Programozási nyelvek Java

Kozsik Tamás előadása alapján
Készítette: Nagy Krisztián

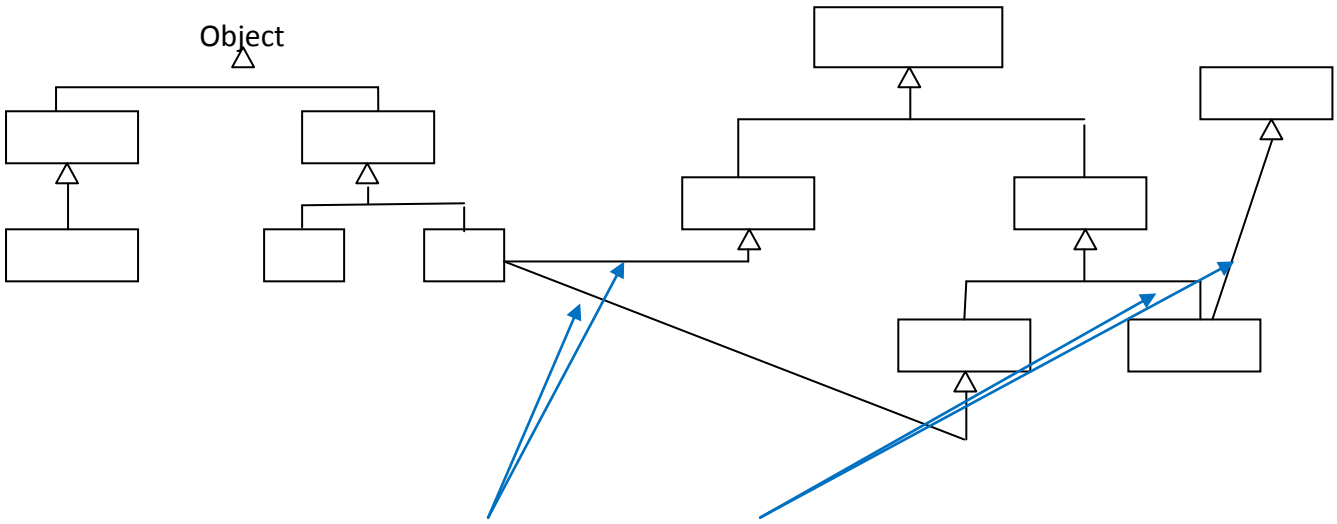
9. előadás

Interface

- típust vezet be, de osztálypéldány nem készíthető belőle (statikus típust ad)
- több osztály is megvalósíthatja

  ↕

  több interfacet is megvalósíthat egy osztály
- minél jobban ki tudjam fejezni magam (ősosztály csak egy van ↔interface lehet több)



többszörös öröklődés „multiple inheritance”

Osztályok közt technikai okok miatt nincs (- 2 egyforma közül melyiket használná?)

Java: - Létezik többszörös öröklődés

$extends_{Interface}$
implements
} többszörös altípusképzés

- Létezik többszörös öröklődés: $extends_{Osztály}$

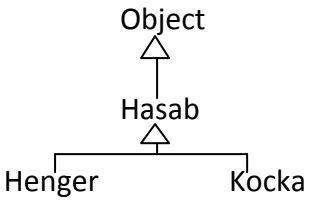
technikai komplikációk
Létezik olyan objektum orientált nyelv, amiben van (ilyenkor bonyolultabb nyelvi szabályok) Például: C++

Interface: publikus, absztrakt példányszintű metódusok
Absztrakt osztályok:

- Ugyan úgy csak típusozásra használhatóak, mint az interfacek
- Lehetnek abstract műveleteik, de tartalmazhatnak mezőket és implementációkat is
- Öröklés: NINCS többszörös
- extends reláció irányított fa marad

konkrét osztály	abstract osztály	interface
egyszeres öröklődés	egyszeres öröklődés	többszörös öröklődés
nem lehet abstract művelet	lehet abstract művelete	lehet abstract művelete
lehet konkrét művelete	lehet konkrét művelete	nem lehet konkrét művelete

```
class C{
        int x;
        void m(); // nélküle értelmes, vele kell az abstract
        { x = 1;}
}

abstract class Hasab{
        double magassag = 1.0;

        double terfogat(){
                return alapterulet() * magassag;
         }

        abstract double alapterulet(); // Abstract osztály esetén kell abstract
                                       // Interface esetén nem kell kiírni

}

class Henger extends Hasab{
      double sugar = 1.0;
      double alapterulet(){ return sugar*sugar*Math.PI;}
}

class Kocka extends Hasab{
      double alapterulet(){ return magassag * magassag;}
}
```



Egy abstract osztály bázisa lehet konkrét osztály, és fordítva.
Továbbá konkrét osztály bázisa konkrét osztály és abstract osztály bázisa abstract osztály.
Csak egyszeres öröklődés!

