PROGRAMOWANIE OBIEKTOWE

Tworzenie interfejsu graficznego w Javie

dr inż. Barbara Fryc

INTERFEJS GRAFICZNY W JAVIE – PODSTAWOWE INFORMACJE

- Kluczowym problemem do rozwiązania przy tworzeniu GUI (Graphical User Interface) w Javie jest przenośność.

- Przenośność można zapewnić wykorzystując komponenty graficzne dostarczane przez biblioteki systemowe. Zaletą takiego rozwiązania jest szybkość i niezawodność działania poszczególnych komponentów. Wadą jest ograniczony zestaw komponentów i ich możliwości (muszą być takie same we wszystkich systemach).

- Innym rozwiązaniem przenośności jest stworzenie biblioteki graficznej całkowicie w Javie. Dzięki temu możemy zapewnić bogaty zestaw komponentów graficznych, które są całkowicie przenośne. Wadą tego rozwiązania jest mniejsza szybkość działania (wyświetlanie grafiki jest obsługiwane przez maszynę wirtualną).

INTERFEJS GRAFICZNY W JAVIE – BIBLIOTEKI

- Do tworzenia interfejsu graficznego w Javie można skorzystać z jednej z trzech bibliotek:
  - **AWT** (Abstract Windowing Toolkit) – najstarsza biblioteka wprowadzona wraz z pierwszą wersją Javy. Dostarcza najuboższego zestawu komponentów. Biblioteka ta nie dostarcza żadnych własnych komponentów, a jedynie wykorzystuje dostarczane przez system operacyjny, w którym uruchamiamy program.
  - **Swing** – wprowadzona wraz z Java2, dostarcza znacznie szerszych możliwości niż AWT (dostarcza własnych komponentów graficznych napisanych w Javie). Jest to obecnie najpopularniejsza biblioteka do tworzenia interfejsu użytkownika w Javie.
  - **SWT** (Standard Widget Toolkit) – stosunkowo nowa biblioteka o porównywalnych możliwościach jak Swing. Została opracowana przez firmę IBM i wykorzystana do stworzenia GUI programu Eclipse.
  - **Java FX** - platforma do tworzenia graficznych interfejsów użytkownika nowej generacji dostarczona została wraz z JDK 7. możliwość definiowania widoku aplikacji (tego jak wygląda) w języku XML, a nie tak w Swingu w kodzie Javy.
- Wymienione biblioteki wykorzystuje się do tworzenia interfejsu graficznego aplikacji desktopowych oraz apletów.
- Do tworzenia interfejsu aplikacji sieciowych używa się innych technologii.

