Programozási Ismeretek

Dr. Gergely Tamás
Dr. Ferenc Rudolf, Dr. Jász Judit, Dr. Kiss Ákos

Szegedi Tudományegyetem
Informatikai Intézet
Szoftverfejlesztés Tanszék

2024

(v0214)

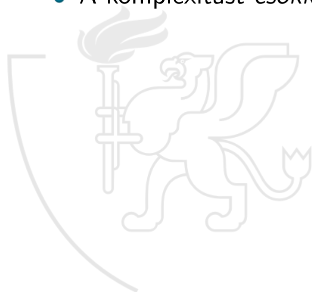
Tartalom

1. Bevezetés
   - Bemutatkozás
2. Modellezés
   - Modellezés
   - UML
   - Modell, nézet és diagram
   - OOP alapfogalmak
3. Objektumorientált programozás
   - Bevezetés
   - OOP
4. Java alkalmazások
   - Bevezetés
   - Alapelvek
   - Típusok
   - Java programok
   - Műveletek
   - Vezérlés
5. Memória-menedzsment
   - Inicializálás
   - Memória felszabadítás
   - Láthatóság
6. Újrafelhasználhatóság
   - Kompozíció és öröklődés
   - Végső dolgok
   - Osztálytagok
   - Hivatkozások és típusuk
7. Polimorfizmus
   - Dinamikus polimorfizmus
   - Absztrakt osztályok
   - Interfészek
   - Felsorolás
   - Belső osztályok
8. Objektumok tárolása
   - Tömbök
   - Kollekciók
   - Generikus kollekciók
9. Java
   - IO
   - Kivételkezelés
   - Reflection
10. Java
    - GUI
    - AWT/Swing
    - JFX
11. Java
    - Generics
    - Anonymous osztályok
    - Lambda kifejezések
    - Annotations
    - Változó paraméterlista

Tartalom

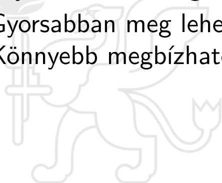
1. Bevezetés
   - Bemutatkozás
2. Modellezés
   - Modellezés
   - UML
   - Modell, nézet és diagram
   - OOP alapfogalmak
3. Objektumorientált programozás
   - Bevezetés
   - OOP
4. Java alkalmazások
   - Bevezetés
   - Alapelvek
   - Típusok
   - Java programok
   - Műveletek
   - Vezérlés
5. Memória-menedzsment
   - Inicializálás
   - Memória felszabadítás
   - Láthatóság
6. Újrafelhasználhatóság
   - Kompozíció és öröklődés
   - Végső dolgok
   - Osztálytagok
   - Hivatkozások és típusuk
7. Polimorfizmus
   - Dinamikus polimorfizmus
   - Absztrakt osztályok
   - Interfészek
   - Felsorolás
   - Belső osztályok
8. Objektumok tárolása
   - Tömbök
   - Kollekciók
   - Generikus kollekciók
9. Java
   - IO
   - Kivételkezelés
   - Reflection
10. Java
    - GUI
    - AWT/Swing
    - JFX
11. Java
    - Generics
    - Anonymous osztályok
    - Lambda kifejezések
    - Annotations
    - Változó paraméterlista

- Az kurzus célja, hogy elsajátítsuk
  - az objektum orientált gondolkodásmódot,
  - az osztálydiagramok használatát, és
  - a Java-t, mint tisztán objektum orientált nyelvet.
- Java
  - Új nyelv, sok jó tulajdonságot átvett: C, C++, Smalltalk, ...
  - A komplexitást *csökkenti* a programozó szempontjából



Komplexitás

- A komplexitást általában növeli:
  - A nyelv „gépközelisége”
  - A követelmények bővülése: a nyelvek is bonyolultabbak lesznek
  - A kompatibilitási problémák: C++ kompatíbilis C-vel, Perl a Sed-del, Visual Basic a BASIC-kel, ...
- A komplexitás kezeléséből adódik:
  - Könnyebb robusztus kódot írni
  - Lassabban fut
- Hátrányai viszont megtérülnek:
  - Gyorsabban meg lehet tanulni a nyelvet
  - Könnyebb megbízható programot írni

