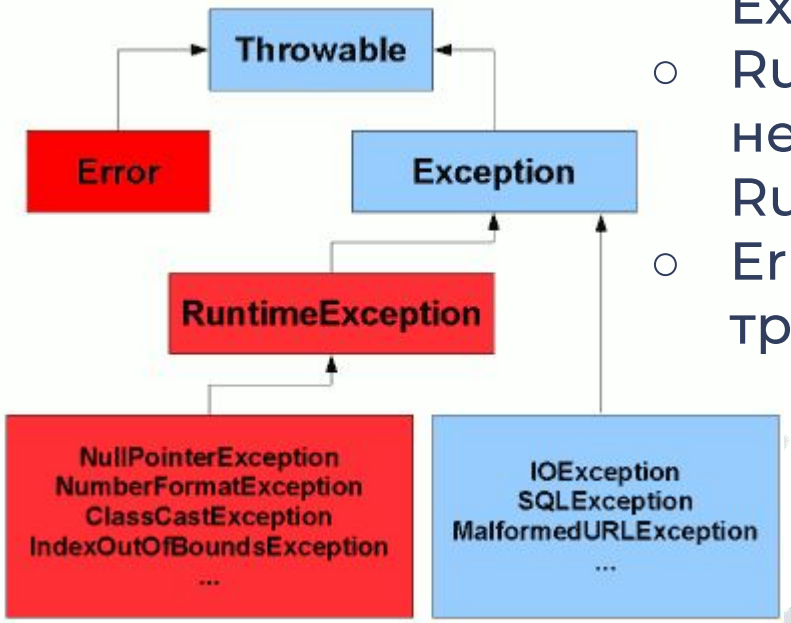
- Понякога кодът хвърля грешки
- Понякога това е нещо очаквано
- Когато код хвърли грешка, тя се предава по стекът на извикване на функциите, докато не стигне main метода
- Когато грешка стигне main метода, тя чупи цялата програма



Method Call Stack
(Last-in-First-out Queue)

Видове грешки

- Има грешки, които са очаквани, има и неочаквани
- Всички грешки наследяват родителският клас Throwable
- Под-класовете се делят на
  - Checked грешки - такива, които са очаквани и трябва да се предпазим от тях. Родителският клас е Exception.
  - Run-time грешки - такива, които се хвърлят неочаквано, от системата. Родителският клас е RuntimeException.
  - Error - анормално поведение на системата, което не трябва да се обработва от програмата.



Хващане на грешки

- Понякога хвърлянето на грешки е част от нормалната работа на програмата
- Тогава искаме да обработим грешките и да продължим изпълнението, а не да чупим програмата
- За целта се използва try / catch клаузата
- Кодът, който може да хвърли грешка се слага в try
- Кодът, който да се изпълни ако е хвърлена грешка се слага в catch

```
try {
    int budget = 1000;
    System.out.println("Success");
}
catch(Exception ex) {
    System.out.println(ex);
}
finally {
    System.out.println("This always runs");
}
```

