

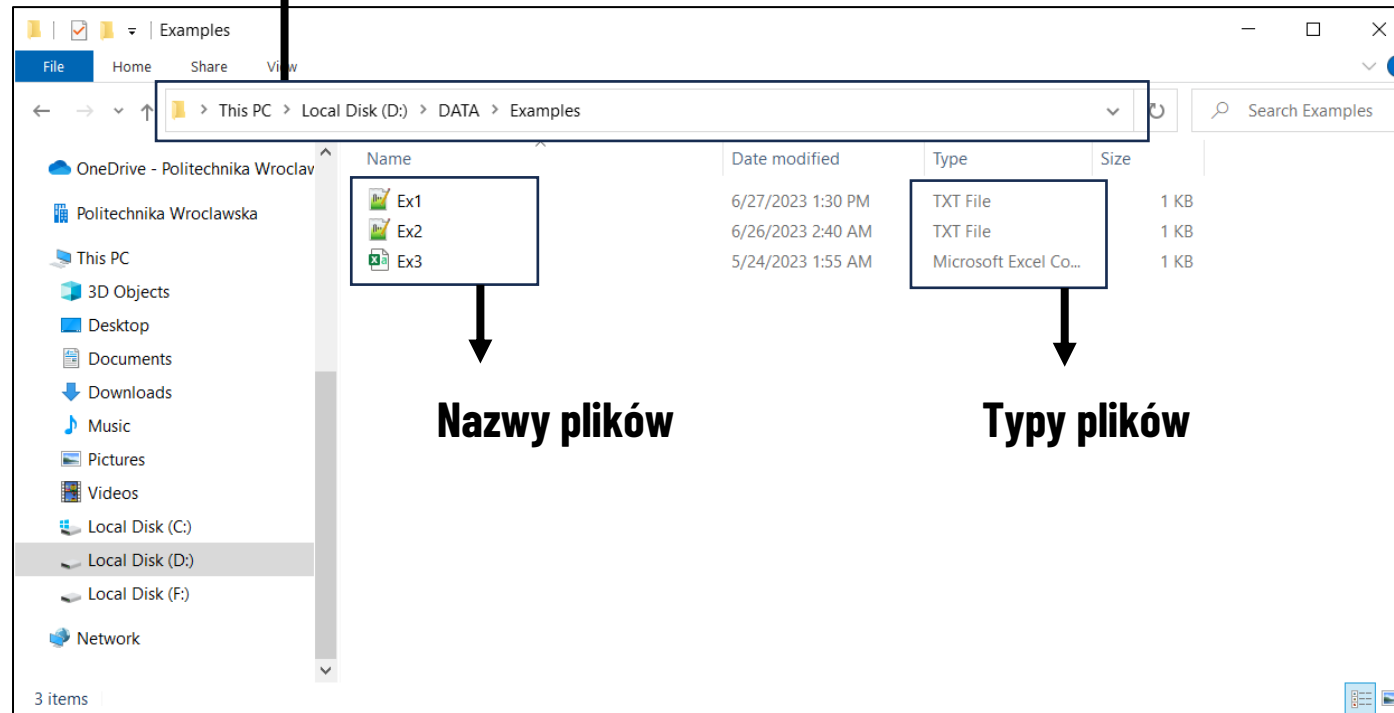
# PRZETWARZANIE DANYCH WEJŚCIOWYCH I WYJŚCIOWYCH

- Ścieżki do plików
- Odczyt z pliku, zapis do pliku
- Formaty JSON i XML

# ŚCIEŻKI DO PLIKÓW

Ścieżka do folderu

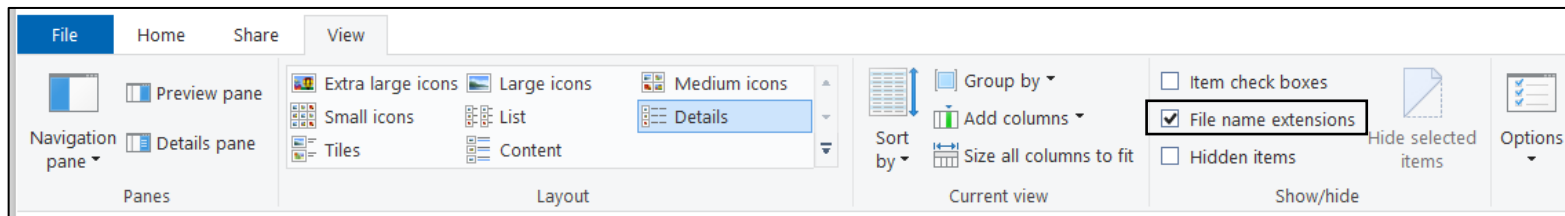
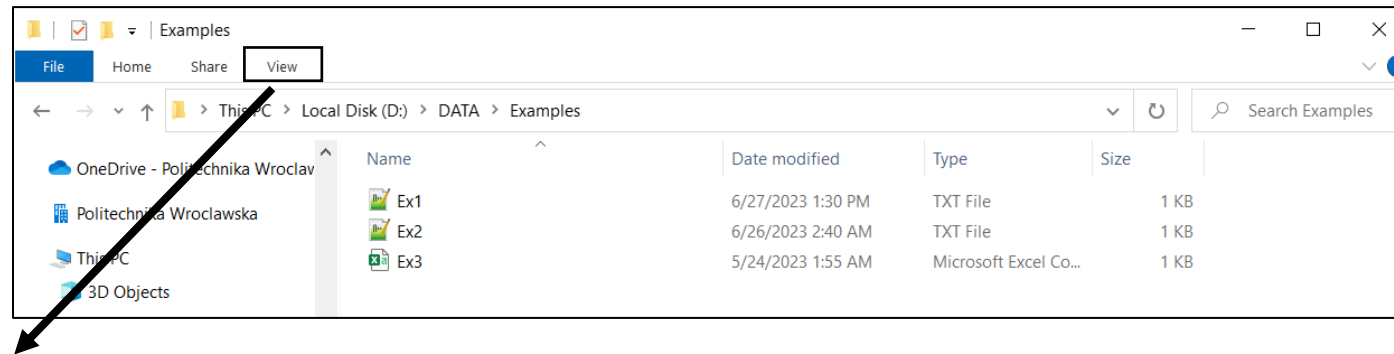
Eksplorator plików Windows\*



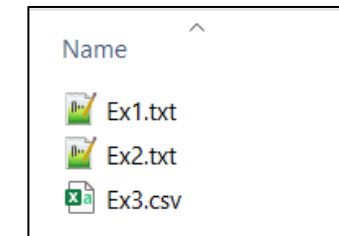
\* Polecane rozwiązanie zastępujące wbudowany eksplorator plików:  
Double Commander (darmowy, open source)

