Tworzenie interfejsów użytkownika za pomocą biblioteki Swing

mgr inż. Jacek Bzdak

20 marca 2012

Spis treści

Email bzdak@poczta.if.pw.edu.pl
Strona domowa: http://lfitj.if.pw.edu.pl/lfitjcms

Biblioteki do tworzenia GUI w Javie

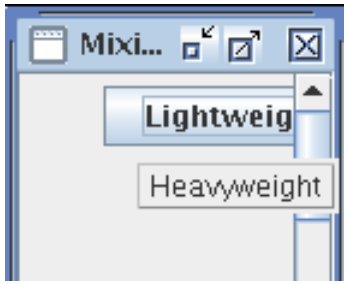
Zacznijmy od ogólnego spojrzenia na biblioteki do tworzenia GUI w Javie.
Ogólnie komponenty dzielimy na:

heavyweight (ciężkie) komponent jest ciężki jeśli jest skonstruowany tak że posiada wskaźnik do komponentu systemu operacyjnego. Czyli guzik na poziomie JVM jest ciężki jeśli posiada odniesienie do systemowego obiektu reprezentującego guzik.

lightweight (lekkie) komponent test lekki jeśli nie posiada referencji do zasobów natywnych.

Jak działają biblioteki lightweight

W swingu tylko okienka są *ciężkie*, pozostałe komponenty są *lekkie*. Lekkie obiekty pojawiają się na ekranie ponieważ delegują ich malowanie na ekranie do komponentów ciężkich do których zostały dodane.
Z tego powodu mogą pojawiać się problemy z położeniem Z (wysokością malowanych nad okienkiem komponentów) kiedy miesza się komponenty ciężkie i lekkie. W najnowszych wersjach JRE problemy te są mniejsze (i pokazywany problem nie będzie się pojawiać).



Rysunek: Problemy przy mieszaniu ciężkich i lekkich komponentów.
`http://java.sun.com/developer/technicalArticles/GUI/mixing_components/index.html`

Lista ciężkich komponentów w Swingu

- `javax.swing.JFrame`
- `javax.swing.JDialog`
- `javax.swing.JWindow`
- `javax.swing.JApplet`

Biblioteki do tworzenia GUI w Javie

AWT (z ang. ***A**bstract **W**idow **T**oolkit*) biblioteka wbudowana w JRE której już w zasadzie nie powinno się używać. Komponenty AWT są **ciężkie**.

Swing biblioteka wbudowana w JRE która jest podstawową metodą budowania GUI w Javie. Komponenty Swinga są **lekkie**.

SWT (*z ang. **S**tandard **W**idget **T**oolkit*) biblioteka rozwijana przez Eclipse Foundation (tak to oni tworzą Eclipse), która ma na celu udostępnienie programiście Javy systemowych komponentów graficznych. Komponenty SWT są **ciężkie**.

JavaFX relatywnie nowa biblioteka która kładzie nacisk na fajerwerki graficzne (na przykład tworzenie wydajnych animacji). Pozwala ona na wykorzystywanie wspomagania sprzętowego. Komponenty JavaFX są **lekkie**.

