Modelowanie i Analiza Systemów informacyjnych (MAS)

dr inż. Mariusz Trzaska
mtrzaska@pjwstk.edu.pl

Wykład 2

Wybrane konstrukcje obiektowych języków programowania (1)

Zagadnienia

- Przykładowe zadania programistyczne
- Podstawy
- Kontrolowanie sterowania
- Klasy
- Interfejsy
- Obsługa błędów
- Pojemniki
- System we/wyj
- Wydajność
- Podsumowanie

Wykorzystano materiały z Thinking in Java (3rd edition) autorstwa Bruce'a Eckel'a (http://www.mindview.net/Books/TIJ)

Przykładowe zadania programistyczne

- Niniejsze zadania programistyczne mają na celu jedynie przypomnienie materiału dotyczącego programowania (przyswojonego w czasie wcześniejszych kursów).
- Ich rozwiązania nie będą oceniane w ramach przedmiotu MAS.
- Mogą być wykorzystane w czasie zajęć „*Wybrane konstrukcje obiektowych języków programowania*".
- Studenci przystępujący do kursu MAS, powinni umieć rozwiązać zdecydowaną większość z nich.

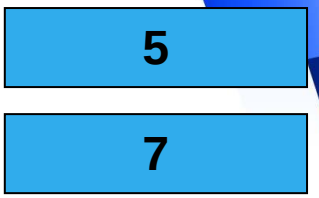
http://www.mtrzaska.com/plik/mas/mas-zadania-programistyczne

Typy podstawowe

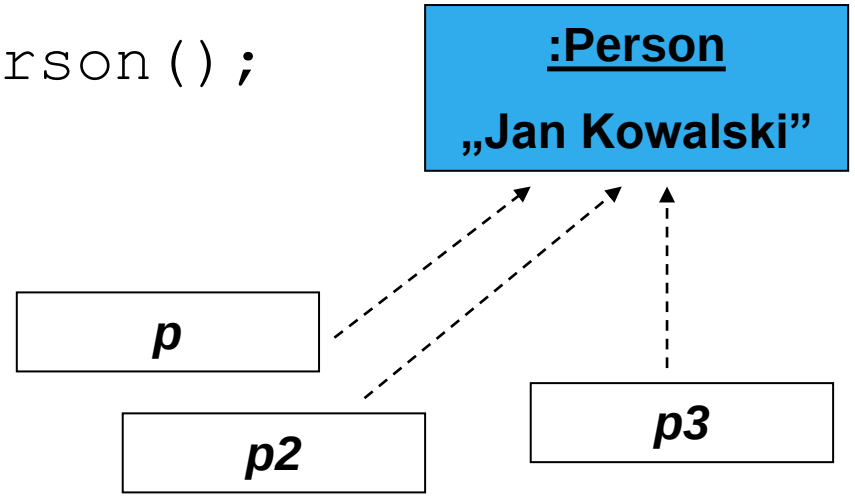
Primitive type	Size	Minimum	Maximum	Wrapper type
`boolean`	—	—	—	`Boolean`
`char`	16-bit	Unicode 0	Unicode 2^{16}- 1	`Character`
`byte`	8-bit	-128	+127	`Byte`
`short`	16-bit	-2^{15}	$+2^{15}$—1	`Short`
`int`	32-bit	-2^{31}	$+2^{31}$—1	`Integer`
`long`	64-bit	-2^{63}	$+2^{63}$—1	`Long`
`float`	32-bit	IEEE754	IEEE754	`Float`
`double`	64-bit	IEEE754	IEEE754	`Double`
`void`	—	—	—	`Void`

Obiekty, a referencje

- Typ podstawowy
  - `int a = 5;`
  - `Int b = 7;`



- Obiekt dostępny przez referencję
  - `Person p = new Person();`
  - `Person p2 = p;`
  - `Person p3 = p;`



Klasy opakowujące (wrapper)

- `char c = 'x';`
- `Character C = new Character(c);`
- `Character C = new Character('x');`
- Powody opakowywania?

Zakres

```
{
  int x = 12;
  // Tylko x jest dostępny
  {
    int q = 96;
    // x & q dostępne
  }
  // Tylko x jest dostępny
}
{
  String s = new String("a string");
} // koniec zakresu
```

Klasy

```
class ClassName {
   /* Ciało klasy */
}

class DataOnly {
      int i;
      float f;
      boolean b;
}
// Stworzenie obiektu
DataOnly d = new DataOnly( );
```

Klasy (2)

Primitive type	Default
boolean	false
char	‘\u0000’ (null)
byte	(byte)0
short	(short)0
int	0
long	0L
float	0.0f
double	0.0d

