Programowanie obiektowe

Wykład 5:
Dziedziczenie i polimorfizm

Kompozycja klas

```
class Drzewo {
…
}

class Krzew {
…
}
```

```
class Las {
Drzewo drzewostan[ ];
Krzew krzewostan[ ];
…
}
```

Mechanizm powszechnie stosowany w językach programowania!

Dziedziczenie

```
class SrodekCzyszczacy {
private String s = new String("Czysc");
public void append(String a) { s += a; }
public void rozciencz() {
  append(" rozciencz");
 }
public void uzyj() {
   append(" uzyj");
 }
public void szoruj() {
   append(" szoruj");
 }
public void print() {
   System.out.println(s);
}
  public static void main(String[ ] args) {
    SrodekCzyszczacy x = new
                  SrodekCzyszczacy();
    x.rozciencz(); x.uzyj(); x.szoruj();
    x.print();
  }
}
```

```
public class Detergent extends
  SrodekCzyszczacy {
 // Modyfikacja metody
 public void szoruj() {
   append(" Detergent.szoruj");
   super.szoruj();
 }
 //  Dodanie metody
 public void piana() { append("piana"); }

 public static void main(String[ ] args) {
   Detergent x = new Detergent();
   x.rozciencz();
   x.uzyj();
   x.szoruj();
   x.piana();
   x.print();
   System.out.println("Test klasy bazowej");
   SrodekCzyszczacy.main(args);
 }
}
```


Inicjalizacja

```
class Artysta {
  Artysta() {
    System.out.println("Konstruktor artysty");
  }
}

class Plastyk extends Artysta {
  Plastyk() {
    System.out.println("Konstruktor plastyka");
  }
}

public class Grafik extends Plastyk {
  Grafik() {
    System.out.println("Konstruktor grafika");
  }
  public static void main(String[ ] args) {
    Grafik x = new Grafik();
  }
}
```

Wynik?

```
Konstruktor artysty
Konstruktor plastyka
Konstruktor grafika
```

Inicjalizacja, c.d.

```
class Gra {
  Gra(int i) {
    System.out.println(„Konstruktor gry");
  }
}

class GraPlanszowa extends Gra {
  GraPlanszowa(int i) {
    super(i);
    System.out.println(„Konstruktor gry planszowej");
  }
}

public class Chinczyk extends GraPlanszowa {
  Chinczyk() {
    super(11);
    System.out.println(„Konstruktor chinczyka");
  }
  public static void main(String[ ] args) {
    Chinczyk x = new Chinczyk();
  }
}
```

Musi być pierwszą instrukcją!!

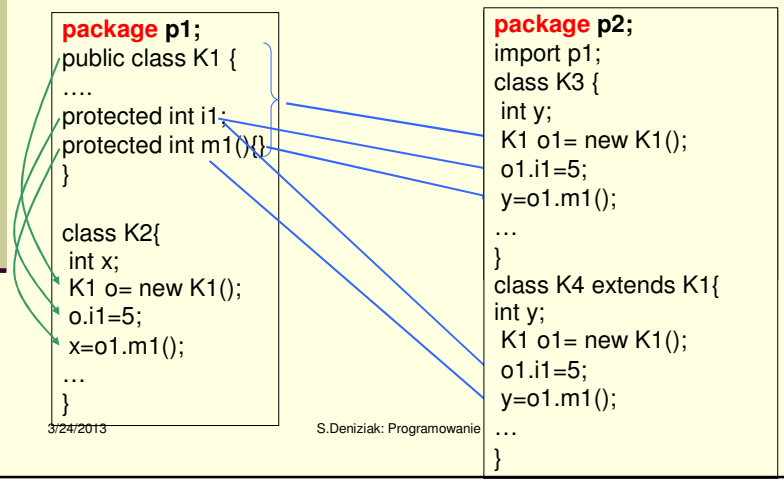
Usuwanie obiektów

```
class K1 {
 …
  finalize() {
    …
  }
}

class K2 extends K1 {
…
finalize(i) {
  …
  super.finalize();
  }
}
```

Modyfikatory dostępu, cd.