Porównanie tych bibliotek

AWT Nie używać

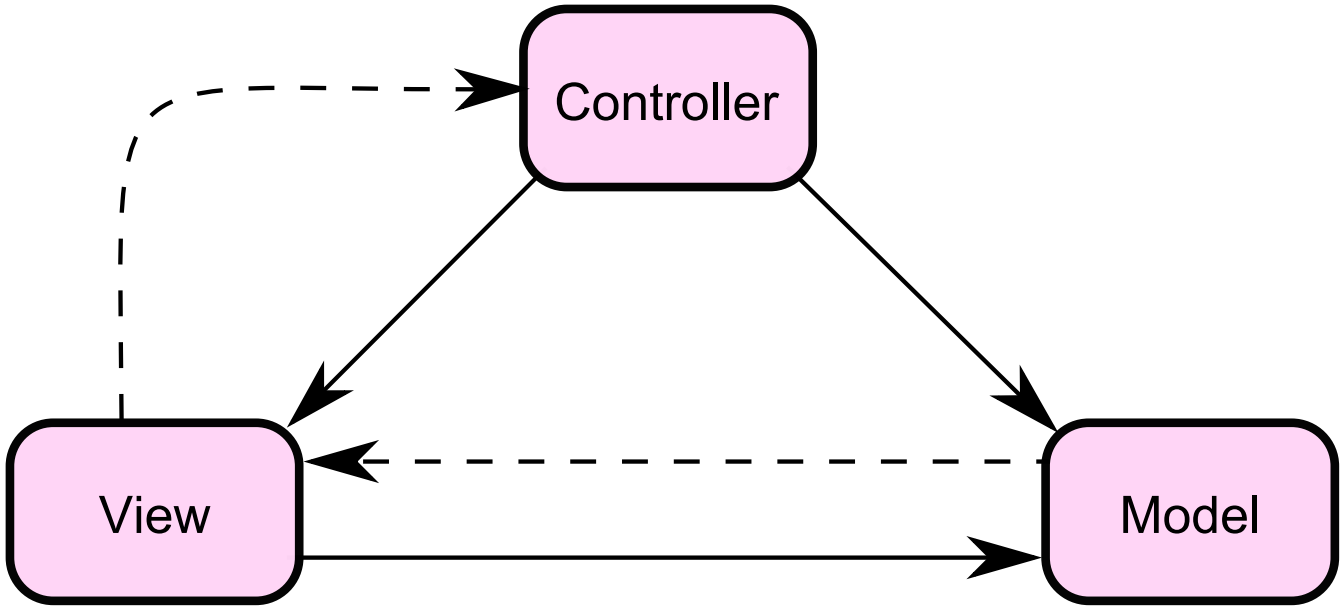
SWT SWT można wykorzystać kiedy zależy nam tym by aplikacja maksymalnie wtapiała w system operacyjny. Niektórzy mówią że SWT jest szybsze niż Swing (i to mogło być prawdą tak do JRE 1.6 kiedy to maszyna wirtualna bardzo przyśpieszyła). Komponenty SWT są też moim zdaniem mniej wygodne w użyciu niż te ze Swinga. Wadą SWT jest to że wymaga ona używania dodatkowych bibliotek (każdy OS ma inny jar)

Swing W zasadzie domyślna opcja tworzenia aplikacji GUI w Javie. API Swinga jest stabilne — było z nami od wersji Javy 1.3 i nie wygląda na to żeby z Javy wyleciało.

JavaFX Użyj JavaFX kiedy interesuje Cie tylko żeby aplikacja była bardzo *ładna*. Wady: na dziś JavaFX2.0 nie ma wsparcia dla Linuksa; API nie jest stabilne — bardzo duże zmiany między wersjami 1.x – 2.0.

Schemat MVC

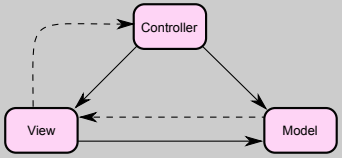
Schemat **M**odel–**V**iew–**C**ontroller jest jednym z najlepiej rozpowszechnionych schematów projektowych w świecie programowania.



Rysunek: Schemat Model View Controller

Schemat MVC

Schemat **M**odel–**V**iew–**C**ontroller jest jednym z najlepiej rozpowszechnionych schematów projektowych w świecie programowania.



Rysunek: Schemat Model View Controller

1. Kontroler jest to ten komponent który odpowiada za interakcje z użytkownikiem — inaczej: reaguje on na zdarzenia od użytkownika.
2. W WWW rolę kontrolera będzie pełnił najpierw serwer który otrzyma pewne żądanie HTTP, a potem mechanizm mapowania ścieżki aplikacji do widoku który wygeneruje stronę.
3. W Swingu, kiedy użytkownik kliknie myszką w okno aplikacji to kontroler wybiera który komponent to zdarzenie otrzyma.
4. Model przechowuje dane — może on notyfikować widok o tym że dane zmieniły się. W aplikacjach WWW jest to generalnie baza danych.
5. Widok wyświetla dane z modelu użytkownikowi.

Dlaczego schemat MVC jest fajny

- Bo można dowolną część z MVC wymienić na inną pozostałe zostawiając nie zmienione.
- Dzięki temu bez specjalnych problemów można napisać Swingową aplikację która służy do edycji bazy danych.
- Analogicznie: zmienianie wyglądu aplikacji jest również dość proste.

Model MVC na przykładzie JTable

- Dlaczego JTable
- W JTable widać wszystkie cechy MVC w Swingu
- Jest chyba najbardziej użytecznym komponentem

Historia

- Załóżmy że chcemy napisać program który pobiera dane o studentach z bazy danych, pozwala je zmienić potem je tam zapisuje.
- Możemy taki program napisać — (no może poza zapisem do bazy danych — zainteresowanych zapraszam na konsultacje, bo to też da się zrobić)

