Osztály, interfész, objektum

OO elvek és elemek
a Java nyelvben

TRAINING360
GET FLOW NOT SKILLS

Tematika

- OO elvek és megvalósításuk
- OO alapelemek
- OO fogalmak
- OO elemek és használatuk

Objektum orientált gondolkodás

- emberi gondolkodás kiterjesztése
  - rendszerezés (Linné és rendszere)
  - absztrakció (közös tulajdonságok felismerése)
- példák az objektum orientált gondolkodásra
  - természettudományos rendszertan
  - tárgyak a környezetünkben
  - alakfelismerés

Strukturális és OO programozás

- Strukturális:
  - algoritmus kiválasztása és globális adatszerkezet azonosítása
  - feladatok finomítása vagy felépítése
  - OK kisebb feladatokra!
- OO alapú:
  - adatok azonosítása és algoritmusok adatokhoz rendelése
  - független és újrahasznosítható blokkok
  - egy közepes alkalmazásban 2000 eljárás vs. 100 osztály egyenként 20 metódussal

OO előtérbe kerülése

- folyamatos szoftverkrízis
- modellezés, tervezés
- nagy rendszerek problémája
- absztrakció igénye
- adatok és adatközpontú alkalmazások
- újrahasznosítás kérdése
  - függvénygyűjtemények és azok rendszerezése
  - programfájlok
  - unitok ("egységek")

OO alapú tervezés

- UML és UP
- meghatározandók az alkalmazás objektumai
- meghatározandók a metódusok
- objektumos világ vs. relációs világ
- osztályok sztereotípiája feladatuk szerint
  - kontroller
  - entitás
  - interfész

OOP alapelvek I

- A-PIE
  - Abstraction
    Csak a legszükségesebb részlet látszódik kifelé. (láthatóság)
  - Polymorphism
    Az objektum többféle formában megjelenhet (method overriding, method overloading)

OOP alapelvek II

- Inheritance
  Új osztály írása egy már meglévő kiterjesztésével.
- Encapsulation
  Az adatok és a rajtuk dolgozó metódusok egységbe zárása, az adatrejtés elvével.

OOP alapelvek III

- S.O.L.I.D
  - Single responsibility principle
    Egy osztálynak csak egy feladata lehet.
  - Open-closed principle
    Nyitott a kiterjesztésre, de zárt a módosításra.
  - Liskov substitution principle
    Helyettesíthetőség elve (leszármazott átveheti az ős szerepét).

OOP alapelvek IV

- Interface segregation principle
  Egy osztály sose kényszerüljön olyan interfész vagy metódus implementálására, amire nincs szüksége.
- Dependency Inversion Principle
  Az osztályoknak az absztrakciótól és nem a megvalósítástól kell függniük. (loose coupling) (interfészre programozunk)

OO elemek Java-ban

- OO elemek
  - osztály
  - objektum
  - interfész
  - speciális osztályok
  - absztrakt osztály
  - final osztály
  - generikus osztály
  - JavaBean

Osztály

- típus és egyedleírás
- absztrakció eredménye
- legkisebb fordítási egység
- csomagokba szervezhetőek
- osztályváltozók és metódusok építik fel
- osztályok közötti viszonyok
  - függőség
  - aggregáció
  - öröklődés

Osztály deklarálása

- osztály felépítése
- egyszeres öröklődés, fa hierarchia, Object osztály
- osztálymódosítók: abstract, final, public
- törzsben változó és metódus deklarációk
- beépített osztályok és saját osztályok

Osztályváltozók, attribútumok, mezők

- [*módosító*] *típus név* [=*kezdőérték*] [,*név* [=*kezdőérték*]]... ;
- kezdőérték kifejezés, implicit kezdőérték
  - kezdőérték többféleképpen megadható
- módosítók:
  - final, static (ritka)

Osztály szintű (statikus) tagok

- osztályváltozó
  - egyszer tárolódik
  - egyszer inicializálódik
  - minősítés nélkül és minősítéssel hivatkozható
- osztálymetódus
  - osztályváltozókhoz férhet hozzá, példányváltozókhoz és metódusokhoz nem
  - minősítés nélkül és minősítéssel hívható

Metódusok I.

- [*módosító*] *fej törzs*
- módosítók a változók metódusain kívül:
  - final
  - abstract
  - static
- paraméterek kiértékelése
  - sorrendi kötés
  - számbeli egyeztetés
  - típusegyeztetés

Metódusok II.

- explicit és implicit paraméter értelmezése
- visszatérési érték
  - utolsó sor végrehajtása után void
  - return utasítás
- minősítés
- metódusok túlterhelése (szignatúra)
- a this kulcsszó szerepe

Metódusok III.

- csak érték szerinti paraméterátadás
- primitív típusú paraméter
- objektum típusú paraméter
- érték szerinti paraméter átadás bizonyítása
  - objektumok cseréje a metódusban

Felülírás

- örökli a metódust a szülőtől
- példánymetódus működésének megváltoztatására
- az eredeti metódusok super minősítővel
- override az angol szakirodalomban
- öröklődésnél:
  - megegyező szignatúrájú metódust definiál felül
  - fordítási idejű döntés

Túlterhelés

- azonos nevű metódus
- azonos funkció különböző paraméterekkel
- azonos funkció többféle visszatérési értékkel
- gyakori konstruktoroknál
- gyakori általános funkcióknál

Konstruktorok

- neve megegyezik az osztály nevével
- nincs visszatérési érték megadva
- paraméterekkel vagy anélkül
- túlterhelhető
- a this és super kulcsszó
- implicit konstruktor

Inicializátor

- osztálydefinícióban a blokkok között
- előfordulás sorrendjében
- konstruktor lefutása előtt
- csak létrejött változók hivatkozhatók
- példány szintű vagy osztály szintű

Destruktor

- Garbage Collector
- objektum megsemmisítése előtt fut le a finalize metódus
- finalize, classFinalize metódus – nem megbízható
- memórián kívüli erőforrások mind külön kezelendők!

