Język programowania Java

Wojciech Rząsa
wrzasa@prz-rzeszow.pl

Katedra Informatyki i Automatyki, Politechnika Rzeszowska

16 kwietnia 2015

Plan

Plan

Wstęp

- Obiektowy język programowania

Wstęp

- Obiektowy język programowania
- James Gosling, Sun Microsystems

Wstęp

- Obiektowy język programowania
- James Gosling, Sun Microsystems
- Powstał na wzór języków C++ i Smalltalk

Wstęp

- Obiektowy język programowania
- James Gosling, Sun Microsystems
- Powstał na wzór języków C++ i Smalltalk
- Maszyna wirtualna

Wstęp

- Obiektowy język programowania
- James Gosling, Sun Microsystems
- Powstał na wzór języków C++ i Smalltalk
- Maszyna wirtualna
- Maszyna wirtualna, przenośność

Wstęp

- Obiektowy język programowania
- James Gosling, Sun Microsystems
- Powstał na wzór języków C++ i Smalltalk
- Maszyna wirtualna
- Maszyna wirtualna, przenośność
- Silne typowanie (brak automatycznego rzutowania)

Java a C++

- Brak automatycznej konwersji typów

Java a C++

- Brak automatycznej konwersji typów
- Brak wskaźników

Java a C++

- Brak automatycznej konwersji typów
- Brak wskaźników
- Zmienne są referencjami

Java a C++

- Brak automatycznej konwersji typów
- Brak wskaźników
- Zmienne są referencjami
- Tworzenie każdego obiektu „dynamicznie”, za pomocą `new`

Java a C++

- Brak automatycznej konwersji typów
- Brak wskaźników
- Zmienne są referencjami
- Tworzenie każdego obiektu „dynamicznie”, za pomocą `new`
- Automatyczne zarządzanie pamięcią

Java a C++

- Brak automatycznej konwersji typów
- Brak wskaźników
- Zmienne są referencjami
- Tworzenie każdego obiektu „dynamicznie”, za pomocą `new`
- Automatyczne zarządzanie pamięcią
- Jednokrotne dziedziczenie

Java a C++

- Brak automatycznej konwersji typów
- Brak wskaźników
- Zmienne są referencjami
- Tworzenie każdego obiektu „dynamicznie”, za pomocą `new`
- Automatyczne zarządzanie pamięcią
- Jednokrotne dziedziczenie
- Interfejsy

Java a C++

- Brak automatycznej konwersji typów
- Brak wskaźników
- Zmienne są referencjami
- Tworzenie każdego obiektu „dynamicznie”, za pomocą `new`
- Automatyczne zarządzanie pamięcią
- Jednokrotne dziedziczenie
- Interfejsy
- Pakiety zamiast `namespace`

Hello world

```
1 public class Hello {
2   public static void main(String[] args) {
3     System.out.println("Hello world");
4   }
5 }
```

Hello world w Javie, plik `Hello.java`

- Nazwa klasy: `Hello`, nazwa pliku: `Hello.java`

Hello world

```
1 public class Hello {
2   public static void main(String[] args) {
3     System.out.println("Hello world");
4   }
5 }
```

Hello world w Javie, plik `Hello.java`

- Nazwa klasy: `Hello`, nazwa pliku: `Hello.java`
- `main`, jak w C/C++

Hello world

```
1 public class Hello {
2   public static void main(String[] args) {
3     System.out.println("Hello world");
4   }
5 }
```

Hello world w Javie, plik `Hello.java`

- Nazwa klasy: `Hello`, nazwa pliku: `Hello.java`
- `main`, jak w C/C++
- `System.out.println` do wypisywania...

Hello world

```
1 public class Hello {
2   public static void main(String[] args) {
3     System.out.println("Hello world");
4   }
5 }
```

Hello world w Javie, plik `Hello.java`

- Nazwa klasy: `Hello`, nazwa pliku: `Hello.java`
- `main`, jak w C/C++
- `System.out.println` do wypisywania...
- Kompilacja: `javac Hello.java` (utworzy plik `Hello.class`)

Hello world

```
1 public class Hello {
2   public static void main(String[] args) {
3     System.out.println("Hello world");
4   }
5 }
```

Hello world w Javie, plik `Hello.java`

- Nazwa klasy: `Hello`, nazwa pliku: `Hello.java`
- `main`, jak w C/C++
- `System.out.println` do wypisywania...
- Kompilacja: `javac Hello.java` (utworzy plik `Hello.class`)
- Uruchomienie `java Hello`

Środowisko programistyczne (IDE)

- Kolorowanie składni
- Podpowiadanie
- Zarządzane projektem (tworzenie plików, usuwanie itp)
- Kompilacja
- Uruchomienie
- Debugowanie
- ...

Plan

Zmienne

```
1 typ nazwa;
```

Składnia deklaracji zmiennej

Zmienne

```
1 typ nazwa;
```

Składnia deklaracji zmiennej

```
1 int i;
```

Przykład deklaracji zmiennej

Zmienne

```
1 typ nazwa;
```

Składnia deklaracji zmiennej

```
1 int i;
```

Przykład deklaracji zmiennej

```
1 typ nazwa = wartosc;
```

Składnia inicjalizacji zmiennej

Zmienne

```
1 typ nazwa;
```

Składnia deklaracji zmiennej

```
1 int i;
```

Przykład deklaracji zmiennej

```
1 typ nazwa = wartosc;
```

Składnia inicjalizacji zmiennej

```
1 int i = 10;
```

Przykład inicjalizacji zmiennej

Typy

`byte` 8 bitów, ze znakiem (-128 do 127)

Typy

`byte` 8 bitów, ze znakiem (-128 do 127)

`short` 16 bitów, ze znakiem (-32768 do 32767)

Typy

`byte` 8 bitów, ze znakiem (-128 do 127)

`short` 16 bitów, ze znakiem (-32768 do 32767)

`int` 32 bitów, ze znakiem (-2^{31} do $2^{31}+1$)

Typy

`byte` 8 bitów, ze znakiem (-128 do 127)

`short` 16 bitów, ze znakiem (-32768 do 32767)

`int` 32 bitów, ze znakiem (-2^{31} do $2^{31}+1$)

`long` 64 bity, ze znakiem (-2^{63} do $2^{63}-1$)

