Java programozási nyelv 2007-2008/ősz
3. óra

Java osztálykönyvtár és használata

Csomagok

Kivételkezelés

Osztálykönyvtárak és használatuk

legradi.gabor@nik.bmf.hu szenasi.sandor@nik.bmf.hu

Java osztálykönyvtár és használata

Témakörök

Csomagok

Kivételkezelés

Osztálykönyvtárak használata

Csomagok definiálása

- Csomagok célja: a szoftvertervezés és a szoftverfejlesztés támogatása, a program megfelelő tagolásának elérése
- Csomagtípusok
  - Névtelen csomag: a fejlesztés induló fázisában, még nem importálható más csomagokból (csak szintén névtelenből látható és érhető el)
  - Névvel ellátott (nevesített) csomag: a már letisztult kód tárolására, valamint hierarchia kialakítására, és az egyértelműség fenntartására (a Pauli-féle tilalmi elv itt is érvényes!)
- A névvel ellátott csomag elején a `package` direktíva után adjuk meg a csomagunk nevét (a hierarchiának megfelően), névtelen csomagnál ez a direktíva hiányzik

Csomagok importálása

- Fordítási egység: olyan önálló kódrészlet, amely önmagában is lefordítható
- A csomagdefiníció fordítási egységek logikai, de nem feltétlenül fizikai halmaza, azaz egy csomagba több különböző fájlban elhelykedő, de azonos csomagba deklarált programegység tartozik
- Egy fordítási egység csak egy csomaghoz tartozhat
- A csomagot felhasználó programegységben szerepelnie kell az `import` direktívának, amely argumentumaként az importálandó csomag nevét tartalmazza
- Például:
  - Konrét osztály importálása: `import java.awt.Frame;`
  - Igény szerinti importálás: `import java.awt.event.*;`
- Megjegyzés: az importálás nem rekurzív folyamat

Csomagok tervezése

- A programban felhasznált csomagokat a kód elején, a csomagdeklarációs direktíva után kell felsorolni, hatása csak az adott programegységre vonatkozik
- A csomagnevek hierarchikus kialakítása csak laza logikai kapcsolatot jelent
- A csomagnevek kiválasztásánál ügyeljünk arra, hogy a csomagnév átgondolt legyen
- A Java-rendszer tartalmaz „gyári” csomagkészletet (Java API), amelyet a programjainkban felhasználhatunk, és újabb csomagokkal is bővíthetünk

Csomagok tárolási lehetőségei

- Alkönyvtárba kódolt megoldás

  Ekkor a csomagnév egy kiindulási ponttól kezdve az adott alkönyvtárstruktúrával azonos, a futtató rendszer számára kiindulási helye(ke)t a `CLASSPATH` rendszerváltozó tartalmazza

  Lehetséges ennek az alkönyvtár struktúrának a tömörített módú tárolása is (pl. `ZIP` állományban vagy Java-archívumban, azaz `JAR` fájlban)

  Tehát a `package sajat;` direktívával lefordított állományt a futtató rendszer az alábbi helyeken keresi:
  - a `sajat` alkönyvtárban
  - a CLASSPATH által definiált könyvtárak `sajat` alkönyvtárban
  - a CLASSPATH által definiált zip/jar fájlok `sajat` alkönyvtárban
- Adatbázisba kódolt megoldás

  Ekkor a csomagok kezelését, tárolását és visszakeresését egy adatbáziskezelő rendszer végzi el

Feladat

3.1. feladat: Készítsünk saját csomagokat, s írjunk meg az őket felhasználó programot is! A csomagokban szereplő osztályok között legyen névütközés!

3.2. feladat: Próbáljuk meg a programot egy másik helyre átmásolva futtatni, értelemszerűen felépítve a csomag hierarchiát többféleképpen (alkönyvtárak, tömörített fájlok)!

