

# PROGRAMOWANIE OBIEKTOWE



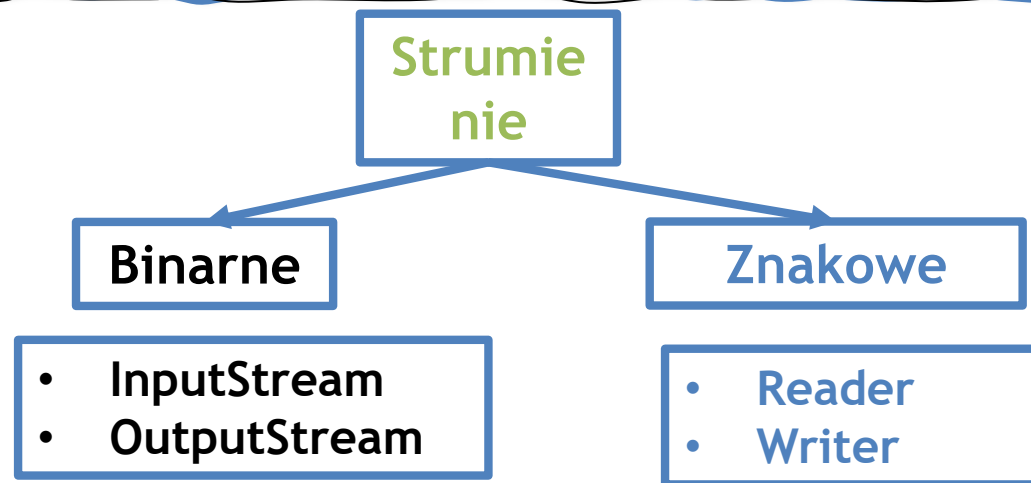
Obsługa wejścia/wyjścia.

dr inż. Barbara Fryc

# OBSŁUGA WEJŚCIA/WYJŚCIA



**Strumień** - w java jest to abstrakcja odczytu/zapisu do urządzeń wejścia/wyjścia, tzn. plików, konsoli, klawiatury, połączeń sieciowych itd.



# STRUMIENIE ZNAKOWE

Strumienie znakowe w Java operują na znakach UNICODE.

## Odczyt danych tekstowych

### Pochodne klasy Reader

- BufferedReader, → readLine
- CharArrayReader,
- FilterReader,
- InputStreamReader
  - FileReader,
- PipedReader,
- StringReader

## Zapis danych tekstowych

### Pochodne klasy Writer

- BufferedWriter,
- CharArrayWriter,
- FilterWriter,
- OutputStreamWriter,
  - FileWriter
- PipedWriter,
- PrintWriter, ← println
- StringWriter

# PRZYKŁAD

```
List<Produkt> orders = new ArrayList<>();

....
try {

    PrintWriter out= new PrintWriter("Zamowienie.txt");

    out.println("Zamawiane składniki: " );

    for (Produkt s : orders) {
        out.print("Składnik: ");
        out.println(s.getNazwa() + " ilość: " + s.getIlosc()+"g");
    }

    if(tp2checkBoxPack.isSelected()==true)
    {
        out.print("\n\nZapakować wszystko");
    }
    out.close();

    JOptionPane.showMessageDialog(null, "Dane zapisane do pliku Zamówienie.txt");
```

# STRUMIENIE BAJTOWE

Strumienie bajtowe odczytują kolejne bajty z danego źródła

## Odczyt danych dowolnych

Pochodne klasy InputStream

- ByteArrayInputStream,
- FileInputStream,
- ObjectInputStream
- StringBufferInputStream

```
byte b = read()
```

```
int read(byte[]
```

```
tablica)
```