Typy

`byte`	8 bitów, ze znakiem (-128 do 127)
`short`	16 bitów, ze znakiem (-32768 do 32767)
`int`	32 bitów, ze znakiem (-2^{31} do $2^{31}+1$)
`long`	64 bity, ze znakiem (-2^{63} do $2^{63}-1$)
`float`	pojedynczej precyzji wg. IEEE 754, 32 bity

Typy

`byte`	8 bitów, ze znakiem (-128 do 127)
`short`	16 bitów, ze znakiem (-32768 do 32767)
`int`	32 bitów, ze znakiem (-2^{31} do $2^{31}+1$)
`long`	64 bity, ze znakiem (-2^{63} do $2^{63}-1$)
`float`	pojedynczej precyzji wg. IEEE 754, 32 bity
`double`	podwójnej precyzji wg. IEEE 754, 64 bity

Typy

`byte`	8 bitów, ze znakiem (-128 do 127)
`short`	16 bitów, ze znakiem (-32768 do 32767)
`int`	32 bitów, ze znakiem (-2^{31} do $2^{31}+1$)
`long`	64 bity, ze znakiem (-2^{63} do $2^{63}-1$)
`float`	pojedynczej precyzji wg. IEEE 754, 32 bity
`double`	podwójnej precyzji wg. IEEE 754, 64 bity
`boolean`	`true`/`false`

Typy

`byte`	8 bitów, ze znakiem (-128 do 127)
`short`	16 bitów, ze znakiem (-32768 do 32767)
`int`	32 bitów, ze znakiem (-2^{31} do $2^{31}+1$)
`long`	64 bity, ze znakiem (-2^{63} do $2^{63}-1$)
`float`	pojedynczej precyzji wg. IEEE 754, 32 bity
`double`	podwójnej precyzji wg. IEEE 754, 64 bity
`boolean`	`true`/`false`
`char`	znak, 16-bit Unicode

Typ `String`

- Nie jest wbudowany w język

Typ `String`

- Nie jest wbudowany w język
- Elementy ułatwiające korzystanie są wbudowane w język

Typ String

- Nie jest wbudowany w język
- Elementy ułatwiające korzystanie są wbudowane w język

```
1 String imie = "Jan";
2 String nazwisko = "Kowalski";
3 int wiek = 34;
4
5 String intro = imie + " " + nazwisko + " " + wiek;
6
7 System.out.println(intro);
```

Przykład użycia String

Typ String

- Nie jest wbudowany w język
- Elementy ułatwiające korzystanie są wbudowane w język

```
1 String imie = "Jan";
2 String nazwisko = "Kowalski";
3 float srednia = 4.5567678f;
4
5 String intro = imie + " " + nazwisko;
6 intro = String.format("%s średnia: %.3f", intro, srednia);
7
8 System.out.println(intro);
```

Przykład użycia `String.format`

Plan

Składnia

- Oparta na C/C++

Instrukcja warunkowa

```
1 int i = 4;
2 if (i % 2 == 0) {
3   System.out.println("Parzyste");
4 } else {
5   System.out.println("Nieparzyste");
6 }
```

Instrukcja `if`

- Składnia jak w C/C++
- Warunek musi być typu logicznego, nie może być: `if (i % 2)`

Instrukcja wyboru

```
1  int dzien = 4;
2  String nazwa;
3  switch(dzien) {
4    case 1: nazwa = "niedziela";
5            break;
6    case 2: nazwa = "poniedziałek";
7            break;
8    case 3: nazwa = "wtorek";
9            break;
10           /* ... i tak dalej */
11   default: nazwa = "??";
12            break;
13 }
```

Instrukcja `switch`

Pętla `for`

```
1 for(int i = 0; i < 10; i++) {
2   System.out.println(i);
3 }
```

Instrukcja `for`

- Składnia jak w C/C++
- Warunek musi być typu logicznego

Pętla while

```
1 int i = 10;
2 while(i > 0) {
3   System.out.println(i);
4   i--;
5 }
```

Instrukcja while

- Składnia jak w C/C++
- Warunek musi być typu logicznego

Pętla do while

```
1 int i = 10;
2 do {
3   System.out.println(i);
4   i--;
5 } while(i > 0);
```

Instrukcja do while

- Składnia jak w C/C++
- Warunek musi być typu logicznego

Plan

Tablice

- Mogą przechowywać wiele wartości
- Wszystkie wartości muszą być tego samego typu
- Do poszczególnych wartości odwołujemy się poprzez jej numer w tablicy
- Numeracja wartości zaczyna się od zera
- Mogą być wielowymiarowe

Tablice

```
1 float[] oceny;
```

Deklaracja zmiennej tablicowej

Tablice

```
1 float[] oceny;
```

Deklaracja zmiennej tablicowej

```
1 oceny = new float[30];
```

Utworzenie zmiennej tablicowej

Tablice

```
1 float[] oceny;
```

Deklaracja zmiennej tablicowej

```
1 oceny = new float[30];
```

Utworzenie zmiennej tablicowej

```
1 float[] oceny = new float[30];
```

Deklaracja i utworzenie zmiennej tablicowej

Tablice – przykład użycia

```
1 float[] oceny = new float[30];
2
3 for (int i = 0; i < oceny.length; i++) {
4   oceny[i] = i % 4 + 2;
5 }
6
7 for (int i = 0; i < oceny.length; i++) {
8   System.out.println("Ocena: "+oceny[i]);
9 }
```

Przykład użycia tablicy

- Można sprawdzić długość tablicy: `tab.length`
- Przekroczenie rozmiaru: `ArrayIndexOutOfBoundsException`

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 30
at Template.main(Template.java:10)
```

Plan

Wypisywanie

`System.out.println` Wypisywanie ze znakiem nowej linii

`System.out.print` Wypisywanie

`System.out.printf` Wypisywanie z formatowaniem

`System.out.write` Wypisywanie kolejnych bajtów z tablicy

Wczytywanie

```
1  import java.util.Scanner;
2  public class ScannerDemo {
3    public static void main(String[] args) {
4
5      Scanner in = new Scanner(System.in);
6
7      int i = in.nextInt();          // wczytaj inta
8      String slowo = in.next();      // wczytaj słowo
9      String linia = in.nextLine(); // wczytaj całą linię
10
11     System.out.println("int: " + i);
12     System.out.println("slowo: " + slowo);
13     System.out.println("linia: " + linia);
14
15   }
16 }
```

Wczytywanie z klawiatury za pomocą Scanner

Wczytywanie za pomocą konsoli