Ramka bez modelu

```
public TableFrame1() throws HeadlessException {
    super("Ramka z tabelką");

    setDefaultCloseOperation(DISPOSE_ON_CLOSE);

    JTable table = new JTable();

    table.setModel(new DefaultTableModel( // Brzydko, ale napiszemy to ładniej!
            new Object[][]{ // Tutaj dodajemy do tabelki dane
                    new Object[]{"Jacek", "Bzdak", "3+", Boolean.TRUE},
                    new Object[]{"Jan", "Kowalski", "5", Boolean.TRUE},
                    new Object[]{"Tomasz", "Frankowski", "4", Boolean.FALSE},
            },
            new Object[]{
                    "Imię", "Nazwisko", "Ocena", "Czy lubi jave" // Tudaj nagłówki
            }
    ));

    this.add(new JScrollPane(table)); // Tabelka która nie jest w
    // JSCroll pane domyślnie nie ma wyświetlonego nagłówka
}
```

Rysunek: Tabelka bez używania MVC

Ramka bez modelu

```
public TableFrame1() throws HeadlessException {
    super("Ramka z tabelką");

    setDefaultCloseOperation(DISPOSE_ON_CLOSE);

    JTable table = new JTable();

    table.setModel(new DefaultTableModel( // Brzydko, ale napiszemy to ładniej
            new Object[][]{ // Tutaj podajemy do tabelki dane
                    new Object[]{"Jacek", "Brzdąk", "3+", Boolean.TRUE},
                    new Object[]{"Jan", "Komalski", "5", Boolean.TRUE},
                    new Object[]{"Tomasz", "Frankowski", "4", Boolean.FALSE},
            },
            new Object[]{
                    "Imię", "Nazwisko", "Ocena", "Czy lubi javę" // Tutaj nagłówki
            }
    ));

    this.add(new JScrollPane(table)); // Tabelka która nie jest w
    // JScroll pane domyślnie nie ma wyświetlonego nagłówka
}
```

Rysunek: Tabelka bez używania MVC

1. Kod jest prosty. Widać tylko że tabelka nie umie ładnie wyświetlić własności Czy lubi Javę. Do tego dodanie do takiej klasy edycji byłoby mordęgą.
2. Jeszcze jedna uwaga: Tabelka zasadniczo zawsze powinna być wewnątrz komponentu `JScrollPane` — jest tak ponieważ bez tego komponentu tabelka nie będzie miała nagłówka — to znaczy można ów nagłówek dodać oddzielnie, ale to jest raczej niewygodne.

Klasa Student

Zacznijmy od klasy która będzie przechowywała informacje o studencie:

```
public class Student implements Serializable{

    private static final long serialVersionUID = 2241878462381472224L;

    String name, surname;

    Double mark;

    Boolean likesJava;

    public Student() {
    }

    Student(String name, String surname, Double mark) {
        this.name = name;
        this.surname = surname;
        this.mark = mark;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getSurname() {
        return surname;
    }
```

Klasa Student

Zacznijmy od klasy która będzie przechowywała informacje o studencie:

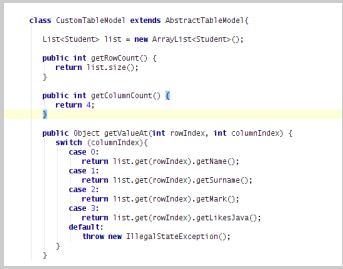
```
public class Student implements Serializable{

   private static final long serialVersionUID = 2241878462381472224L;

   String name, surname;

   Double mark;

   Boolean ItKesJava;

   public Student() {
   }

   Student(String name, String surname, Double mark) {
      this.name = name;
      this.surname = surname;
      this.mark = mark;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getSurname() {
      return surname;
   }
```

1. W zasadzie jest to standardowe ziarno Javy — w którym nie ma żadnej większej logiki
2. Instancje tej klasy będę *bezpośrednio wyświetlane w tabelce*.
3. Proszę państwa w Javie stworzenie takiego ziarna jest raczej standardową praktyką. Istnieją mechanizmy które są w stanie umieszczać instancje takich klas w relacyjnej bazie danych.

Model

```
class CustomTableModel extends AbstractTableModel{

   List<Student> list = new ArrayList<Student>();

   public int getRowCount() {
      return list.size();
   }

   public int getColumnCount() {
      return 4;
   }

   public Object getValueAt(int rowIndex, int columnIndex) {
      switch (columnIndex){
         case 0:
            return list.get(rowIndex).getName();
         case 1:
            return list.get(rowIndex).getSurname();
         case 2:
            return list.get(rowIndex).getMark();
         case 3:
            return list.get(rowIndex).getLikesJava();
         default:
            throw new IllegalStateException();
      }
   }
```

