Programowanie obiektowe

Wykład 6:
Klasy abstrakcyjne. Interfejsy.

Klasy ostateczne

```
class Ogon {}

final class Dinozaur {
  int i = 7;
  int j = 1;
  Ogon x = new Ogon();
  void f() {}
}

//! class Potomek extends Dinozaur {}
// Błąd: Nie można rozszerzać klasy ostatecznej „Dinozaur”

public class Jura {
  public static void main(String[] args) {
    Dinozaur n = new Dinozaur();
    n.f();
    n.i = 40;
    n.j++;
  }
}
```

Metody ostateczne

```
class WithFinals {
  // To samo co "private":
  private final void f() {
    System.out.println("WithFinals.f()");
  }
  // Automatycznie "final":
  private void g() {
    System.out.println("WithFinals.g()");
  }
}

class OverridingPrivate extends WithFinals {
  private final void f() {

System.out.println("OverridingPrivate.f()");
  }
  private void g() {

System.out.println("OverridingPrivate.g()");
  }
}
```

```
class OverridingPrivate2
  extends OverridingPrivate {
  public final void f() {
System.out.println("OverridingPrivate2.f()");
  }
  public void g() {
System.out.println("OverridingPrivate2.g()");
  }
}
public class FinalOverridingIllusion {
  public static void main(String[] args) {
    OverridingPrivate2 op2 =
      new OverridingPrivate2();
    op2.f();
    op2.g();
    // Można rzutować w górę:
    OverridingPrivate op = op2;
    // Ale nie można wywoływać tych metod:
    //! op.f();
    //! op.g();
    // Podobnie:
    WithFinals wf = op2;
    //! wf.f();
    //! wf.g();
  }
}
```


Zmienne ostateczne

```
class Value {
  int i = 1;
}
public class FinalData {
  final int i1 = 9;
  static final int VAL_TWO = 99;
  public static final int VAL_THREE = 39;
  final int i4 = (int)(Math.random()*20);
  static final int i5 = (int)(Math.random()*20);
  Value v1 = new Value();
  final Value v2 = new Value();
  static final Value v3 = new Value();
  final int[] a = { 1, 2, 3, 4, 5, 6 };
  public void print(String id) {
    System.out.println(
      id + ": " + "i4 = " + i4 + ", i5 = " + i5);
  }
```

```
public static void main(String[] args) {
    FinalData fd1 = new FinalData();
    fd1.i1++;
    fd1.v2.i++;
    fd1.v1 = new Value();
    for(int i = 0; i < fd1.a.length; i++)  fd1.a[i]++;
    fd1.v2 = new Value();
    fd1.v3 = new Value();
    fd1.a = new int[3];
    fd1.print("fd1");
    System.out.println("Creating new FinalData");
    FinalData fd2 = new FinalData();
    fd1.print("fd1");
    fd2.print("fd2");
  }
}
```

Argumenty ostateczne

```
class Gizmo {
  public void spin() {}
}

public class FinalArguments {
  void with(final Gizmo g) {
    //! g = new Gizmo();  // Niedozwolone: g jest ostateczny
  }
  void without(Gizmo g) {
    g = new Gizmo();      // OK: g nie jest ostateczy
    g.spin();
  }
  // void f(final int i) { i++; } // Nie można zmieniać „i”
  // Można tylko czytać wartości ostateczne:
  int g(final int i) { return i + 1; }
  public static void main(String[] args) {
    FinalArguments bf = new FinalArguments();
    bf.without(null);
    bf.with(null);
  }
}
```


Klasy i metody abstrakcyjne

```
abstract class Instrument {
  public void graj(Nuta n) {
   System.out.println("Instrument.graj()");
  }
}
```

```
abstract class Ksztalt {
  abstract void rysuj();
  abstract void usun();
}
```

Metody polimorficzne w konstruktorach

```
abstract class Ksztalt {
  abstract void rysuj();
  Ksztalt() {
    System.out.println("Ksztalt() przed rysuj()");
    rysuj();
    System.out.println("Ksztalt() po rysuj()");
  }
}

class Kolo extends Ksztalt {
  int radius = 1;
  Kolo(int r) {
    radius = r;
    System.out.println(
      "Kolo.Kolo(), radius = " + radius);
  }
  void rysuj() {
    System.out.println(
      "Kolo.rysuj(), radius = " + radius);
  }
}
```

```
public class PolyConstructors {
  public static void main(String[ ] args) {
    new Kolo(5);
  }
}
```

Wynik?

Ksztalt() przed rysuj()
Kolo.rysuj(), radius = 0
Ksztalt() po rysuj()
Kolo.Kolo(), radius = 5

Problem wielodziedziczenia

```
class Pojazd {
…
int ustaw_pozycję(..){ …}
}
```

```
class Broń {
…
int ustaw_pozycję(…){ …}
}
```

```
class Czołg extends Pojazd, Broń {
…
void  jedź() {
…
 ustaw_pozycję(kierunek, szybkość);
…
}
```

?

Interfejsy

- Jak klasa „czysto” abstakcyjna (tzn. wszystkie metody bez implementacji)
- wszystkie pola statyczne (static)
- wszystkie pola ostateczne (final)
- wszystkie pola i metody publiczne
- można implementować wiele interfejsów

Deklaracja interfejsu

klasa	class	extends
interfejs	interface	implements

```
interface A {
....
}
class B implements A {
...
}
 interface C extends A {
 ....
 }
```

```
interface D {
....
}
class E extends B implements C,D {
...
}
```

„Wielodziedziczenie”

```
interface Walczący {
  void fight();
}

interface Pływający {
  void swim();
}

interface Latający {
  void fly();
}

class Wojownik {
  public void fight() {}
}

class Bohater extends Wojownik
    implements Walczący, Pływający, Latający {
  public void swim() {}
  public void fly() {}
}
```

```
public class GraPrzygodowa {
  static void t(Walczący x) { x.fight(); }
  static void u(Pływający x) { x.swim(); }
  static void v(Latający x) { x.fly(); }
  static void w(Wojownik x) { x.fight(); }
  public static void main(String[] args) {
    Bohater h = new Bohater();
    t(h);
    u(h);
    v(h);
    w(h);
  }
}
```


Kolizje nazw metod

```
interface I1 { void f(); }
interface I2 { int f(int i); }
interface I3 { int f(); }

class C { public int f() { return 1; } }

class C2 implements I1, I2 {
  public void f() {}
  public int f(int i) { return 1; }
                    //przeciążenie
}

class C3 extends C implements I2 {
  public int f(int i) { return 1; }
                   // przeciążenie
}
```

```
class C4 extends C implements I3 {
  //  Identyczna deklaracja
  public int f() { return 1; }
}

// To jest niepoprawne:
//! class C5 extends C
            implements I1 {}

//! interface I4 extends I1, I3 {}
```

Podsumowanie

- Klasy abstrakcyjne:
  - Klasy bazowe z niekompletną implementacją
  - Wspólny interfejs dla klas pochodnych
- Zastosowanie interfejsów:
  - „wielodziedziczenie”
  - polimorfizm
  - deklaracje stałych

Pytania

1. Czy klasa abstrakcyjna może zawierać metody ostateczne?
2. Co może zawierać interfejs?
3. Czy metody statyczne mogą być ostateczne, abstrakcyjne lub polimorficzne?

Koniec