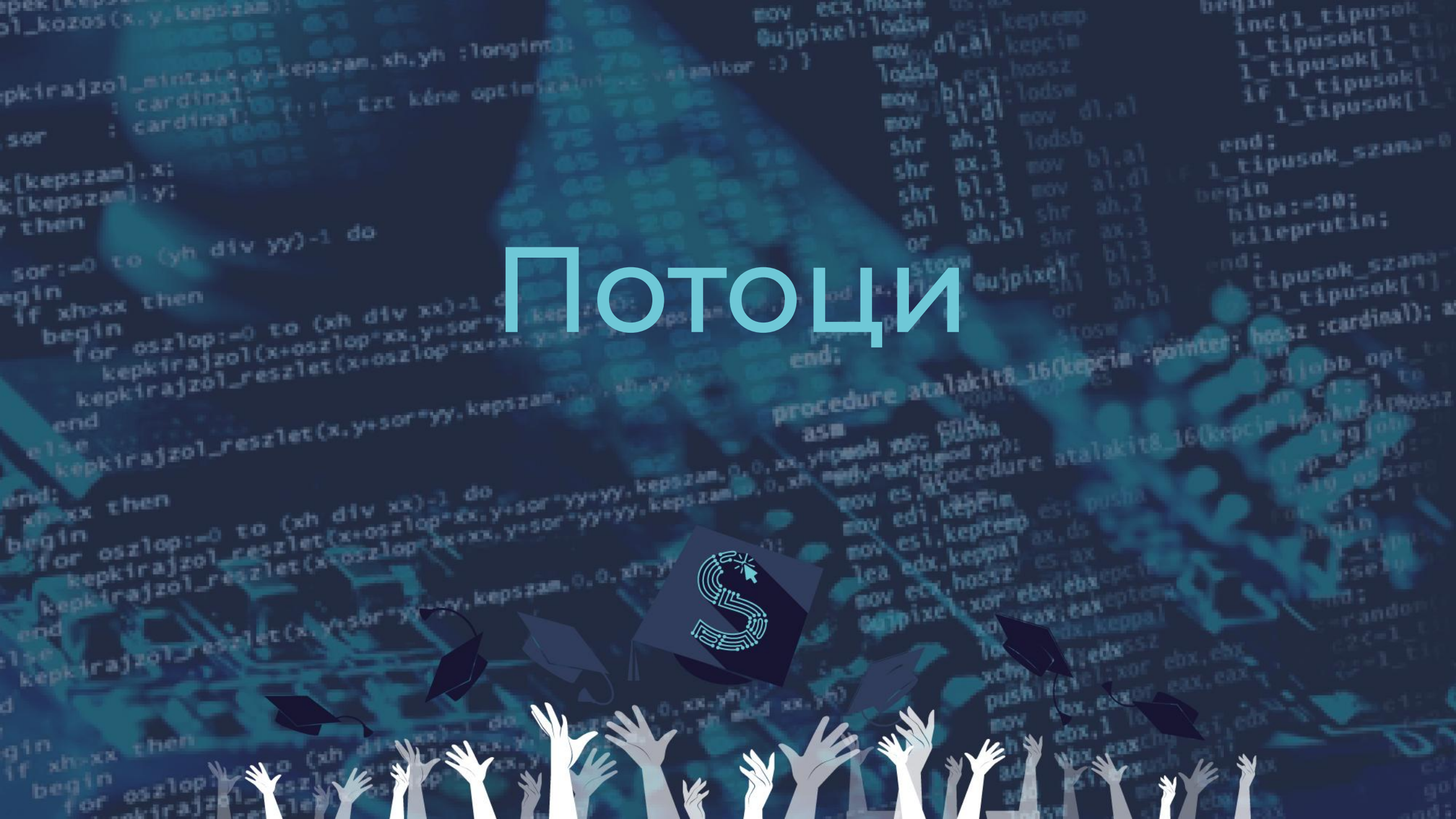
Хващане на грешки

- Catch клаузата изисква да упоменем коя точно грешка хващаме
- Можем да хващаме поредица от грешки, като навържем няколко catch клаузи последователно

- Finally - това е трети вид клауза, която можем да сложим след catch. Кодът в нея се изпълнява независимо дали е била хвърлена грешка или не.

```
try {
    int budget = 1000;
    System.out.println("Success");
}
catch(Exception ex) {
    System.out.println(ex);
}
finally {
    System.out.println("This always runs");
}
```