```
1  import java.io.Console;
2  public class ReadPasswordDemo {
3    public static void main(String[] args) {
4      Console console = System.console();
5
6      // wczytaj nazwę uzytkownika, normalnie
7      String login = console.readLine("Login: ");
8
9      // wczytaj hasło, nie pokazując go
10     String haslo = new String(console.readPassword("Haslo: "));
11
12     System.out.println(login + " : " + haslo);
13   }
14 }
```

Wczytywanie z klawiatury za pomocą `Console`

- Obiekt `Console` nie jest dostępny jeśli uruchomimy program z IDE!
- Trzeba uruchomić z wiersza poleceń systemu operacyjnego!

Plan

Programowanie obiektowe

- W jednym programie: wiele operacji o tej samej nazwie, polegających na czymś innym
- A w życiu?
  - *otwórz* drzwi
  - *otwórz* plik
  - *otwórz* konto bankowe
- Operacja *otwórz* oznacza różne rzeczy, zależnie od tego *na czym* ją wykonujemy!
- Jak uporządkować operacje i dane, zwłaszcza w dużym programie?

Programowanie obiektowe

- Świat składa się z *obiektów*

Programowanie obiektowe

- Świat składa się z *obiektów*
- Każdy obiekt potrafi wykonywać pewne *operacje*

Programowanie obiektowe

- Świat składa się z *obiektów*
- Każdy obiekt potrafi wykonywać pewne *operacje*
- Operacje mogą zmieniać *stan* obiektów

Programowanie obiektowe

- Świat składa się z *obiektów*
- Każdy obiekt potrafi wykonywać pewne *operacje*
- Operacje mogą zmieniać *stan* obiektów
- Obiekty mogą wysyłać między sobą *komunikaty* (współpracować)

Programowanie obiektowe

- Świat składa się z *obiektów*
- Każdy obiekt potrafi wykonywać pewne *operacje*
- Operacje mogą zmieniać *stan* obiektów
- Obiekty mogą wysyłać między sobą *komunikaty* (współpracować)
- Obiekt po odebraniu komunikatu wykonuje odpowiednią operację (reaguje)

Programowanie obiektowe

- Świat składa się z *obiektów*
- Każdy obiekt potrafi wykonywać pewne *operacje*
- Operacje mogą zmieniać *stan* obiektów
- Obiekty mogą wysyłać między sobą *komunikaty* (współpracować)
- Obiekt po odebraniu komunikatu wykonuje odpowiednią operację (reaguje)
- Na przykład:

Programowanie obiektowe

- Świat składa się z *obiektów*
- Każdy obiekt potrafi wykonywać pewne *operacje*
- Operacje mogą zmieniać *stan* obiektów
- Obiekty mogą wysyłać między sobą *komunikaty* (współpracować)
- Obiekt po odebraniu komunikatu wykonuje odpowiednią operację (reaguje)
- Na przykład:
  - Obiekt `przełożony` → `mów wykład` → `wykładowca`

Programowanie obiektowe

- Świat składa się z *obiektów*
- Każdy obiekt potrafi wykonywać pewne *operacje*
- Operacje mogą zmieniać *stan* obiektów
- Obiekty mogą wysyłać między sobą *komunikaty* (współpracować)
- Obiekt po odebraniu komunikatu wykonuje odpowiednią operację (reaguje)
- Na przykład:
  - Obiekt `przełożony` → `mów wykład` → `wykładowca`
  - Obiekt `wykładowca` → `zdałeś na 5.0` → `student`

Obiekty

- Mają swój zestaw operacji (które potrafią wykonać)

Obiekty

- Mają swój zestaw operacji (które potrafią wykonać)
  - `wykładowca` – `mów wykład`

Obiekty

- Mają swój zestaw operacji (które potrafią wykonać)
  - `wykładowca` – `mów wykład`
  - `student` – `zdałeś na`

Obiekty

- Mają swój zestaw operacji (które potrafią wykonać)
  - `wykładowca` – `mów wykład`
  - `student` – `zdałeś na`
- Mają swój stan (który może się zmieniać)

Obiekty

- Mają swój zestaw operacji (które potrafią wykonać)
  - `wykładowca` – `mów wykład`
  - `student` – `zdałeś na`
- Mają swój stan (który może się zmieniać)
  - `student` – `lista ocen`

Obiekty

- Mają swój zestaw operacji (które potrafią wykonać)
  - `wykładowca` – `mów wykład`
  - `student` – `zdałeś na`
- Mają swój stan (który może się zmieniać)
  - `student` – `lista ocen`
  - `student` – `humor`

Obiekty

- Mają swój zestaw operacji (które potrafią wykonać)
  - `wykładowca` – `mów wykład`
  - `student` – `zdałeś na`
- Mają swój stan (który może się zmieniać)
  - `student` – `lista ocen`
  - `student` – `humor`
- W programach:

Obiekty

- Mają swój zestaw operacji (które potrafią wykonać)
  - `wykładowca` – `mów wykład`
  - `student` – `zdałeś na`
- Mają swój stan (który może się zmieniać)
  - `student` – `lista ocen`
  - `student` – `humor`
- W programach:
  - stan jest zapamiętywany w...

Obiekty

- Mają swój zestaw operacji (które potrafią wykonać)
  - `wykładowca` – `mów wykład`
  - `student` – `zdałeś na`
- Mają swój stan (który może się zmieniać)
  - `student` – `lista ocen`
  - `student` – `humor`
- W programach:
  - stan jest zapamiętywany w. . . zmiennych

Obiekty

- Mają swój zestaw operacji (które potrafią wykonać)
  - `wykładowca` – `mów wykład`
  - `student` – `zdałeś na`
- Mają swój stan (który może się zmieniać)
  - `student` – `lista ocen`
  - `student` – `humor`
- W programach:
  - stan jest zapamiętywany w. . . zmiennych
  - operacje są zapisywane jako. . .

Obiekty

- Mają swój zestaw operacji (które potrafią wykonać)
  - `wykładowca` – `mów wykład`
  - `student` – `zdałeś na`
- Mają swój stan (który może się zmieniać)
  - `student` – `lista ocen`
  - `student` – `humor`
- W programach:
  - stan jest zapamiętywany w. . . zmiennych
  - operacje są zapisywane jako. . . funkcje

Obiekty

- Mają swój zestaw operacji (które potrafią wykonać)
  - `wykładowca` – `mów wykład`
  - `student` – `zdałeś na`
- Mają swój stan (który może się zmieniać)
  - `student` – `lista ocen`
  - `student` – `humor`
- W programach:
  - stan jest zapamiętywany w. . . zmiennych
  - operacje są zapisywane jako. . . funkcje
- W programowaniu obiektowym, wewnątrz obiektu umieszczamy:

Obiekty

- Mają swój zestaw operacji (które potrafią wykonać)
  - `wykładowca` – `mów wykład`
  - `student` – `zdałeś na`
- Mają swój stan (który może się zmieniać)
  - `student` – `lista ocen`
  - `student` – `humor`
- W programach:
  - stan jest zapamiętywany w. . . zmiennych
  - operacje są zapisywane jako. . . funkcje
- W programowaniu obiektowym, wewnątrz obiektu umieszczamy:
  - zmienne i nazywamy je polami

Obiekty

- Mają swój zestaw operacji (które potrafią wykonać)
  - `wykładowca` – `mów wykład`
  - `student` – `zdałeś na`
- Mają swój stan (który może się zmieniać)
  - `student` – `lista ocen`
  - `student` – `humor`
- W programach:
  - stan jest zapamiętywany w. . . zmiennych
  - operacje są zapisywane jako. . . funkcje
- W programowaniu obiektowym, wewnątrz obiektu umieszczamy:
  - zmienne i nazywamy je polami
  - funkcje i nazywamy je metodami

Obiekty

- Mają swój zestaw operacji (które potrafią wykonać)

Obiekty

- Mają swój zestaw operacji (które potrafią wykonać)