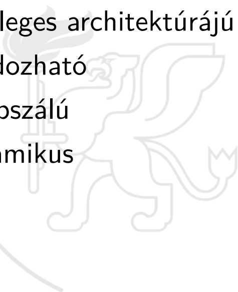


A Java nyelv

- Magas szintű programozási nyelv
  - Nem a kódoláson van a hangsúly, hanem a *tervezésen*
  - Elvontabb fogalmakat ábrázol (pl. interfész)
- Web programozás elősegítése
  - Platform-független
  - Beépített biztonsági rendszerek
- Nem „csak” egy nyelv
  - Számos egyéb alkalmazhatóság (applet, beágyazott rendszerek, ...)

A Java fő jellemzői

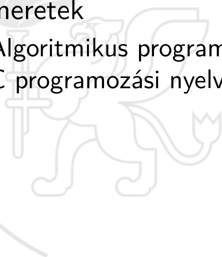
- Egyszerű
- Objektumorientált
- Előfordított
- Értelmezett
- Robusztus
- Biztonságos
- Semleges architektúrájú
- Hordozható
- Többszálú
- Dinamikus



A Java nyelv

A kurzus anyaga

- Java nyelv 1.2-1.6(-1.9) verziója
- Az alapokat mutatjuk be és néhány kiegészítő könyvtárat
- Részletes referenciát és dokumentációt nem adunk, az könyvekben elérhető, pl.:
  - Bruce Eckel: Thinking in Java
    `http://www.mindview.net/Books/TIJ/Solutions`
  - `https://docs.oracle.com/javase/6/docs/api/`
- Előismeretek
  - Algoritmikus programozás alapjai
  - C programozási nyelv



- Kódolási stílus:
  - Ajánlatos követni a Sun stílusát
- Programozási irányelvek:
  - Sok jó anyag van, egy jó gyűjtemény van pl. az Eckel könyv végén

Tartalom

1 Bevezetés
- Bemutatkozás

2 Modellezés
- Modellezés
- UML
- Modell, nézet és diagram
- OOP alapfogalmak

3 Objektumorientált programozás
- Bevezetés
- OOP

4 Java alkalmazások
- Bevezetés
- Alapelvek
- Típusok
- Java programok
- Műveletek
- Vezérlés

5 Memória-menedzsment
- Inicializálás
- Memória felszabadítás
- Láthatóság

6 Újrafelhasználhatóság
- Kompozíció és öröklődés
- Végső dolgok
- Osztálytagok
- Hivatkozások és típusuk

7 Polimorfizmus
- Dinamikus polimorfizmus
- Absztrakt osztályok
- Interfészek
- Felsorolás
- Belső osztályok

8 Objektumok tárolása
- Tömbök
- Kollekciók
- Generikus kollekciók

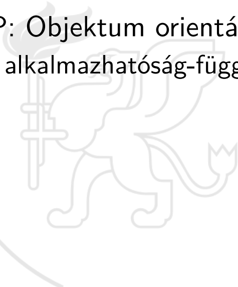
9 Java
- IO
- Kivételkezelés
- Reflection

10 Java
- GUI
- AWT/Swing
- JFX

11 Java
- Generics
- Anonymous osztályok
- Lambda kifejezések
- Annotations
- Változó paraméterlista

Absztrakció

- Minél bonyolultabb a programozás problémája, annál absztraktabb megoldások kellenek
- Assembly: absztrakció a gép felett
- Fortran, C: absztrakció assembly felett
  - még mindig gép-orientált elvonatkoztatás
- Probléma orientált absztrakció: LISP, APL
  - korlátozott az alkalmazhatóság
- OOP: Objektum orientált programozás
  - alkalmazhatóság-független absztrakció



Objektum és osztály

- **Objektum**
  - A probléma egy elemének alkalmazhatóság-független absztrakciója
- Egy tisztán OO programban minden objektum
- **Program**
  - Egymással kommunikáló objektumok összessége
- Minden objektumot kisebb objektumokból állítunk össze (akár alaptípusúakból)
- **Osztály**
  - Az objektumok típusa
  - A `class` kulcsszóval definiáljuk
- Ugyanolyan típusú objektumok ugyanolyan üzeneteket fogadhatnak
  - a gyakorlatban ez relaxálható $\rightarrow$ polimorfizmus