protected – dostęp chroniony (tylko w pakiecie i podklasach)



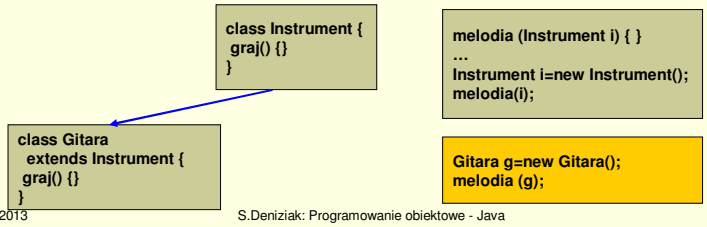
Porównanie modyfikatorów dostępu

Modyfikator	Klasa	Pakiet	Podklasa	Wszędzie
public	T	T	T	T
protected	T	T	T	N
brak	T	T	N	N
private	T	N	N	N

Klasy: tylko publiczne lub „przyjazne”!!!

Zastosowanie dziedziczenia

- Dziedziczenie a kompozycja
  - dziedziczenie: gdy nowa klasa jest pewną wersją innej klasy
  - kompozycja: gdy nowa klasa ma inną klasę
- Przyrostowe tworzenie oprogramowania
- Rzutowanie w górę, polimorfizm



Przykład 1: rzutowanie w górę

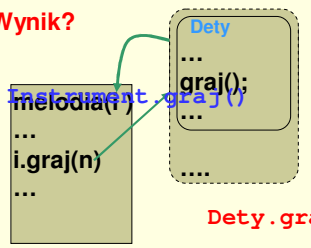
```
class Nuta {
  private int value;
  private Nuta(int val) { value = val; }
  public static Nuta
    C = new Nuta(0),
    Cis  = new Nuta(1),
    D   = new Nuta(2);
}

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

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

```
public class Muzyka {
  public static void melodia(Instrument i) {
    // ...
    i.graj(Nuta.C);
  }
  public static void main(String[] args) {
    Dety flet = new Dety();
    melodia(flet);
  }
}
```

Wynik?



run-time binding

Dety.graj()

Przykład 2: bez rzutowania w górę

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

Rzutowanie w górę

```
public class Muzyka {                    Instrument i
  public static void melodia(Dety i) {
    // ...
    i.graj(Nuta.C);
  }

public static void melodia(Strunowy i) {
    // ...
    i.graj(Nuta.C);
  }

  public static void main(String[] args) {
    Dety flet = new Dety();
    Strunowy skrzypce = new Strunowy();
    melodia(flet);
    melodia(skrzypce)
  }
}
```


Przykład 3: Przydatność polimorfizmu

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

class Kolo extends Ksztalt {
  void rysuj() {
    System.out.println("Kolo.rysuj()");
  }
  void usun() {
    System.out.println("Kolo.usun()");
  }
}
```

```
class Trojkat extends Ksztalt {
  void rysuj() {
    System.out.println("Trojkat.rysuj()");
  }
  void usun() {
    System.out.println("Trojkat.usun()");
  }
}
```

```
public class Ksztalty {
  public static Ksztalt generuj() {
    switch((int)(Math.random() * 3)) {
      default:
      case 0: return new Kolo();
      case 1: return new Kwadrat();
      case 2: return new Trojkat();
    }
  }
  public static void main(String[] args) {
    Ksztalt[] s = new Ksztalt[9];
    for(int i = 0; i < s.length; i++)
      s[i] = generuj();
    for(int i = 0; i < s.length; i++)
      s[i].rysuj();
  }
}
```

Znaczenie polimorfizmu

- Programowanie obiektowe:
  - enkapsulacja
  - dziedziczenie
  - polimorfizm !!!
- Zastosowanie:
  - separacja interfejsu od implementacji
  - tworzenie rozszerzalnych programów

Pytania

1. Co jest dziedziczone z klasy bazowej?
2. Zastosowania dziedziczenia.
3. Zasady inicjalizacji w podklasach i klasach bazowych.
4. Znaczenie i zastosowanie kwalifikatora protected.
5. Jakie warunki musi spełniać metoda aby mogła być metodą polimorficzną?

Koniec