## Zapis danych dowolnych

Pochodne klasy OutputStream

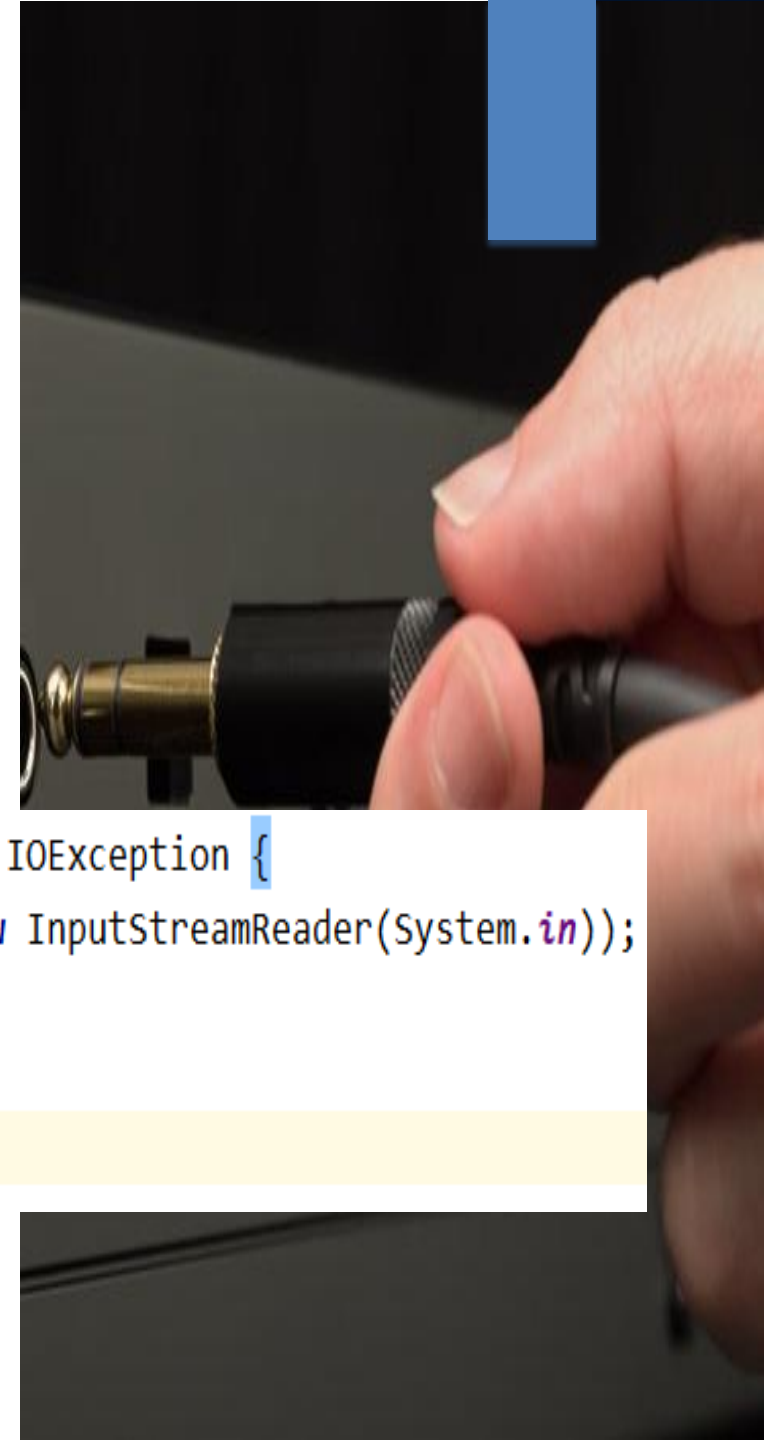
- ByteArrayOutputStream,
- FileOutputStream,
- ObjectOutputStream
- StringBufferOutputStream

```
write(byte b)
```

```
write(byte[] tab)
```

# ŁĄCZENIE STRUMIENI

Strumień można łączyć, w celu rozbudowywania funkcjonalności. W tym celu jako argument konstruktora strumienia podajemy inny strumień



```
public static void main(String[] args) throws IOException {  
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  
    String linia = br.readLine();  
    System.out.println("Linia "+linia);  
}
```

# KLASA FILE

Klasa pomocnicza umożliwiająca operacje na plikach i katalogach

- `isDirectory()`
- `isFile()`
- `length()`
- `list()`
- `mkdir()`
- `exists()`
- ...