Osztály interfésze

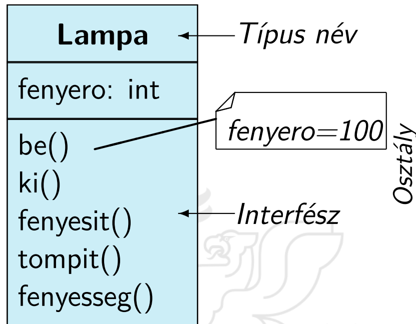
- A sok egyedi objektum között vannak olyanok, melyeknek közös tulajdonságaik és viselkedéseik vannak, vagyis egyazon családba – osztályba tartoznak
- A Simula-67 volt az első ilyen nyelv
- Az osztály egyben egy *absztrakt adattípus* is
  - Adatok és a rajtuk végzett műveletek *egységbezárása* (encapsulation)
- Ugyanúgy viselkedik mint minden egyéb primitív típus
  - pl. változó (objektum) hozható létre

Osztály interfésze
(folyt.)

- **Osztály**: tulajdonság + viselkedés
  - Tulajdonság = *attribútumok* (adattagok, mezők)
    - Mi lehet az állapot?
  - Viselkedés = *operációk* (metódusok, tagfüggvények)
    - Milyen üzenetekre és hogyan reagál?
- **Objektum**: az osztály egy *példánya*, állapot + reakció
  - Állapot = *attribútumok értéke*
    - Mi az állapot?
  - Üzenetküldés ≈ *függvényhívás*
    - Mire kell reagálnia?
  - Reakció = *operáció végrehajtása*
    - Mit reagál az üzenetre?

Interfész és implementáció

- **Interfész**: mi az amit üzenhetünk (≈*metódus deklaráció*)
- **Implementáció**: ami teljesíti a kérést (≈*metódus definíció*)



Osztály

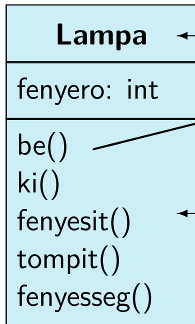
```
class Lampa {
    private int fenyero = 100;
    public void be() {fenyero = 100;}
    public void ki() {fenyero = 0;}
    public void fenyesit() {
        if (fenyero <= 90) fenyero += 10;
    }
    public void tompit() {
        if (fenyero >= 10) fenyero -= 10;
    }
    public int fenyesseg() {
        return fenyero;
    }
}
```

Objektum

```
Lampa lampa1 = new Lampa();
lampa1.tompit();
System.out.println(lampa1.fenyesseg());
lampa1.ki();
System.out.println(lampa1.fenyesseg());
lampa1.be();
System.out.println(lampa1.fenyesseg());
```

Interfész és implementáció

- **Interfész**: mi az amit üzenhetünk (≈*metódus deklaráció*)
- **Implementáció**: ami teljesíti a kérést (≈*metódus definíció*)



Típus név

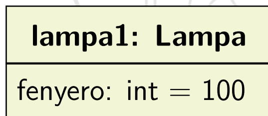
fenyero=100

Interfész

Osztály

```
class Lampa {
    private int fenyero = 100;
    public void be() {fenyero = 100;}
    public void ki() {fenyero = 0;}
    public void fenyesit() {
        if (fenyero <= 90) fenyero += 10;
    }
    public void tompit() {
        if (fenyero >= 10) fenyero -= 10;
    }
    public int fenyesseg() {
        return fenyero;
    }
}
```

Probléma eleme a valóságban

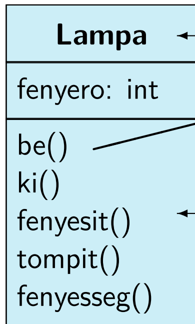


Objektum

```
Lampa lampa1 = new Lampa();
lampa1.tompit();
System.out.println(lampa1.fenyesseg());
lampa1.ki();
System.out.println(lampa1.fenyesseg());
lampa1.be();
System.out.println(lampa1.fenyesseg());
```

Interfész és implementáció

- **Interfész**: mi az amit üzenhetünk (≈*metódus deklaráció*)
- **Implementáció**: ami teljesíti a kérést (≈*metódus definíció*)