Dostęp do pól

```
d.i = 47;
d.f = 1.1f;
d.b = false;
```

Inicjalizacja - konstruktor

Metody

- Składnia

```
returnType nazwaMetody( /* Argumenty*/ ) {
   /* Ciało */
}
```

- Metoda pobiera string, a zwraca int:

```
int storage(String s) {
  return s.length( ) * 2;
}
```

- Wiele argumentów dla metody

```
objectName.methodName(arg1, arg2, arg3);
```

Pierwszy program

```
package com.mt.mas;

import java.time.LocalDate;

public class Main {
    // metoda „startowa”
    public static void main(String[] args) {
        System.out.println("Hello, it's: ");
        System.out.println(LocalDate.now());
    }
}
```

Komentarze

```
/* To jest komentarz
      zawierający
      wiele linii. */

// A to jest komentarz w jednej linii
```

- Dokumentowanie kodu źródłowego (javadoc)

```
/** Opis klasy */
public class DocTest {

      /** opis atrybutu */
      public int i;

      /** Opis metody */
      public void f( ) {}
}
```

Jakość kodu

- Język: polski, angielski?
- Nazwy:
  - Klas
  - Metod
  - Zmiennych
- Podział na mniejsze fragmenty

Jakość kodu (2)

- Samodokumentujący/czytelny kod
  - *Robert C. Martin: „Czysty kod. Podręcznik dobrego programisty”*
- Konwencje
- Formatowanie kodu źródłowego
  - Wcięcia,
  - Nawiasy { } – czy zawsze warto ich używać?
  - tabs vs spaces.
- Refaktoryzacja kodu (*refactoring)*.

Rzutowanie (casting)

- Po co?

- Przykład

```
void casts( ) {
  int i = 200;
  long l = (long)i;
  long l2 = (long)200;
}
```

Kontrola sterowania

- If

```
if(Boolean-expression)
   statement
else
   statement
```

- return
- Iteracja

```
while(Boolean-expression)
   statement
```

Kontrola sterowania (2)

- do-while

```
do
  statement
while(Boolean-expression);
```

- `break, continue`
- for

```
for(initialization; Boolean-expression; step)
  statement
```

Kontrola sterowania (4)

- switch

```
switch(integral-selector) {
        case integral-value1:
            statement;
            break;
        case integral-value2:
            statement;
            break;
        // ...

        default: statement;
}
```

Interfejsy

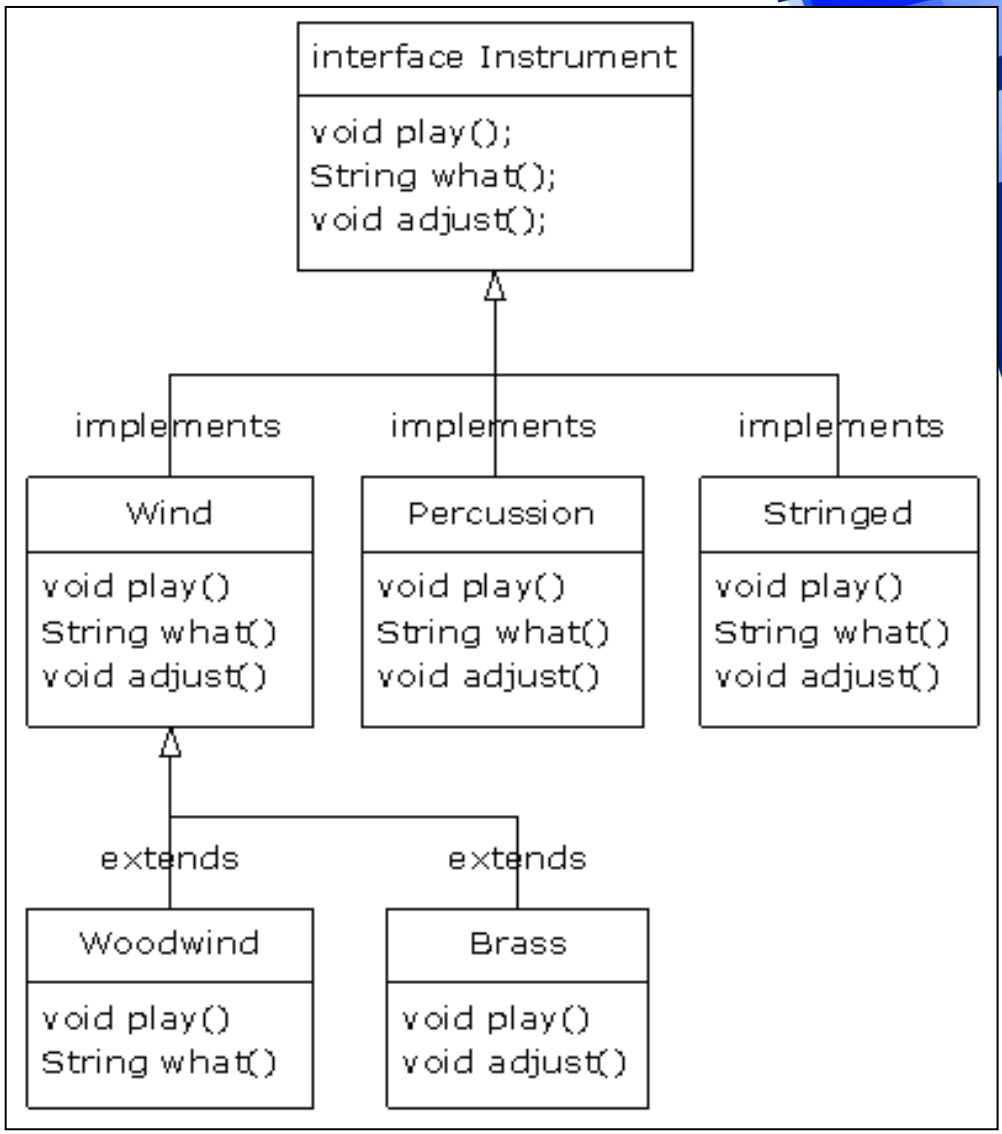
- Cel
- Przykład

```
interface Instrument {
  // Stała:
  // static & final
  int I = 5;

  // tylko deklaracja
  // metoda publiczna
  void play(Note n);

  String what();

  void adjust();
}
```

