Programozás III

OOP ÖRÖKLŐDÉS,
INTERFÉSZ - folytatás

OOP ALAPELVEK – ÖSSZEFOGLALVA

Objektumok: egy egységben az adatok és a rajtuk végzett műveletek. (Egyszerre a modularitás és a struktúra elemei.)

Egységbezárás (**az információ elrejtése**): Az objektumokat nem lehet kívülről nem várt módon manipulálni. Csak meghatározott metódusokon keresztül módosítható az állapot.

Öröklés: Létrehozhatók már létező típusok specializációi, melyek használhatják (és bővíthetik) a létező típusok műveleteit anélkül, hogy újra meg kellene valósítani őket.

Polimorfizmus: A referenciák különböző típusú objektumokra hivatkozhatnak, és a hivatkozott objektumoktól függő viselkedést produkálhatnak.

ÖRÖKLŐDÉS A JAVA-BAN

ÖRÖKLŐDÉS A JAVA-BAN

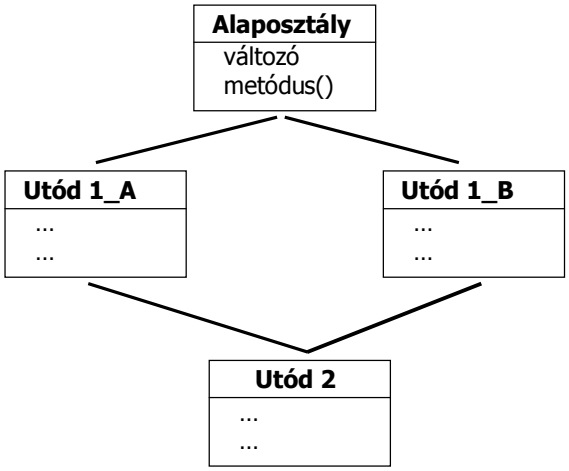
JAVA-ban csak egyszeres öröklődés van – pedig néha kellene többszörös is. (Pl. lakókocsi)

DE megvalósítható a többszörös öröklődés is.

Mi az oka a többszörös öröklődéstől való idegenkedésnek?

- a többszörös öröklődés problémát jelenthet a fordító számára (pl.: káró típusú öröklési lánc)
- biztonsági okok
(a hackerek könnyen kihasználhatják ezt a problémát)

KÁRÓ TÍPUSÚ ÖRÖKLÉSI LÁNC



Probléma: az Utód2 osztály kétszeresen örökölné az alaposztály tulajdonságait, ez pedig nem lehetséges.

TÖBBSZÖRÖS ÖRÖKLÉS – MEGOLDÁS: INTERFÉSZ

Az interfész konstans értékek és absztrakt metódusok deklarációjának az összessége.

vagyis:

- a metódusok törzs nélkül vannak deklarálva
- önmagában nem használhatók, implementálni kell őket
- az implementációt egy-egy osztály végzi

TÖBBSZÖRÖS ÖRÖKLÉS – MEGOLDÁS: INTERFÉSZ

Interfész létrehozása:

[*módosítók*] interface InterfeszNev [*extends*...]
{
//absztrakt metódusok
}

– az interfész módosítója csak **publikus** lehet
– az absztrakt metódusok csak publikusak lehetnek
– az interfész alapértelmezésben absztrakt
(nem lehet példányosítani)

TÖBBSZÖRÖS ÖRÖKLÉS – MEGOLDÁS: INTERFÉSZ

Interfész implementálása:

[*módosítók*] class OsztályNév implements InterfészNév{
//absztrakt metódusok
}

Több intrefész implementálása esetén vesszővel felsoroljuk az implements kulcsszó után az összes interfész nevét

KÖTELEZŐ AZ ÖSSZES ABSZTRAKT METÓDUS IMPLEMENTÁLÁSA!!!

TÖBBSZÖRÖS ÖRÖKLÉS – MEGOLDÁS: INTERFÉSZ

Egy osztály több interfészt is implementálhat:

```
public class Osztaly implements Egyik, Masik {
```

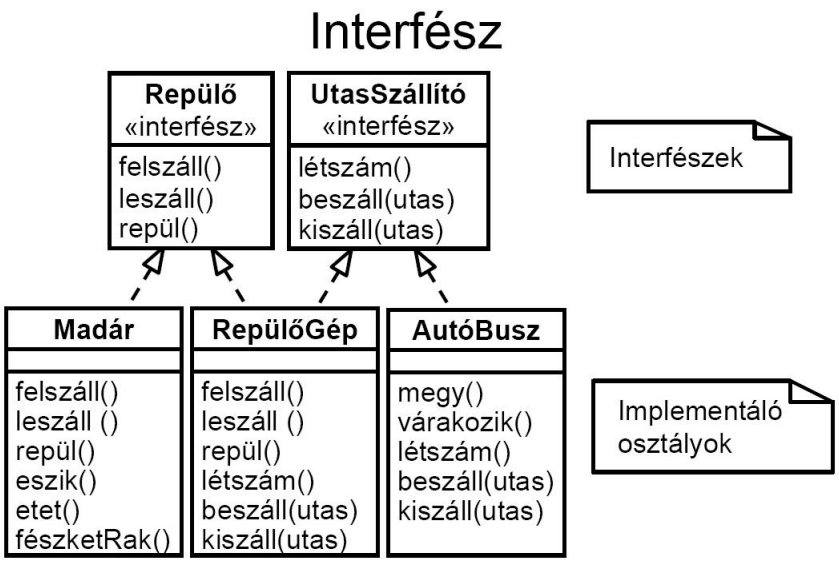
Interfészek között is létezik öröklődés – jelölése szintén az **extends** kulcsszó.