# ŚCIEŻKI DO PLIKÓW

Eksplorator plików Windows\*



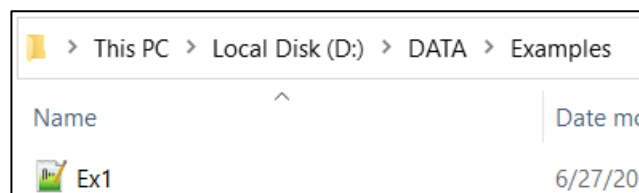
**Nazwy plików z rozszerzeniami**



\* Polecane rozwiązanie zastępujące wbudowany eksplorator plików:  
Double Commander (darmowy, open source)

# ŚCIEŻKI DO PLIKÓW

- *ścieżka* (ang. path) to ciąg znaków określający jednoznacznie lokalizację danego obiektu w strukturze katalogów na dysku twardym lub innym nośniku danych
- elementy składowe ścieżki określają miejsce danego obiektu w drzewie *katalogów* (ang. directory)

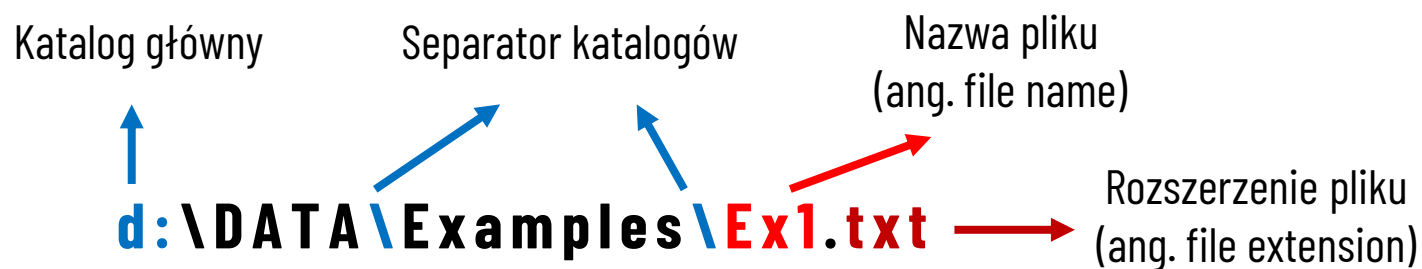


```
C:\Users\akazimierska>tree d: /f
Folder PATH listing
Volume serial number is 00000085 EE5C:6E50
D:\
  KERNEL.SYS
  LICENSE.TXT
  README.TXT
  SRC
    COMMANDS.ZIP
    freedos.tag
    KE2026AS.ZIP
  DATA
    Examples
      Ex1.txt
      Ex2.txt
      Ex3.csv
```

**Ścieżka:**  
**d:\DATA\Examples\Ex1.txt**

# ŚCIEŻKI DO PLIKÓW

- *ścieżka bezwzględna* (ang. absolute path) uwzględnia wszystkie katalogi od katalogu głównego (np. dysku)



- *ścieżka względna* (ang. relative path) opisuje lokalizację obiektu względem aktualnego katalogu roboczego (np. katalogu, z którego uruchamiany jest program)

Obecny katalog      ← **.\Examples\Ex1.txt**

# ŚCIEŻKI DO PLIKÓW A SYSTEM OPERACYJNY

System	Windows	“Unix-like”, w tym Linux i macOS
Katalog główny	<ul style="list-style-type: none"><li>dysk może być podzielony na partycje, z których każda ma swój własny katalog główny (C:, D:)</li><li>katalog główny zwykle nazywany dyskiem (ang. drive)</li></ul>	<ul style="list-style-type: none"><li>jeden katalog główny (/)</li><li>katalog główny nazywany root</li></ul>
Separator katalogów	backslash \	(forward) slash /
Nazwy plików i folderów	<ul style="list-style-type: none"><li>wielkość liter nie jest uwzględniana</li><li>spacje są dozwolone</li></ul>	<ul style="list-style-type: none"><li>wielkość liter jest uwzględniana,</li><li>spacje są dozwolone, ale mogą powodować problemy</li></ul>
	C:\user\docs\test.txt	/home/user/docs/test.txt

# KLASA FILE (java.io)

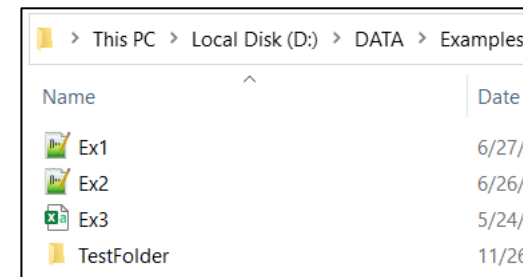
- służy do tworzenia obiektowej reprezentacji ścieżki do pliku lub katalogu i podstawowych interakcji z systemem plików

Najważniejsze metody:

- *exists()*, *isFile()*, *isDirectory()*
- *length()*
- *list()*, *listFiles()*
- *renameTo()*, *delete()*
- *createNewFile()*, *mkdir()*

# KLASA FILE (java.io)

- służy do tworzenia obiektowej reprezentacji ścieżki do pliku lub katalogu i podstawowych interakcji z systemem plików



Najważniejsze metody:

- `exists()`, `isFile()`, `isDirectory()`
- `length()`
- `list()`, `listFiles()`
- `renameTo()`, `delete()`
- `createNewFile()`, `mkdir()`

```
String myFileStr = "d:\\DATA\\Examples\\Ex1.txt";  
File myFile = new File(myFileStr);  
  
boolean fileExists = myFile.exists();  
System.out.println("File exists? " + fileExists);  
long fileSize = myFile.length();  
System.out.println("Length in bytes: " + fileSize);
```

```
File exists? true  
Length in bytes: 75
```

```
File myFile2 = new File("d:\\DATA\\Examples\\Ex4.txt");  
System.out.println(myFile2.exists());
```

```
false
```



# KLASA FILE (java.io)

```
File myFile = new File("d:\\DATA\\Examples\\Ex1.txt");  
System.out.println("Is file? " + myFile.isFile());  
System.out.println("Is dir? " + myFile.isDirectory());
```

```
Is file? true  
Is directory? false
```

```
File myDir = new File("d:\\DATA\\Examples");  
System.out.println("Is file? " + myDir.isFile());  
System.out.println("Is dir? " + myDir.isDirectory());
```

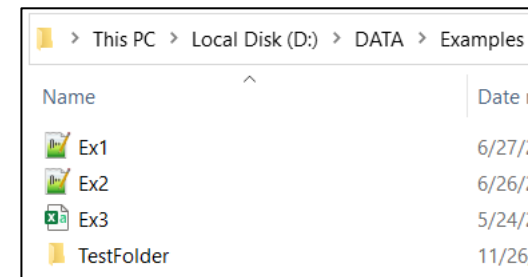
```
Is file? false  
Is directory? true
```

```
String[] dirContentsStr = myDir.list();  
System.out.println(Arrays.toString(dirContentsStr));
```

```
[Ex1.txt, Ex2.txt, Ex3.csv, TestFolder]
```