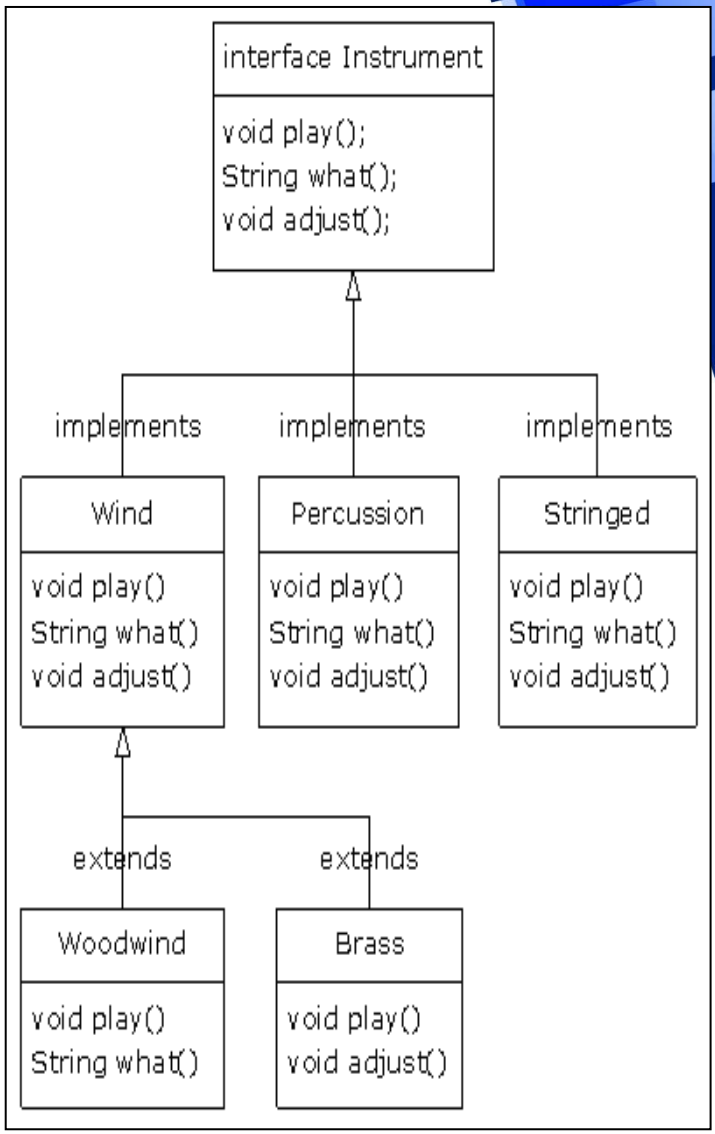


Interfejsy (2)

```
Implementacja interfejsu
class Wind implements Instrument {
  // wind - instrumenty dęte
  public void play(Note n) {
    System.out.println("Wind.play() " + n);
  }
  public String what() { return "Wind"; }
  public void adjust() {}
}

class Brass extends Wind {
  public void play(Note n) {
    System.out.println("Brass.play() " + n);
  }
  public void adjust() {
    System.out.println("Brass.adjust()");
  }
}

class Woodwind extends Wind {
  public void play(Note n) {
    System.out.println("Woodwind.play() " + n);
  }
  public String what() { return "Woodwind"; }
```



