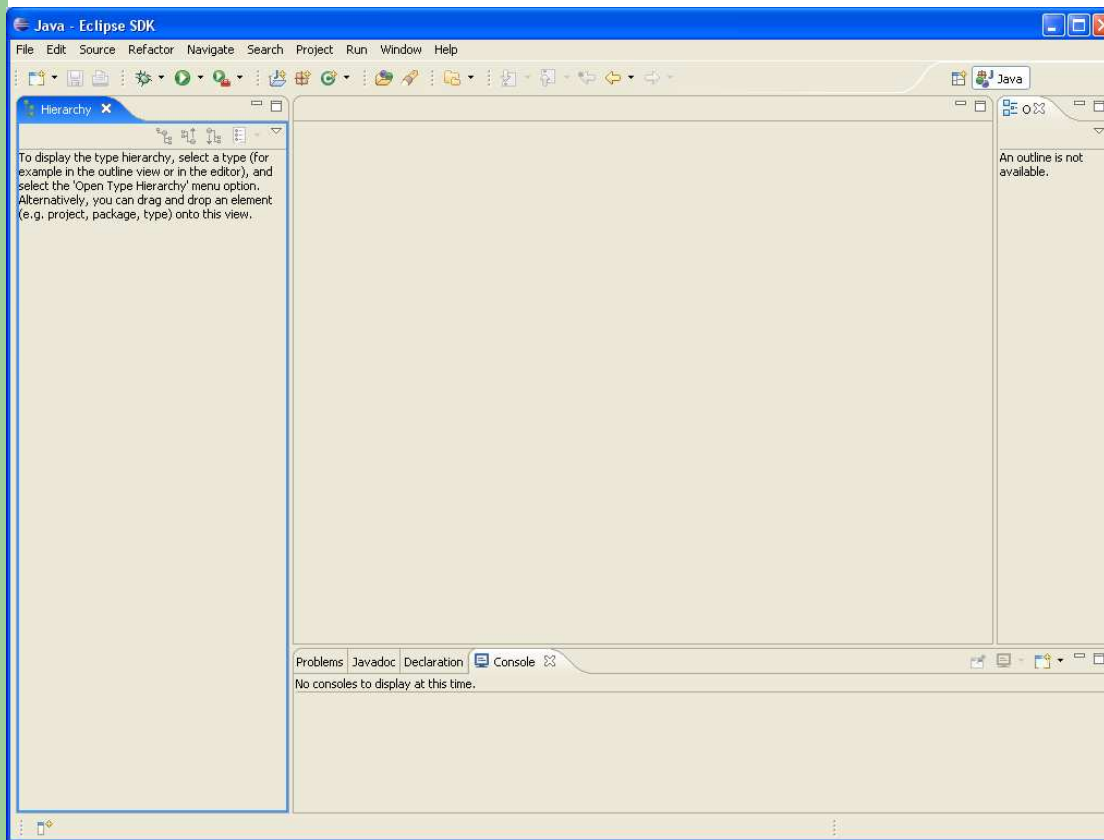


Zaawansowane programowanie obiektywne

Wykład 3 część 2

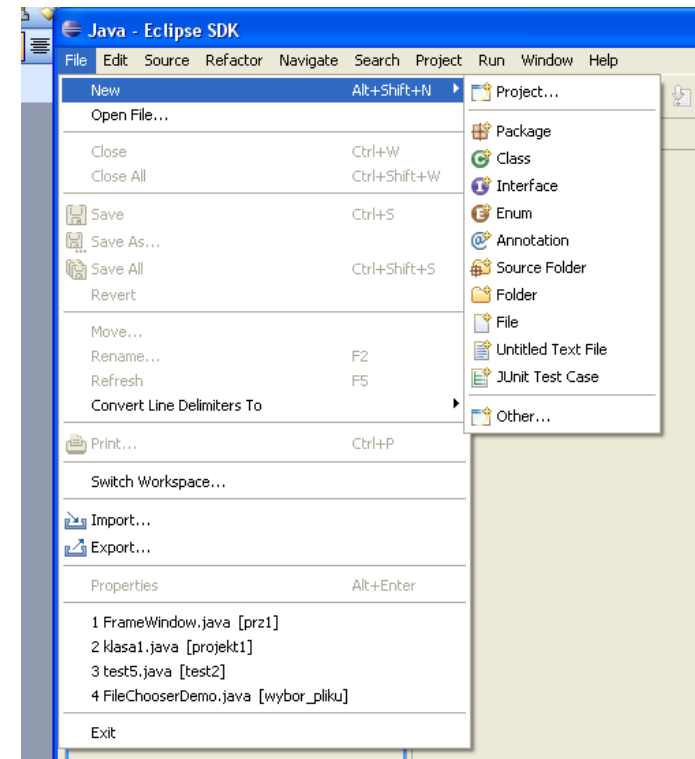


Środowisko Eclipse



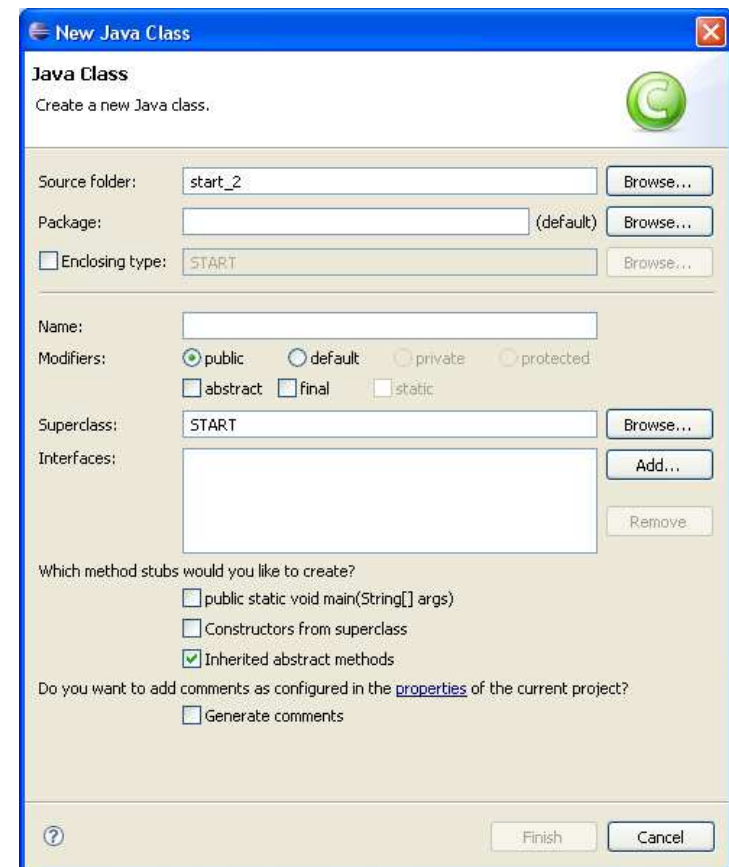
Tworzenie projektu

- File → New → project
- Java project



Dodanie klasy

- File → New → Class

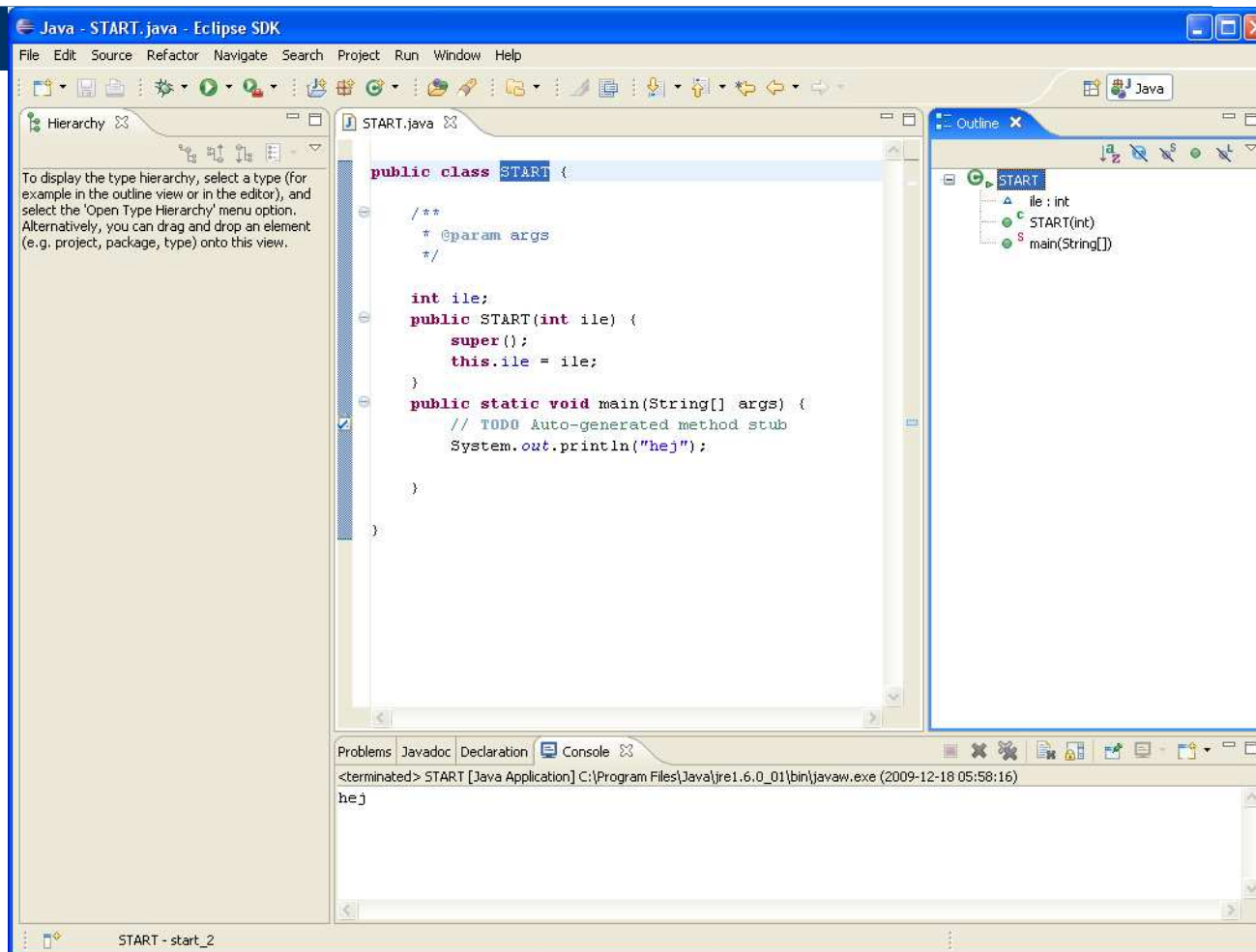


The screenshot shows the 'New Java Class' dialog box. The title bar is 'New Java Class'. The main heading is 'Java Class' with a subtitle 'Create a new Java class.' and a green 'C' icon. The dialog contains several fields and options:

- Source Folder:** 'start_2' with a 'Browse...' button.
- Package:** '(default)' with a 'Browse...' button.
- Enclosing type:** 'START' with a 'Browse...' button.
- Name:** An empty text field.
- Modifiers:** Radio buttons for 'public' (selected), 'default', 'private', and 'protected'. Checkboxes for 'abstract', 'final', and 'static'.
- Superclass:** 'START' with a 'Browse...' button.
- Interfaces:** An empty list box with 'Add...' and 'Remove' buttons.
- Which method stubs would you like to create?** Checkboxes for 'public static void main(String[] args)', 'Constructors from superclass', and 'Inherited abstract methods' (checked).
- Do you want to add comments as configured in the [properties](#) of the current project?** A checkbox for 'Generate comments'.

At the bottom, there is a help icon (?), a 'Finish' button, and a 'Cancel' button.