Típus név

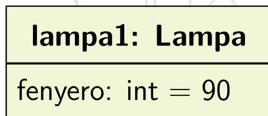
fenyero=100

Interfész

Osztály

```
class Lampa {
    private int fenyero = 100;
    public void be() {fenyero = 100;}
    public void ki() {fenyero = 0;}
    public void fenyesit() {
        if (fenyero <= 90) fenyero += 10;
    }
    public void tompit() {
        if (fenyero >= 10) fenyero -= 10;
    }
    public int fenyesseg() {
        return fenyero;
    }
}
```

Probléma eleme a valóságban

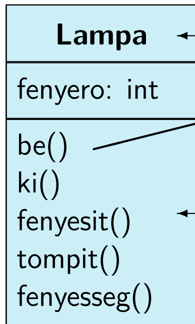


Objektum

```
Lampa lampa1 = new Lampa();
lampa1.tompit();
System.out.println(lampa1.fenyesseg());
lampa1.ki();
System.out.println(lampa1.fenyesseg());
lampa1.be();
System.out.println(lampa1.fenyesseg());
```

Interfész és implementáció

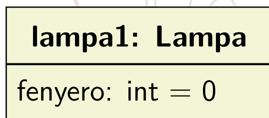
- **Interfész**: mi az amit üzenhetünk (≈*metódus deklaráció*)
- **Implementáció**: ami teljesíti a kérést (≈*metódus definíció*)



Osztály

```java
class Lampa {
    private int fenyero = 100;
    public void be() {fenyero = 100;}
    public void ki() {fenyero = 0;}
    public void fenyesit() {
        if (fenyero <= 90) fenyero += 10;
    }
    public void tompit() {
        if (fenyero >= 10) fenyero -= 10;
    }
    public int fenyesseg() {
        return fenyero;
    }
}
```

Probléma eleme a valóságban

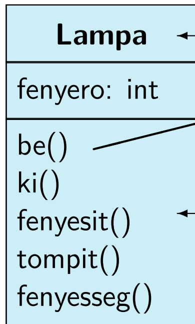


Objektum

```java
Lampa lampa1 = new Lampa();
lampa1.tompit();
System.out.println(lampa1.fenyesseg());
lampa1.ki();
System.out.println(lampa1.fenyesseg());
lampa1.be();
System.out.println(lampa1.fenyesseg());
```

Interfész és implementáció

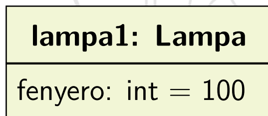
- **Interfész**: mi az amit üzenhetünk (≈*metódus deklaráció*)
- **Implementáció**: ami teljesíti a kérést (≈*metódus definíció*)



Osztály

```
class Lampa {
    private int fenyero = 100;
    public void be() {fenyero = 100;}
    public void ki() {fenyero = 0;}
    public void fenyesit() {
        if (fenyero <= 90) fenyero += 10;
    }
    public void tompit() {
        if (fenyero >= 10) fenyero -= 10;
    }
    public int fenyesseg() {
        return fenyero;
    }
}
```

Probléma eleme a valóságban



Objektum

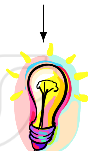
```
Lampa lampa1 = new Lampa();
lampa1.tompit();
System.out.println(lampa1.fenyesseg());
lampa1.ki();
System.out.println(lampa1.fenyesseg());
lampa1.be();
System.out.println(lampa1.fenyesseg());
```

Implementáció elrejtése

- OO programozás közben két feladatot szoktunk megoldani
  - osztályt gyártunk (magunknak vagy másoknak)
  - osztályt használunk (miénket vagy másét)
- Ha osztályt gyártunk, el kell rejtenünk az implementációt
  - a használója nem kell hogy ismerje, nem tud róla, így nem használhatja rosszul és nem is teheti tönkre → kevesebb lesz a programhiba

Probléma eleme a valóságban

lampa1: Lampa
fenyero: int = 100



```java
Lampa lampa1 = new Lampa();
lampa1.fenyero = -100;
System.out.println(lampa1.fenyesseg());
```

Implementáció elrejtése

- OO programozás közben két feladatot szoktunk megoldani
  - osztályt gyártunk (magunknak vagy másoknak)
  - osztályt használunk (miénket vagy másét)
- Ha osztályt gyártunk, el kell rejtenünk az implementációt
  - a használója nem kell hogy ismerje, nem tud róla, így nem használhatja rosszul és nem is teheti tönkre → kevesebb lesz a programhiba

Probléma eleme a valóságban???