Hasáb: egy paraméteres definíció (itt paraméter az alapterulet()), a paramétert használhatom.

kóddal paraméterezhető osztálydefiníció
- formális paraméter: alapterulet()
A Henger és a Kocka aktuális paramétert feleltetnek meg a formálisnak.
(metódus implementációt)

A konkrét alosztály teljessé teszi a definíciót, felparaméterezi kóddal az abstract bázisosztályt.
Megjegyzés: Objektum-orientált módja a kóddal való paraméterezésnek.

Statikus típus: compiler eldönti, mely programok legálisak
Dinamikus típus: run-time eldönti, hogy egy példány szintű metódus melyik implementációját kell végrehajtani

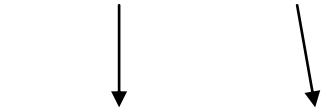
```
class Point{
        int x,y;
        @Override public String toString(){return x+";"+y;} // régi művelethez új törzs
        // megjelöljük a felüldefiniált függvényt „annotáció", nem kötelező, de ajánlott

        void move(int dx, int dy){ x+= dx; y += dy; } // új művelet
}

Object o = new Point();
o.move(3,7);
/*
Fordítási hiba!, statikus típus alapján
Nem minden „jó" program helyes
*/
o.toString();
/*
helyes, Object-ben van ilyen, ezért legális. Run-time a Pointban szereplő fut majd le
(dinamikus típus)
*/
```

(Point) o // statikus típusa Point lett
(Point) ”pityu” // fordítási hiba, nincs kapcsolat a Point és a String között
// nem downcast (String nem bázistípusa Point-nak)

Típuskényszerítés: (type cast): downcast, általánosabb típusból szeretnénk speciálisabb típust.
((Point) o).move(3,7); // helyes compiler szerint, lefordul

ha o = ”pityu”;
((Point) o).move(3,7); // a compiler szerint helyes, futás közben lesz baj
Futási hiba: ClassCastException
(a dinamikus típus, new megfelelő)

Néha kell: Statikus és dinamikus típus ellenörzésének kombinálása.



biztonság (le se fordul) flexibilitás (futásidejű típusok)

A statikus típus becslése a dinamikusnak, a dinamikus a pontos, ezért rugalmas a dinamikus típusellenörzés.

instanceof

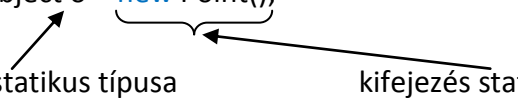
if (o instanceof Point) ((Point) o).move(3,7);
≡
o != null & dinamikus_tipus(o) $<: Point$

altípusa

Ez akkor is jó, ha o egy (pl) ColoredPoint()
dinamikus_tipus(o):= o által hivatkozott objektum létrehozásához használt típus.

Automatikus upcast: REFERENCIÁK!

Példa: Object o = new Point();

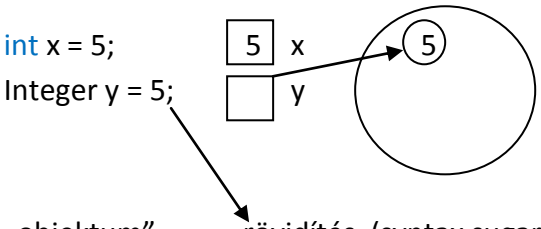


változó statikus típusa Object kifejezés statikus típusa Point

Ha altípusból megyek a bázisa: Point <: Object

Speciálisabb értéket általánosabb változóba:
Primitív típusokon automatikus konverzió
byte <: short <: int <: long <: float <: double
- értékreferenciája megváltozik (más bitek)

Átjárható a primitív típus és a referencia típus közti világ:
$int \sim java.lang.Integer$
$double \sim java.lang.Double$
$short \sim java.lang.Short$
$char \sim java.lang.Character$
… 8 darab Wrapper(csomagoló) osztályok



„objektum” rövidítés (syntax sugar ≡ szintaktikus cukorka)

```
Integer y = new Integer(5); // Igazság

class C{
        void m(int x){…}
        void n(Integer x){…}
}
new C().m(5);
new C().n(new Integer(5))
new C().n(5)
new C().m(new Integer(5))
```

autoboxing
auto-unboxing
} Automatikus konverziók

ugyanaz