  - metody (funkcje)

Obiekty

- Mają swój zestaw operacji (które potrafią wykonać)
  - metody (funkcje)
- Mają swój stan (który może się zmieniać)

Obiekty

- Mają swój zestaw operacji (które potrafią wykonać)
  - metody (funkcje)
- Mają swój stan (który może się zmieniać)
  - pola (zmienne)

Grupy „podobnych” obiektów

- Można wyróżnić grupy obiektów, które są do siebie podobne

Grupy „podobnych” obiektów

- Można wyróżnić grupy obiektów, które są do siebie podobne
  - Potrafią wykonywać te same operacje (`przedstaw_sie`)

Grupy „podobnych” obiektów

- Można wyróżnić grupy obiektów, które są do siebie podobne
  - Potrafią wykonywać te same operacje (`przedstaw_sie`)
  - Odpowiadają na te same komunikaty (`przedstaw_sie`)

Grupy „podobnych” obiektów

- Można wyróżnić grupy obiektów, które są do siebie podobne
  - Potrafią wykonywać te same operacje (`przedstaw_sie`)
  - Odpowiadają na te same komunikaty (`przedstaw_sie`)
  - Mają takie same cechy (`imie`)

Grupy „podobnych" obiektów

- Można wyróżnić grupy obiektów, które są do siebie podobne
  - Potrafią wykonywać te same operacje (`przedstaw_sie`)
  - Odpowiadają na te same komunikaty (`przedstaw_sie`)
  - Mają takie same cechy (`imie`)
- Jak wygodnie tworzyć takie „podobne" obiekty?

Grupy „podobnych" obiektów

- Można wyróżnić grupy obiektów, które są do siebie podobne
  - Potrafią wykonywać te same operacje (`przedstaw_sie`)
  - Odpowiadają na te same komunikaty (`przedstaw_sie`)
  - Mają takie same cechy (`imie`)
- Jak wygodnie tworzyć takie „podobne" obiekty?
- Grupy „podobnych" obiektów nazywamy *klasami*

Grupy „podobnych" obiektów

- Można wyróżnić grupy obiektów, które są do siebie podobne
  - Potrafią wykonywać te same operacje (`przedstaw_sie`)
  - Odpowiadają na te same komunikaty (`przedstaw_sie`)
  - Mają takie same cechy (`imie`)
- Jak wygodnie tworzyć takie „podobne" obiekty?
- Grupy „podobnych" obiektów nazywamy *klasami*
- *Klasy* służą do wygodnego tworzenia „podobnych" obiektów

Klasa

- Klasa pozwala wygodnie tworzyć obiekty

Klasa

- Klasa pozwala wygodnie tworzyć obiekty
- Mówimy, że są to *obiekty danej klasy*:

Klasa

- Klasa pozwala wygodnie tworzyć obiekty
- Mówimy, że są to *obiekty danej klasy*:
  - `jozek` to obiekt klasy `Osoba`

Klasa

- Klasa pozwala wygodnie tworzyć obiekty
- Mówimy, że są to *obiekty danej klasy*:
  - `jozek` to obiekt klasy `Osoba`
  - `franek` to obiekt klasy `Osoba`

Klasa

- Klasa pozwala wygodnie tworzyć obiekty
- Mówimy, że są to *obiekty danej klasy*:
  - `jozek` to obiekt klasy `Osoba`
  - `franek` to obiekt klasy `Osoba`
- Każdy z obiektów ma swoje własne pola i może je dowolnie ustawiać

Klasa

- Klasa pozwala wygodnie tworzyć obiekty
- Mówimy, że są to *obiekty danej klasy*:
  - `jozek` to obiekt klasy `Osoba`
  - `franek` to obiekt klasy `Osoba`
- Każdy z obiektów ma swoje własne pola i może je dowolnie ustawiać
  - `jozek` ma swoje własne pole `imie`

Klasa

- Klasa pozwala wygodnie tworzyć obiekty
- Mówimy, że są to *obiekty danej klasy*:
  - `jozek` to obiekt klasy `Osoba`
  - `franek` to obiekt klasy `Osoba`
- Każdy z obiektów ma swoje własne pola i może je dowolnie ustawiać
  - `jozek` ma swoje własne pole `imie`
  - niezależnie od tego `franek` na swoje pole `imie`

Klasa

- Klasa pozwala wygodnie tworzyć obiekty
- Mówimy, że są to *obiekty danej klasy*:
  - `jozek` to obiekt klasy `Osoba`
  - `franek` to obiekt klasy `Osoba`
- Każdy z obiektów ma swoje własne pola i może je dowolnie ustawiać
  - `jozek` ma swoje własne pole `imie`
  - niezależnie od tego `franek` na swoje pole `imie`
- Obiekty danej klasy mają taką samą listę metod i pól

Klasa

- Klasa pozwala wygodnie tworzyć obiekty
- Mówimy, że są to *obiekty danej klasy*:
  - `jozek` to obiekt klasy `Osoba`
  - `franek` to obiekt klasy `Osoba`
- Każdy z obiektów ma swoje własne pola i może je dowolnie ustawiać
  - `jozek` ma swoje własne pole `imie`
  - niezależnie od tego `franek` na swoje pole `imie`
- Obiekty danej klasy mają taką samą listę metod i pól
  - każdy obiekt klasy `Osoba` ma pole `imie`

Klasa

- Klasa pozwala wygodnie tworzyć obiekty
- Mówimy, że są to *obiekty danej klasy*:
  - `jozek` to obiekt klasy `Osoba`
  - `franek` to obiekt klasy `Osoba`
- Każdy z obiektów ma swoje własne pola i może je dowolnie ustawiać
  - `jozek` ma swoje własne pole `imie`
  - niezależnie od tego `franek` na swoje pole `imie`
- Obiekty danej klasy mają taką samą listę metod i pól
  - każdy obiekt klasy `Osoba` ma pole `imie`
  - każdy obiekt klasy `Osoba` ma takie same metody:

Klasa

- Klasa pozwala wygodnie tworzyć obiekty
- Mówimy, że są to *obiekty danej klasy*:
  - `jozek` to obiekt klasy `Osoba`
  - `franek` to obiekt klasy `Osoba`
- Każdy z obiektów ma swoje własne pola i może je dowolnie ustawiać
  - `jozek` ma swoje własne pole `imie`
  - niezależnie od tego `franek` na swoje pole `imie`
- Obiekty danej klasy mają taką samą listę metod i pól
  - każdy obiekt klasy `Osoba` ma pole `imie`
  - każdy obiekt klasy `Osoba` ma takie same metody:
    - `przywitaj_sie`

Klasa

- Klasa pozwala wygodnie tworzyć obiekty
- Mówimy, że są to *obiekty danej klasy*:
  - `jozek` to obiekt klasy `Osoba`
  - `franek` to obiekt klasy `Osoba`
- Każdy z obiektów ma swoje własne pola i może je dowolnie ustawiać
  - `jozek` ma swoje własne pole `imie`
  - niezależnie od tego `franek` na swoje pole `imie`
- Obiekty danej klasy mają taką samą listę metod i pól
  - każdy obiekt klasy `Osoba` ma pole `imie`
  - każdy obiekt klasy `Osoba` ma takie same metody:
    - `przywitaj_sie`
    - `nazywasz_sie`

Klasa

- Klasa pozwala wygodnie tworzyć obiekty
- Mówimy, że są to *obiekty danej klasy*:
  - `jozek` to obiekt klasy `Osoba`
  - `franek` to obiekt klasy `Osoba`
- Każdy z obiektów ma swoje własne pola i może je dowolnie ustawiać
  - `jozek` ma swoje własne pole `imie`
  - niezależnie od tego `franek` na swoje pole `imie`
- Obiekty danej klasy mają taką samą listę metod i pól
  - każdy obiekt klasy `Osoba` ma pole `imie`
  - każdy obiekt klasy `Osoba` ma takie same metody:
    - `przywitaj_sie`
    - `nazywasz_sie`
    - `przedstaw_sie`

Klasa

- Obiekty danej klasy mają:

Klasa

- Obiekty danej klasy mają:
  - Taki sam zestaw cech

Klasa

- Obiekty danej klasy mają:
  - Taki sam zestaw cech
  - Reagują na taki sam zestaw komunikatów

Klasa

- Obiekty danej klasy mają:
  - Taki sam zestaw cech
  - Reagują na taki sam zestaw komunikatów
  - Potrafią podejmować takie same działania

Klasa w Javie