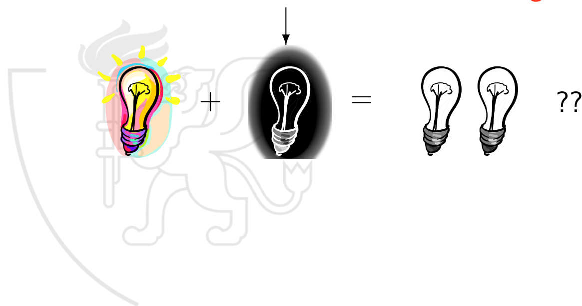
lampa1: Lampa
fenyero: int = -100

```java
Lampa lampa1 = new Lampa();
lampa1.fenyero = -100;
System.out.println(lampa1.fenyesseg());
```

Implementáció elrejtése

- OO programozás közben két feladatot szoktunk megoldani
  - osztályt gyártunk (magunknak vagy másoknak)
  - osztályt használunk (miénket vagy másét)
- Ha osztályt gyártunk, el kell rejtenünk az implementációt
  - a használója nem kell hogy ismerje, nem tud róla, így nem használhatja rosszul és nem is teheti tönkre → kevesebb lesz a programhiba

Probléma eleme a valóságban???



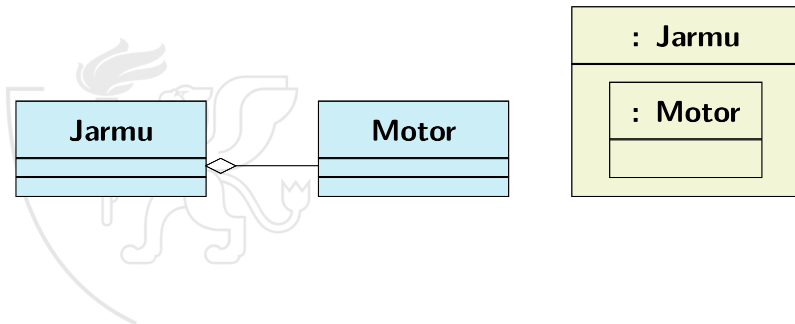
Implementáció elrejtése

Láthatóság

- Elérés vezérlése (láthatóság) – access specifiers
  - elrejtésre
  - implementáció biztonságos módosítása
  - `public`, `protected`, `private`, alapértelmezett: „friendly” (*package private*: csomagon belül public, egyébként private)

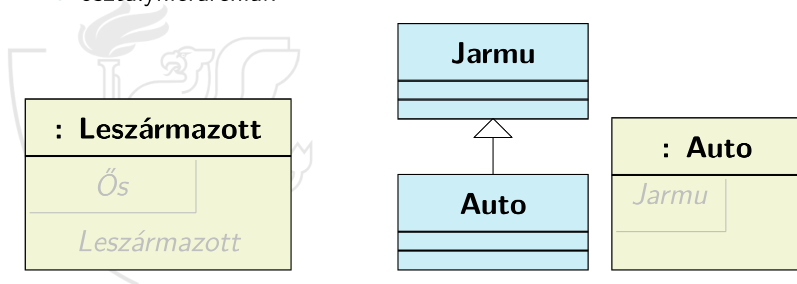
Implementáció újrafelhasználása

- Az újrafelhasználhatóság az OOP egyik legfontosabb előnye
- Kompozíció, aggregáció
  - objektum használata másik osztályban
  - futás közben változtatható
  - általában private
- Öröklődés is egy alternatíva, de nem feltétlenül az a legjobb