Java osztálykönyvtár és használata

Témakörök

Csomagok

Kivételkezelés

Osztálykönyvtárak használata

Kivételkezelés folyamata

- Célja: megbízható programtermékek létrehozásának támogatása, a futásidejű hibák felismerésének, feltárásának és kijavításának biztosítása
- Futásidejű hiba: a program futása során, a forráskód alapján tervezhető és elvárható működéstől való eltérés
- Futásidejű hibák felismerését a JVM végzi
- Az ilyen módon felmerülő problémák megoldási lehetőségei
  - a program „normál üzemű” működésének helyreállítása
  - amennyiben ez sem lehetséges, úgy a program „minimális vérveszteséggel” történő leállásának biztosítása
- A kivételkezelés előnyei
  - A hibakezelő kód jól elkülönül a tényleges kódtól
  - A hiba könnyen eljut oda, ahol kezelni kell
  - A hibatípusok csoportosíthatók

Kivételkezelés folyamata

- Kivétel keletkezésének módjai
  - A program futása során rendellenes dolog történik (ez a program tetszőleges pontján előfordulhat)
  - A program egy `throw` utasítást hajt végre (kivételt dob)
  - Aszinkron hiba lép fel (többszálú alkalmazás esetén az egyik programszál futása megszakad)
- A hibát okozó metódus felfüggesztődik, és a JVM a megfelelő kivételkezelőt megkeresi (kivétel kezelő kódblokk)
- A kivételkezelő kódblokk megfelelő, ha:
  - a kezelő által kezelt kivétel megegyezik a kiváltott kivétellel
  - vagy őse annak az osztályhierarchiában
- A kivételkezelők egymásba ágyazhatók (kivételkezelés alatt is keletkezhet kivétel)

Kivételkezelés folyamata

- Kivétel elkapása: ha a JVM megtalálta a kivételkezelő kódblokkot, a kódblokk megkapja a kivételobjektumot feldolgozásra
- A kódblokk:
  - felderítheti a kivétel okát
  - kezelheti a problémát
  - továbbíthatja a kivételt (akár más típusú kivételt is kelthet)
- A kivétel kezelése után a vezérlés a kivételkezelő kódblokkot követő utasításnál folytatódhat (és nem a kivételt kiváltó utasítások után)
- A Java kivételkezelés tulajdonságai
  - Objektum-orientált
  - Bővíthető

Kivételek típusai

- **Ellenőrzött kivételek**

  Ha egy metódus a fejlécében jelzi, hogy elképzelhető ilyen kivétel dobása, akkor az összes hívásánál kötelező a programot felkészíteni a kivétel kezelésére (vagy a hívó metódusnak is jelölnie kell a kivétel továbbítását)

  A *java.lang.Exception* osztályból származnak
- **Nem ellenőrzött kivételek**

  Az előzővel ellentétben ezeket a kivételeket nem kötelező elkapni (hasonlóan a C# kivételekhez)

  Két ősből is származhatnak:
  - *java.lang.Error*
    Kritikus hibák, pl. nincs elég memória (a programon belül bárhol előfordulhatnak, és igazából ellenszer sincs ellenük)
  - *java.lang.RuntimeException*
    Szintén gyakran előforduló kivételek, amelyeket tervezési okokból nem kötelező kezelni, mert olvashatatlanná tennék a programkódot

Kivételkezelés Java nyelven

- A kivételkezelés módjai:
  - A `try-catch` blokk alkalmazása,
  - A `try-catch` blokk alkalmazása `finally` kiegészítéssel,
  - A kivételkezelés „elhalasztása” a kivétel specifikálásával
- A kivételkezelő blokk alkalmazása:

```
try {
  // „gyanús” utasítássorok
}
catch(kivételOsztály kivételReferencia) {
  // ide kerül a kivételt kezelő kódrész
}
finally {
  // ide kerülő kódrészlet mindig lefut
}
```

Megjegyzések

- Egy `try` blokkhoz több, egymás után következő `catch` ág is tartozhat (ha több különböző lehetséges kivétel is keletkezhet a `try` blokkon belül)
- Több `catch` blokk esetén az első, a megadott kivételt kezelni képes blokk fut le
- A kivételosztályok között van lehetőség az *Exception* osztály alkalmazására is, mivel ez minden kivétel őse, ezért ez bármelyik kivétel esetén lefuthat
- Mivel az *Exception* osztállyal definiált kivételkezelő minden leszármazottját „elnyeli”, így vagy ne alkalmazzuk (csak valamelyik leszármazottját), vagy ha mégis szükséges, akkor a `catch` ágak közül ezt a kivételt kezeljük legutoljára
  Így ez csak akkor fut le, ha előtte nem soroltuk fel egyik speciálisabb leszármazottját sem

Kivételek specifikálása

- A kivétel specifikálása (a kivételkezelés „elhalasztása”):

```
visszaadott_érték metódusnév(paraméterek) throws
keltett_kivételosztályok_felsorolása {
   // metódus kódja
}
```