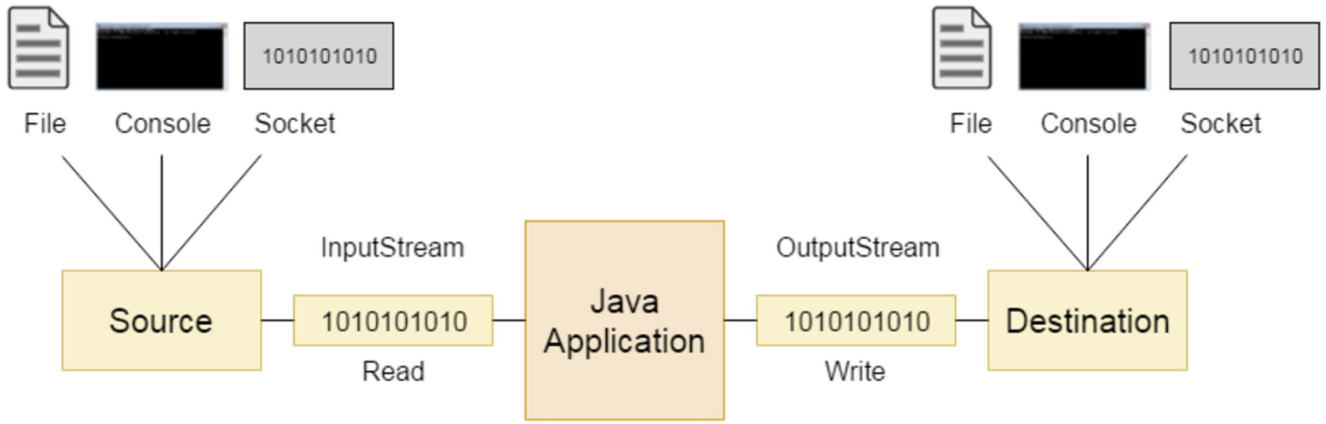
Потоци

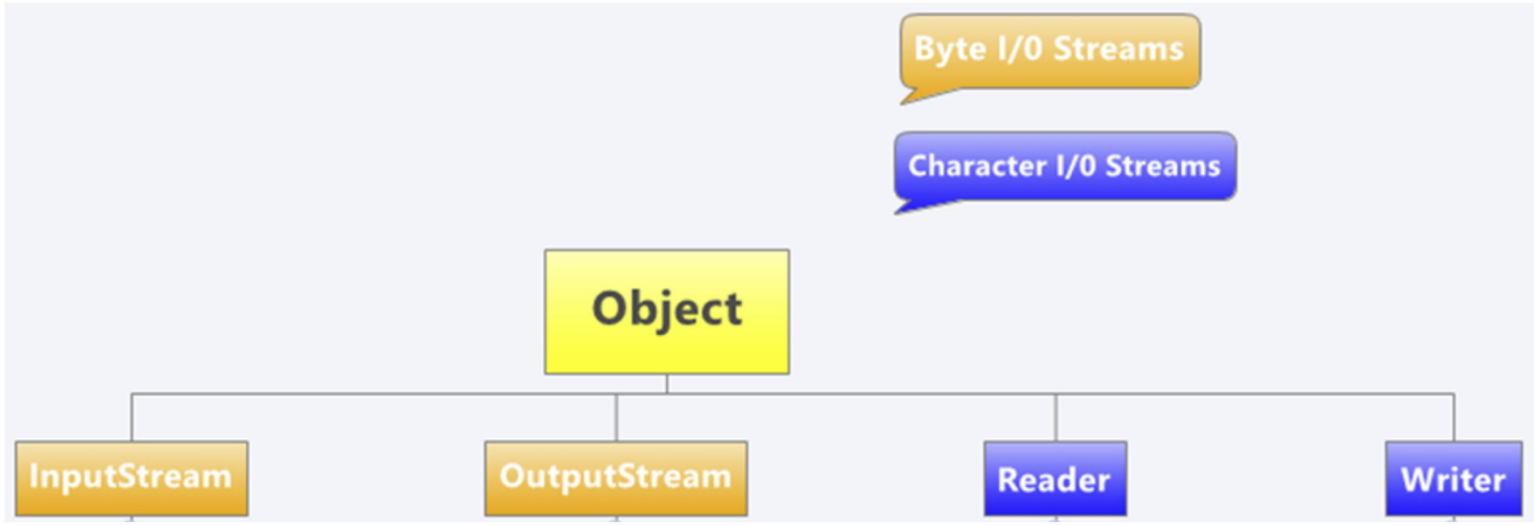


Потоци

- 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
- Имат подредба

Потоци


1010101010
File
Console
Socket
1010101010
File
Console
Socket
InputStream
OutputStream
Source
1010101010
Read
Java Application
1010101010
Write
Destination


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

Видове Потоци

- Тип на данните
  - Byte Streams (Input/Output Streams) - извършват операции със символи до 8 бита включително (1 байт)
  - Char Streams (Reader / Writer) - извършват операции със символи до 16 бита включително (2 байта)
- Тип на операцията
  - InputStream / Reader - за четене
  - OutputStream / Write - за писане