```
1  public class Osoba {
2    // pola, czyli zmienne wewnątrz klasy
3    private String imie;
4    private String nazwisko;
5
6
7    // metody, czyli funkcje wewnątrz klasy
8    public void nazywasz_sie(String imie, String nazwisko) {
9      this.imie = imie;           // this.imie -- pole imie
10     this.nazwisko = nazwisko; // this.nazwisko -- pole nazwisko
11   }
12
13   public void przywitaj_sie() {
14     System.out.println("Cześć");
15   }
16
17   public void przedstaw_sie() {
18     System.out.println("Jestem " + imie + " " + nazwisko);
19   }
20 }
```

Definicja klasy Osoba

Obiekty w Javie

```
1  public class OsobaDemo {
2    public static void main(String[] args) {
3      Osoba jozek;           // referencja do obiektu osoba
4      jozek = new Osoba(); // utworzenie i zapamietanie
5                           // nowego obiektu
6
7      jozek.nazywasz_sie("Józef", "Nowak"); // wywołanie metody
8                                            // na obiekcie
9
10     Osoba franek = new Osoba();
11     franek.nazywasz_sie("Franciszek", "Kowalski");
12
13     jozek.przywitaj_sie();
14     jozek.przedstaw_sie();
15
16     franek.przywitaj_sie();
17     franek.przedstaw_sie();
18   }
19 }
```

Użycie obiektów klasy Osoba

Klasy i obiekty

- Nazwy klas z wielkiej litery

Klasy i obiekty

- Nazwy klas z wielkiej litery
- Nazwy zmiennych z małej litery

Klasy i obiekty

- Nazwy klas z wielkiej litery
- Nazwy zmiennych z małej litery
- Nazwy pól z małej litery

Klasy i obiekty

- Nazwy klas z wielkiej litery
- Nazwy zmiennych z małej litery
- Nazwy pól z małej litery
- Nazwy metod z małej litery

Konstruktor

- (Prawie) zwyczajna metoda
- Wywoływana w momencie tworzenia obiektu
- Pozwala ustawić początkowe wartości pól
- Musi się nazywać tak samo jak klasa
- Nie zwraca wartości

Klasa z konstruktorem