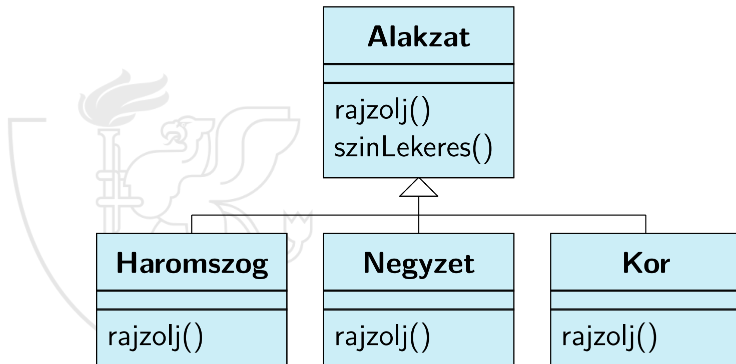
Interfész újrafelhasználása

- „Hasonló” osztályokat ne kelljen mindig újra és újra létrehozni, elég legyen „klónozni”, majd bővíteni/módosítani.
- Ezt **öröklődésnek** nevezzük
  - ős, szülő, alap (base, super) / gyerek, leszármaztatott (derived, child)
  - ha az ős változik, a származtatott is „módosul”
  - az elérhetőség fontos: a *private* nem elérhető (de az objektum része!), a *protected* és *public* viszont igen
  - osztályhierarchiák



Öröklődés jelentése

- Hasonlóság kifejezése az ős felé: **általánosítás**
- Különbség a gyerek felé: **specializálás**
- A származtatott új típus lesz
  - az ős interfészét duplikálja
  - azonos típusú az őssel (a kör az egy alakzat)



Osztály bővítései

- Attribútumok:
  - Új attribútumok felvételével finomíthatjuk a lehetséges *állapotok* halmazát
  - (Az örökölt attribútumok nem módosíthatóak vagy törölhetők)
- Operációk:
  - Új operációk felvételével bővíthetjük az osztály *interfészét*
  - Meglévő operációk **felüldefiniálásával** (**overriding**) módosíthatjuk az ős *viselkedését*
  - (Az örökölt operációk nem törölhetők)

Polimorfizmus

Dinamikus polimorfizmus

- Többalakúság
- Objektumok felcserélhetőségét biztosítja
- Az objektumot az őstípusa alapján kezeljük
  - a kód nem függ a specifikus típusoktól
  - utólag is lehet definiálni leszármazottakat
- Az ős osztály interfészét használjuk
- A fordító nem tudja melyik konkrét operáció hívódik (az ősé vagy a leszármazotté)
  - a programozó nem is kívánja megmondani
  - futás közben derül ki konkrét típus alapján

Polimorfizmus

Dinamikus polimorfizmus (folyt.)

- Mikor döntjük el, hogy egy hívás hatására melyik függvény/metódus hívódik meg?
  - későn – futás közben
  - korán – fordításkor
- OOP – kései kötés
  - Fordítási időben csak az operáció kandidátusok adottak (virtuális táblák)
  - A konkrét hívás futási időben dől el (objektum típus alapján)
  - Java-ban minden operáció hívás ilyen
- Nem OOP – korai kötés
  - pl. Pascal, C
  - A hívott eljárás fordítási időben beazonosítható (a hívási cím fordítási időben adott)

Polimorfizmus
Példa

```
class Alakzat {
    public void rajzolj() {
        System.out.println("Alakzat");
    }
}
```

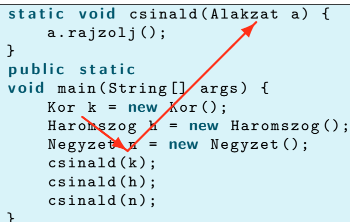
```
class Haromszog extends Alakzat {
    public void rajzolj() {
        System.out.println("Haromszog");
    }
}
```

```
class Negyzet extends Alakzat {
    public void rajzolj() {
        System.out.println("Negyzet");
    }
}
```

```
class Kor extends Alakzat {
    public void rajzolj() {
        System.out.println("Kor");
    }
}
```

```
public class AlakzatPelda {
    static void csinald(Alakzat a) {
        a.rajzolj();
    }
    public static
    void main(String[] args) {
        Kor k = new Kor();
        Haromszog h = new Haromszog();
        Negyzet n = new Negyzet();
        csinald(k);
        csinald(h);
        csinald(n);
    }
}
```