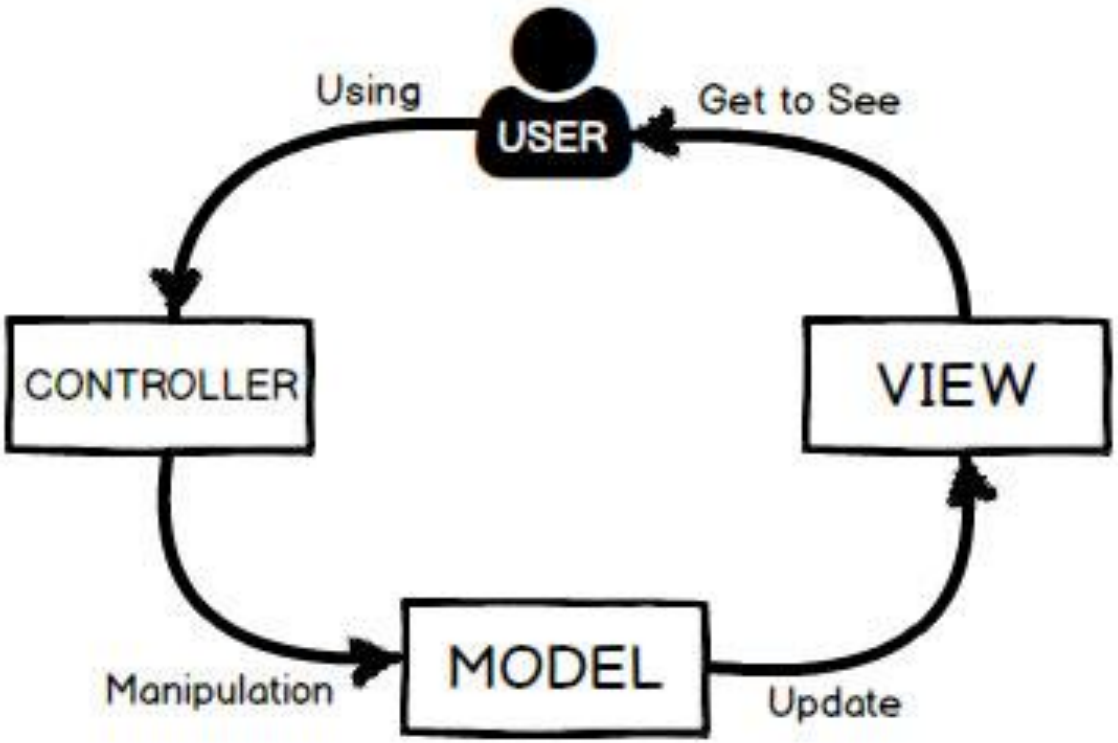
PAKIETY JAVAFX

Cała zawartość platformy JavaFX została umieszczona w pakietach, których nazwy zaczynają się od javafx. Zaczynając od JDK 9, pakiety JavaFX zostały zorganizowane w formie modułów javafx.base, javafx.graphics oraz javafx.controls.

JAVA FX - ZALETY

- **Tworzenie GUI przy pomocy arkuszy CSS.** Dzięki temu możemy dowolnie zmieniać wygląd aplikacji, nie wprowadzając żadnych zmian w jej kodzie. Pozwala to zachować jego czystość oraz łatwo tworzyć wiele szablonów graficznych, wczytywanych według potrzeb.
- **Biblioteki dla bogatych interfejsów.** JavaFX udostępnia odświeżone, znane wcześniej ze Swinga jak i nowe komponenty. Pozwala łatwo implementować ciekawe efekty graficzne oraz szybko tworzyć grafikę 2D oraz 3D.
- **Integracja z innymi technologiami.** JavaFX jest kompatybilna z najnowszymi JDK (od wersji 7). Jej kod można swobodnie łączyć z natywnymi bibliotekami Java, Swingiem, czy aplikacjami webowymi - JavaScript, HTML. Tak jak inne aplikacje Java, zapewnia działanie na każdej platformie z zainstalowaną JVM (Java Virtual Machine).
- **Sprzętowe przetwarzanie grafiki.** Dzięki wykorzystaniu potokowego (sekwencyjnego) przetwarzania grafiki z wykorzystaniem kart graficznych, można tworzyć efektowne i płynne efekty wizualne.

JAVAFX A WZORZEC MVC



WIDOK

- Widoki mają postać plików FXML (mogą być też zapisane w Javie). Kontroler, który ma przypisany jakiś FXML (właściwie w JavieFX jest troszkę na odwrót bo to w FXML'u przypisujemy kontroler do widoku, ale z logicznego punktu widzenia to widok przypisywany jest do kontrolera nie odwrotnie - to kontroler ma zarządzać widokiem, nie widok kontrolerem).
- Przykładowy FXML aplikacji do dodawania dwóch liczb:

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.Pane?>
<?import javafx.scene.text.Font?>
<Pane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-
    Infinity" prefHeight="400.0"
      prefWidth="600.0" xmlns="http://javafx.com/javafx/8.0.111"
    xmlns:fx="http://javafx.com/fxml/1"
      fx:controller="mvc.controller.Controller">
    <children>
        <TextField fx:id="number1" layoutX="147.0" layoutY="53.0" promptText="Liczba 1"/>
        <TextField fx:id="number2" layoutX="147.0" layoutY="105.0" promptText="Liczba 2"/>
        <Label fx:id="resultLabel" layoutX="223.0" layoutY="250.0">
            <font>
                <Font size="24.0"/>
            </font>
        </Label>
        <Button layoutX="208.0" layoutY="169.0" mnemonicParsing="false"
    onAction="#buttonOnAction" text="Dodaj"/>
    </children>
</Pane>
```

KONTROLER

- W kontrolerze można przypisać konkretne kontrolki do zmiennych Javowych.
- Tutaj będzie potrzebny dostęp do pól tekstowych aby pobrać z nich wprowadzone przez użytkownika dane oraz do etykiety (Label) aby wyświetlić wynik. Dodatkowo w fxmlu można przypisać od razu metody akcji. **W kontrolerze musi znaleźć się jej sygnatura bo inaczej JavaFx nie zadziała.**