```
1  public class Osoba {
2    // pola, czyli zmienne wewnątrz klasy
3    private String imie;
4    private String nazwisko;
5
6    public Osoba(String imie, String nazwisko) {
7      this.imie = imie; this.nazwisko = nazwisko;
8    }
9
10   // metody, czyli funkcje wewnątrz klasy
11   public void nazywasz_sie(String imie, String nazwisko) {
12     this.imie = imie;           // this.imie -- pole imie
13     this.nazwisko = nazwisko; // this.nazwisko -- pole nazwisko
14   }
15
16   public void przywitaj_sie() {
17     System.out.println("Cześć");
18   }
19
20   public void przedstaw_sie() {
21     System.out.println("Jestem " + imie + " " + nazwisko);
22   }
23 }
```

Użycie klasy z konstruktorem

```
1  public class OsobaDemo {
2    public static void main(String[] args) {
3      Osoba jozek = new Osoba("Józef", "Nowak");
4      Osoba franek = new Osoba("Franciszek", "Kowalski");
5
6      jozek.przywitaj_sie();
7      jozek.przedstaw_sie();
8
9      franek.przywitaj_sie();
10     franek.przedstaw_sie();
11   }
12 }
```

Użycie klasy `Osoba` z konstruktorem

Dziedziczenie

- Różne klasy mogą mieć wspólne cechy

Dziedziczenie

- Różne klasy mogą mieć wspólne cechy
- `Osoba`

Dziedziczenie

- Różne klasy mogą mieć wspólne cechy
- `Osoba`
  - `Student`

Dziedziczenie

- Różne klasy mogą mieć wspólne cechy
- `Osoba`
  - `Student`
  - `Pracownik` uczelni

Dziedziczenie

- Różne klasy mogą mieć wspólne cechy
- `Osoba`
  - `Student`
  - `Pracownik` uczelni
- Ale mają też cechy, ktore je różnią

Dziedziczenie

- Różne klasy mogą mieć wspólne cechy
- `Osoba`
  - `Student`
  - `Pracownik` uczelni
- Ale mają też cechy, ktore je różnią
  - `Student` – lista ocen

Dziedziczenie

- Różne klasy mogą mieć wspólne cechy
- `Osoba`
  - `Student`
  - `Pracownik` uczelni
- Ale mają też cechy, ktore je różnią
  - `Student` – lista ocen
  - `Pracownik` –

Dziedziczenie

- Różne klasy mogą mieć wspólne cechy
- `Osoba`
  - `Student`
  - `Pracownik` uczelni
- Ale mają też cechy, ktore je różnią
  - `Student` – lista ocen
  - `Pracownik` – tytuł! ;-)

Dziedziczenie

- Różne klasy mogą mieć wspólne cechy
- `Osoba`
  - `Student`
  - `Pracownik` uczelni
- Ale mają też cechy, ktore je różnią
  - `Student` – lista ocen
  - `Pracownik` – tytuł! ;-)
- Można utworzyć osobne klasy:
  - `Student`
  - `Pracownik`

Dziedziczenie

- Różne klasy mogą mieć wspólne cechy
- `Osoba`
  - `Student`
  - `Pracownik` uczelni
- Ale mają też cechy, ktore je różnią
  - `Student` – lista ocen
  - `Pracownik` – tytuł! ;-)
- Można utworzyć osobne klasy:
  - `Student`
  - `Pracownik`
- Każdej z tych klas dodać wszystkie cechy, które ma `Osoba`

Dziedziczenie

- Różne klasy mogą mieć wspólne cechy
- `Osoba`
  - `Student`
  - `Pracownik` uczelni
- Ale mają też cechy, ktore je różnią
  - `Student` – lista ocen
  - `Pracownik` – tytuł! ;-)
- Można utworzyć osobne klasy:
  - `Student`
  - `Pracownik`
- Każdej z tych klas dodać wszystkie cechy, które ma `Osoba`
- Mówimy, że

Dziedziczenie

- Różne klasy mogą mieć wspólne cechy
- `Osoba`
  - `Student`
  - `Pracownik` uczelni
- Ale mają też cechy, ktore je różnią
  - `Student` – lista ocen
  - `Pracownik` – tytuł! ;-)
- Można utworzyć osobne klasy:
  - `Student`
  - `Pracownik`
- Każdej z tych klas dodać wszystkie cechy, które ma `Osoba`
- Mówimy, że
  - klasa `Student` dziedziczy po klasie `Osoba`

Dziedziczenie

- Różne klasy mogą mieć wspólne cechy
- `Osoba`
  - `Student`
  - `Pracownik` uczelni
- Ale mają też cechy, ktore je różnią
  - `Student` – lista ocen
  - `Pracownik` – tytuł! ;-)
- Można utworzyć osobne klasy:
  - `Student`
  - `Pracownik`
- Każdej z tych klas dodać wszystkie cechy, które ma `Osoba`
- Mówimy, że
  - klasa `Student` dziedziczy po klasie `Osoba`
  - klasa `Pracownik` dziedziczy po klasie `Osoba`

Student i Pracownik

```
1 public class Student extends Osoba {
2   private float oceny[];
3   private int ile_ocen;
4 }
```

Klasa Student dziedzicząca po klasie Osoba

```
1 public class Pracownik extends Osoba {
2   private String tytul;
3 }
```

Klasa Pracownik dziedzicząca po klasie Osoba

Klasa dziedzicząca

- Dziedziczy pola i metody klasy nadrzędnej
- (W Javie) nie dziedziczy konstruktorów
- Może dodać swoje pola i metody
- Może nadpisać odziedziczone pola i metody

Kompletna klasa Student

```
1  public class Student extends Osoba {
2    private float oceny[];
3    private int ile_ocen;
4
5    public Student(String imie, String nazwisko) {
6      super(imie, nazwisko);
7      oceny = new float[50];
8      ile_ocen = 0;
9    }
10
11   public void zdales_na(float ocena) {
12     oceny[ile_ocen] = ocena;
13     ile_ocen++;
14   }
15 }
```

Klasa Student z własnymi polami i metodami

Kompletna klasa Pracownik