Model



```
class CustomTableModel extends AbstractTableModel{

    List<Student> list = new ArrayList<Student>();

    public int getRowCount() {
        return list.size();
    }

    public int getColumnCount() {
        return 4;
    }

    public Object getValueAt(int rowIndex, int columnIndex) {
        switch (columnIndex){
            case 0:
                return list.get(rowIndex).getName();
            case 1:
                return list.get(rowIndex).getSurname();
            case 2:
                return list.get(rowIndex).getMark();
            case 3:
                return list.get(rowIndex).getLikesJava();
            default:
                throw new IllegalStateException();
        }
    }
```

1. Model dla tabeli musi dziedziczyć po interfejsie `javax.swing.table.TableModel`, ale najlepiej dziedziczyć po klasie `javax.swing.table.AbstractTableModel`.
2. Należy nadpisać trzy metody: `getRowCount`, `getColumnCount oraz getValueAt`.
3. Nasz model tabeli przechowuje dane w zwykłej liście, ale nic nie stoi na przeszkodzie inna klasa pobierała je z bazy danych!
4. Dodatkowo na tym etapie definiujemy jeszcze metodę `getColumnName` która wyświetla nam nazwę danej kolumny.
5. Listę studentów pobieramy i zapisujemy do prostej bazy danych (ale to dopiero na końcu wykładu pokażę)

Tworzenie tabelki z modelem

```
public TableFrame2() throws HeadlessException {
    CustomTableModel model = new CustomTableModel();
    model.addStudents(Database.getStudentsFromDatabase());
    JTable view = new JTable(model);
    this.add(new JScrollPane(view));
    setDefaultCloseOperation(DISPOSE_ON_CLOSE);
}
```

Renderer

- Powiedzmy że chcemy żeby pole „czy lubi Javę“ przyjmowało trzy wartości: Prawda i Fałsz, oraz stan pośredni (reprezentowany w klasie Student wartością null) który oznacza że nie wiadomo czy dana osoba lubi Javę.
- Musimy zatem zmienić sposób w jaki trzecia kolumna jest wyświetlana
- Służą do tego instancje klasy `TableCellRenererr`.

Jak działa renderowanie tabelek

- Wyobraźmy sobie tabelkę która ma rozmiar 10 kolumn i 1000 wierszy. Gdyby każda komórka tabelki była sama w sobie komponentem Swinga podczas jej tworzenia należałoby stworzyć 10 000 obiektów. To za dużo.
- Mądrzy ludzie z Sun wymyślili zatem że każda kolumna będzie renderowana za pomocą *jednego* komponentu swinga, który będzie działał tak taki stempel — ustawiamy jego własności potem jest on malowany w odpowiednim miejscu, a zaraz potem zmieniamy jego własności i malujemy wiersz niżej.

Interfejs `TableCellRenderer`

```
public class CheckboxRenderer implements TableCellRenderer{

   private final Border noFocusBorder = new EmptyBorder(1, 1, 1, 1);

   private final TristateCheckBox checkBox = new TristateCheckBox("");

   public CheckboxRenderer() {
      checkBox.setHorizontalAlignment(SwingConstants.CENTER);
   }

   public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected,
                                                  boolean hasFocus, int row, int column) {
      fixBorder(table, value, isSelected, hasFocus, row, column);
      if (value == null){
         checkBox.setIndeterminate();
      }else{
         checkBox.setSelected((Boolean) value);
      }
      return checkBox;
   }
```

Rysunek:

Interfejs `TableCellRenderer`

```
public class CheckboxRenderer implements TableCellRenderer{

  private final Border noFocusBorder = new EmptyBorder(1, 1, 1, 1);

  private final TristateCheckBox checkBox = new TristateCheckBox("");

  public CheckboxRenderer() {
    checkBox.setHorizontalAlignment(SwingConstants.CENTER);
  }

  public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected,
                                                 boolean hasFocus, int row, int column) {
    fixBorder(table, value, isSelected, hasFocus, row, column);
    if (value == null){
      checkBox.setIndeterminate();
    }else{
      checkBox.setSelected((Boolean) value);
    }
    return checkBox;
  }
```

Rysunek:

1. Metoda `getTableCellRendererComponent` jest wykonywana *przed* namalowaniem każdej komórki w danej kolumnie. W metodzie tej ustawie się własności komponentu, a następnie się ów komponent maluje na tabelce. Po jego namalowaniu przy pomocy tego samego komponentu maluje się następny rząd!
2. Metoda `fixBorder` robi magię która powouje że komponent ma stan który odzwiercielda włąsnosci `hasFocus` i `isSelected`.