Praca w Eclipse



Przykład 1

- `import java.awt.*;`
- `public class rysik {`
- `public static void main(String[] args) {`
 `Frame okno = new Frame("Okno");`
 `okno.add(new Plansza());`
 `okno.setSize(100,100);`
 `okno.setVisible(true); }`
 `}`
- `class Plansza extends Canvas {`
 - `Plansza() {`
 `System.out.println(""+this.getSize().width);`
 `}`
 - `public void paint(Graphics g) {`
 - `g.drawLine(10,10,20,30);`
 `}`
- `}`

AWT – *abstract windowing toolkit*

- pierwsza biblioteka okienkowa Javy;
- dostępna we wszystkich wersjach Javy;
- mało przyjazne API;
- brak wielu pożytecznych kontrolek;
- pakiet `java.awt`

SWT (Standard Widget Toolkit)

- nie dołączane standardowo do Javy, należy dołączyć do programu;
- bardzo wydajna oferuje bogata paletę komponentów
- korzysta z komponentów systemowych wszędzie, gdzie to możliwe.

Swing

- nowsza wersja biblioteki okienkowej, dostępna od wersji 1.2 języka Java.
- jest wygodny, dobrze przemyślany interfejs i szeroka gama dostępnych komponentów.
- nie ma jej w starszych wersjach Javy,
- nie jest ona szczególnie wydajna.
- pakiet javax.swing

Nazewnictwo

- Dla łatwiejszego **odróżniania** klas **Swing** od klas **AWT** w bibliotece Swing wprowadzono następująca konwencje nazewnicza:
- nazwy wszystkich klas biblioteki **Swing** zaczynają się od litery **J**, po czym następuje właściwa nazwa komponentu, np.
- JButton oznacza przycisk (ang. *button*).

MainFrameAWT

- **import** java.awt.*;
- **import** java.awt.event.*;
- **public class** MainFrameAWT **extends** Frame {
 - **public** MainFrameAWT() {
 - **super**("Aplikacja AWT");
 - **init**();
 - **show**();
 - }
 - **private void** **init**() {
 - **setLayout**(**null**);
 - **setSize**(400, 300);
 - **Label** label = **new** **Label**("Aplikacja AWT");
 - label.**setBounds**(10, 20, 100, 25);
 - label.**add**(label);
 - **addWindowListener**(**new** **WindowAdapter**() {
 - **public void** **windowClosing**(**WindowEvent** e) {
 - **System.exit**(0);
 - }
 - **});**
 - }
 - }

```
public class AppAWT {  
  
    public static void main(String[] args) {  
        MainFrameAWT f = new  
            MainFrameAWT();  
    }  
}
```

Swing

- **import** java.awt.event.*;
- **import** javax.swing.*;
- **public class** MainFrameSwing **extends** JFrame {
 - **public** MainFrameSwing() {
 - **super**("Aplikacja Swing");
 - **init**();
 - **show**();
 - }
 - **private void** **init**() {
 - **setSize**(400, 300);
 - **JPanel** panel = (**JPanel**) **getContentPane**();
 - panel.**setLayout**(**null**);
 - **JLabel** label = **new** **JLabel**("Aplikacja Swing");
 - label.**setBounds**(10, 10, 100, 25);
 - panel.**add**(label);
 - **addWindowListener**(**new** **WindowAdapter**() {
 - **public void** **windowClosing**(**WindowEvent** e) {
 - **System.exit**(0);
 - }
 - **});**
 - }
 - }

```
public class appswing {  
  
    public static void  
    main(String[] args) {  
        MainFrameSwing f = new  
        MainFrameSwing();  
    }  
}
```

Podstawowe elementy graficznego interfejsu użytkownika (GUI)

- Komponenty

- Przyciski
- Etykiety
- Listy
- Pola edycyjne

- Kontenery

- okna

Etapy tworzenia GUI

- Główne okno programu
- Komponent
- Umieszczanie komponentu na oknie
- Wymiana danych komponent/program

javax.swing / java.awt

1. `import java.awt.*`
2. `extends JFrame`
3. `show(), setSize()...`

```
import javax.swing.*;
class start_gui extends JFrame {
}
class start_program {
    public static void main(String[] s){
        start_gui okienko = new start_gui();
        okienko.setSize(500,500);
        okienko.show();
    }
}
```

Aplikacja okienkowa musi posiadać swoje okno. Okno jest klasa dziedzicząca po klasie `java.awt.Window`.

Jednak w aplikacjach Swing będziemy najczęściej dziedziczyć po klasie `javax.swing.JFrame`, która jest jej klasa pochodna.

Metody klasy JFrame

- **JFrame()** – konstruktor domyślny – tworzy nowe okienko,
- **JFrame(String tytuł)** – konstruktor tworzący okienko z ustalonym napisem na belce tytułowej,
- **Container getContentPane()** – zwraca panel zawartości, w którym należy umieszczać wszystkie komponenty, które mają się znaleźć w oknie aplikacji,
- **void setContentPane (Container cp)** – ustawia panel zawartości dla aplikacji,

- **void setDefaultCloseOperation(int operation)** – ustawia operację wykonywaną po zamknięciu aplikacji przez użytkownika, możliwe wartości, to:

- **JFrame.DO NOTHING ON CLOSE** – nic nie robi (nie zamyka okna) – musimy sami obsłużyć zdarzenie związane z zamykaniem aplikacji,
- **JFrame.HIDE ON CLOSE** – ukrywa okno (nie zamyka go), jest to wartość domyślna, jeśli jej nie zmienimy tak właśnie zareaguje program na próbę zamknięcia przez użytkownika,
- **JFrame.DISPOSE ON CLOSE** – zamyka okno i zwalnia jego zasoby (zalecane dla okien potomnych),
- **JFrame.EXIT ON CLOSE** – zamyka program za pomocą metody `System.exit` – jest to zalecana reakcja dla okna głównego aplikacji,

- **void setJMenuBar(JMenuBar menubar)** – ustawia główne menu dla okna
- **void setLayout(LayoutManager manager)** – ustawia menedżera układu dla okna –domyślnie użyty jest menedżer java.awt.BorderLayout,
- **void addWindowListener(WindowListener l)** – dodaje odbiorcę zdarzeń związanych z oknem (takich jak na przykład zamknięcie okna).
- **void setBounds(int x, int y, int width, int height)** - ustawia pozycję i wymiary okna (po kolei: odległość od lewej krawędzi ekranu, odległość od góry ekranu, wysokość i szerokość; wszystkie odległości podane w pikselach),
- **void setVisible(boolean widoczne)** – ustawia widzialność okienka.

zalecenia

- metoda ***main***, która stworzy obiekt okna i wyświetli go. Metodę umieszczamy w klasie głównego okna,
- operacje związane z wyglądem okna umieszczamy w **konstruktorze** klasy tego okna.