- A `throws` kulcsszó használatának esetei:
  - A metóduson belül egy olyan kivétel keletkezhet, aminek a kezelését nem tudjuk (akarjuk) megoldani. A kulcsszó segítségével jelezhetjük a hívó metódus számára, hogy a felsorolt kivételek továbbítására számíthat
  - A metóduson belül egy kötelező kivételt dobni képes (amit egy `throws` kulcsszóval jelez számunkra) metódust hívunk, azonban a kivételt nem tudjuk ezen a szinten kezelni. Mivel ez fordítási hibához vezetne, a `throws` kulcsszóval jelöljük, hogy ezt a kivételt majd az ezt meghívónak kell kezelnie (persze ő is specifikálhatja, de végül valakinek kezelnie kell)

Kivételek dobása

- A kivétel eldobása:

```
throw kivételobjektum
throw new kivételosztály(konstr. param.)
```

- Lehetséges okai
  - A program ezen a szinten nem tudja kezelni az előállt hibát, ezért egy kivételdobással ezt a feladatot továbbítja a hívási láncban visszafelé
  - Kivétel kezelése során más típusú kivétel létrehozása és dobása
  - Program tesztelése
- Megjegyzés: a saját, az *Exception* osztályból, a *RuntimeException* osztályból, vagy azok bármely leszármazottából származó kivételosztály alkalmazása esetén is lehet ilyen módon kivételt „dobni”, ennek kezelése teljesen egyezik a korábban említettekkel

Feladat

3.2. feladat: Készítsünk olyan programot, amelyben egész számok osztását mutatjuk be, és a nevező legyen zérus!

3.3.a. feladat: A „nullával osztó” programban kezeljük le a kivételt!

3.3.b. feladat: A „nullával osztó” programban használjuk a finally kiegészítést!

3.4. feladat: Készítsünk saját kivételosztályt, és a fenti példát egészítsük ki azzal, hogy egyel való osztáskor dobjon egy *DivisionByOneException* kivételt!

3.5. feladat: A saját kivételt kezelő programban készítsünk olyan metódust, amelyben a kivételt specifikáljuk!

Java osztálykönyvtár és használata

Témakörök

Csomagok

Kivételkezelés

Osztálykönyvtárak használata

Osztálykönyvtárak használata

- Osztálykönyvtárak: osztályok és típusok összessége, amelyek segítségével összetett feladatokat is meg tudunk oldani
- Java osztálykönyvtárak
  - „gyári” osztálykönyvtár
  - saját osztálykönyvtár
- Használatukhoz a megfelelő csomagot importálni kell (kivétel a *java.lang* csomag)
- Futtatás során a betöltendő lefordított osztályok elérhetőségét biztosítani kell (aktuális könyvtár, CLASSPATH, szerver konfigurációs állományok)
- A „gyári” osztálykönyvtárak esetén mindig figyelembe kell venni, hogy a futtató környezet mit tud biztosítani (kiadás, verziószám stb.)

Alapvető osztálykönyvtárak

- Részletes leírás elérhető: http://java.sun.com/javase/6/docs/api/

Csomag	Tartalma
javax.applet	Appletek
java.awt	Vizuális felület
java.beans	Javabeans létrehozás/kezelés
java.io	Kommunikáció
java.lang	Alapvető nyelvi elemek
java.math	Matematikai osztályok
java.net	Hálózatok
java.rmi	Távoli metódushívás
java.security	Biztonságpolitika
java.sql	Adatbáziskezelés
java.util	Segédosztályok

- További lehetőségek: javax. csomagok

java.lang csomag – csomagoló osztályok

- A *java.lang* csomag tartalmazza az egyes primitív típusokhoz használatos csomagoló osztályokat. Ezekkel:
  - Primitív típus objektumként használható (pl. gyűjteményekben)
  - Típussal kapcsolatos statikus műveletek/konstansok
- Pl. *Integer* osztály
  - public Integer(int value)
  - public Integer(String s) throws NumberFormatException
  - public int intValue()
  - public double doubleValue()
  - public static String toString(int i)
  - public static String toString(int i, int radix)
  - public static int parseInt(String s, int radix) throws NumberFormatException
  - public static final int MIN_VALUE