Ustawianie renderera w tabelce

```
public TableFrame3() throws HeadlessException {
    setDefaultCloseOperation(DISPOSE_ON_CLOSE);
    studentsModel = new StudentsModel();
    table = new JTable(studentsModel);
    table.setDefaultRenderer(Boolean.class, new CheckboxRenderer());
    this.add(new JScrollPane(table));
}
```

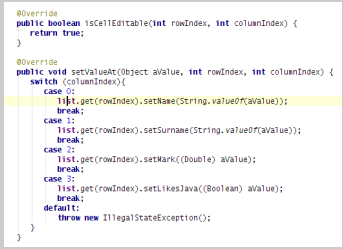
Ustawianie renderera w tabelce

```
public TableFrame3() throws HeadlessException {
    setDefaultCloseOperation(DISPOSE_ON_CLOSE);
    studentsModel = new StudentsModel();
    table = new JTable(studentsModel);
    table.setDefaultRenderer(Boolean.class, new CheckboxRenderer());
    this.add(new JScrollPane(table));
}
```

1. Są dwie metody na ustawienie renderera w tabelce. Pierwsza jest taka że ustalamy że dany renderer będzie renderował wszystkie kolumny w których są instancje klasy `Boolean`. Skąd tabelka ma wiedzieć że w danej kolumnie są instancje `Boolean`? Model informuje o tym tabelkę za pomocą metody `getColumnClass`.
2. Można też w klasie JFrame nadpisać metodę `getRenderer`.

Edytowanie tabelki

```
@Override
public boolean isCellEditable(int rowIndex, int columnIndex) {
   return true;
}

@Override
public void setValueAt(Object aValue, int rowIndex, int columnIndex) {
   switch (columnIndex){
      case 0:
         list.get(rowIndex).setName(String.valueOf(aValue));
         break;
      case 1:
         list.get(rowIndex).setSurname(String.valueOf(aValue));
         break;
      case 2:
         list.get(rowIndex).setMark((Double) aValue);
         break;
      case 3:
         list.get(rowIndex).setLikesJava((Boolean) aValue);
         break;
      default:
         throw new IllegalStateException();
   }
}
```



```
@Override
public boolean isCellEditable(int rowIndex, int columnIndex) {
  return true;
}

@Override
public void setValueAt(Object aValue, int rowIndex, int columnIndex) {
  switch (columnIndex){
    case 0:
      list.get(rowIndex).setName(String.valueOf(aValue));
      break;
    case 1:
      list.get(rowIndex).setSurname(String.valueOf(aValue));
      break;
    case 2:
      list.get(rowIndex).setMark((Double) aValue);
      break;
    case 3:
      list.get(rowIndex).setLikesJava((Boolean) aValue);
      break;
    default:
      throw new IllegalStateException();
  }
}
```

1. Można też tworzyć w tabelce własne edytory — czyli komponenty odpowiedzialne za edycje poszczególnych komórek. Działa to zupełnie analogicznie do Rendererów.

Dodawanie wierszy do tabelki

Do modelu dodajemy dwie metody które zmieniają jego zawartość

```
JMenuItem addNew = new JMenuItem("Dodaj nowego");

addNew.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
      studentsModel.addStudent(new Student());
   }
});

jMenuBar.add(addNew);
```

Podpięcie do akcji w menu

Kiedy mamy model sama manipulacja zawartością jest bardzo prosta:

Zapis do ’bazy danych’

Nasza baza danych korzysta z serializacji:

```
public static void saveToDatabase(List<Student> students) throws IOException{
    ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream(STUDENTS_FILE));
    try{
        outputStream.writeObject(students);
    }finally {
        outputStream.close();
    }
}

private static List<Student> readStudents() throws IOException{
    ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream(STUDENTS_FILE));
    try {
        return (List<Student>) inputStream.readObject();
    } catch (ClassNotFoundException e) {
        throw new IOException("Coulnd't load students", e);
    }finally {
        inputStream.close();
    }
}
```

Inne komponenty Swinga

- Nie mówiłem o tym żeby Państwa nie nudzić! Te komponenty działają dokładnie tak samo jak te znane z AWT.
  - Tak samo rejestruje się na nich listenery
  - Listenery są takich samych klas

Komponenty swinga

`JTextField` Krótkie pole tekstowe

`JComboBox` ComboBox z którego wybiera się jedną wartość

`JButton` Guzik

`JCheckbox` Guzik który jest polem do zanzanczania

`JLabel` krótki napis

Wszystkie są opisane na stronie:
`http://docs.oracle.com/javase/tutorial/ui/features/components.html`,

Tabelka z sortowaniem (i JTextField i JComboBox)

- To wrzuciłem tylko dlatego że komuś kiedyś coś takiego może się przydać
- Jest to w klasie `TableFrame5`.
- Pokaz

JEditorPane

- Komponent który reprezentuje prosty edytor który pozwala na przetwarzanie tekstu który posiada style.
- Pozwala na ładowanie stron WWW.
- Pozwala na stworzenie prostego edytora który potrafi edytować pliki HTML i RTF.
- Generalnie tutorial do JEditor pane jest na stronie: `http://docs.oracle.com/javase/tutorial/uiswing/components/generaltext.html`.

