



Programozási technológia I.

Osztályszintű elérés, kivételkezelés, fájlkezelés

Osztályszintű elérés

Osztályszintű attribútumok

Osztályszintű műveletek

Segédosztályok

Singleton

Kivételkezelés

Fájlkezelés

Példák

Sorosítás

Serializáció

Deserializáció

Felhasználás

Dr. Szendrei Rudolf
Informatikai Kar
Eötvös Loránd Tudományegyetem



Tartalom

- 1 Osztályszintű elérés**
Osztályszintű attribútumok
Osztályszintű műveletek
Segédosztályok
Singleton
- 2 Kivételkezelés**
- 3 Fájlkezelés**
Példák
- 4 Sorosítás**
Szerializáció
Deszerializáció
Felhasználás



Osztályszintű elérés

Osztályszintű attribútumok

Osztályszintű műveletek

Segédosztályok

Singleton

Kivételkezelés

Fájelkezelés

Példák

Sorosítás

Szerializáció

Deszerializáció

Felhasználás

Osztályszintű elérés

Osztályszintű elérés (statikusság)

- Az osztályban definiált attribútumok és műveletek az osztályból példányosított objektumok sajátjai lesznek, így azok csak az objektumokon keresztül érhetőek el
- Ez alól kivételt képeznek azok az attribútumok és műveletek, amelyeknek osztályszintű elérést adtunk meg, azaz amelyeket statikussá tettünk
- A statikus attribútumok és műveletek függetlenek az illető osztályból példányosított objektumoktól, helyettük csak az osztályhoz tartoznak, az osztályon keresztül érhetőek el



Osztályszintű elérés

Osztályszintű attribútumok

Osztályszintű műveletek
Segédosztályok
Singleton

Kivételkezelés

Fájlkezelés

Példák

Sorosítás

Serializáció
Deszerializáció
Felhasználás

Osztályszintű elérés

Osztályszintű attribútumok

- Az osztályszintű attribútumok nem az osztályból példányosított objektumokhoz, hanem magához az osztályhoz tartozik, így akkor is léteznek, ha egyetlen objektumot sem hoztunk létre az osztályból
- Mivel egyetlen osztályhoz tartoznak, így csak egyetlen példányuk létezik, vagyis az összes objektum ugyanezt az egy értéket használja közösen
- Szokásos használatuk:
 - Konstans értékek
 - Jelzőértékek (flag-ek)
 - Az adott osztályba tartozó objektumok közösen használható értékei



Osztályszintű elérés

Osztályszintű műveletek

- Hasonlóan az osztályszintű attribútumokhoz, az osztályhoz tartoznak, és az osztály példányosítása nélkül is használhatók
- Meghívhatók csupán az osztályra hivatkozva is (objektum létrehozása nélkül)
- Fontos megkötés, hogy az osztályszintű műveletek kizárólag osztályszintű, azaz statikus attribútumokat és műveleteket használhatnak
- Szokásos használatuk:
 - Segédfüggvények
 - Matematikai függvények



Osztályszintű elérés

Segédosztályok

A segédosztályok (**utility class**) kizárólag statikus attribútumokat és műveleteket tartalmaznak, általában nem szokás objektumokat példányosítani belőlük.

```
public final class Math{
    private Math() {...}

    public static final double E = 2.7182818284590452354;
    public static final double PI = 3.14159265358979323846;

    public static double sin(double a) {...}
    public static double cos(double a) {...}

    public static double toRadians(double a) {...}
    public static double toDegrees(double a) {...}

    public static double sqrt(double a) {...}

    public static int abs(int a) {...}
}
```



Osztályszintű elérés

Példa: Singleton osztály

- Néha szükséges, hogy egy osztályból csak egyetlen példányt hozhassunk létre
- Az ilyen osztályt singletonnak nevezzük
- Az osztályt a konstruktora helyett egy speciális `getInstance()` művelettel kell példányosítanunk

```
public class MySingleton{
    private static MySingleton instance;
    private MySingleton(){...}
    public static MySingleton getInstance()
    {
        if (instance == null)
            instance = new MySingleton();
        return instance;
    }
}
```