Komponenty w oknie aplikacji

- Layout managers → menadżer rozmieszczenia
 - Typy rozmieszczenia:
 - **Border** Layout, **Flow** Layout, **Box** Layout, **Grid** Layout,
1. Dla kontenera określ sposób rozmieszczania elementów,
 2. *Dodaj komponenty → dodaje do okna programu za pomocą utworzonego menedżera (metoda kontenera `add(...)`).*

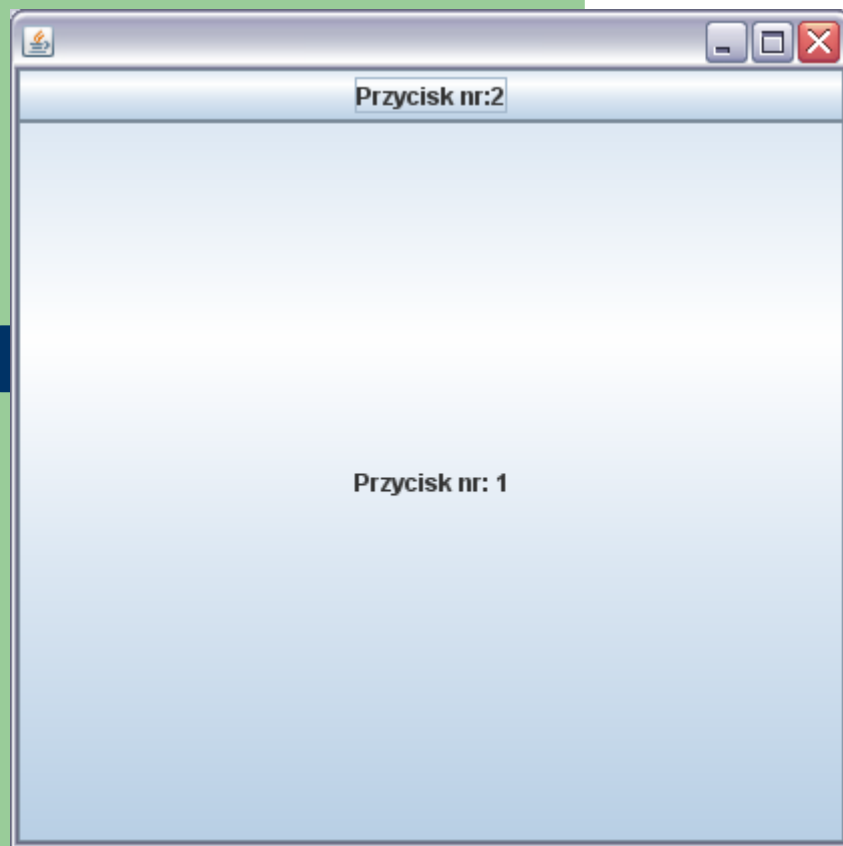
- Obiekty klasy **JFrame** składają się z czterech płaszczyzn (ang. *panes*),
- płaszczyzna nazwana **ContentPane** jest kontenerem, zawiera elementy graficznego interfejsu użytkownika.
- `JFrame frame = new JFrame();`
- `Container cp = frame.getContentPane();`
- `cp.add(...);`


użycie menedżera BorderLayout

- **new BorderLayout()** → domyślny dla JFrame
- Rozmieszczanie elementów:
- pięć regionów, identyfikowanych etykietami "North", ..., "South", "Center",
- dodawanie elementów:
 `add(nazwa_komponentu, region)`
- Wypełnianie regionów: całkowite
- Zmiana rozmiarów okna: cały przyrost
 powierzchni zajmowany przez element "Center"

```
public static void main (String[] args){  
    new layout_start();  
}
```