Otwieranie strony w JFrame

By załadować stronę w HTML należy wywołać metodę setPage .

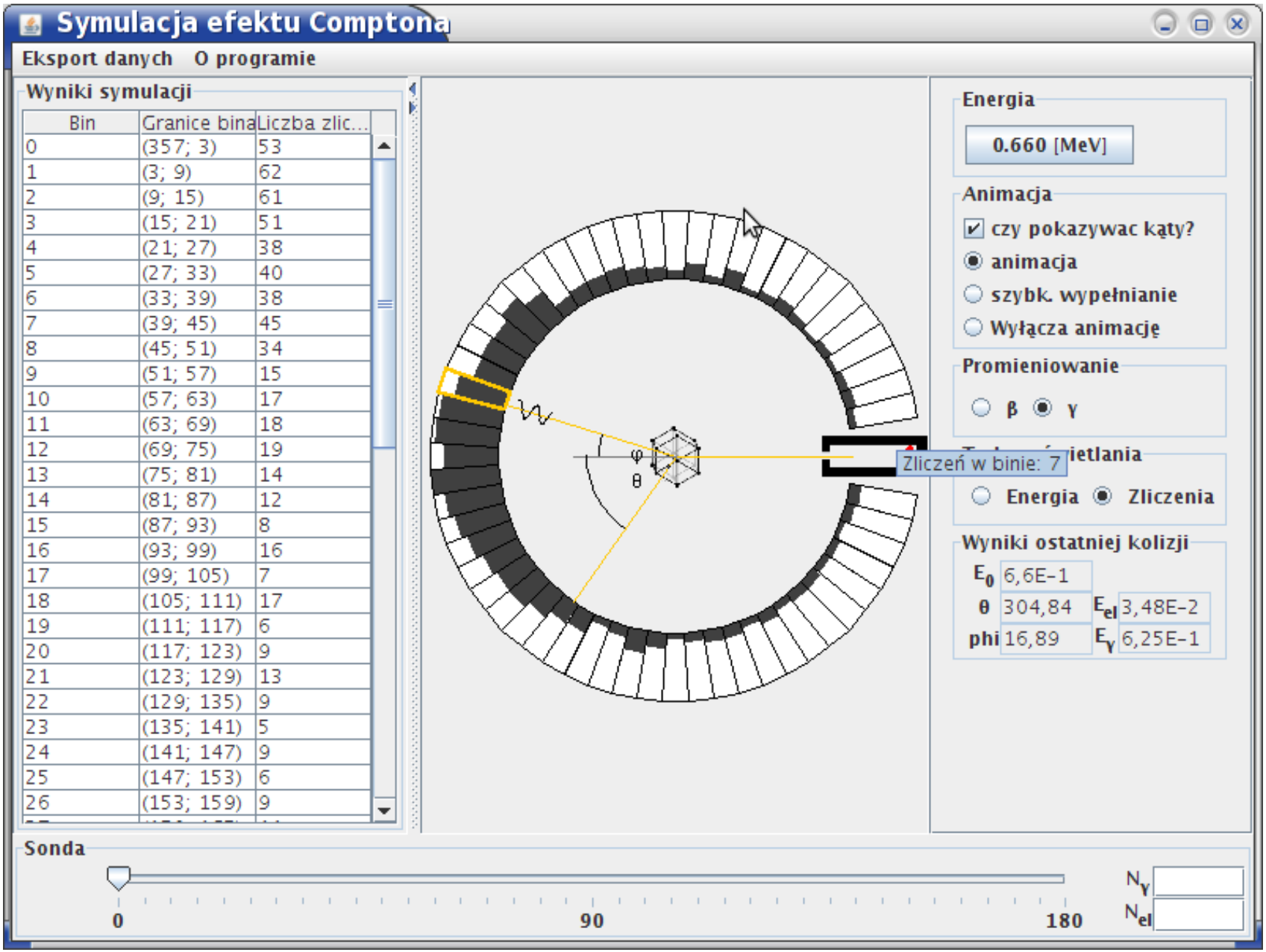
```
editorPane = new JEditorPane();
  URL resource = getClass().getResource("javawiki.html");
URL resource = new URL("http://en.wikipedia.org/wiki/Java_(programming_language)");
editorPane.setPage(resource);
```