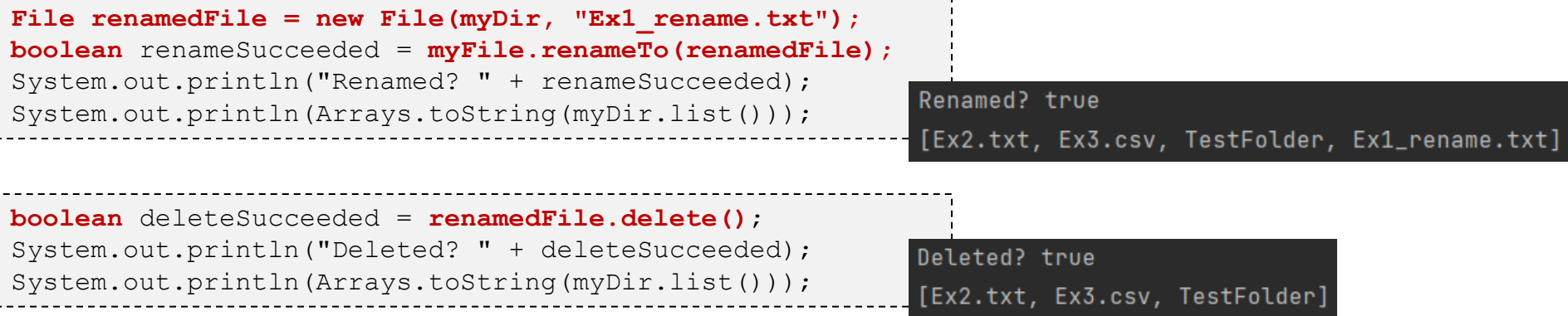
```
File[] dirContentsFile = myDir.listFiles();  
System.out.println(Arrays.toString(dirContentsFile));
```

```
[d:\\DATA\\Examples\\Ex1.txt, d:\\DATA\\Examples\\Ex2.txt,  
d:\\DATA\\Examples\\Ex3.csv, d:\\DATA\\Examples\\TestFolder]
```



# KLASA FILE (java.io)

new File (File parent, String s) lub new File (String parent, String s)



```
File renamedFile = new File(myDir, "Ex1_rename.txt");
boolean renameSucceeded = myFile.renameTo(renamedFile);
System.out.println("Renamed? " + renameSucceeded);
System.out.println(Arrays.toString(myDir.list()));
```

```
Renamed? true
[Ex2.txt, Ex3.csv, TestFolder, Ex1_rename.txt]
```

```
boolean deleteSucceeded = renamedFile.delete();
System.out.println("Deleted? " + deleteSucceeded);
System.out.println(Arrays.toString(myDir.list()));
```

```
Deleted? true
[Ex2.txt, Ex3.csv, TestFolder]
```

# KLASA FILE (java.io)

```
File renamedFile = new File(myDir, "Ex1_rename.txt");  
boolean renameSucceeded = myFile.renameTo(renamedFile);  
System.out.println("Renamed? " + renameSucceeded);  
System.out.println(Arrays.toString(myDir.list()));
```

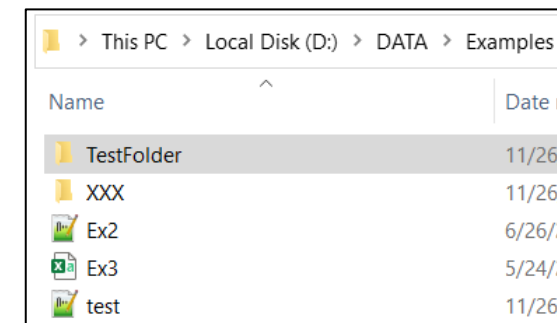
```
Renamed? true  
[Ex2.txt, Ex3.csv, TestFolder, Ex1_rename.txt]
```

```
boolean deleteSucceeded = renamedFile.delete();  
System.out.println("Deleted? " + deleteSucceeded);  
System.out.println(Arrays.toString(myDir.list()));
```

```
Deleted? true  
[Ex2.txt, Ex3.csv, TestFolder]
```

```
File newFile = new File(myDir, "test.txt");  
boolean createFileSucceeded = newFile.createNewFile();  
  
File newDir = new File(myDir, "XXX");  
boolean createDirSucceeded = newDir.mkdir();  
  
System.out.println(Arrays.toString(myDir.list()));
```

```
[Ex2.txt, Ex3.csv, TestFolder,  
test.txt, XXX]
```



# KLASA FILES (java.nio.file)

- zapewnia rozszerzone funkcjonalności do interakcji z systemem plików
- tylko metody statyczne

Najważniejsze metody:

- *exists()*, *isRegularFile()*, *isDirectory()*
- *size()*
- *list()*, *walk()*
- *createFile()*, *createDirectory()*
- *move()*, *copy()*, *delete()*
- *readAllLines()*, *write()*

Paths.get (String s)

```
Path myFilePath =  
    Paths.get("d:\\DATA\\Examples\\Ex1.txt");  
  
boolean fileExists = Files.exists(myFilePath);  
System.out.println("File exists? " + fileExists);  
long fileSize = Files.size(myFilePath);  
System.out.println("Length in bytes: " + fileSize);
```

```
File exists? true  
Length in bytes: 75
```

```
Path myDirPath = Paths.get("d:\\DATA\\Examples");  
  
System.out.println("Is file? "  
    + Files.isRegularFile(myDirPath));  
System.out.println("Is dir? "  
    + Files.isDirectory(myDirPath));
```

```
Is file? false  
Is dir? true
```

# KLASA FILES (java.nio.file)

```
Path myDirPath = Paths.get("d:\\DATA\\Examples");  
  
Stream<Path> allContents = Files.list(myDirPath);  
allContents.forEach(System.out::println);
```

```
d:\DATA\Examples\Ex2.txt  
d:\DATA\Examples\Ex3.csv  
d:\DATA\Examples\TestFolder  
d:\DATA\Examples\Ex1.txt
```

```
Stream<Path> allContents2 = Files.walk(myDirPath);  
allContents2.forEach(System.out::println);
```

```
d:\DATA\Examples  
d:\DATA\Examples\Ex2.txt  
d:\DATA\Examples\Ex3.csv  
d:\DATA\Examples\TestFolder  
d:\DATA\Examples\TestFolder\test_file1.txt  
d:\DATA\Examples\TestFolder\test_file2.txt  
d:\DATA\Examples\Ex1.txt
```