Потоци

Абстрактните класове InputStream, OutputStream, Reader и Writer

- се делят на такива за байтове (-Stream) и за символи (Reader & Writer)
- имат много наследници, създадени за различни цели:
  - Достъп до файлове
  - Достъп до мрежи
  - Достъп до буфери в паметта
  - Междунишкова комуникация (Pipes)
- Могат да извършват:
  - Буфериране
  - Филтриране
  - Парсване
  - Четене и писане на текст
  - Четене и писане на примитивни типове данни
  - Четене и писане на обекти

Видове потоци

	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`		

Видове потоци

SOFTWARE ACADEMY
Educational Center Net It

- FileStreams - за писане и четене от файлове
- DataInputStream - за писане и четене на променливи
- ObjectOutputStream - за писане и четене на обекти

Жизнен цикъл

- Създават
  - Създават се от файл, данни или друг тип входно или изходно устройство
  - Един поток може да приеме друг поток в конструктура си
  - Това се казва wrap-ване (опаковане). Така може да добавяме специални свойства към потока.

new **BufferedReader**(new **CharArrayReader**(“Pesho”.toByteArray()));

- Използват
- Затварят
  - Това става като се извика .close()

Четене

- Използва се метода read
- Всеки поток си има “глава” - кой е следващият знак за четене
- При прочитане на текущият знак, главата се мести на следващият

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

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

inputStream.close();
```

Писане

- Използва се метода write
- Можем да дадем true/false като втори параметър
  - True - добавя към файла
  - False - презаписва файла
- Не трябва да забравяме да затворим потока.

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

Почивка

до 20:30

SOFTWARE ACADEMY
Educational Center Net It

- Напишете програма, която чете списък от имена от един текстов файл, сортира ги по азбучен ред и ги запазва в друг файл. Имената да са с латински букви. На всеки ред от файла, където са записани имената, има точно по едно име. На всеки ред от файла с резултата също трябва да има само по едно име.

Ами сега?

Задача за упражнение

Грешки

- Използването на потоци може да хвърли IOException.
- Трябва да се обгражда в try/catch блок.
- Трябва задължително да се затваря потока, дори да е хвърлил грешка.
- Това може да става с finally
- Това може да става и със специален try - try with resource
- Каквото е дефинирано в скобите на try-а ще бъде автоматично затворено