# JAVADOC

**Javadoc** - narzędzie automatycznie generujące dokumentację na podstawie zamieszczonych w kodzie źródłowym znaczników w komentarzach. Javadoc został stworzony specjalnie na potrzeby języka programowania Java przez firmę Sun Microsystems.



# STRUKTURA KOMENTARZA JAVADOC

Komentarz Javadoc oddzielony jest znacznikami

```
/**
```

```
i
```

```
*/
```

które sprawiają, że ich zawartość (czyli to, co znajduje się między nimi), jest ignorowana przez kompilator. Pierwsza jego linia to opis metody lub klasy, która zadeklarowana jest poniżej. Dalej znajduje się pewna liczba opcjonalnych tagów, które z kolei opisują parametry metody (@param), wartość zwracaną (@return) itp.

# STRUKTURA KOMENTARZA JAVADOC

```
/**
 *
 * @param imie imie osoby
 * @param nazwisko nazwisko osoby
 * @param data_ur dataurodzenia w formacie dd/MM/yyyy
 */
public Osoba(String imie, String nazwisko, LocalDate data_ur) {
    this.imie = imie;
    this.nazwisko = nazwisko;
    this.data_ur = data_ur;
}
```

# JAVADOC TAGI

## Popularne tagi

- |            |                                   |
|------------|-----------------------------------|
| ■ @author  | Author projektu/aplikacji         |
| ■ @version | wersja projektu/ aplikacji        |
| ■ @since   | początek                          |
| ■ @param   | znaczenie/opis parametru          |
| ■ @return  | znaczenie/opis wartości zwracanej |
| ■ @throws  | znaczenie/opis wyjątku            |
| ■ @see     | link do innych opcji              |

# EXAMPLE

```
/** An abstract class representing various kinds of
 *  shapes. This class represents common features
 *  of all shapes such as ...
 *
 *  @author Barbara Fryc
 *  @version 1.0 (26/23)
 *  @since version 0.5
 */
public abstract class Shape {
    // ...
}
```