# KLASA FILES (java.nio.file)

```
Path myDirPath = Paths.get("d:\\DATA\\Examples");  
  
Stream<Path> allContents = Files.list(myDirPath);  
allContents.forEach(System.out::println);
```

```
d:\DATA\Examples\Ex2.txt  
d:\DATA\Examples\Ex3.csv  
d:\DATA\Examples\TestFolder  
d:\DATA\Examples\Ex1.txt
```

```
Stream<Path> allContents2 = Files.walk(myDirPath);  
allContents2.forEach(System.out::println);
```


```
d:\DATA\Examples  
d:\DATA\Examples\Ex2.txt  
d:\DATA\Examples\Ex3.csv  
d:\DATA\Examples\TestFolder  
d:\DATA\Examples\TestFolder\test_file1.txt  
d:\DATA\Examples\TestFolder\test_file2.txt  
d:\DATA\Examples\Ex1.txt
```

```
C:\Users\akazimierska>tree d: /f  
Folder PATH listing  
Volume serial number is 00000041 EE5C:6E50  
D:\  
|   KERNEL.SYS  
|   LICENSE.TXT  
|   README.TXT  
|  
|---SRC  
|   COMMANDS.ZIP  
|   freedos.tag  
|   KE2026AS.ZIP  
|  
|---DATA  
|   |  
|   |---Examples  
|   |   |   Ex2.txt  
|   |   |   Ex3.csv  
|   |   |   Ex1.txt  
|   |   |  
|   |   |---TestFolder  
|   |       |   test_file1.txt  
|   |       |   test_file2.txt
```

**walk()** (red box around Examples and TestFolder)  
**list()** (red box around Ex2.txt, Ex3.csv, Ex1.txt)

# KLASA FILES (java.nio.file)

Paths.get (String s, ..., String z)



```
Path renamedFilePath  
    = Paths.get("d:\\DATA\\Examples", "Ex1_renamed.txt");  
Path filePath = Files.move(myFilePath, renamedFilePath);  
Files.list(myDirPath).forEach(System.out::println);
```

```
d:\DATA\Examples\Ex2.txt  
d:\DATA\Examples\Ex3.csv  
d:\DATA\Examples\TestFolder  
d:\DATA\Examples\Ex1_renamed.txt
```

```
Files.delete(renamedFilePath);  
Files.list(myDirPath).forEach(System.out::println);
```

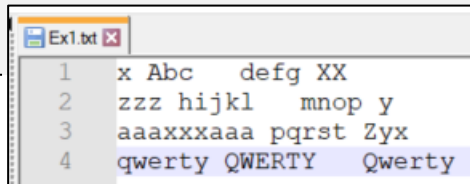
```
d:\DATA\Examples\Ex2.txt  
d:\DATA\Examples\Ex3.csv  
d:\DATA\Examples\TestFolder
```

```
Path newFilePath  
    = Paths.get("d:\\DATA\\Examples", "test.txt");  
Path filePath = Files.createFile(newFilePath);  
  
Path newDirPath = Paths.get("d:\\DATA\\Examples", "XXX");  
Path dirPath = Files.createDirectory(newDirPath);  
  
Files.list(myDirPath).forEach(System.out::println);
```

```
d:\DATA\Examples\Ex2.txt  
d:\DATA\Examples\Ex3.csv  
d:\DATA\Examples\TestFolder  
d:\DATA\Examples\test.txt  
d:\DATA\Examples\XXX
```

# ODCZYT PLIKÓW TEKSTOWYCH

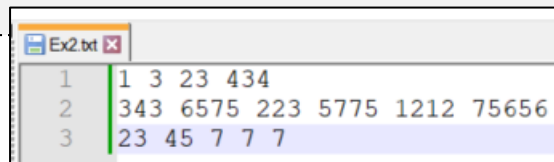
```
File myFile = new File("d:\\DATA\\Examples\\Ex1.txt");
Scanner sc1 = new Scanner(myFile);
while (sc1.hasNextLine()) {
    System.out.println(sc1.nextLine());
}
sc1.close();
```



```
1 x Abc   defg XX
2 zzz hijkl  mnop y
3 aaaxxxaaa pqrst Zyx
4 qwerty QWERTY Qwerty
```

```
x Abc   defg XX
zzz hijkl  mnop y
aaaxxxaaa pqrst Zyx
qwerty QWERTY Qwerty
```

```
Path myPath = Paths.get("d:\\DATA\\Examples\\Ex2.txt");
Scanner sc2 = new Scanner(myPath);
List<Integer> myNumbers = new ArrayList<>();
while (sc2.hasNextInt()) {
    myNumbers.add(sc2.nextInt());
}
System.out.println(myNumbers);
sc2.close();
```



```
1 1 3 23 434
2 343 6575 223 5775 1212 75656
3 23 45 7 7 7
```

```
[1, 3, 23, 434, 343, 6575, 223, 5775, 1212, 75656, 23, 45, 7, 7, 7]
```

## KLASA SCANNER

- zamiast standardowego wejścia (*System.in*) może przyjąć obiekt klas *File* lub *Path*
- posiada metody do odczytu podstawowych typów danych rozdzielanych dowolnymi białymi znakami (*next()*, *nextInt()*, itd.) lub kolejnych linii (*nextLine()*)
- metody *hasNext()*, *hasNextInt()* itd. pozwalają na zabezpieczenie przed wyjątkiem *NoSuchElementException*



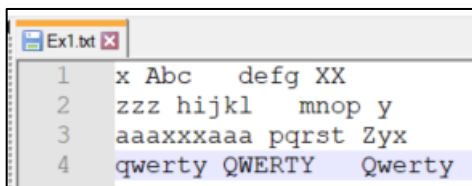
# ODCZYT PLIKÓW TEKSTOWYCH

new FileReader (File f) lub new FileReader (String s)

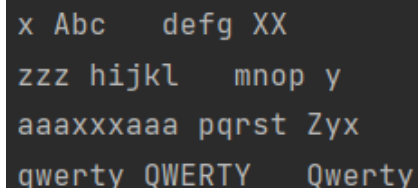
```
File myFile = new File("d:\\DATA\\Examples\\Ex1.txt");
FileReader fr = new FileReader(myFile);
BufferedReader br = new BufferedReader(fr);

String line = br.readLine();
while (line != null) {
    System.out.println(line);
    line = br.readLine();
}
br.close();
```

```
String line;
while ((line = br.readLine()) != null) {
    System.out.println(line);
}
```



```
1 x Abc   defg XX
2 zzz hijkl  mnop y
3 aaaxxxaaa pqrst Zyx
4 qwerty QWERTY  Qwerty
```



```
x Abc   defg XX
zzz hijkl  mnop y
aaaxxxaaa pqrst Zyx
qwerty QWERTY  Qwerty
```

## KLASA BUFFEREDREADER

- służy do wydajnego odczytu ciągu znaków z wykorzystaniem bufora o określonej wielkości
- przyjmuje obiekt klasy Reader (lub klasy pochodnej, np. FileReader)
- najczęściej służy do odczytu pliku linia po linii za pomocą metody *readLine()*

# ODCZYT PLIKÓW TEKSTOWYCH A OBSŁUGA WYJĄTKÓW

```
public static void main(String[] args) {  
  
    File myFile  
        = new File("d:\\DATA\\Examples\\Ex1.txt");  
    FileReader fr = new FileReader(myFile);  
    BufferedReader br = new BufferedReader(fr);  
  
    String line;  
    while ((line = br.readLine()) != null) {  
        System.out.println(line);  
    }  
    br.close();  
}
```

Unhandled exception: java.io.FileNotFoundException

Unhandled exception: java.io.IOException

Problem:  
Dlaczego ten kod się nie kompiluje?