Ezért alkalmas a többszörös öröklés megvalósítására!!!

Sőt, interfészek között többszörös öröklődés is lehet:

```
public interface Utod extends Egyik, Masik {
```

INTERFÉSZ – UML (PÉLDA)



INTERFÉSZ – PÉLDA

```
interface Ital {
    void koccint();
}

interface Alkohol extends Ital{
    boolean megart(int pohar);
}

class Sor implements Alkohol {

    public boolean megart (int pohar){
        if (pohar < 2) return false; else return true;
    }

    public void koccint() {
        System.out.println("Nem koccintunk.");
    }
}

public class Pelda {

    public static void main(String args[]){
        Sor b = new Sor();
        int i=3;
        if (b.megart(i)) System.out.println("Vigyázz, megárt!");
        b.koccint();
    }
}
```


```
interface Ital {
    void koccint();
}

interface Alkohol extends Ital{
    boolean megart(int pohar);
}

class Sor implements Alkohol {

    public boolean megart (int pohar){
        if (pohar < 2) return false; else return true;
    }

    public void koccint() {
        System.out.println("Nem koccintunk.");
    }
}
```

INTERFÉSZ – PÉLDA

```
public class Pelda {

    public static void main(String args[]){
        Sor b = new Sor();
        int i=3;
        if (b.megart(i)){
           System.out.println("Vigyázz, megárt!");
        }
        b.koccint();
    }
}
```

Lehet-e egy forrásfájlban az összes?

INTERFÉSZ – PÉLDA

Fordításkor létrejött fájlok:

- Ital.class
- Alkohol.class
- Sor.class
- Pelda.class

Eredmény:

```
--------------------
Vigyázz, megárt!
Nem koccintunk.

Process completed.
```

INTERFÉSZ ALKALMAZÁSOK

Interfész „saját” alkalmazása:
Ha nagy a projekt, és többen dolgoznak rajta, akkor először célszerű felületeket tervezni interface alapon, azután jöhet az osztálygyártás. – Az interfész egy vázat ad (egy tervrajz).

A Java sok „kész” interfészt használ:
A Java fejlődése: Egy új igény esetén születik egy jsr (Java Service Request) felület-definíció (interface), majd a gyártók elkészítik a saját megvalósításaikat.

Ilyen pl. jdbc. A jdbc egy interface gyűjtemény (felület), amelyet minden adatbázisgyártó implementál.
A grafikus csomagok is sok interfészt használnak.

ELEGÁNS PROJECT-SZERKEZET

Javasolt felépítés:

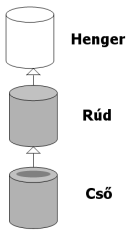
– interface (esetleg extends Comparable)

– az interface-t implementáló abstract osztály

– az abstract osztályt kiterjesztő további osztályok

Pl.

```
public interface AlakzatInterface {
public abstract class AbsztraktAlakzat implements AlakzatInterface {
public class Henger extends AbsztraktAlakzat{
    public class Rud extends Henger {
        public class Cso extends Rud {
```

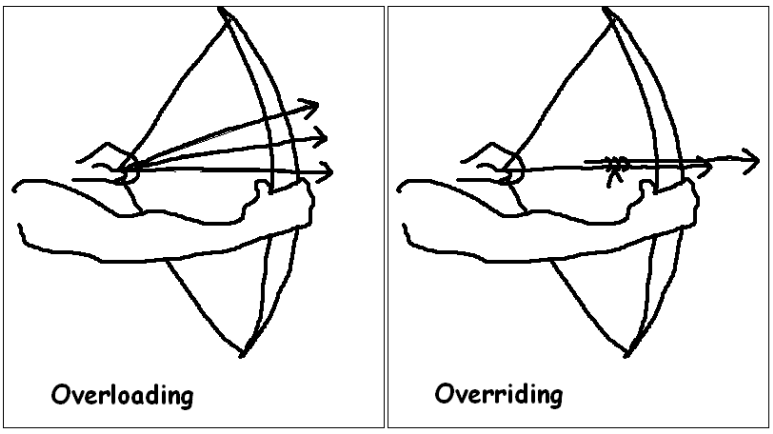


KÉRDÉSEK

1. Min múlik, hogy interfész vagy absztrakt osztály?

2. Mi a különbség az overload és override között?

OVERLOAD vs OVERRIDE



OVERLOAD, OVERRIDE

Kiválasztás fordítási időben. (statikus kötés)	Kiválasztás futási időben. (dinamikus kötés)
private, static, final túlterhelhető.	private, static, final nem definiálható felül.
Azonos név, eltérő szignatúra. (szignatúra: név + paraméterek típusa, száma)	Név, szignatúra, visszatérési típus azonos.
Azonos osztályban vannak.	Különböző osztályokban vannak.