Konverziós lehetőségek

- Típuskényszerítés (casting)

```
int a; double b;
b = (double)a;
a = (int)b;
```

- Boxing/unboxing

```
int a; Integer b;
b = Integer.valueOf(a);
b = new Integer(a)
a = b.intValue()
```

- String konverziók

```
int a; String b;
b = Integer.toString(a);
b = new Integer(a).toString();
b = "" + a;
a = Integer.parseInt(b);
```

java.lang csomag – Szövegkezelés

- *java.lang.String* osztály
  Nem módosítható Unicode szöveget tárol
  - public String(String original)
  - public String(byte[] bytes, Charset charset)
  - public int length()
  - public boolean isEmpty()
  - public char charAt(int index)
  - public boolean equals(Object anObject)
  - public boolean equalsIgnoreCase(String anotherString)
  - public int compareTo(String anotherString)
  - public int indexOf(String str)
  - public String substring(int beginIndex, int endIndex)
  - public String toUpperCase()
    stb.

java.lang csomag – Szövegkezelés

- *java.lang.StringBuffer* osztály
  Változtatható Unicode szöveget tárol
  - public StringBuffer()
  - public StringBuffer(int capacity)
  - public StringBuffer(String str)
  - public int length()
  - public int capacity()
  - public void ensureCapacity(int minimumCapacity)
  - public char charAt(int index)
  - public StringBuffer append(String str)
  - public StringBuffer replace(int start, int end, String str)
  - public StringBuffer insert(int offset, String str)
  - public String toString()
    stb.

java.util csomag – Szövegkezelés

- *java.lang.StringTokenizer* osztály

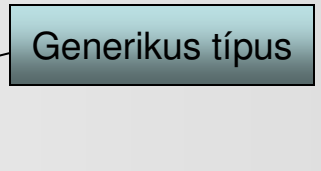
  Szöveget felbontja elválasztójelek mentén
  - public StringTokenizer(String str, String delim)
  - public boolean hasMoreTokens()
  - public String nextToken()
  - public int countTokens()

    stb.
- Példa

```
String s = ”Rövid példa szöveg”;
StringTokenizer st = new StringTokenizer(s, ” ”);
while (st.hasMoreTokens()) {
   System.out.println(st.nextToken());
}
```

java.util csomag – Gyűjtemények

- *java.util.Vector* osztály
  Adatok dinamikus tárolását teszi lehetővé
  - public Vector()
  - public int size()
  - public int capacity()
  - public boolean contains(Object o)
  - public int indexOf(Object o)
  - public E elementAt(int index)
  - public void addElement(E obj)
  - public void insertElementAt(E obj, int index)
  - public void setElementAt(E obj, int index)
  - public void removeElementAt(int index)
  - public boolean removeElement(Object obj)
    stb.

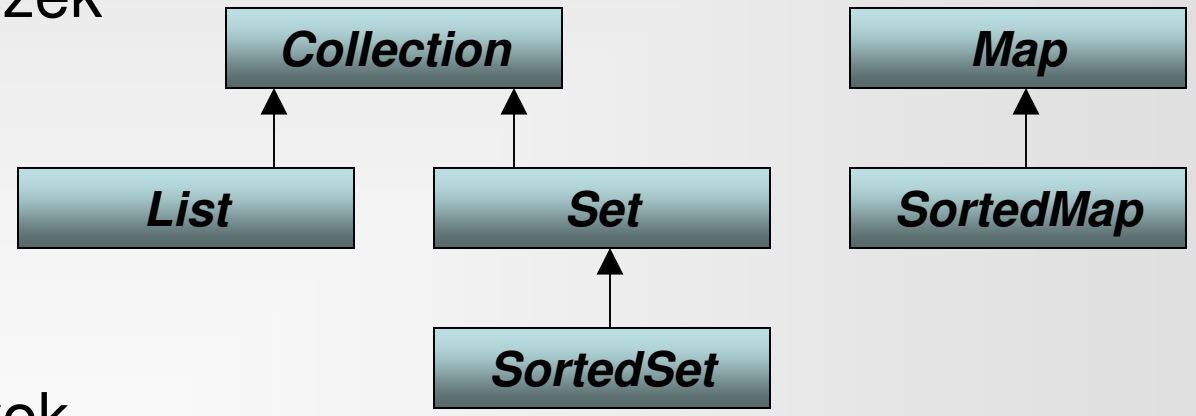


java.util csomag – Gyűjtemények

- *java.util.HashMap* osztály
  Adatok dinamikus tárolását teszi lehetővé kulcs alapján
  - public HashMap()
  - public int size()
  - public V get(Object key)
  - public V put(K key, V value)
  - public boolean containsKey(Object key)
  - public boolean containsValue(Object value)
  - public V remove(Object key)
  - public Set<K> keySet()
  - public Collection<V> values()
  - public void clear()
    stb.