```
try (BufferedReader br =
                   new BufferedReader(new FileReader(path))) {
        return br.readLine();
    }
```

Сравняване

	Time to read data file (in milliseconds)						
Reading Method	**1KB**	**10KB**	**100KB**	**1MB**	**10MB**	**100MB**	**1GB**
FileReader.read()	3	9	29	95	512	4,279	43,635
BufferedFileReader.readLine()	1	2	8	30	81	492	4,498
FileInputStream.read()	2	13	133	1,247	12,603	124,413	1,261,190
BufferedInputStream.read()	0	1	6	24	122	1,138	24,643
Files.readAllBytes()	3	3	4	4	15	102	969
Files.readAllLines()	5	6	12	39	120	866	OutOfMemoryError
Files.lines()	26	31	35	59	112	465	3,588
Scanner.nextLine()	6	15	38	107	376	2,346	21,539

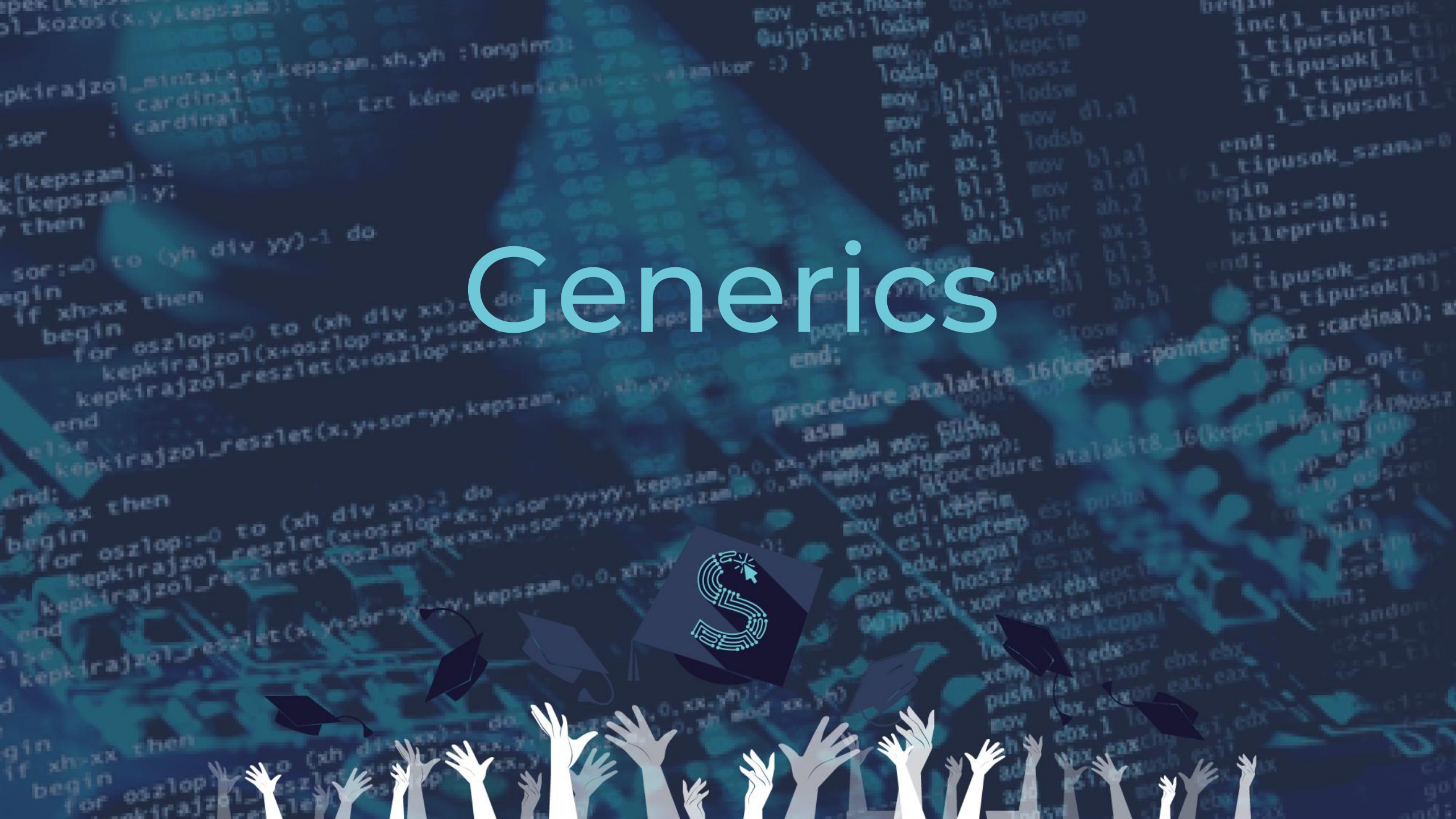
Файлове

Класът Path

- Класът Path служи за представяне на път
- Обозначава файл или директория
- Файлът или директорията може физически да съществуват или не
- Инстанции се създават чрез статични методи на java.nio.file.Paths, а от Java 11 - и на java.nio.file.Path
- В зависимост каква е ОС, разделителите на директориите са различни
  - В UNIX, Linux и MacOS е forward slash: /
  - В Windows е back slash: \
  - Може да се вземе в Java код чрез File.separator или FileSystem.getSeparator()

```
Path pathToUserHomeDir = Path.of("C:\\Users\\joe");                    // C:\Users\joe
Path pathToAFile = Path.of("C:\\Users\\joe\\orders.txt");    // C:\Users\joe\orders.txt
Path relPathToAFile = Path.of("Users", "joe", "orders.txt");  // \Users\joe\orders.txt
Path linuxPathToAFile = Paths.get("/home", "joe", "file.txt");   // /home/joe/file.txt
Path linuxRelativePath = Paths.get("documents", "FileIO.odp");  // documents/FileIO.odp
```

Generics



Какво е?

- Generics терминът се използва, когато в клас, метод или интерфейс използваме ТИП НА ОБЕКТ като параметър.
- Чрез създаването на Generic класове, можем да използваме типове (други класове или интерфейси) като параметри на Generic класа. Това позволява един и същи код да бъде изпълняван върху различни типове обекти.