```
@FXML
private TextArea studentTextArea;
2 usages
@FXML
private TextField textName;
```

MODEL

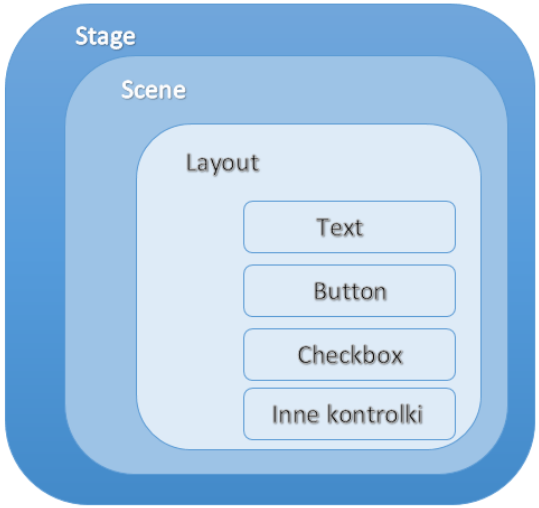
- Model czyli logika aplikacji. Logiką jest tutaj dodanie dwóch liczb.
- Przykładowy model:

```
public class Model
{
    public int add(int a, int b)
    {
        return a+b;
    }
}
```

KLASY STAGE ORAZ SCENE

- Główną metaforą zaimplementowaną w JavaFX jest obszar roboczy (ang. stage). W przypadku prawdziwych sztuk scenicznych obszar roboczy zawiera scenę (ang. scene).

- A zatem potocznie mówiąc: obszar roboczy definiuje przestrzeń, a scena określa, co się w tej przestrzeni pojawi.



https://github.com/koduj-z-klasa/java-krok-po-kroku/blob/master/docs/source/gui.rst

KLASY STAGE ORAZ SCENE

- Obiekt **Stage** jest pojemnikiem najwyższego poziomu. Wszystkie aplikacje JavaFX automatycznie mają dostęp do jednego obiektu **Stage**, nazywanego głównym obszarem roboczym (ang. *Primarystage*).

- Obiekt **Scene** jest pojemnikiem dla wszystkich elementów, które składają się na scenę (są nimi kontrolki, takie jak przyciski, pola wyboru, pola tekstowe oraz inne elementy graficzne). W celu utworzenia sceny dodać należy wszystkie te elementy do obiektu klasy **Scene**.

KLASY STAGE ORAZ SCENE

- Główną metaforą zaimplementowaną w JavaFX jest obszar roboczy (ang. stage). W przypadku prawdziwych sztuk scenicznych obszar roboczy zawiera scenę (ang. scene).
- A zatem potocznie mówiąc: obszar roboczy definiuje przestrzeń, a scena określa, co się w tej przestrzeni pojawi.
- Obiekt Stage jest pojemnikiem najwyższego poziomu. Wszystkie aplikacje JavaFX automatycznie mają dostęp do jednego obiektu Stage, nazywanego głównym obszarem roboczym (ang. Primary stage).
- Obiekt Scene jest pojemnikiem dla wszystkich elementów, które składają się na scenę (są nimi kontrolki, takie jak przyciski, pola wyboru, pola tekstowe oraz inne elementy graficzne). W celu utworzenia sceny dodać należy wszystkie te elementy do obiektu klasy Scene.

Węzły i graf sceny

- Pojedyncze elementy sceny są nazywane węzłami (ang. node). Na przykład takim węzłem będzie kontrolka przycisku.
- Węzły mogą także zawierać grupy innych węzłów. Węzeł może także zawierać inny węzeł potomny. W takim przypadku węzeł zawierający jakieś dziecko jest nazywany węzłem rodzica (ang. parent node) lub węzłem gałęzi (ang. branch mode).
- Z kolei węzły, które nie zawierają węzłów potomnych, są nazywane węzłami końcowymi (ang. terminal node) lub liśćmi.

Węzły i graf sceny

- Kolekcja wszystkich węzłów tworzących scenę jest określana jako graf sceny (ang. scene graph) i stanowi drzewo.
- W grafie sceny występuje jeden, specjalny typ węzła — korzeń (ang. root node). Jest to najwyższy węzeł w grafie sceny i jednocześnie jedyny, który nie ma żadnego rodzica. A zatem wszystkie inne węzły należące do grafu sceny mają rodzica i wszystkie bezpośrednio lub pośrednio są potomkami korzenia.
- Klasą bazową dla wszystkich węzłów jest Node.
  - Istnieje kilka różnych klas, które bezpośrednio lub pośrednio są jej klasami pochodnymi; między innymi należą do nich: Parent, Group, Region oraz Control.

UKŁADY

- JavaFX udostępnia kilka paneli układu obsługujących proces rozmieszczania innych elementów na scenie.
- Na przykład klasa FlowPane tworzy układ rozmieszczający elementy jeden za drugim, a klasa GridPane — układ pozwalający na umieszczanie elementów w wierszach i kolumnach. Dostępnych jest także kilka innych układów, takich jak BorderPane. Każdy z nich dziedziczy po klasie Node.
- Wszystkie klasy układów zostały umieszczone w pakiecie javafx.scene.layout.

KLASA APPLICATION ORAZ METODY CYKLU ŻYCIA

- Aplikacja JavaFX musi być klasą pochodną klasy Application zdefiniowanej w pakiecie javafx.application.
- Klasa ta definiuje trzy metody cyklu życia, które aplikacja może przesłaniać: init(), start() oraz stop().
- Metoda init() jest wywoływana na samym początku działania aplikacji. Służy ona do wykonywania różnych czynności inicjalizacyjnych. Niemniej jednak nie można jej używać do tworzenia obiektu sceny ani do jej konstruowania.
- Jeśli aplikacja nie potrzebuje żadnych czynności inicjalizacyjnych, to metody init() nie trzeba przesłaniać, gdyż istnieje jej wersja domyślna

KLASA APPLICATION ORAZ METODY CYKLU ŻYCIA

- Po metodzie init() wywoływana jest metoda start() – stanowi ona początek działania aplikacji i może posłużyć do skonstruowania i przygotowania sceny.
- Parametrem tej metody jest obiekt klasy Scene dostarczony przez środowisko uruchomieniowe i stanowi scenę główną aplikacji.
- Jest to metoda abstrakcyjna i aplikacja musi ją nadpisać.

KLASA APPLICATION ORAZ METODY CYKLU ŻYCIA

- Kiedy aplikacja kończy działanie, wywoływana jest z kolei metoda stop(), która pozwala na wykonanie wszelkich czynności porządkowych związanych z zamykaniem aplikacji.
- W przypadkach, gdy wykonywanie takich czynności nie jest potrzebne, można skorzystać z pustej, domyślnej wersji tej metody.

URUCHAMIANIE APLIKACJI JAVAFX

- Aby uruchomić niezależną aplikację JavaFX, należy wywołać metodę launch() klasy Application:
  - public static void launch(String ... args)
- Parametr args reprezentuje listę (być może pustą) łańcuchów znakowych, które zazwyczaj będą określały argumenty wiersza poleceń.
- Wywołanie metody launch() powoduje utworzenie aplikacji, a następnie wywołanie jej metod init() i start().
- Wywołanie metody launch() zakończy się dopiero po zakończeniu aplikacji.

SZKIELET APLIKACJI JAVAFX