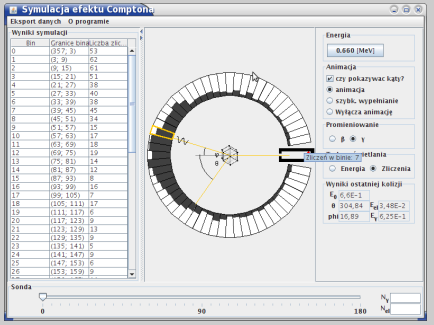
Zmiana stylów

By stworzyć prosty edytor należy po prostu dodać do menu odpowiednie akcje.

```
JMenuBar jMenuBar = new JMenuBar();
JMenu jMenu = new JMenu("Format");

Action action = new StyledEditorKit.BoldAction();
action.putValue(Action.NAME, "Bold");

jMenu.add(action);
jMenuBar.add(jMenu);
setJMenuBar(jMenuBar);
```

Malowanie w wielu warstwach





1. To jest symulacja efektu Comptona którą zrobiłem kiedyś tam dawno dla Wydziału
2. W wielu symulacjach (takich jakie Państwo macie przy zaliczaniu projektów) pojawia się problem malowania w wielu warstwach — jak państwo widzicie w symulacji (pokaz) żółta obwoluta która wskazuje bina do którego wpadnie foton nachodzi (jest nad) innymi obiektami, ale kiedy ona znika ujawniają się one niejako spod niej. Zaimplementowanie takiego zachowania może być kłopotliwe

Przemalowywanie całej ramki

Pierwszym rozwiązaniem jest napisanie metody `paintComponent` tak by działała ona tak:

- Kasujemy całe cały obraz
- Rysujemy wszystko od nowa

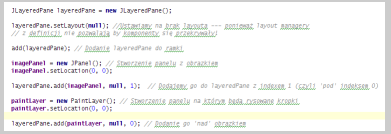
Tutaj jest problem z wydajnością: jeśli mamy symulację w rozdzielczości 1000 x 1000 to 25 razy na sekundę wyświetlamy na ekranie jeden megapiksel nowych pikseli. Biorąc pod uwagę że malowanie w swingu nie wykorzystuje wsparcia sprzętowego — to Państwa symulacja nie będzie bardzo szybka.

Wielowarstwowy komponent

W swingu można tworzyć wielowarstwowe komponenty.

```
JLayeredPane layeredPane = new JLayeredPane();

layeredPane.setLayout(null); //Ustawiamy na brak layouta --- ponieważ layout managery
// z definicji nie pozwalają by komponenty się przekrywały!

add(layeredPane); // Dodanie layeredPane do ramki

imagePanel = new JPanel(); // Stworzenie panelu z obrazkiem
imagePanel.setLocation(0, 0);

layeredPane.add(imagePanel, null, 1);  // Dodajemy go do layeredPane z indexem 1 (czyli 'pod' indeksem 0)

paintLayer = new PaintLayer(); // Stworzenie panelu na którym będą rysowane kropki
paintLayer.setLocation(0, 0);

layeredPane.add(paintLayer, null, 0); // Dodanie go 'nad' obrazkiem
```

W swingu można tworzyć wielowarstwowe komponenty.



1. Pokaz
2. To też nie będzie wydajne! Tak na prawdę będzie gorsze niż przemalowywanie całości! Bowiem OBA komponenty zostaną przemalowane.

Opaque

- Własność `opaque` mówi czy dany komponent zapełnia wszystkie piksele pod nim.
- Jeśli jakiś komponent jest opaque — znaczy to że komponenty *pod nim* nie będą w ogóle odświeżane.
- W naszym przykładzie trzeba było ustawić na *górnym* komponencie własność *`opaque`* na true.