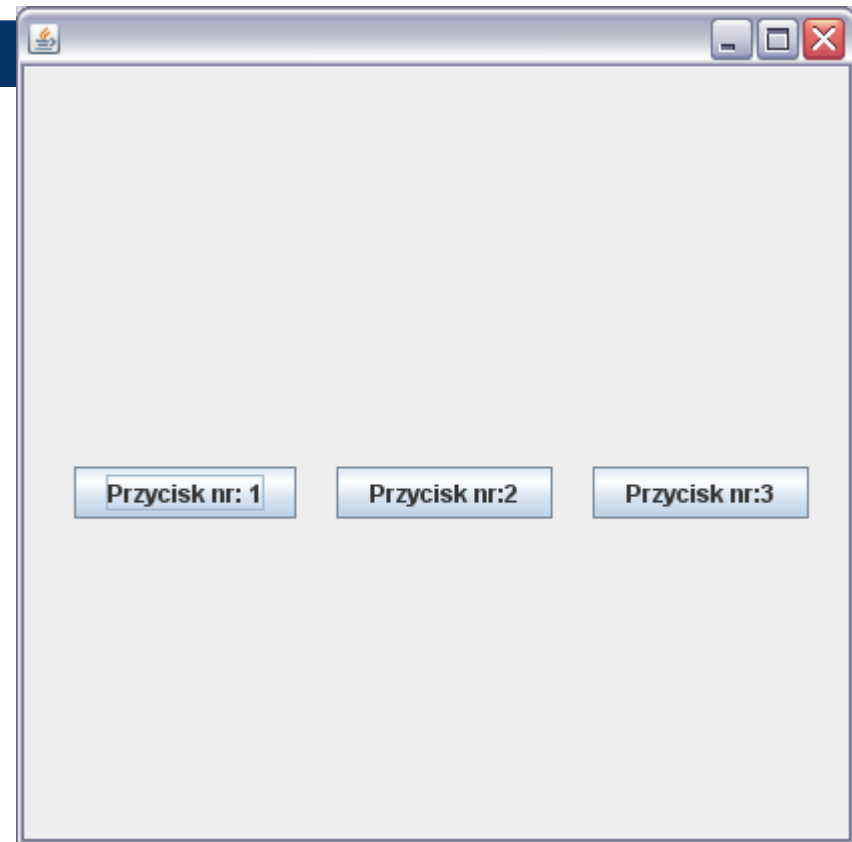
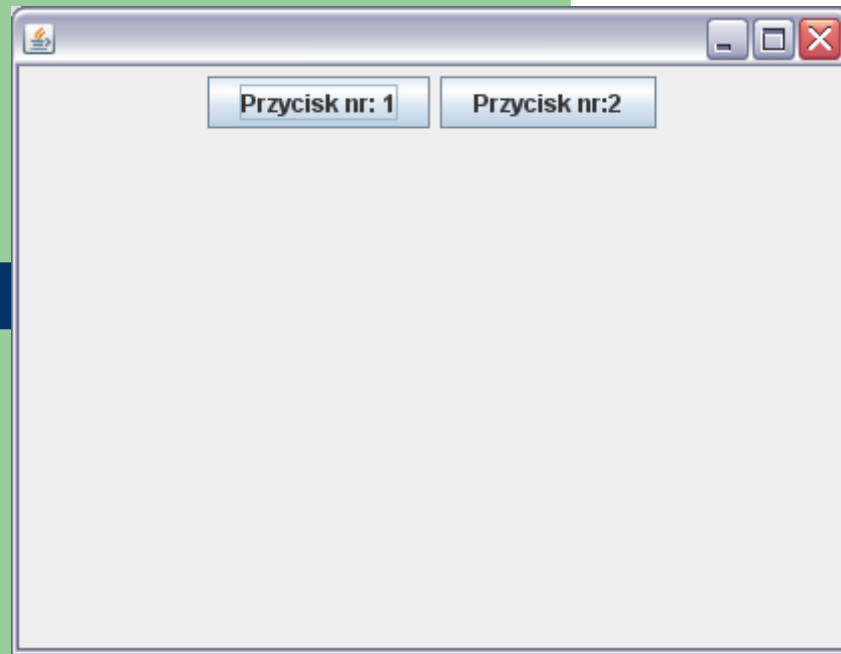
```
1. public layooy_start() {  
2. JFrame frame = new JFrame();  
3. Container cp = frame.getContentPane();  
4. cp.setLayout(new BorderLayout());  
5. int i=1;  
6. JButton b = new JButton("Przycisk nr: " + (i));  
7. JButton b1 = new JButton("Przycisk nr: " +  
    (i+1));  
8. cp.add(b,"Center");  
9. cp.add(b1,"North");  
10.}  
// "North", "South", "East", "West", "Center"
```



- 
- `frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);`
 - `frame.setSize(420,420);`
 - `frame.setVisible(true);`

użycie menedżera FlowLayout

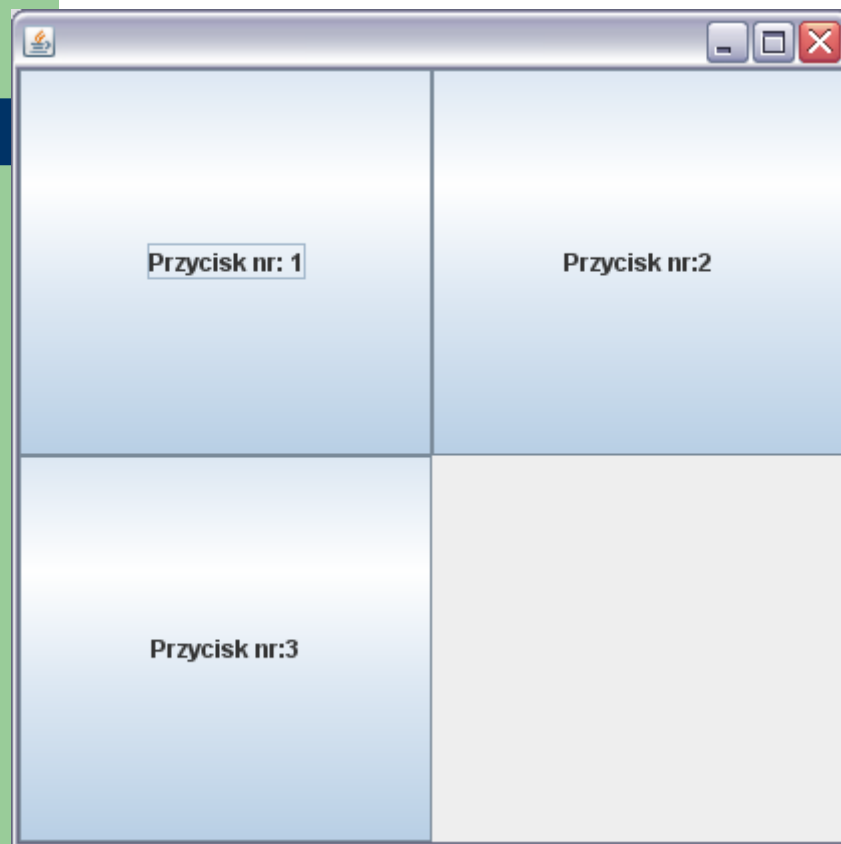
- domyślny dla kontenera typu **JPanel**
- **new FlowLayout()**
Rozmieszczanie elementów: umieszczane
kolejno, jeden za drugim dodawanie elementów:
`add(nazwa_komponentu)`
- Wypełnianie regionów: określone rozmiarem
domyslnym komponentu
- Zmiana rozmiarów okna: komponenty nie
zmieniają rozmiarów