- Könnyen bővíthető a program (nincs típus-specifikusság)
- `Alakzat` helyett `Kor` van átadva
  - Ez megtehető, mert a kör az egy alakzat
  - Ez az **upcasting** (beleolvasztás)
- A `csinald()` függvényben nincs megkülönböztetés a típusok szerint!



```
static void csinald(Alakzat a) {
    a.rajzolj();
}
public static
void main(String[] args) {
    Kor k = new Kor();
    Haromszog h = new Haromszog();
    Negyzet n = new Negyzet();
    csinald(k);
    csinald(h);
    csinald(n);
}
```

- Melyik `rajzolj()` operáció fog hívódni?
  - Fordítási időben nem dönthető el!
  - Futási időben derül ki
- Egy speciális mechanizmus működik, amely futás közben rendeli hozzá a híváshoz a megfelelő implementációt (kései kötés)

```
class Alakzat {
    public void rajzolj() {
        System.out.println("Alakzat");
    }
}
```

```
class Negyzet extends Alakzat {
    public void rajzolj() {
        System.out.println("Negyzet");
    }
}
```

```
class Kor extends Alakzat {
    public void rajzolj() {
        System.out.println("Kor");
    }
}
```

```
public class AlakzatPelda {
    static void csinald(Alakzat a) {
        a.rajzolj();
    }
    public static
    void main(String[] args) {
        Kor k = new Kor();
        Haromszog h = new Haromszog();
        Negyzet n = new Negyzet();
        csinald(k);
        csinald(h);
        csinald(n);
    }
}
```

Polimorfizmus

Kései kötés

- Melyik `rajzolj()` operáció fog hívódni?
  - Fordítási időben nem dönthető el!
  - Futási időben derül ki
- Egy speciális mechanizmus működik, amely futás közben rendeli hozzá a híváshoz a megfelelő implementációt (kései kötés)

```
class Alakzat {
    public void rajzolj() {
        System.out.println("Alakzat");
    }
}
```

```
class Negyzet extends Alakzat {
    public void rajzolj() {
        System.out.println("Negyzet");
    }
}
```

```
class Kor extends Alakzat {
    public void rajzolj() {
        System.out.println("Kor");
    }
}
```

```
public class AlakzatPelda {
    static void csinald(Alakzat a) {
        a.rajzolj();
    }
    public static
    void main(String[] args) {
        Kor k = new Kor();
        Haromszog h = new Haromszog();
        Negyzet n = new Negyzet();
        csinald(k);
        csinald(h);
        csinald(n);
    }
}
```

Polimorfizmus

Kései kötés

- Melyik `rajzolj()` operáció fog hívódni?
  - Fordítási időben nem dönthető el!
  - Futási időben derül ki
- Egy speciális mechanizmus működik, amely futás közben rendeli hozzá a híváshoz a megfelelő implementációt (kései kötés)

```
class Alakzat {
    public void rajzolj() {
        System.out.println("Alakzat");
    }
}
```

```
class Negyzet extends Alakzat {
    public void rajzolj() {
        System.out.println("Negyzet");
    }
}
```

```
class Kor extends Alakzat {
    public void rajzolj() {
        System.out.println("Kor");
    }
}
```

```
public class AlakzatPelda {
    static void csinald(Alakzat a) {
        a.rajzolj();
    }
    public static
    void main(String[] args) {
        Kor k = new Kor();
        Haromszog h = new Haromszog();
        Negyzet n = new Negyzet();
        csinald(k);
        csinald(h);
        csinald(n);
    }
}
```

- Melyik `rajzolj()` operáció fog hívódni?
  - Fordítási időben nem dönthető el!
  - Futási időben derül ki
- Egy speciális mechanizmus működik, amely futás közben rendeli hozzá a híváshoz a megfelelő implementációt (kései kötés)

```
class Alakzat {
    public void rajzolj() {
        System.out.println("Alakzat");
    }
}
```

```
class Negyzet extends Alakzat {
    public void rajzolj() {
        System.out.println("Negyzet");
    }
}
```

```
class Kor extends Alakzat {
    public void rajzolj() {
        System.out.println("Kor");
    }
}
```

```
public class AlakzatPelda {
    static void csinald(Alakzat a) {
        a.rajzolj();
    }
    public static
    void main(String[] args) {
        Kor k = new Kor();
        Haromszog h = new Haromszog();
        Negyzet n = new Negyzet();
        csinald(k);
        csinald(h);
        csinald(n);
    }
}
```