# ODCZYT PLIKÓW TEKSTOWYCH A OBSŁUGA WYJĄTKÓW

## Rozwiązanie 1: Dodanie wyjątku do sygnatury metody

```
public static void main(String[] args) throws IOException {  
  
    File myFile  
        = new File("d:\\DATA\\Examples\\Ex1.txt");  
    FileReader fr = new FileReader(myFile);  
    BufferedReader br = new BufferedReader(fr);  
  
    String line;  
    while ((line = br.readLine()) != null) {  
        System.out.println(line);  
    }  
    br.close();  
}
```

## Rozwiązanie 2: Blok try-catch-finally

```
public static void main(String[] args) {  
  
    File myFile  
        = new File("d:\\DATA\\Examples\\Ex1.txt");  
    BufferedReader br = null;  
    try {  
        br = new BufferedReader(new FileReader(myFile));  
        String line;  
        while ((line = br.readLine()) != null) {  
            System.out.println(line);  
        }  
    } catch (IOException e) {  
        e.printStackTrace();  
    } finally {  
        try {  
            if (br != null) { br.close(); }  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

# ODCZYT PLIKÓW TEKSTOWYCH A OBSŁUGA WYJĄTKÓW

```
public static void main(String[] args) {  
  
    File myFile = new File("d:\\DATA\\Examples\\Ex1.txt");  
    try (BufferedReader br  
        = new BufferedReader(new FileReader(myFile))) {  
  
        String line;  
        while ((line = br.readLine()) != null) {  
            System.out.println(line);  
        }  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
  
}
```

## BLOK TRY-WITH-RESOURCES

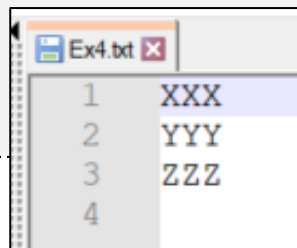
- zapewnia automatyczne zamknięcie zasobów zadeklarowanych przy instrukcji *try*
- poprawia czytelność kodu i ogranicza niebezpieczeństwo wycieków zasobów
- automatyczne zamknięcie dotyczy obiektów implementujących interfejs *Closeable* i polega na wywołaniu ich metody *close()*

# ZAPIS DO PLIKÓW TEKSTOWYCH

```
File myFile = new File("d:\\DATA\\Examples\\Ex4.txt");
String[] myData = new String[]{"XXX", "YYY", "ZZZ"};

try (BufferedWriter bw
    = new BufferedWriter(new FileWriter(myFile))) {

    for (String myDataElement : myData) {
        bw.write(myDataElement);
        bw.newLine();
    }
} catch (IOException e) {
    e.printStackTrace();
}
```



## KLASA BUFFEREDWRITER

- klasa “analogiczna” do `BufferedReader`
- służy do wydajnego zapisu ciągu znaków
- przyjmuje obiekt klasy `Writer` (lub klasy pochodnej, np. `FileWriter`)
- najczęściej służy do zapisu pliku linia po linii za pomocą metod `write()` i `newLine()`

### Uwaga:

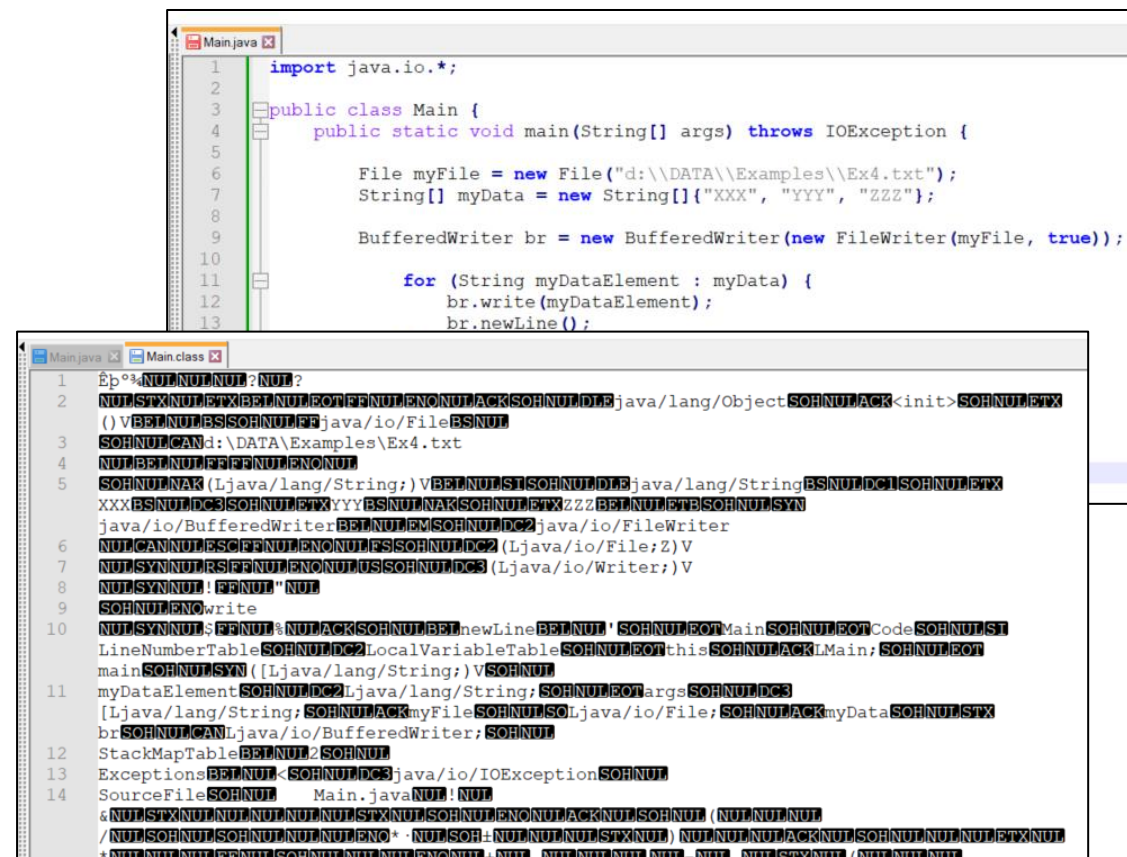
- obiekt klasy `FileWriter` można stworzyć w trybie nadpisywania lub dopisywania