```
1  public class Pracownik extends Osoba {
2    private String tytul;
3
4    public Pracownik(String imie, String nazwisko) {
5      super(imie, nazwisko);
6      tytul = "";
7    }
8
9    public Pracownik(String tytul, String imie, String nazwisko) {
10     super(imie, nazwisko);
11     this.tytul = tytul;
12   }
13
14 }
```

Klasa Pracownik z własnymi polami i metodami

Kompletna klasa Pracownik

```
1  public class Pracownik extends Osoba {
2    private String tytul;
3
4    public Pracownik(String imie, String nazwisko) {
5      super(imie, nazwisko);
6      tytul = "";
7    }
8
9    public Pracownik(String tytul, String imie, String nazwisko) {
10     super(imie, nazwisko);
11     this.tytul = tytul;
12   }
13
14   // To nie zadziała: imie i nazwisko są prywatne!
15   public void przedstaw_sie() {
16     System.out.println("Jestem " + tytul + " "
17         + imie + " " + nazwisko);
18   }
19 }
```

Klasa Pracownik nie ma dostępu do prywatnych pól klasy Osoba

Dostęp do pól i metod

`private` dostęp tylko dla tej klasy

`public` dostep dla wszystkich

`protected` dostęp tylko dla tej klasy i tych, które z niej dziedziczą

(brak modyfikatora) – o tym kiedy indziej ;)

Klasa z polami protected

```
1  public class Osoba {
2    // pola, czyli zmienne wewnątrz klasy
3    protected String imie;
4    protected String nazwisko;
5
6    public Osoba(String imie, String nazwisko) {
7      this.imie = imie; this.nazwisko = nazwisko;
8    }
9
10   // metody, czyli funkcje wewnątrz klasy
11   public void nazywasz_sie(String imie, String nazwisko) {
12     this.imie = imie;           // this.imie -- pole imie
13     this.nazwisko = nazwisko; // this.nazwisko -- pole nazwisko
14   }
15
16   public void przywitaj_sie() {
17     System.out.println("Cześć");
18   }
19
20   public void przedstaw_sie() {
21     System.out.println("Jestem " + imie + " " + nazwisko);
22   }
23 }
```

Użycie klas dziedziczących i dziedziczonych – zmienne

```
1  public class OsobaDemo {
2
3    public static void main(String[] args) {
4      Osoba pan_jozek = new Osoba("Józef", "Nowak");
5      Pracownik prof =
6        new Pracownik("prof. dr hab.", "Krzysztof", "Kowalski");
7      Student tomek = new Student("Tomasz", "Kwiatkowski");
8
9      Osoba[] ludzie = new Osoba[3];
10     ludzie[0] = tomek;
11     ludzie[1] = prof;
12     ludzie[2] = pan_jozek;
13
14     for(int i = 0; i < ludzie.length; i++) {
15       ludzie[i].przedstaw_sie();
16     }
17   }
18 }
```

Użycie klas dziedziczących po klasie Osoba

Użycie klas dziedziczących i dziedziczonych – metody

```
1  public class OsobaDemo {
2
3    static void przedstaw_czlowieka(Osoba o) {
4      o.przedstaw_sie();
5    }
6
7    public static void main(String[] args) {
8      Osoba pan_jozek = new Osoba("Józef", "Nowak");
9      Pracownik prof =
10       new Pracownik("prof. dr hab.", "Krzysztof", "Kowalski");
11     Student tomek = new Student("Tomasz", "Kwiatkowski");
12
13     przedstaw_czlowieka(pan_jozek);
14     przedstaw_czlowieka(prof);
15   }
16 }
```

Użycie klas dziedziczących po klasie Osoba

Dziedziczenie

- Pozwala napisać kod raz, używać wiele razy
- W Javie dziedziczyć można tylko po jednej klasie bazowej
- Po danej klasie może dziedziczyć wiele klas...
- ...z których znowu dziedziczyć mogą kolejne klasy

Interfejsy

- Niezależnie od dziedziczenia
- Klasa może implementować wiele interfejsów
- Interfejs deklaruje metody, ale ich nie implementuje
- Nie da się utworzyć obiektu z interfejsu

Interfejsy

- Niezależnie od dziedziczenia
- Klasa może implementować wiele interfejsów
- Interfejs deklaruje metody, ale ich nie implementuje
- Nie da się utworzyć obiektu z interfejsu

```
1 interface Mieszkaniec {
2   public void wypiszAdres();
3   public String podajAdres();
4 }
```

Definicja interfejsu `Mieszkaniec`

Implementacja interfejsu

```
1  public class Student extends Osoba implements Mieszkaniec {
2    private float oceny[];
3    private int ile_ocen;
4
5    public Student(String imie, String nazwisko) {
6      super(imie, nazwisko);
7      oceny = new float[50];
8      ile_ocen = 0;
9    }
10
11   public void zdales_na(float ocena) {
12     oceny[ile_ocen] = ocena;
13     ile_ocen++;
14   }
15
16   public String podajAdres() {
17     return "Akademik";
18   }
19
20   public void wypiszAdres() {
21     System.out.println(podajAdres());
22   }
23 }
```

Implementacja interfejsu

```
1  public class Pracownik extends Osoba implements Mieszkaniec {
2    private String tytul;
3
4    /*
5     * (...)
6     * tu się nic nie zmienia
7     * (...)
8     */
9
10   public String podajAdres() {
11     return "Moja Willa!";
12   }
13
14   public void wypiszAdres() {
15     System.out.println(podajAdres());
16   }
17 }
```

Klasa `Pracownik` implementująca interfejs `Mieszkaniec`

Wykorzystanie interfejsu

```
1  public class MieszkaniecDemo {
2
3    public static void main(String[] args) {
4      Osoba pan_jozek = new Osoba("Józef", "Nowak");
5      Pracownik prof =
6        new Pracownik("prof. dr hab.", "Krzysztof", "Kowalski");
7      Student tomek = new Student("Tomasz", "Kwiatkowski");
8
9      Mieszkaniec[] ludzie = new Mieszkaniec[3];
10     ludzie[0] = tomek;
11     ludzie[1] = prof;
12     // ludzie[2] = pan_jozek; tego nie można zrobić!
13
14     for(int i = 0; i < ludzie.length; i++) {
15       ludzie[i].wypiszAdres(); // błąd dla i == 2
16     }
17   }
18 }
```

Użycie klas implementujących interfejs `Mieszkaniec`

Plan

Wyjątki

- Służą do obsługi błędów

Wyjątki

- Służą do obsługi błędów
- Program generuje wyjątek w przypadku błędu

Wyjątki

- Służą do obsługi błędów
- Program generuje wyjątek w przypadku błędu
- Wyjątek jest obiektem

Wyjątki

- Służą do obsługi błędów
- Program generuje wyjątek w przypadku błędu
- Wyjątek jest obiektem
- Wyjątek może być przechwycony

Wyjątki

- Służą do obsługi błędów
- Program generuje wyjątek w przypadku błędu
- Wyjątek jest obiektem
- Wyjątek może być przechwycony
- Nieprzechwycony wyjątek powoduje zakończenie programu