- Példányosítás:

```
MySingleton s1 = new MySingleton();           // hiba!
MySingleton s2 = MySingleton.getInstance();    // rendben!
MySingleton s3 = MySingleton.getInstance();    // s3 == s2
```



Kivételkezelés

Kivétel (exception)

Olyan váratlan hiba (kivételes helyzet), melynek oka nem a futó programban keresendő, hanem külső körülményre vezethető vissza

Kivételkezelés (exception handling)

Egy esetlegesen fellépő kivétel futás közbeni megoldása - akár hibajelzés formájában, akár a hiba kijavítása formájában

Példa

- Jelszó ellenőrzés belépéskor
 - Kivétel: a felhasználó rossz jelszót adott meg
 - Kivételkezelés: a program jelzi a hibát és újra kéri a jelszót
- A felhasználó menteni akarja a munkáját
 - Kivétel: a megadott fájl már létezik
 - Kivételkezelés: a program rákérdez, hogy felülírja-e a fájlt



Kivételkezelés

Kivételkezelés Java-ban

- Java-ban a kivételkezelés kivétel (`Exception`, `RuntimeException`) típusú objektumok használatával valósul meg
- Ha a program futása közben kivétel lép fel, de a program az adott helyen nem képes kezelni a kivételt, akkor eldob egy kivétel típusú objektumot
- A kivétel objektum a függvényhívásokon keresztül (hívási lánc) visszafelé vándorol, egészen addig, amíg egy kivételkezelő blokk el nem kapja és nem kezeli
- Ha a kivétel kijut egészen a virtuális gépig, akkor a program futása leáll
- Minden `Exception` típusú kivételt muszáj vagy kezelni, vagy jelezni a továbbadását
- A `RuntimeException` típusú kivételeket nem kötelező sem elkapni, sem kezelni, de egy jól működő program ezeket is kezeli



Osztályszintű elérés

Osztályszintű attribútumok

Osztályszintű műveletek

Segédosztályok

Singleton

Kivételkezelés

Fájelkezelés

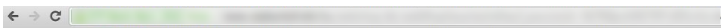
Példák

Sorosítás

Serializáció

Deszerializáció

Felhasználás



Error 500–Internal Server Error

```

javax.servlet.ServletException: java.lang.NullPointerException
    at hu.igsoft.homebank.action.BaseAction.execute(BaseAction.java:157)
    at hu.igsoft.homebank.action.webshop.WebShopVasarlasInditas.execute(WebShopVasarlasInditas.java:314)
    at org.apache.struts.chain.commands.servlet.ExecuteAction.execute(ExecuteAction.java:53)
    at org.apache.struts.chain.commands.AbstractExecuteAction.execute(AbstractExecuteAction.java:64)
    at org.apache.struts.chain.commands.ActionCommandBase.execute(ActionCommandBase.java:48)
    at org.apache.commons.chain.impl.ChainBase.execute(ChainBase.java:190)
    at org.apache.commons.chain.generic.LookupCommand.execute(LookupCommand.java:304)
    at org.apache.struts.chain.impl.ChainBase.execute(ChainBase.java:190)
    at org.apache.struts.chain.ComposableRequestProcessor.process(ComposableRequestProcessor.java:280)
    at org.apache.struts.action.ActionServlet.process(ActionServlet.java:1858)
    at org.apache.struts.action.ActionServlet.doGet(ActionServlet.java:446)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:707)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:820)
    at weblogic.servlet.internal.StubSecurityHelper$ServletServiceAction.run(StubSecurityHelper.java:226)
    at weblogic.servlet.internal.StubSecurityHelper.invokeServlet(StubSecurityHelper.java:124)
    at weblogic.servlet.internal.ServletStubImpl.execute(ServletStubImpl.java:283)
    at weblogic.servlet.internal.TailFilter.doFilter(TailFilter.java:26)
    at weblogic.servlet.internal.FilterChainImpl.doFilter(FilterChainImpl.java:42)
    at hu.igsoft.homebank.servlet.SessionReplicatorFilter.doFilter(SessionReplicatorFilter.java:53)
    at weblogic.servlet.internal.FilterChainImpl.doFilter(FilterChainImpl.java:42)
    at hu.igsoft.commonclient.security.SecurityFilter.doFilter(SecurityFilter.java:115)
    at hu.igsoft.commonclient.util.BaseFilter.doFilter(BaseFilter.java:42)
    at weblogic.servlet.internal.FilterChainImpl.doFilter(FilterChainImpl.java:42)
    at hu.igsoft.homebank.servlet.LocaleFilter.doFilter(LocaleFilter.java:90)
    at weblogic.servlet.internal.FilterChainImpl.doFilter(FilterChainImpl.java:42)
    at weblogic.servlet.internal.WebAppServletContext$ServletInvocationAction.run(WebAppServletContext.java:3402)
    at weblogic.security.acl.internal.AuthenticatedSubject.doAs(AuthenticatedSubject.java:321)
    at weblogic.servlet.internal.WebAppServletContext.securedExecute(WebAppServletContext.java:2140)
    at weblogic.servlet.internal.WebAppServletContext.execute(WebAppServletContext.java:2046)
    at weblogic.servlet.internal.ServletRequestImpl.run(ServletRequestImpl.java:1398)
    at weblogic.work.ExecuteThread.execute(ExecuteThread.java:200)
    at weblogic.work.ExecuteThread.run(ExecuteThread.java:172)
Caused by: java.lang.NullPointerException
    
```