```
Пример за такъв клас: ArrayList
ArrayList<String>, ArrayList<Cat>, ArrayList<DatabaseRecord>,
ArrayList<ArrayList<ArrayList<String>>>
```

Защо е?

- Проверка на типа по време на компилирането предотвратява грешки по време на изпълнението
- Премахва нуждата от кастване. Представете си, ако ArrayList приемаше само обекти:
- List list = new ArrayList();
- list.add("hello");
- String s = (String) list.get(0);
- Позволява имплементирането на generic алгоритми, които работят върху различни типове и са типово-безопасни

```
Пример за такъв клас: ArrayList
ArrayList<String>, ArrayList<Cat>, ArrayList<DatabaseRecord>,
ArrayList<ArrayList<ArrayList<String>>>
```

Как?

SOFTWARE ACADEMY
Educational Center Net It

Без Generic:

```
public class Box {
    private Object object;

    public void set(Object object) {
       this.object = object;
    }

    public Object get() { return object; }
}
```

Как?

SOFTWARE ACADEMY
Educational Center Net It

```
C Generic:
public class Box<T> {
    // T stands for "Type"
    private T t;

    public void set(T t) { this.t = t; }
    public T get() { return t; }
}
```

Generics с повече типове

- Всеки нов тип му се измисля буква.
- Изреждат се със запетайки

```
public interface Pair<K, V> {
    public K getKey();
    public V getValue();
}
```

```
public class OrderedPair<K, V> implements Pair<K, V> {
    private K key;
    private V value;

    public OrderedPair(K key, V value) {
        this.key = key;
        this.value = value;
    }
    public K getKey()  { return key; }
    public V getValue() { return value; }
}
```

Generics без тип

Raw тип се получава когато извикаме generic клас без да указваме конкретен тип за параметър

```
Box rawBox = new Box();
```

В този случай когато извикваме put и get ще работим с типа Object. Пробвайте да използвате ArrayList и вашият List без да конкретизирате типа на данните.

Generic методи

```
public static <K, V> boolean compare(Pair<K, V> p1, Pair<K, V> p2) {
        return p1.getKey().equals(p2.getKey()) &&  p1.getValue().equals(p2.getValue());
 }


Pair<Integer, String> p1 = new Pair<>(1, "apple");
Pair<Integer, String> p2 = new Pair<>(2, "pear");
boolean same = Util.<Integer, String>compare(p1, p2);
```

Ограничаване на Generics

- За да ограничите какви типове могат да бъдат задавани на вашите класове и методи, специфицирайте ги чрез интерфейси и наследяване.

```
public class Box<T extends Number> {
    private T t;