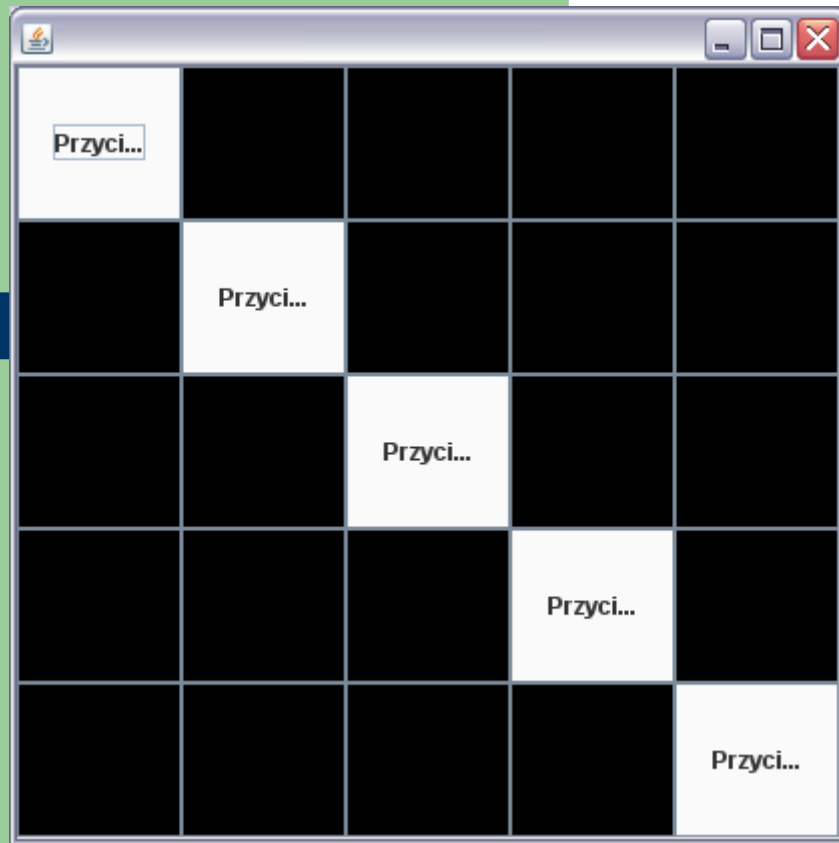
```
cp.setLayout(new FlowLayout(FlowLayout.RIGHT,20,200));
```

użycie menedżera GridLayout

- new **GridLayout**(liczba_wierszy,liczba_kolumn)
- **Rozmieszczanie elementów:** umieszczane w kolejnych komórkach hipotetycznej siatki o strukturze określonej w konstruktorze.
- **Dodawanie elementu:**
add(nazwa_komponentu)
- **Wypełnianie regionów:** maksymalne możliwe
- **Zmiana rozmiarów okna:** równomierne dla wszystkich komponentów



```
cp.setLayout(new GridLayout(2,2));
```



```
b.setBackground(new Color(250,250,250));
```

użycie menedżera GridBagLayout

- Z każdym komponentem jest kojarzony obiekt o nazwie **GridBagConstraints**, opisujący wszystkie szczegóły jego położenia i zachowania podczas zmian rozmiaru okna.
1. położenie elementu –
gridx, gridy (np. gridx=2)
 2. rozmiar elementu -
gridwidth, gridheight
 3. zmiennosc rozmiaru elementu -
fill (np. GridBagConstraints.HORIZONTAL)
 4. zmiennosc rozmiaru komórki hipotetycznej siatki -
weightx, weighty (np. weightx=100)
 5. usytuowanie elementu w komórce siatki -
anchor (np. GridBagConstraints.NORTH)

Manager *GridBagLayout*

- Klasa `GridBagLayout`, podobnie jak klasa `GridLayout` pozwala na układanie komponentów w „tabelce”. Ma ona jednak większe możliwości:
 - ustalanie wielkości komórek
 - łączenie ich.

przykład

- Container cp = getContentPane();
JButton b= new JButton(„przycisk”);
GridBagLayout gbl=new GridBagLayout();
- cp.setLayout(gbl);
- GridBagConstraints gbc=new GridBagConstraints();
- GridBagConstraints gbc=new GridBagConstraints();
- gbc.gridx=0; gbc.gridy=0;
gbc.gridwidth=2; gbc.gridheight=1;
gbc.weightx=100; gbc.weighty=100;
gbc.fill=GridBagConstraints.NONE;
gbc.anchor=GridBagConstraints.NORTHWEST;
- gbl.setConstraints(b1,gbc);
cp.add(b1);

Manager *CardLayout*

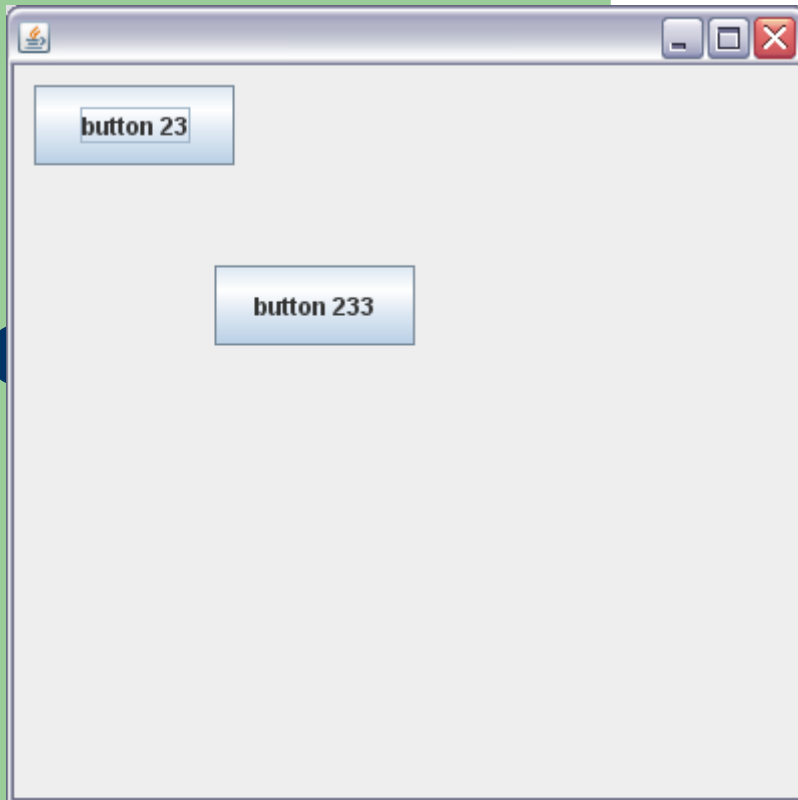
- Układa dodawane komponenty na stosie. Pierwszy umieszczony w panelu komponent zostanie umieszczony na wierzchu, pozostałe poniżej niego (będą niewidoczne).
- przełączanie się między komponentami możliwe jest za pomocą metod klasy *CardLayout*:

klasy CardLayout

- void first(Container parent) – przewija na pierwszy komponent,
- • void last(Container parent) – przewija na ostatni komponent,
- • void next(Container parent) – przewija na następny komponent,
- • void previous(Container parent) – przewija na poprzedni komponent,
- • void show(Container parent, String name) – przewija na komponent o podanej nazwie.

bez menedżera rozmieszczenia

- Elementy mogą być umieszczone w oknie programu bezpośrednio, poprzez wskazanie ich położenia i rozmiaru.
- **setBounds(...)** - przyjmuje jako parametry położenie (wsp. x i y w pikselach) i rozmiar (szerokość, wysokość):
- ```
 JButton b1= new JButton("button 2");
 Container cp = getContentPane();
 cp.setLayout(null);
 b1.setBounds(10,10,100,40);
```



```
import java.awt.*;
import javax.swing.*;
```

```
public class bezy {
```

```
 public static void main(String[] args) {
 new bezy();
 }
```

```
 public bezy(){
 JButton b1= new JButton("button 23");
 JButton b2= new JButton("button 233");
```

```
 JFrame frame = new JFrame();
 Container cp = frame.getContentPane();
 cp.setLayout(null);
```

```
 b1.setBounds(10,10,100,40);
 b2.setBounds(100,100,100,40);
```

```
 cp.add(b1);
 cp.add(b2);
```

```
 frame.pack();
 frame.setSize(400, 400);
 frame.setVisible(true);
 }
}
```

# użycie menedżera BorderLayout

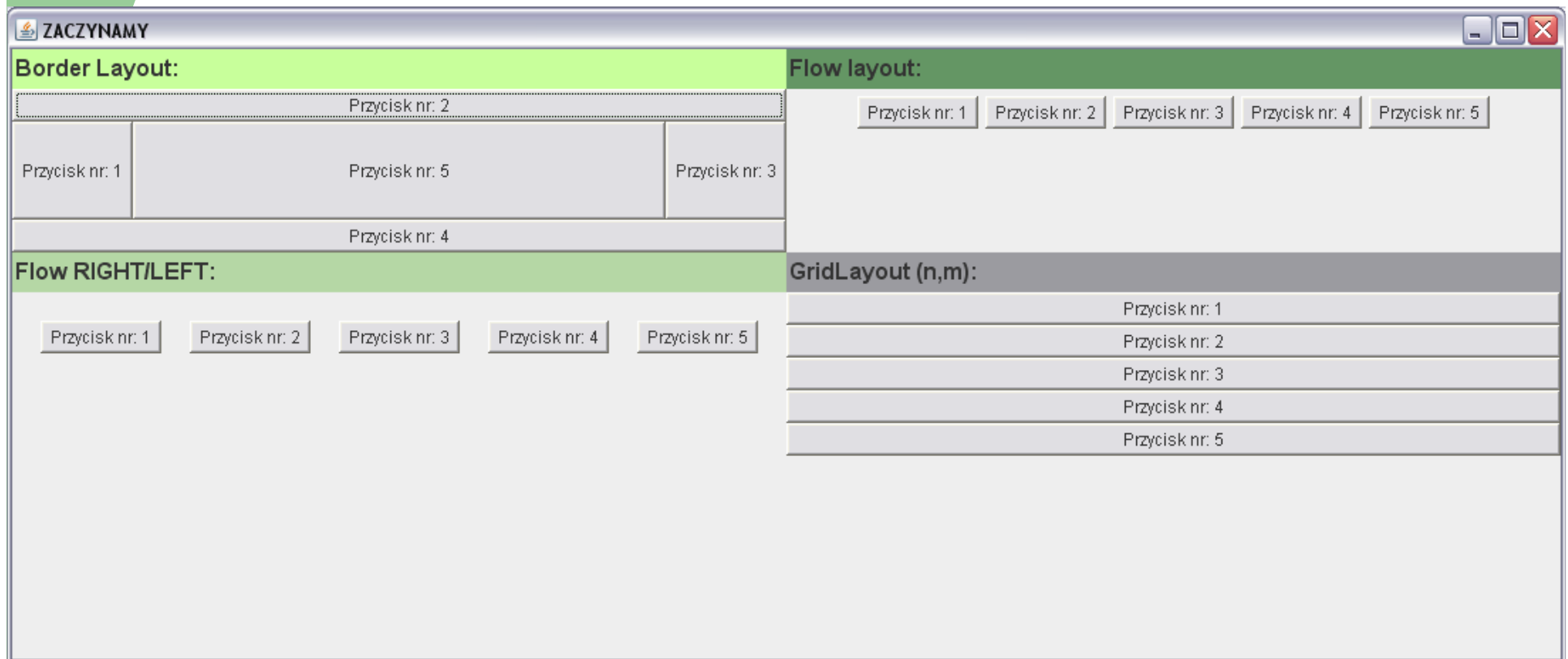
- **new BorderLayout(nazwa\_kontenera, orientacja)**
- orientacja - **BoxLayout.X\_AXIS, BorderLayout.Y\_AXIS**
- Rozmieszczanie elementów:  
kolejno, pionowo lub poziomo.
- Dodawanie elementu:  
**add(nazwa\_komponentu)**
- Wypełnianie regionów:  
położenie określone dla każdego komponentu indywidualnie -  
**setAlignment(położenie)**  
**Component.CENTER\_ALIGNMENT,**  
**Component.LEFT\_ALIGNMENT,...**
- Zmiana rozmiarów okna:  
zmiana rozmiaru kontrolowana indywidualnie dla komponentu -  
**setMaximumSize(new Dimension(w,h))**
- Przerwy między elementami:  
przerwy o stałym: **Box.createRigidArea(new Dimension(w,h))**  
zmiennym rozmiarze: **Box.createVerticalGlue(), Box.createHorizontalGlue()**