Kivételkezelés

Saját kivételtípus definiálása

```
class EtrException extends Exception{  
    public EtrException(String message) { super(message); }  
}
```

Kivétel eldobása

```
throw new EtrException("Namespace_'"+nameFract+"'_is_full");
```

Kivétel továbbengedésének jelzése

```
private String generateEtr(EtrPerson aThis) throw EtrException{...}
```

Kivétel kezelése

```
try {  
    etrPersons.add(new EtrPerson("Teszt", "Elek", "IK"));  
} catch (EtrException etrException) {  
    System.err.println(etrException.getMessage());  
}
```



Osztályszintű elérés

Osztályszintű attribútumok

Osztályszintű műveletek

Segédosztályok

Singleton

Kivételkezelés

Fájlkezelés

Példák

Sorosítás

Serializáció

Deserializáció

Felhasználás

Fájlkezelés

Fájlkezelés

- Emlékeztető:
 - Mit jelent a fájlkezelés?
 - Mikor és mire használjuk?
- Milyen Java osztályok segítségével és hogyan működik?
- Fájlkezelés asztrakciója:
 - `java.io.InputStream`: bájt-csatornák olvasásához
 - `java.io.OutputStream`: bájt-csatornák írásához
 - `java.io.Reader`: karaktercsatornák olvasásához
 - `java.io.Writer`: karaktercsatornák írásához



Fájlkezelés

Fájlkezelés

- Az előbbi absztrakt osztályokból származnak a konkrét csatornákhöz használatos osztályok, így pl. fájlkezelésre:
 - `java.io.FileInputStream`: bájtok olvasása fájlból
 - `java.io.FileOutputStream`: bájtok írása fájlba
 - `java.io.FileReader`: karakterek olvasása fájlból
 - `java.io.FileWriter`: karakterek írása fájlba
- Ezek mindössze a megnyitott fájl bájtjainak/karaktereknek egyenkénti olvasására/írására használhatóak.
- A fentieket érdemes összetettebb adatkezelésre alkalmas objektumokkal kombinálni.
- Sose feledkezzünk meg a kivételkezelésről, hiszen bármikor előfordulhat, hogy egy fájlt nem tudunk megnyitni, nincs jogunk az írására vagy nem az elvárt tartalommal rendelkezik



Osztályszintű elérés
Osztályszintű attribútumok
Osztályszintű műveletek
Segédosztályok
Singleton