```
Exception in thread "main" java.lang.NullPointerException
at MieszkaniecDemo.main(mainMieszkaniecDemo.java:15)
```

Wyjątki

- Służą do obsługi błędów
- Program generuje wyjątek w przypadku błędu
- Wyjątek jest obiektem
- Wyjątek może być przechwycony
- Nieprzechwycony wyjątek powoduje zakończenie programu

```
Exception in thread "main" java.lang.NullPointerException
at MieszkaniecDemo.main(mainMieszkaniecDemo.java:15)
```

- Dokumentacja do klas i metod mówi jakie wyjątki mogą one generować

Wyjątki

- Służą do obsługi błędów
- Program generuje wyjątek w przypadku błędu
- Wyjątek jest obiektem
- Wyjątek może być przechwycony
- Nieprzechwycony wyjątek powoduje zakończenie programu

```
Exception in thread "main" java.lang.NullPointerException
at MieszkaniecDemo.main(mainMieszkaniecDemo.java:15)
```

- Dokumentacja do klas i metod mówi jakie wyjątki mogą one generować
- Java może zażądać przechwycenia wyjątku w programie

Obsługa wyjątku

```
1  public class MieszkaniecDemo {
2
3    public static void main(String[] args) {
4      Osoba pan_jozek = new Osoba("Józef", "Nowak");
5      Pracownik prof =
6        new Pracownik("prof. dr hab.", "Krzysztof", "Kowalski");
7      Student tomek = new Student("Tomasz", "Kwiatkowski");
8
9      Mieszkaniec[] ludzie = new Mieszkaniec[3];
10     ludzie[0] = tomek;
11     ludzie[1] = prof;
12     // ludzie[2] = pan_jozek; tego nie można zrobić!
13
14     try {
15       for(int i = 0; i < ludzie.length; i++) {
16         ludzie[i].wypiszAdres();
17       }
18     } catch(NullPointerException e) {
19       System.out.println("Złapałem: " + e
20           + " tablica nie jest wypełniona!?");
21     }
22   }
23 }
```

Obsługa wyjątku – stack trace

```
1  public class MieszkaniecDemo {
2
3    public static void main(String[] args) {
4      Osoba pan_jozek = new Osoba("Józef", "Nowak");
5      Pracownik prof =
6        new Pracownik("prof. dr hab.", "Krzysztof", "Kowalski");
7      Student tomek = new Student("Tomasz", "Kwiatkowski");
8
9      Mieszkaniec[] ludzie = new Mieszkaniec[3];
10     ludzie[0] = tomek;
11     ludzie[1] = prof;
12     // ludzie[2] = pan_jozek; tego nie można zrobić!
13
14     try {
15       for(int i = 0; i < ludzie.length; i++) {
16         ludzie[i].wypiszAdres();
17       }
18     } catch(NullPointerException e) {
19       e.printStackTrace();
20     }
21   }
22 }
```

Obsługa wyjątku – stack trace

```
java.lang.NullPointerException
        at MieszkaniecDemo.main(MieszkaniecDemo.java:16)
```

Obsługa wyjątków – uwagi

- `catch` przechwyci tylko wyjątek odpowiedniej klasy

Obsługa wyjątków – uwagi

- `catch` przechwyci tylko wyjątek odpowiedniej klasy
- Po jednym `try` może być wiele `catch` dla różnych klas wyjątków

Obsługa wyjątków – uwagi

- `catch` przechwyci tylko wyjątek odpowiedniej klasy
- Po jednym `try` może być wiele `catch` dla różnych klas wyjątków
- Można użyć bloku `finally`, który wykona się zawsze, niezależnie od wyjątków

Obsługa wyjątków – uwagi

- `catch` przechwyci tylko wyjątek odpowiedniej klasy
- Po jednym `try` może być wiele `catch` dla różnych klas wyjątków
- Można użyć bloku `finally`, który wykona się zawsze, niezależnie od wyjątków
- Metoda może deklarować, że generuje wyjątek pewnej klasy (nawet jeśli w rzeczywistości tego nie robi)

```
1 public void przedstaw_sie() throws NameNotSetException {
2   System.out.println("Jestem " + imie + " " + nazwisko);
3 }
```

Metoda może wygenerować wyjątek

Generowanie wyjątku

```
1 public void przedstaw_sie() {
2   if (imie == null || nazwisko == null) {
3     throw new RuntimeException("name not known!");
4   }
5   System.out.println("Jestem " + imie + " " + nazwisko);
6 }
```

Generowanie wyjątku

- Własny wyjątek: klasa dziedzicząca po `Exception` lub `RuntimeException`

Generowanie wyjątku

```
1 public void przedstaw_sie() {
2   if (imie == null || nazwisko == null) {
3     throw new RuntimeException("name not known!");
4   }
5   System.out.println("Jestem " + imie + " " + nazwisko);
6 }
```

Generowanie wyjątku

- Własny wyjątek: klasa dziedzicząca po `Exception` lub `RuntimeException`
- Metoda generująca `RuntimeException` lub pochodny nie musi go deklarować

Generowanie wyjątku

```
1 public void przedstaw_sie() {
2   if (imie == null || nazwisko == null) {
3     throw new RuntimeException("name not known!");
4   }
5   System.out.println("Jestem " + imie + " " + nazwisko);
6 }
```

Generowanie wyjątku

- Własny wyjątek: klasa dziedzicząca po `Exception` lub `RuntimeException`
- Metoda generująca `RuntimeException` lub pochodny nie musi go deklarować
- Metoda generująca `Exception` lub pochodny musi go deklarować w nagłówku