Wielodziedziczenie klas a interfejsów

Piotr Laskowski, Winicjusz Szyszka

Plan

- Czym jest wielodziedziczenie?
- Wielodziedziczenie klas – teoria i praktyka
- Interfejsy – teoria i praktyka
- Wady tych rozwiązań
- Co lepsze i dlaczego?

WIELODZIEDZICZENIE

- Cóż to takiego?
- Wielodziedziczenie klas (C++, Perl, Python)
- Interfejsy (Java, PHP, C#)

Wielodziedziczenie klas

- Jedno dziecko wiele matek
- Prostota i szybkość tworzenia
- Intuicyjność
- Łatwe zmiany
- Zobaczmy jak to wygląda w praktyce

Praktyka

Interfejsy

- Inny rodzaj dziedziczenia
- Pełna enkapsulacja
- Wygoda

Praktyka

- package java.util;
- **public class ArrayList<E>**
- **extends AbstractList<E>**
- **implements List<E>, RandomAccess, Cloneable, Serializable**

Praktyka

```
class ObjectListHandler extends Object implements
    ActionListener, ListSelectionListener {
   void actionPerformed(ActionEvent e) { // zaznaczenie
    checkbox-a
      this.isOptionEnabled !=  this.isOptionEnabled;
      this.showHideList(this.isOptionEnabled);
   }
   void valueChanged(ListSelectionEvent e) { // zmiana
    zaznaczenia listy
     if(this.isOptionEnabled) this.refilterObjects();
   }
   ...
}
```

Praktyka

```
class VideoList extends AbstractListModel implements TableModel, ListModel {
    private List<Video> source;
    public VideoList(List<Video> source) {
        this.source = source;
    }
    public Object getElementAt(int i) { // for ListModel
        return source[i].getTitle();
    }
    public Object getValueAt(int i, int j) { // for TableModel
        Video obj = source[i];
        switch(j) {
            case 0: return obj.getTitle();
            case 1: return obj.getThumb();
            case 2: return obj.getAuthor();
            case 3: return obj.getDate();
        }
    }
}
```

Praktyka

```
package javax.swing;
interface SwingConstants {
  static int  BOTTOM;
  static int  CENTER;
  static int  EAST;
  static int  HORIZONTAL;
  static int  LEADING;
  static int  LEFT;
  static int  NEXT;
  ...
}
```

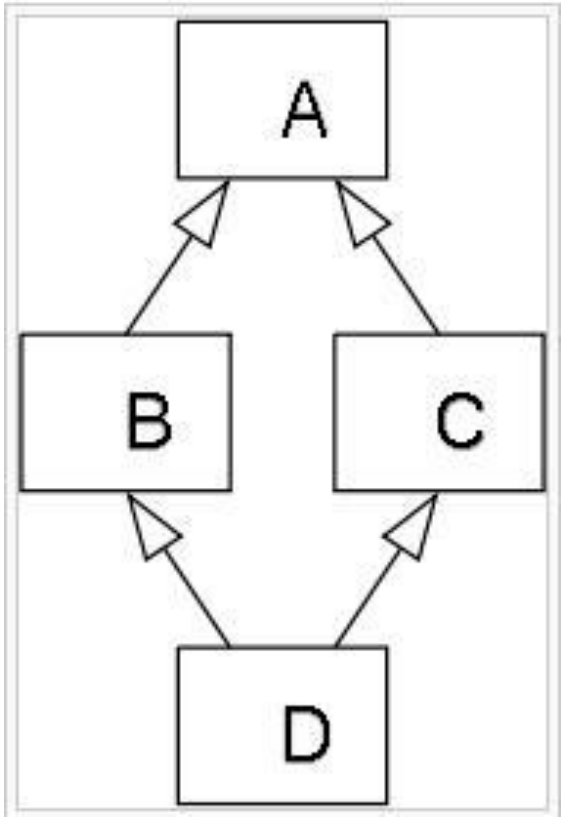
Nic nie jest idealne, nawet interfejsy

- Pracochłonność przy tworzeniu
- Pracochlonnośc przy zmianach
- Kolizje nazw

Wielodziedziczenie klas jest gorsze!

- Niejednoznaczność
- Diamond problem
- Dużo błędów (potencjalnie)

Diamond problem



Klasy > Interfejsy

- Łatwość tworzenia i intuicyjność
- Szybkość wprowadzania zmian

Interfejsy > Klasy

- Większa przejrzystość
- Swoboda implementacji
- Większa modularność/niezależność

Czas na podsumowanie