Kivételkezelés

Fájlkezelés

Példák

Sorosítás

Serializáció
Deserializáció
Felhasználás

Fájlkezelés

Példa: számok bináris kiírása fájlba

```
DataOutputStream dos =  
    new DataOutputStream(  
        new FileOutputStream(  
            "num.dat"));  
dos.writeInt(20);  
dos.writeInt(31);  
dos.close();
```

Példa: szöveg kiírása fájlba

```
List<String> textLines = someUtility.getTextLines();  
PrintWriter pw = new PrintWriter(new FileWriter("text.txt"));  
for (String line : textLines) {  
    pw.println(line);  
}
```



Fájlkezelés

Példa: zip fájl beolvasása

```
ZipInputStream zis =  
    new ZipInputStream(  
        new FileInputStream(  
            "someFile.zip"));  
ZipEntry entry = zis.getNextEntry();  
while (entry != null) {  
    System.out.println(entry.getName());  
    doSomethingWithInput(zis);  
    entry = zis.getNextEntry();  
}  
zis.close();
```

Példa: tokenizált adatok beolvasása fájlból

```
Scanner sc = new Scanner(new FileReader("raceResults.txt"));  
List<RaceResult> raceResults = new ArrayList<>();  
while (sc.hasNext()) {  
    RaceResult rr = new RaceResult();  
    rr.setName(sc.next());  
    if (!sc.hasNextInt()) break;  
    rr.setTime(sc.nextInt());  
    if (!sc.hasNextBoolean()) break;  
    rr.setValid(sc.nextBoolean());  
    raceResults.add(rr);  
}
```



Sorosítás

Sorosítás - szerializáció

- Lehetőség van teljes Java objektumok fájlba való kiírására
- Ehhez a virtuális gép sorosítja az adott objektumot, ezért az objektum osztályának meg kell valósítania a `Serializable` interfészt. Ez rekurzívan érvényes az objektum összes adattagjára is

- Szerializálható objektum:

```
public class EtrPerson implements Serializable { ... }
```

- Az objektum szerializálása:

```
ObjectOutputStream oos = new ObjectOutputStream(  
    new FileOutputStream(  
        "etrPersons.obj"));  
oos.writeObject(etrPersons);  
oos.close();
```




Sorosítás

Sorosított adat visszafejtése - deszerializáció

- A sorosított adatok természetesen a fájlokból beolvashatóak, visszafejthetőek objektumokká
- Ehhez teljesülni kell a következő feltételeknek:
 - a beolvasott objektum tényleg olyan típusú legyen, mint amire számítunk
 - legyen betöltve a beolvasott objektum osztálya a virtuális gépbe - pontosan az az osztály legyen az

```
ObjectInputStream ois = new ObjectInputStream(  
    new FileInputStream(  
        "etrPersons.obj"));  
  
try{  
    etrPersons = (List<EtrPerson>) ois.readObject();  
} catch (ClassNotFoundException ex) {  
    System.err.println("Class_not_found:" + ex.getMessage());  
} catch (ClassCastException ex) {  
    System.err.println("Not_a_List<EtrPerson>");  
}  
ois.close();
```



Sorosítás

Felhasználás

- A sorosítás és az adatok mentése eltérő jelentéssel bír
 - A sorosítás az objektum állapotát menti, a sorosított adat visszatöltéskor „ugyanazt” az objektumot kapjuk vissza, ugyanabban az állapotban
 - Az adatok mentése nem feltétlenül jelenti az állapotok mentését is (pl. egy focistának elmentjük a nevét, de azt nem, hogy ő az éppen soron következő, aki tizenegyest fog lőni)
- A sorosítás segítségével pillanatképet készíthetünk a program objektumainak aktuális állapotáról és a programot újraindíthatjuk máskor ugyanabból az állapotból (feltéve, ha...)
- A sorosítással csak az objektumokat tudjuk menteni, a rájuk mutató referenciákat nem
- A sorosítás alapvetően szükséges, ha távoli metódushívásokkal akarunk dolgozni