Polimorfizmus

Kései kötés

- Melyik `rajzolj()` operáció fog hívódni?
  - Fordítási időben nem dönthető el!
  - Futási időben derül ki
- Egy speciális mechanizmus működik, amely futás közben rendeli hozzá a híváshoz a megfelelő implementációt (kései kötés)

```java
class Alakzat {
    public void rajzolj() {
        System.out.println("Alakzat");
    }
}
```

```java
class Negyzet extends Alakzat {
    public void rajzolj() {
        System.out.println("Negyzet");
    }
}
```

```java
class Kor extends Alakzat {
    public void rajzolj() {
        System.out.println("Kor");
    }
}
```

```java
public class AlakzatPelda {
    static void csinald(Alakzat a) {
        a.rajzolj();
    }
    public static
    void main(String[] args) {
        Kor k = new Kor();
        Haromszog h = new Haromszog();
        Negyzet n = new Negyzet();
        csinald(k);
        csinald(h);
        csinald(n);
    }
}
```

Polimorfizmus

Kései kötés

- Melyik `rajzolj()` operáció fog hívódni?
  - Fordítási időben nem dönthető el!
  - Futási időben derül ki
- Egy speciális mechanizmus működik, amely futás közben rendeli hozzá a híváshoz a megfelelő implementációt (kései kötés)

```
class Alakzat {
    public void rajzolj() {
        System.out.println("Alakzat");
    }
}
```

```
class Negyzet extends Alakzat {
    public void rajzolj() {
        System.out.println("Negyzet");
    }
}
```

```
class Kor extends Alakzat {
    public void rajzolj() {
        System.out.println("Kor");
    }
}
```

```
public class AlakzatPelda {
    static void csinald(Alakzat a) {
        a.rajzolj();
    }
    public static
    void main(String[] args) {
        Kor k = new Kor();
        Haromszog h = new Haromszog();
        Negyzet n = new Negyzet();
        csinald(k);
        csinald(h);
        csinald(n);
    }
}
```

Polimorfizmus

Kései kötés

- Melyik `rajzolj()` operáció fog hívódni?
  - Fordítási időben nem dönthető el!
  - Futási időben derül ki
- Egy speciális mechanizmus működik, amely futás közben rendeli hozzá a híváshoz a megfelelő implementációt (kései kötés)

```java
class Alakzat {
    public void rajzolj() {
        System.out.println("Alakzat");
    }
}
```

```java
class Negyzet extends Alakzat {
    public void rajzolj() {
        System.out.println("Negyzet");
    }
}
```

```java
class Kor extends Alakzat {
    public void rajzolj() {
        System.out.println("Kor");
    }
}
```

```java
public class AlakzatPelda {
    static void csinald(Alakzat a) {
        a.rajzolj();
    }
    public static
    void main(String[] args) {
        Kor k = new Kor();
        Haromszog h = new Haromszog();
        Negyzet n = new Negyzet();
        csinald(k);
        csinald(h);
        csinald(n);
    }
}
```

- Melyik `rajzolj()` operáció fog hívódni?
  - Fordítási időben nem dönthető el!
  - Futási időben derül ki
- Egy speciális mechanizmus működik, amely futás közben rendeli hozzá a híváshoz a megfelelő implementációt (kései kötés)

```java
class Alakzat {
    public void rajzolj() {
        System.out.println("Alakzat");
    }
}
```

```java
class Negyzet extends Alakzat {
    public void rajzolj() {
        System.out.println("Negyzet");
    }
}
```

```java
class Kor extends Alakzat {
    public void rajzolj() {
        System.out.println("Kor");
    }
}
```

```java
public class AlakzatPelda {
    static void csinald(Alakzat a) {
        a.rajzolj();
    }
    public static
    void main(String[] args) {
        Kor k = new Kor();
        Haromszog h = new Haromszog();
        Negyzet n = new Negyzet();
        csinald(k);
        csinald(h);
        csinald(n);
    }
}
```

Objektum orientáltság

- **Egységbezárás** – absztrakt adattípus
  - Adatok és a rajtuk végzett műveletek egységbezárása
- **Öröklődés**
  - Interfész újrafelhasználása
- **Polimorfizmus** – többalakúság
  - Kései kötés