# SPECIFYING PARAMETERS AND RETURN VALUE

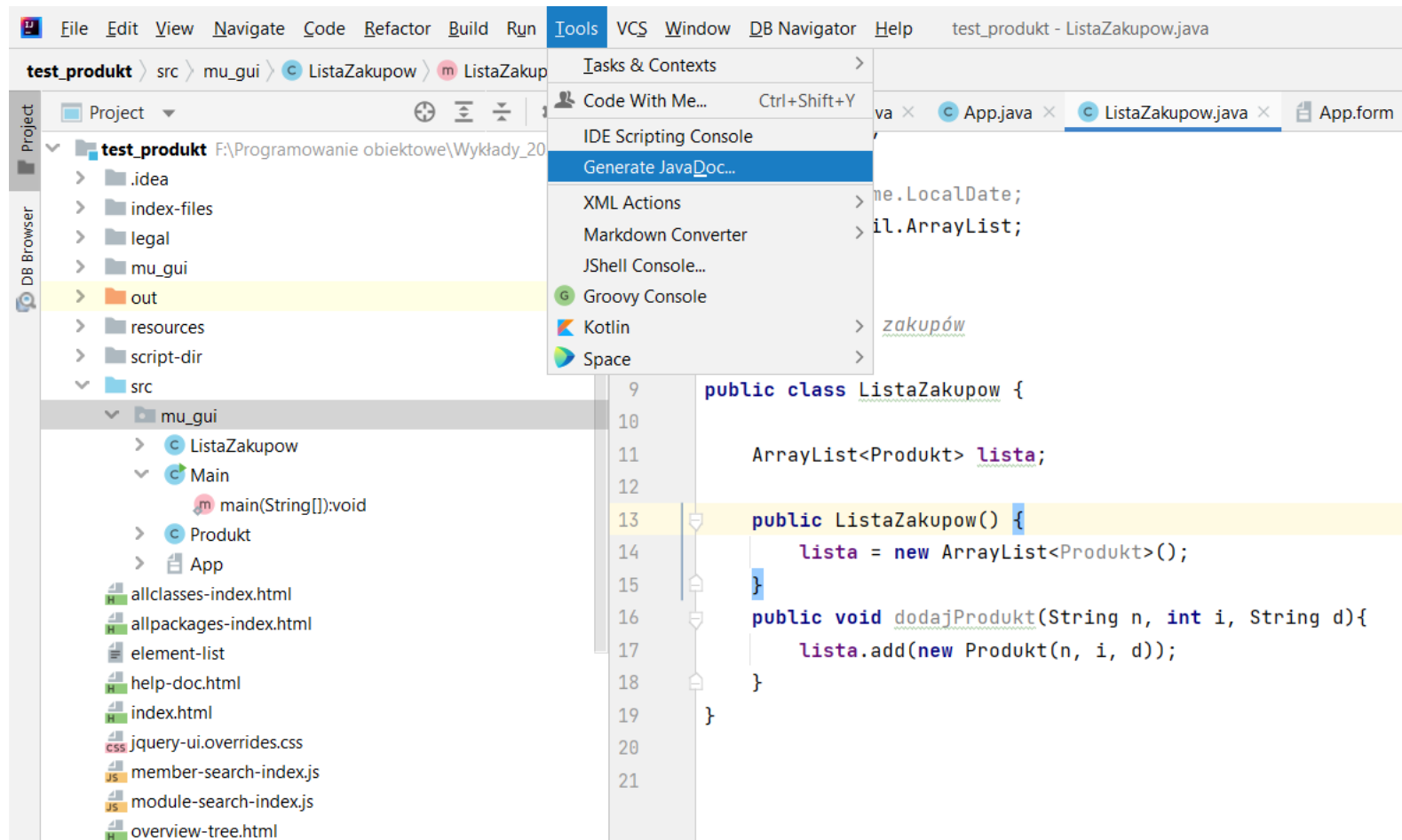
## ◉ Syntax

- @param *name description*
- @return *description*
- @throws *exception description*

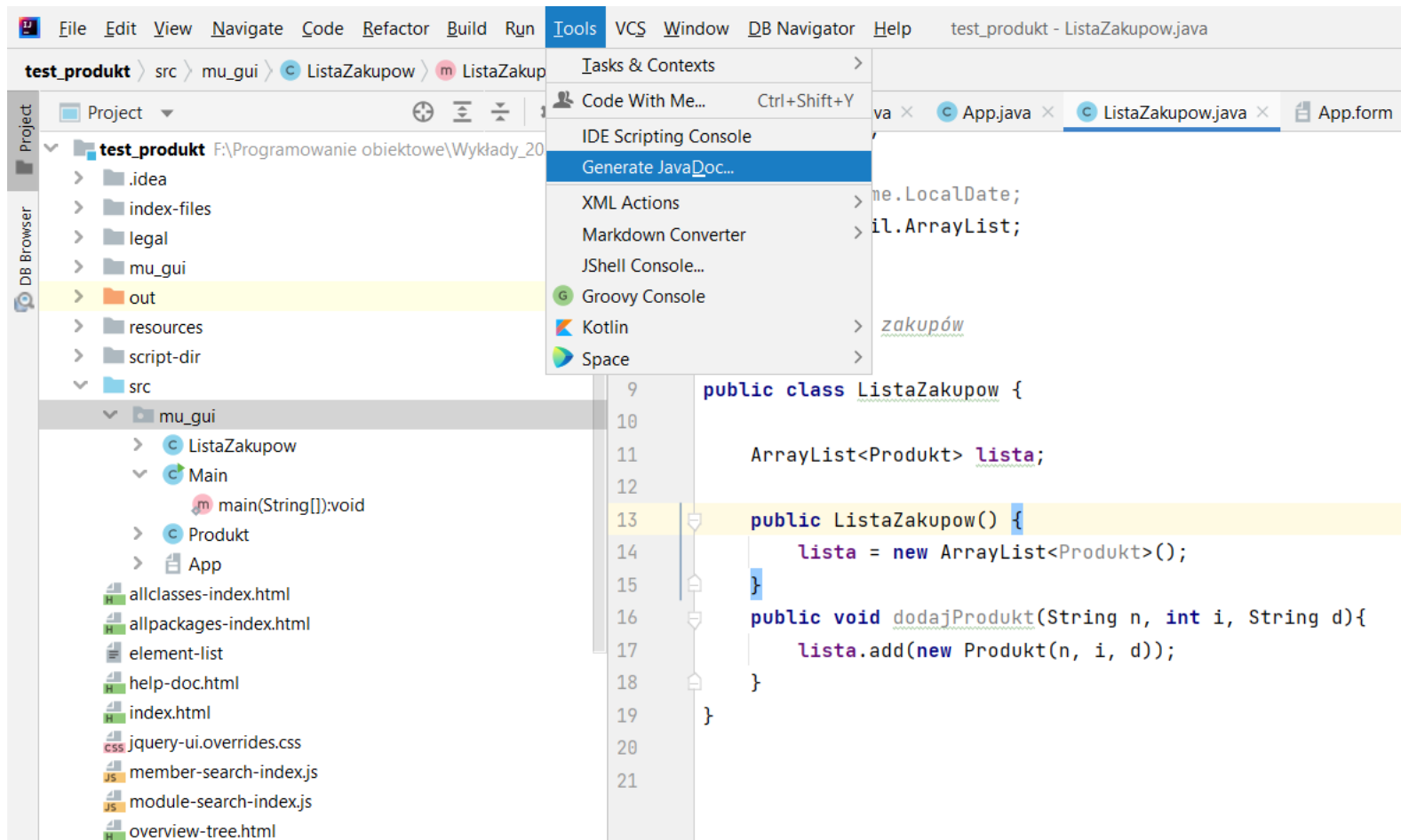
## ◉ Example

```
/** Returns the definition of a given word in this dictionary.  
 *  
 * @param word a word whose definition is being looked up.  
 * @return the definition of the word; null if no definition is  
 * found.  
 * @throws NullPointerException if the word is null.  
 */  
public String lookup(String word) { /* ... */ }
```