```
public class HelloApplication extends Application {
    public HelloApplication() {
    }

    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(HelloApplication.class.getResource("hello-view.fxml"));
        Scene scene = new Scene((Parent)fxmlLoader.load(), 320.0, 240.0);
        stage.setTitle("Hello!");
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
            // Uruchamia aplikację JavaFX, wywołując metodę launch().
        launch(new String[0]);
    }
}
```

INTERFEJS ACTIONLISTENER

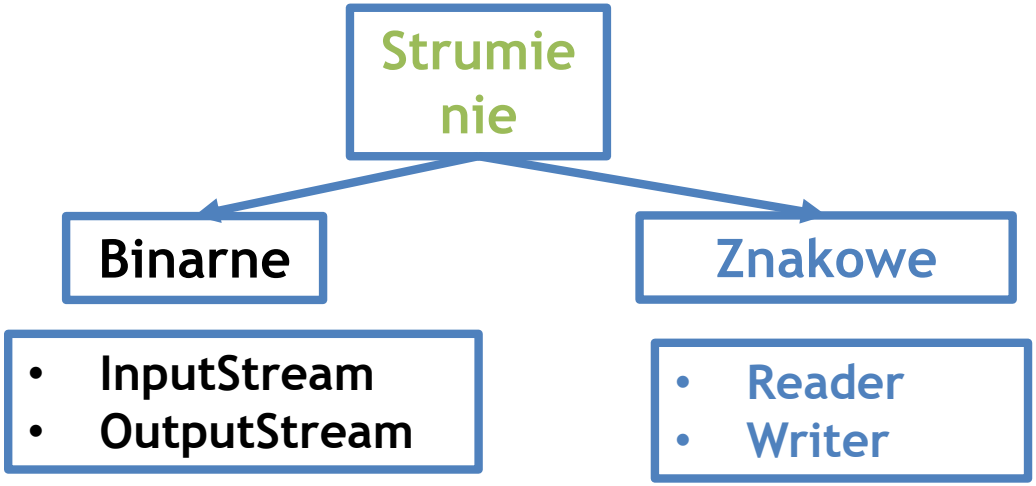
- java.awt.event.ActionListener jest interfejsem służącym do obsługi zdarzeń specyficznych dla komponentu (np. dla przycisku będzie to kliknięcie myszy). Jedyną metodą jaką on posiada jest actionPerformed()
- Przykładowa implementacja interfejsu:

```
class Sluchacz implements ActionListener {
    public void actionPerformed(ActionEvent e) {
            String nazwa = ((Button)e.getSource()).getText();
            txt.setText(nazwa);
    }
}
```

- Powyższy słuchacz będzie pobierał tekst z jednego komponentu (typu Button) i umieszczał ten tekst w innym komponencie (txt) za każdym razem kiedy użytkownik kliknie w pierwszy komponent.

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**,
- CharArrayReader,
- FilterReader,
- **InputStreamReader**
  - FileReader,
- PipedReader,
- StringReader

readLine

Zapis danych tekstowych

Pochodne klasy Writer

- BufferedWriter,
- CharArrayWriter,
- FilterWriter,
- OutputStreamWriter,
  - **FileWriter**
- PipedWriter,
- **PrintWriter**,
- **StringWriter**

println

PRZYKŁAD - ZAPIS

```
try {
        PrintWriter out= new PrintWriter("Zamowienie.txt");
       out.println("Zamawiane składniki: " );
       out.print("\nmasło \n chleb");
       out.close();
}
catch (IOException a)
{
       System.out.println("Błąd zapisu pliku");
}
```

STRUMIENIE BAJTOWE

Strumienie bajtowe odczytują kolejne bajty z danego źródła

Odczyt danych dowolnych

Pochodne klasy InputStream

- ByteArrayInputStream,
- FileInputStream,
- ObjectInputStream
- StringBufferInputStream

Zapis danych dowolnych

Pochodne klasy OutputStream

- ByteArrayOutputStream,
- FileOutputStream,
- ObjectOutputStream
- StringBufferOutputStream

byte b = **read**()

Int **read**(byte[]

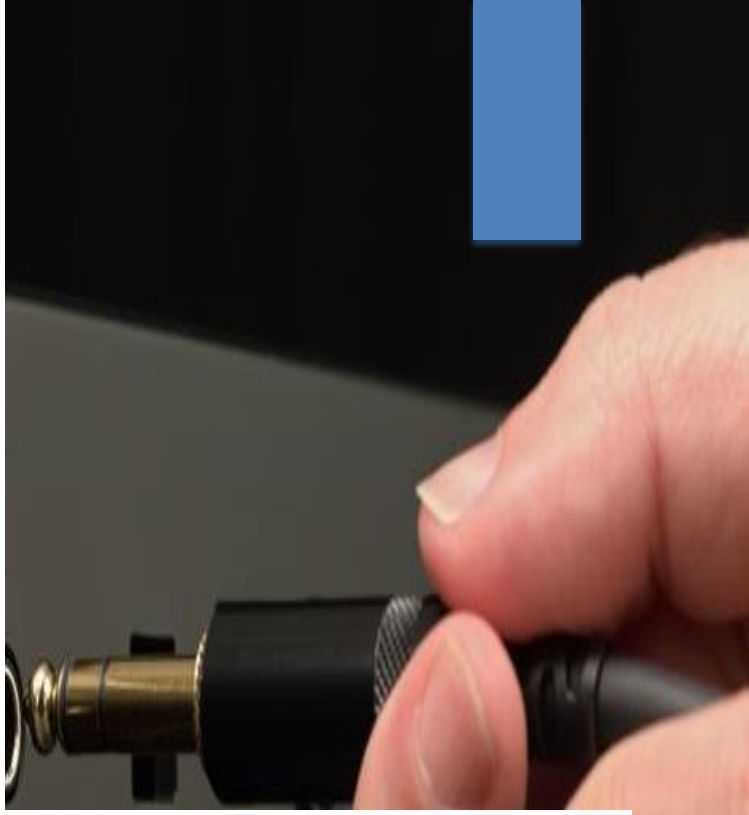
tablica)

write(byte b)

write(byte[] tab)

ŁĄCZENIE STRUMIENI

Strumienie 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()
- …

KLASA FILECHOOSER

```
FileChooser fil_chooser = new FileChooser();
File file = fil_chooser.showSaveDialog(
            new Stage());
String path = file.getAb;solutePath()
```

CIEKAWE TUTORIALE

- http://qbisiek.blogspot.com/2013/12/bardzo-bogate-klienty-javafx-tutorial-01.html
- https://github.com/koduj-z-klasa/java-krok-po-kroku/blob/master/docs/source/gui.rst
- https://ggoralski.pl/1907/
- https://www.youtube.com/watch?v=dLyDJdYGKWY&t=28s