```
FileWriter fr = new FileWriter(myFile);
FileWriter fr = new FileWriter(myFile, true);
```

- zawierają dane zapisane jako ciągi bajtów
- wymagają określonego sposobu przetwarzania, właściwego dla danego formatu pliku

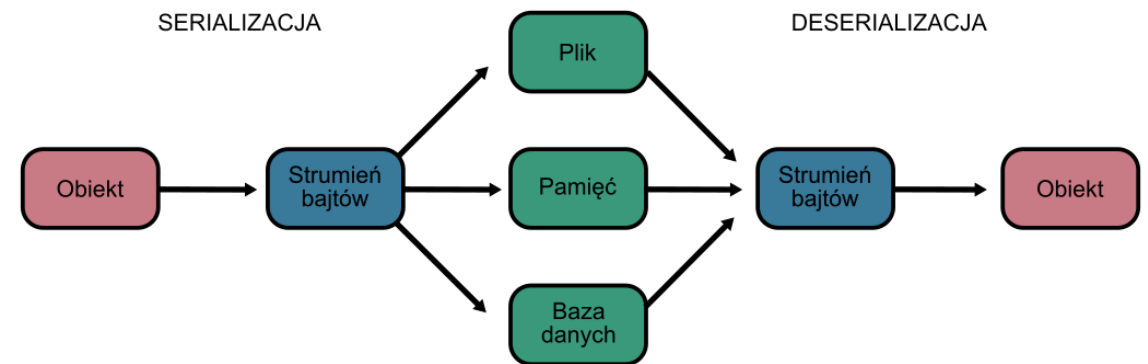
- popularne formaty plików mają w Javie odpowiednie klasy służące do ich przetwarzania:

- obrazy, np. *ImageIO*, *BufferedImage*
- pliki audio i wideo, np. zewnętrzne biblioteki *JavaZoom*, *Xuggle*



# SERIALIZACJA I DESERIALIZACJA (ang. (De)Serialization)

- serializacja to proces konwersji stanu obiektu na strumień bajtów – deserializacja to proces odwrotny: rekonstrukcji obiektu ze strumienia bajtów
- serializacja pozwala na zapis i przenoszenie obiektów, np. pomiędzy maszynami wirtualnymi Javy
- klasy implementujące interfejs znacznikowy *Serializable* można serializować za pomocą wbudowanych mechanizmów



# SERIALIZACJA I DESERIALIZACJA (ang. (De)Serialization)

```
public class Person implements Serializable {
    String firstName;
    String lastName;
    int yearOfBirth;

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

    @Override
    public String toString() {
        return String.format("%s %s (born %d)",
            firstName, lastName, yearOfBirth);
    }
}
```

```
Person p1 = new Person("XXX", "YYY", 1980);
System.out.println("Original: " + p1);

// serialization to .ser file
File saveFile = new File("person.ser");
ObjectOutputStream out = new ObjectOutputStream(
    new BufferedOutputStream(
        new FileOutputStream(saveFile)));
out.writeObject(p1);
out.close();

// deserialization from .ser file
ObjectInputStream in = new ObjectInputStream(
    new BufferedInputStream(
        new FileInputStream(saveFile)));
Person p2 = (Person) in.readObject();
System.out.println("Read: " + p2);
```

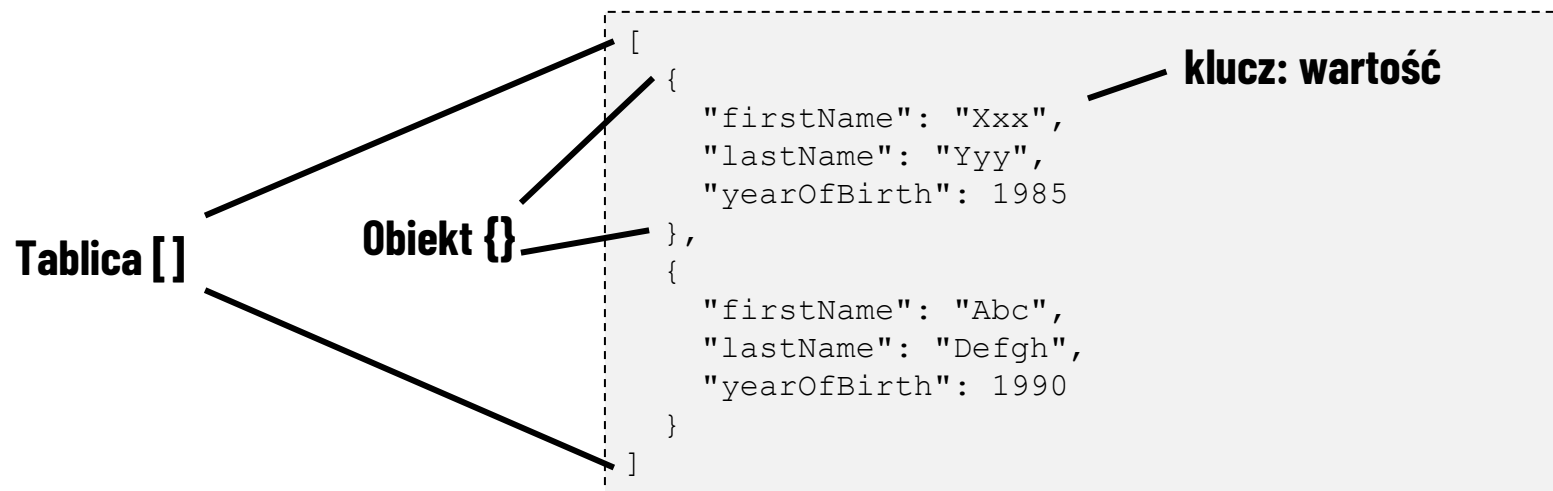
Original: XXX YYY (born 1980)

Read: XXX YYY (born 1980)



# FORMAT JSON, z ang. JavaScript Object Notation

- standard przechowywania danych i format plików tekstowych, niezależny od języka
- opiera się na zapisie danych w formie par klucz-wartość, obiektów i tablic
- powszechnie używany do wymiany danych (np. w aplikacjach webowych), przechowywania ustrukturyzowanych danych, tworzenia plików konfiguracyjnych



# FORMAT JSON, z ang. JavaScript Object Notation

```
"person": {  
  "firstName": "Xxx",  
  "middleNames": null,  
  "lastName": "Yyy",  
  "yearOfBirth": 2001,  
  "isStudent": true,  
  "grades": [85, 90, 80, 95],  
  "address": {  
    "city": "Aaa", "street": "123 Bbb",  
    "postalCode": "00-000"}  
}
```

## Uwagi:

- klucze i tekst są umieszczane w cudzysłowach
- elementy w tablicach i obiektach są oddzielane przecinkami
- białe znaki nie mają znaczenia
- wielkość liter ma znaczenie

## Dopuszczalne typy wartości:

- łańcuchy znaków
- liczby: całkowite, zmiennoprzecinkowe
- logiczne: true, false
- tablice (w nawiasach kwadratowych)
- obiekty (w klamrach)
- null

```
"person": {"firstName": "Xxx", "middleNames": null,  
"lastName": "Yyy", "yearOfBirth": 2001, "isStudent":  
true, "grades": [85, 90, 80, 95], "address": {"city":  
"Aaa", "street": "123 Bbb", "postalCode": "00-000"}}
```

# FORMAT JSON, z ang. JavaScript Object Notation