java.util csomag – Gyűjtemények

- A Java osztálykönyvtár a .NET-hez hasonlóan kiterjedt osztálykönyvtárral rendelkezik az adatok tárolásához (Collection Framework)
- Ezen tárgynak nem célja ennek teljes áttekintése, azonban javasolt a hierarchia tanulmányozása
- Interfészek



- Osztályok
  - *LinkedList*, *ArrayList*
  - *HashSet*, *TreeSet*
  - *HashMap*, *TreeMap*

java.math és java.lang.Math

- *java.math* **csomag** tartalmaz néhány matematikai osztályt

  pl. *BigInteger* / *BigDecimal* - Nagypontosságú aritmetika
- *java.lang.Math* **osztály** konstainsai/metódusai
  - public static final double PI
  - public static double sin(double a)
  - public static double sqrt(double a)
  - public static double ceil(double a)
  - public static double floor(double a)
  - public static int round(float a)
  - public static double pow(double a, double b)
  - public static double random()
  - public static int abs(int a)
  - public static int max(int a, int b)

Feladat

3.6. feladat: készítsünk programot, amely a köregyenlet ($r^2 = x^2 + y^2$) alapján kiszámolja (és kijelzi), egy adott 0,0 koordinátájú, egység sugarú kör x,y pontjainak koordinátáit!

3.7. feladat: egészítsük ki úgy a programot, hogy az így nyert pontokat tároljuk el egy *Vector* objektumban!

3.8. feladat: legyen lehetőség megadott x és y koordináták alapján szűrni és a képernyőn jelenítsük meg a meghatározott téglalapba eső pontokat!

Házi feladat

- Készítsen egy *Berelszamolo* csomagot, benne egy *Dolgozo* osztályt:
  - név (pl. „Nagy Béla”)
  - előző munkahelyei (pl. „BKV;FTC;BMF;FKF”)
  - fizetés (pl. 67000)
  - getKifizetendo() – visszaadja a fizetést levonva az adókat (50%-ot)
- Ugyanebben a névtérben készítse el a *Dolgozo* osztály *GyerekesDolgozo* leszármazottját, ami kiegészíti a fenti osztály egy családipótlék (pl. 30000) mezővel. Módosítsa a getFizetendo() metódust, hogy ezt is vegye figyelembe
- Készítsen menüvezérelt programot, ami megvalósítja az alábbi funkciókat:
  - Billentyűzetről megadott adatokkal feltölt egy vektort *Dolgozo* vagy *GyerekesDolgozo* objektumokkal
  - Listázza azokat, akiknek a kifizetendő összeg nagyobb mint az átlagos
  - Törölje a listából azokat, akik egy megadott helyen dolgoztak már előtte
- Készítsen egy *FeketeMunkas* osztályt, ami csak egy nevet tartalmaz, és mivel nincs fix fizetése, ezért a fizetéseknek egy listáját. Egy metódussal legyen lehetőség ebbe felvenni egy új kifizetést
- Készítsen egy *IKirughato* interfészt, aminek van egy vegKielegites() metódusa. Mindhárom osztály valósítsa meg ezt az interfészt, dolgozóknál adja vissza a fizetés háromszorosát, feketemunkásnál pedig az átlagos kifizetés négyzetgyöke * Π értéket lefelé kerekítve
- Egy tömbbe helyezzen néhány kirúgható embert és számolja ki a költségeket. Ha valakinél ez több mint 1M, dobjon *NemRughatoKi* kivételt!

Ajánlott irodalom

Az óra anyagához kapcsolódó irodalom

- Nyékyné Gaizler Judit: Java 2 útikalauz programozóknak 1.3 I.; ELTE TTK Hallgatói alapítvány, Budapest
  97 – 105. o.
- The Java™ API Reference:
  http://java.sun.com/javase/6/docs/api/
- Java Collection Framework:
  http://java.sun.com/docs/books/tutorial/collections/index.html