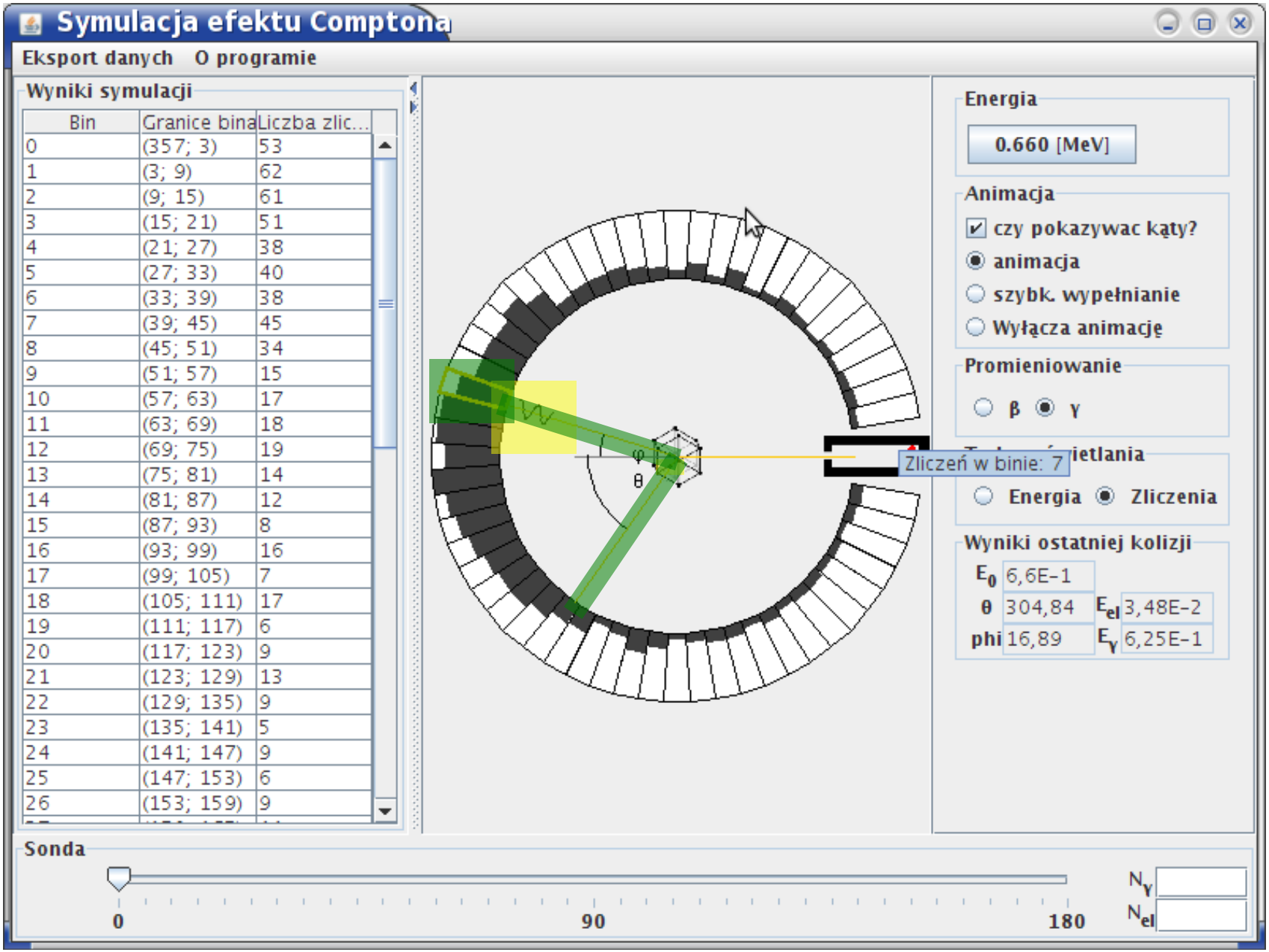
Wydajne malowanie

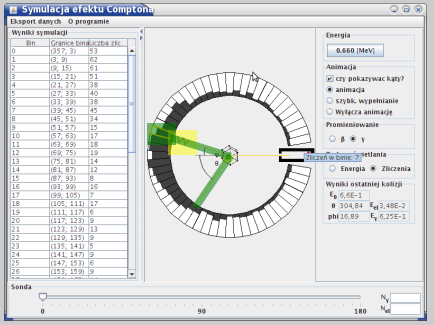
- Udostępniona jest metoda `repaint(int x, int y, int width, int height)` która odświerza tylko zadany kawałek panelu.
- Po zastosowaniu jej zamiast `repaint()` odświeżone zostaną tylko wybrane kawałki panelu.

Wydajne malowanie (przykład)

```
if ((redSquare.getX()!=x) || (redSquare.getY()!=y)) {

    // The square is moving, repaint background
    // over the old square location.
   int OLD_X = redSquare.getX();
   int OLD_Y = redSquare.getY();

   // Update coordinates.
   redSquare.setX(x);
   redSquare.setY(y);

   repaint(OLD_X, OLD_Y,redSquare.getWidth()+1,redSquare.getHeight()+1);

    // Repaint the square at the new location.
    repaint(redSquare.getX(), redSquare.getY(),
            redSquare.getWidth()+1,
            redSquare.getHeight()+1);
}
```

Co wymaga przemalowania





1. Co klatkę trzeba jeszcze raz namalować obszary które są zaznaczone na żółto, po zmianie konta takie które są zaznaczone na niebiesko.
2. Gdybym o tym wiedział kiedy pisałem tą symulację ;)