```
{
  "firstName": "Xxx",
  "middleNames": null,
  "lastName": "Yyy",
  "yearOfBirth": 2001,
  "isStudent": true,
  "grades": [85, 90, 80, 95],
  "address": {
    "city": "Aaa", "street": "123 Bbb",
    "postalCode": "00-000" }
}
```

Pakiet org.json (biblioteka JSON-java):  
<https://stleary.github.io/JSON-java/index.html>

```
File jsonFile = new File("d:\\DATA\\Examples\\person.json");
try (FileReader reader = new FileReader(jsonFile))
{
    JSONTokener jsonTokener = new JSONTokener(reader);

    JSONObject personObj = new JSONObject(jsonTokener);

    String firstName = personObj.getString("lastName");
    String middleNames = personObj.optString("middleNames", null);
    String lastName = personObj.getString("lastName");
    int yearOfBirth = personObj.getInt("yearOfBirth");
    boolean isStudent = personObj.getBoolean("isStudent");

    JSONArray grades = personObj.getJSONArray("grades");
    List<Integer> gradesList = new ArrayList<>();
    for (int i = 0; i < grades.length(); i++) {
        gradesList.add(grades.getInt(i);); }

    JSONObject addressObj = personObj.getJSONObject("address");
    String city = addressObj.getString("city");
    String street = addressObj.getString("street");

    // catch block etc.
}
```

# FORMAT JSON, z ang. JavaScript Object Notation

```
[
{
  "firstName": "Xxx", "lastName": "Yyy",
  "yearOfBirth": 2001,
  "grades": [85, 90, 80, 95]
},
{
  "firstName": "Abc", "lastName": "Def",
  "yearOfBirth": 2000,
  "grades": [0, 75, 80, 60]
}
]
```

Pakiet org.json (biblioteka JSON-java):  
<https://stleary.github.io/JSON-java/index.html>

```
File jsonFile = new File("d:\\DATA\\Examples\\students.json");
try (FileReader reader = new FileReader(jsonFile))
{
    JSNTokener jsonTokener = new JSNTokener(reader);

    JSONArray studentsArr = new JSONArray(jsonTokener);
    for (int i = 0; i < studentsArr.length(); i++) {
        JSONObject studentObj = studentsArr.getJSONObject(i);

        String firstName = studentObj.getString("firstName");
        String lastName = studentObj.getString("lastName");
        int yearOfBirth = studentObj.getInt("yearOfBirth");

        JSONArray grades = studentObj.getJSONArray("grades");
        List<Integer> gradesList = new ArrayList<>();
        for (int j = 0; j < grades.length(); j++) {
            gradesList.add(grades.getInt(j)); }

        System.out.println(String.format("%s %s (born %d)",
            firstName, lastName, yearOfBirth));
        System.out.println("Grades: " + gradesList);
    }
    // catch block etc.
```

# FORMAT JSON, z ang. JavaScript Object Notation

```
JSONArray personArr = new JSONArray();

JSONObject p1 = new JSONObject();
p1.put("firstName", "Xxx");
p1.put("lastName", "Yyy");
p1.put("yearOfBirth", 2001);

JSONArray g1 = new JSONArray();
g1.put(85);
g1.put(90);
p1.put("grades", g1);

personArr.put(p1);

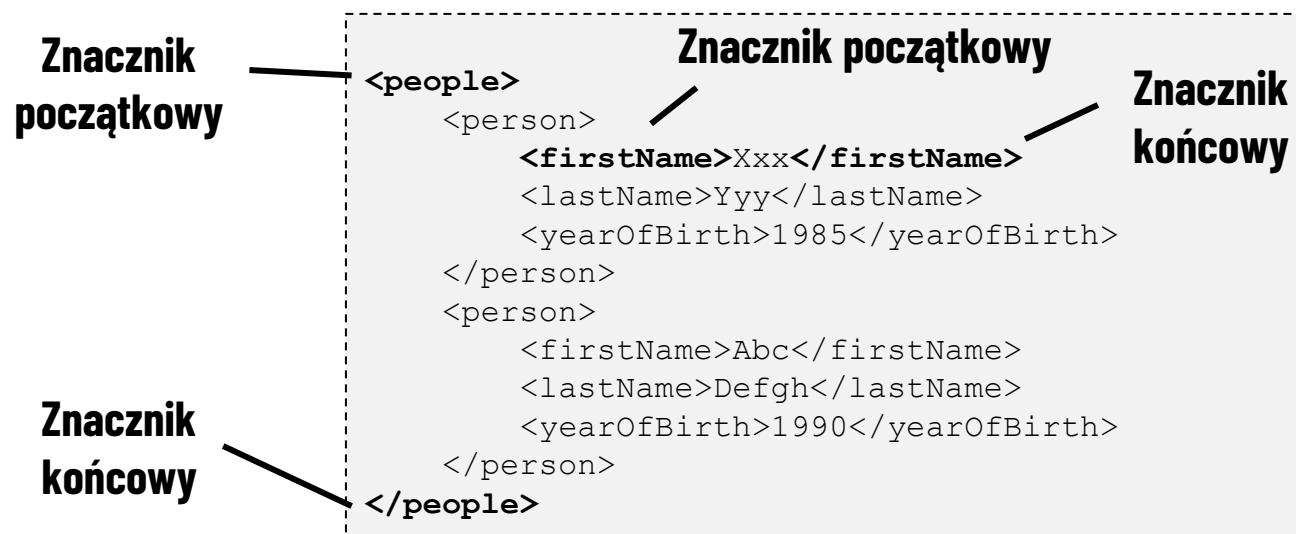
File saveFile = new File("d:\\DATA\\Examples\\person2.json");
try (FileWriter fileWriter = new FileWriter(saveFile)) {
    fileWriter.write(personArr.toString(1));
} catch (IOException e) {
    e.printStackTrace();
}
```



Pakiet org.json (biblioteka JSON-java):  
<https://stleary.github.io/JSON-java/index.html>

# FORMAT XML, z ang. eXtensible Markup Language

- język znaczników (ang. markup language) i format plików tekstowych, niezależny od języka
- opiera się na zapisie danych w znacznikach (ang. tags)
- powszechnie używany do wymiany danych (np. w aplikacjach webowych), przechowywania ustrukturyzowanych danych, tworzenia plików konfiguracyjnych



# FORMAT XML, z ang. eXtensible Markup Language

```
<person>
  <firstName>Xxx</firstName>
  <middleNames/>
  <lastName>Yyy</lastName>
  <yearOfBirth>2001</yearOfBirth>
  <isStudent>true</isStudent>
  <grades>
    <grade>85</grade>
    <grade>90</grade>
    <grade>80</grade>
    <grade>95</grade>
  </grades>
  <address>
    <city>Aaa</city>
    <street>123 Bbb</street>
    <postalCode>00-000</postalCode>
  </address>
</person>
```

Dane są przechowywane jako tekst – wybór odpowiedniego typu danych leży po stronie odczytującego

Uwagi:

- tekst nie jest umieszczany w cudzysłowach
- znaczniki są umieszczane w nawiasach ostrych
- znaczniki mogą zawierać atrybuty w formie par klucz-wartość
- tablice są reprezentowane przez serie powtarzających się elementów
- obiekty są reprezentowane przez zagnieżdżone elementy

# FORMAT XML, z ang. eXtensible Markup Language