# Panele

- Panele to kolejny typ "kontenerów" Javy.
- Możliwe jest ich zagnieżdżanie;
- Możliwość umieszczania na nich grafiki;

```
JPanel panel1=new JPanel();
panel1.setPreferredSize(new Dimension(460,468));
panel1.setBackground(white);
panel1.setLayout(new BorderLayout());
...
add(panel1);
```

# Przykład





# 1. Dodanie bibliotek

- **import** java.awt.\*;
- **import** javax.swing.\*;
- **public class** rozmieszczenie **extends** JFrame
- {
- **static final int** *col\_num*=5;
- **public static void** main(String args[])
- { ...

## Przygotowanie definicji layout'ow -2

- String **rodzaj\_layout**[]={"Border Layout",
- "Flow layout",
- "Flow RIGHT/LEFT",
- "GridLayout (n,m),,
- };
- LayoutManager **Menadzer\_loyaout**[]={
- new BorderLayout(),
- new FlowLayout(),
- new FlowLayout(FlowLayout.*RIGHT*,20,20),
- new GridLayout(*col\_num*,*col\_num*),
- };

## Cd. - 3

- String **uklad**[]=  
{"West","North","East","South", "Center"};
- Color **kolory**[]=  
{
  - new Color(200,255,155),
  - new Color(100,150,100),
  - new Color(181,215,165),
  - new Color(155,155,160),
- };

# Przygotowanie okna - 4

- //mamy: rodzaje layout'ów → string
  - //uklad → string
  - //kolory → Color
  - //menadzer layout'ów → LayoutManager
- 
- JFrame frame=**new** JFrame("ZACZYNAMY");
  - frame.setLayout(**new** GridLayout(3,2));

- **for**(int i=0;i<rodzaj\_layout.length;i++)
- {
- **Panel** p=**new Panel**();
- p.setLayout(Menadzer\_loyaout[i]);
- **for**(int j=0;j<col\_num;j++)
- {
- Button b=**new** Button("Przycisk nr: "+(j+1));
- **if**(i==0)p.add(b,uklad[k++]);
- **else** p.add(b);
- }

## dalej w pętli for ...

- Panel **p1=new** Panel();
- p1.setLayout(**new** BorderLayout());
- Label **lab=new** Label(rodzaj\_layout[i]+":");
- lab.setFont(**new** Font("Dialog",Font.*BOLD*,26));
- lab.setBackground(kolory[i]);
  
- p1.add(lab,"North");
- p1.add(p,"Center");
- **frame.add(p1);**

# kończymy

- `frame.setDefaultCloseOperation(frame.EXIT_ON_CLOSE);`
- 
- `frame.pack();`
- `frame.setVisible(true);`

# Zdarzenia i obsługa zdarzeń

- Zdarzenie niskiego poziomu to wciśnięcie klawisza klawiatury lub kliknięcie przycisku myszki.

**KeyEvent, MouseEvent, MouseMotionEvent**



- Jeżeli jesteśmy zainteresowani reagowaniem na wystąpienie konkretnego zdarzenia, musimy w kodzie programu zidentyfikować źródło zdarzenia,
- Dla akcji użytkownika polegających na wciśnięciu przycisku programowego, źródłem jest przycisk (obiekt typu JButton)

- *Add*???*Listener*(???)

Źródło zdarzenia

Kod obsługi umieszczony w obiekcie ???

- zdarzeniem generowanym przez obiekty typu '**przycisk**' są zdarzenia typu '**akcja**'
- **addActionListener(.)**
- dla zdarzeń typu akcja, metodą uruchamianą przez JVM jest metoda
- **public void actionPerformed(ActionEvent e)**
- **{...}**
- informacja o implementacji odpowiedniego interfejsu programowego
- **java.awt.event.**

# przykład

- public class przyklad implements ActionListener{
- public void **actionPerformed**(ActionEvent e){
- String tekst=e.getActionCommand();
- System.out.println(tekst);
- }
- ...przykład(){
- JButton b = new JButton("przycisk");
- ...b.addActionListener(this);
- ...
- }

- **import** java.awt.event.\*;
- **import** javax.swing.\*;
- **class** obrazek **extends** JFrame **implements** ActionListener{
- **int** count=0;
- JButton b;
- obrazek(){
  - b = **new** JButton(„licznik: 0”);
  - getContentPane().add(b);
  - b.addActionListener(**this**);
  - pack();
  - }

```
public void
actionPerformed(ActionEvent e){

 count++;

 b.setText(„licznik:"+count);

}
```

```
public static void main(String[] args) {

 obrazek okno = new obrazek();

 okno.show();

} }
```

# pola tekstowe

- pola służące do wprowadzania tekstu klasa **TextField** , 2 najciekawsze metody :
- `getText()`, `setText`.
- Różne odmiany konstruktorów umożliwiają podanie wielu parametrów w trakcie powstawania obiektu.
- Przyciśnięcie klawisza ENTER klawiatury powoduje generację zdarzenia typu 'akcja',
- reakcja identyczna jak dla przycisków programowych.