# INTELLIJ IDEA GENEROWANIE DOKUMENTACJI



# INTELLIJ IDEA GENEROWANIE DOKUMENTACJI



# POLSKIE ZNAKI W DOKUMENTACJI

 **Generate JavaDoc** ✕

Generate JavaDoc Scope

☒ Whole project

☐ File '...\src\mu\_gui\ListaZakupow.java'

☐ Custom scope  ...

☐ Include JDK and library sources in -sourcepath

☐ Link to JDK documentation (use -link option)

Output directory:  📁

private

package

protected

public

☒ Generate hierarchy tree  
☒ Generate navigation bar  
☒ Generate index  
☒ Separate index per letter

☒ @use  
☒ @author  
☒ @version  
☒ @deprecated  
☒ deprecated list

Locale:

Other command line arguments:

Maximum heap size (Mb):

☒ Open generated documentation in browser

? OK Cancel



# FORMAT JAR

- JAR (Java ARchive) jest formatem archiwum wykorzystującym kompresję ZIP. Używa się go do strukturalizacji i kompresji skompilowanych programów w Javie.
- Archiwum JAR zawiera plik manifestu umieszczony w ścieżce META-INF/MANIFEST.MF, który informuje o sposobie użycia i przeznaczeniu archiwum.
- W pliku manifestu możemy wyszczególnić klasę główną projektu. W takim przypadku plik JAR będzie stanowił samodzielnie uruchamialną aplikację (poprzez kliknięcie).
- Archiwum JAR możemy stworzyć poprzez polecenie *jar* wpisywane z wiersza poleceń (dostępne w JDK razem z kompilatorem i maszyną wirtualną). Należy przy tym podać odpowiednie parametry w zależności, czy chcemy utworzyć, czy też rozpakować archiwum.

# FORMAT JAR W INTELLIJ IDEA

- W środowisku IntelliJ idea możemy wyeksportować naszą aplikację do wykonywalnego. Ustawiamy w tym celu ścieżkę do pliku wynikowego, następnie wybieramy klasę główną, która zawiera metodę main.