    public void set(T t) {
        this.t = t;
    }
    public T get() {
        return t;
    }
}
```

ВЪПРОСИ?

Резюме

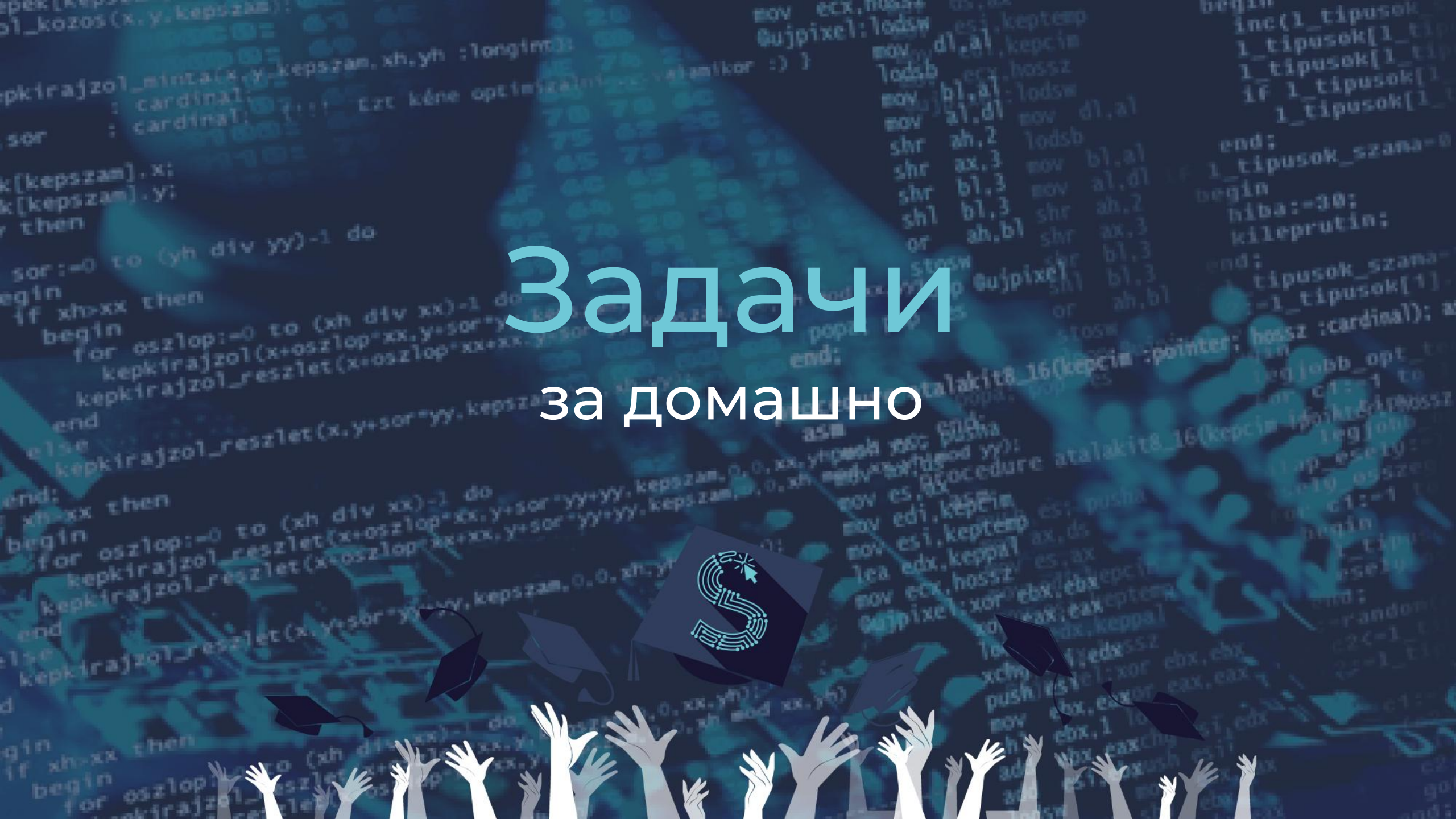
- Има разнообразни класове за четене и писане на файлове.
- Според случая, трябва да избираме най-подходящият.

Ресурси

- Docs
- GitHub Repo with Demos
-

Задачи

за домашно



Задача 1

Напишете програма, която заменя най-често срещаната дума в текстов файл с *******, броя звездички отговаря на дължината на думата.

Ами сега?

Задача за упражнение

Задача 2

Напишете проста програма, която брои колко пъти се среща дума в даден текстов файл (за дума считаме всеки подниз от текста). В примера, нека текстът изглежда така:

This is our "Intro to Programming in Java" book.
In it you will learn the basics of Java programming.
You will find out how nice Java is.

Ами сега?

Задача за упражнение

Задача 3

Напишете generic клас, който да симулира записване в база данни. Класът да работи само с типове, които имат метод getId() и getValue(). Класът да има метод addToTable, който приема обекти от съответния тип и ги записва в подходяща структура. Да има и метод writeToDatabase, който да принтира ид и стойност от всички добавени обекти.

```
public static void main( String args[] ) {
    Database<User> db = new Database<>();
      User u1 = new User(1, “pesho”);
      User u2 = new User(2, “nepesho”);
      db.addToTable(u1);
      db.addToTable(u2);
      db.writeToDatabase();
}
```

Ами сега?

Задача за упражнение

БЛАГОДАРЯ
ЗА ВНИМАНИЕТО!