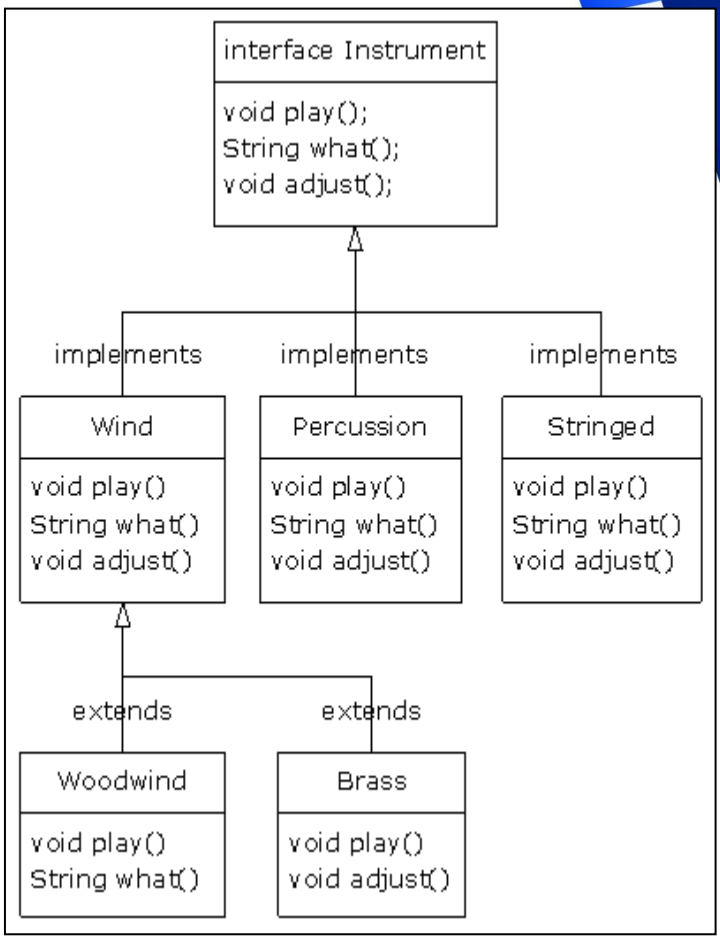
Interfejsy (3)

- Wykorzystanie interfejsów

```
Instrument currentInstrument = null;

currentInstrument = new Brass();
currentInstrument.play(1);


currentInstrument = new Woodwind();
currentInstrument.play(1);
```



- Rozwiązanie z:
  - Klasą bazową,
  - Interfejsami.

Obsługa błędów

- Błędy:
  - Kompilacji,
  - Czasu wykonania
- Jak to robiono kiedyś?
  - Ustawianie zmiennej globalnej,
  - Zwracanie specjalnej wartości (jakiej?)
- Współczesne podejście – wyjątki
  - Zalety
  - Wady (?)

Wyjątki

- Klasa
- Dziedziczenie z klasy Exception
- Argumenty
- Łapanie wyjątków i ich obsługa

```
try {
    // kod, który może spowodować wyjątek
} catch(Type1 id1) {
    // Obsługa wyjątków typu Type1
} catch(Type2 id2) {
    // Obsługa wyjątków typu Type2
}
```

Wyjątki (2)

- Metoda, która nie rzuca wyjątków na zewnątrz i obsługuje je „wewnątrz”

```
void f( ) { // ...
```

- Metoda, która „ostrzega”, że może rzucić wyjątek

```
void f( ) throws TooBigException,DivZeroException {
   //...
```

- Które podejście jest lepsze?

Wyjątki (3)

- Łapanie wszystkich wyjątków

```
catch(Exception e) {
  System.err.println("Caught an exception");
}
```

- Ponowne rzucanie wyjątku

```
catch(Exception e) {
    System.err.println("An exception was thrown");
    throw e;
}
```

- Specjalny przypadek `NullPointerException`

```
person.showName();
```

Wyjątki (4)

- Sprzątanie z finally

```
try {
  // Chroniony region: spodziewamy się wyjątków:
   // A, B, lub C
} catch(A a1) {
  // Obsługa A
} catch(B b1) {
  // Obsługa B
} catch(C c1) {
  // Obsluga C
} finally {
  // Wykonywany zawsze!
}
```

Używanie stałych

- Różne zastosowania
- Rozwiązanie klasyczne (nie polecane!)

```
public class MainMenu {
  public static final int MENU_FILE   = 0;
  public static final int MENU_EDIT   = 1;
  public static final int MENU_FORMAT = 2;
  public static final int MENU_VIEW   = 3;
}
```

- Nowe rozwiązanie z typem wyliczeniowym (enum)

```
public enum MainMenu {FILE, EDIT, FORMAT, VIEW};
```

- Ciąg dalszy na następnym wykładzie…