```
<person>
  <firstName>Xxx</firstName>
  <middleNames/>
  <lastName>Yyy</lastName>
  <yearOfBirth>2001</yearOfBirth>
  <isStudent>true</isStudent>
  <grades>
    <grade>85</grade>
    <grade>90</grade>
    <grade>80</grade>
    <grade>95</grade>
  </grades>
  <address>
    <city>Aaa</city>
    <street>123 Bbb</street>
    <postalCode>00-000</postalCode>
  </address>
</person>
```

DOM API:

<https://docs.oracle.com/javase/8/docs/api/org/w3c/dom/Document.html>

```
File xmlFile = new File("d:\\DATA\\Examples\\person.xml");
try {
    DocumentBuilder builder
        = DocumentBuilderFactory.newInstance().newDocumentBuilder();

    Document document = builder.parse(xmlFile);
    document.getDocumentElement().normalize();

    NodeList nodeList = document.getElementsByTagName("person");
    for (int i = 0; i < nodeList.getLength(); i++) {
        Node node = nodeList.item(i);

        if (node.getNodeType() == Node.ELEMENT_NODE) {
            Element element = (Element) node;

            String firstName = element
                .getElementsByTagName("firstName")
                .item(0).getTextContent();
            String yearOfBirth = element
                .getElementsByTagName("yearOfBirth")
                .item(0).getTextContent();
        }
    }
}
```



# FORMAT XML, z ang. eXtensible Markup Language

```
<person>
  <firstName>Xxx</firstName>
  <middleNames/>
  <lastName>Yyy</lastName>
  <yearOfBirth>2001</yearOfBirth>
  <isStudent>true</isStudent>
  <grades>
    <grade>85</grade>
    <grade>90</grade>
    <grade>80</grade>
    <grade>95</grade>
  </grades>
  <address>
    <city>Aaa</city>
    <street>123 Bbb</street>
    <postalCode>00-000</postalCode>
  </address>
</person>
```

DOM API:

<https://docs.oracle.com/javase/8/docs/api/org/w3c/dom/Document.html>

```
// continued
```

```
NodeList gradesList = element
    .getElementsByTagName("grades");
Element gradesElement = (Element) gradesList.item(0);
NodeList grades = gradesElement
    .getElementsByTagName("grade");
```

```
List<String> gradesList = new ArrayList<>();
for (int j = 0; j < grades.getLength(); j++) {
    gradesList.add(grades.item(j)
        .getTextContent()); }
```

```
String city = element
    .getElementsByTagName("city")
    .item(0).getTextContent();
String street = element
    .getElementsByTagName("street")
    .item(0).getTextContent();
```

```
// catch block etc.
```

# FORMAT XML, z ang. eXtensible Markup Language

```
<person>
  <firstName>Xxx</firstName>
  <middleNames/>
  <lastName>Yyy</lastName>
  <yearOfBirth>2001</yearOfBirth>
  <isStudent>true</isStudent>
  <grades>
    <grade>85</grade>
    <grade>90</grade>
    <grade>80</grade>
    <grade>95</grade>
  </grades>
  <address>
    <city>Aaa</city>
    <street>123 Bbb</street>
    <postalCode>00-000</postalCode>
  </address>
</person>
```

DOM API:

<https://docs.oracle.com/javase/8/docs/api/org/w3c/dom/Document.html>

```
// continued
```

```
NodeList gradesList = element
    .getElementsByTagName("grades");
Element gradesElement = (Element) gradesList.item(0);
NodeList grades = gradesElement
    .getElementsByTagName("grade");
```

```
List<String> gradesList = new ArrayList<>();
for (int j = 0; j < grades.getLength(); j++) {
    gradesList.add(grades.item(j)
        .getTextContent()); }
```

```
String city = element
    .getElementsByTagName("city")
    .item(0).getTextContent();
String street = element
    .getElementsByTagName("street")
    .item(0).getTextContent();
```

```
// catch block etc.
```

# SERIALIZACJA DO FORMATU JSON

```
public class Person {
    String firstName;
    String lastName;
    int yearOfBirth;

    public Person() {}

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

    @Override
    public String toString() {
        return String.format("%s %s (born %d)",
            firstName, lastName, yearOfBirth);
    }
}
```

```
public String getFirstName() {
    return firstName;
}

public void setFirstName(String firstName) {
    this.firstName = firstName;
}

public String getLastName() {
    return lastName;
}

public void setLastName(String lastName) {
    this.lastName = lastName;
}

public int getYearOfBirth() {
    return yearOfBirth;
}

public void setYearOfBirth(int yearOfBirth) {
    this.yearOfBirth = yearOfBirth;
}
}
```

# SERIALIZACJA DO FORMATU JSON

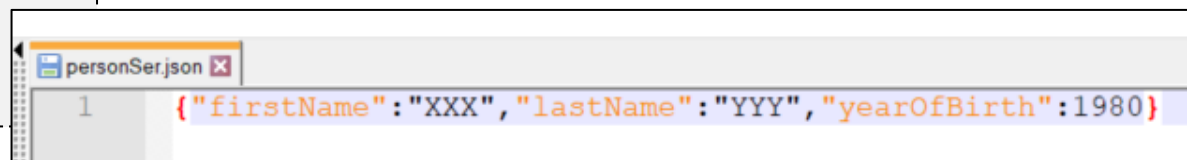
```
Person p1 = new Person("XXX", "YYY", 1980);

try {
    ObjectMapper objectMapper = new ObjectMapper();

    String jsonString
        = objectMapper.writeValueAsString(p1);
    System.out.println(jsonString);

    File saveFile =
        new File("d:\\DATA\\Examples\\personSer.json");
    objectMapper.writeValue(saveFile, p1);
} catch (IOException e) {
    e.printStackTrace();
}
```

```
{"firstName":"XXX","lastName":"YYY","yearOfBirth":1980}
```



Biblioteka Jackson:  
<https://github.com/FasterXML/jackson>

# DESERIALIZACJA Z FORMATU JSON

```
Person p1 = new Person("XXX", "YYY", 1980);
System.out.println("Original: " + p1);

try {
    ObjectMapper objectMapper = new ObjectMapper();

    File saveFile =
        new File("d:\\DATA\\Examples\\personSer.json");
    Person p2 =
        objectMapper.readValue(saveFile, Person.class);

    System.out.println("Read: " + p2);
} catch (IOException e) {
    e.printStackTrace();
}
```

```
Original: XXX YYY (born 1980)
Read: XXX YYY (born 1980)
```

Biblioteka Jackson:

<https://github.com/FasterXML/jackson>