Öröklődés I.

- hajtóerő: az újrafelhasználhatóság
- kulcsszó: extends - kiterjesztés
- aszimmetrikus viszony az osztályok között
- szülő- és gyerekosztályok
- a private tagokból van példánya, de nem férhet hozzá

Öröklődés II.

- két nézőpont: specializáció és generalizáció
- láthatóság és annak érvényesülése az öröklődés során
- egyszeres vs. többszörös öröklődés - a Java csak egyszereset enged
- szülő osztály minden tagjával rendelkezik, de csak azt láthatja, melyre a szülő engedélyt ad

Osztályhierarchia

- osztályok rokonsági kapcsolatai
- egyszeres öröklődés
- faszerkezet
- gyökér osztály (ősosztály): java.lang.Object

Mikor használjunk öröklődést?

- közös mezők és metódusok
- ne használjunk protected mezőket
- öröklődés == olyan-mint reláció
- ha az örökölt metódusoknak van értelme
- az elvárt viselkedést nem szabad megváltoztatni
- használjunk polimorfizmust

Konstruktorok és az öröklődés viszonya

- konstruktorok nem öröklődnek
- super hívás
- implicit super() hívás
- a super() hívás az inicializáló blokkok előtt hívódik meg
- ha csak paraméteres konstruktor van, az implicit hívás nem megy

Polimorfizmus

- olyan-mint szabály:
  - öröklődés reális, ha az osztály olyan mint egy már meglévő osztály
- helyettesítési elv:
  - ahol a szülő osztály használható, a gyermek osztály is használható
- automatikus/explicit konverzió
- késői kötés

Osztálymetódusok és változók elfedése

- statikus kötés
- elérhető:
  - minősített névvel
  - a super minősítővel
  - típuskényszerítéssel

Láthatósági szintek

- private
- jelöletlen (package vagy félnyilvános)
  - a default erősen kérdéses!
- protected
- public

Protected hozzáférési kategória

- a félnyilvános kategória kiterjesztése
- hozzáférhető az azonos csomagban szereplő osztályok számára
- ezen kívül hozzáférhető még a leszármazott osztályok számára is, még akkor is, ha azok másik csomagban vannak

Objektum (példány) I.

- az objektumnak van
  - viselkedése
  - állapota
  - azonossága
- adott osztály egy példánya
- tárrész foglalás

Objektum (példány) II.

- aktuális példány létrehozása
- mutator és accessor metódusok
- élettartama van
  - implicit - amíg referencia van rá
  - explicit - meghatározható

Objektumok létrehozása

- példányosítás
- változódeklaráció, értékadás
- minden objektumra közös tagok
- referencia típusú változó
- metódushívás minősítéssel
- nincs változóra hivatkozás kivülről! – (egységbezárás - amit lehet, el kell dugni)

Enum

- Felsorolás típus objektum orientált megvalósítása.
- Metódusai:
  - valueOf()
  - name()
  - ordinal()
- Lehetnek attribútumai, konstruktora (privát) és metódusai
- Nincs öröklődés, de van interfész implementáció!

Interfész I

- nem a határfelület értelemben vett interfész!
- felületet definiál – lehetőségeket ad meg
- osztályként viselkedik, mint típus, de csak metódus fejeket tartalmaz
  - interfész típusú változó létrehozható
- tetszés szerinti számban implementálható
- kötelező viselkedést biztosít az osztálynak

Interfész II

- magasabb absztrakciós szint
- konstansok és absztrakt metódusok halmaza
- önmagában nem értelmezhető, egy más használati módot ad meg
- marker interfész szerepe
- polimorfizmus
- UML:
  - interfész öröklődés: nyíl
  - interfész implementálás: szaggatott nyíl

Interfész deklarációja

```
[public] [abstract] interface InterfaceNev {
   // Konstansok
   // Metódusok
}
```

- név: -ható, -hető (-able)

```
public interface Comparable {
   public int compareTo(Object o);
}
```

- rendezés szempontjából fontos tulajdonság kiemelése: összehasonlíthatóság

Interfész deklarációja - változók

- konstansok
- módosítók: public static final
- inicializáció kötelező
- elfedés
- névütközés problémája

Interfész deklarációja - metódusok

- definíció adott, implementáció nem
- módosítók: alapértelmezetten public és abstract
- névütközés
  - felülírás: szignatúra egyezés
  - túlterhelés: csak név egyezés

Interfész implementációja

- kulcsszó: implements
- tetszőleges számú interfészt implementálhat
- minden absztrakt metódust implementálnia kell
- többszörös öröklődés feloldása

Interfész használata

- típusként használható
- az instanceof operátor szerepe
- nem hierarchikus osztályok közös tulajdonságainak kiemelése
- alkalmazás igényeinek megfelelő kötelező elemek megadása

Interfészek hierarchiája

- öröklődés – kiterjesztés
- kulcsszó: extends
- többszörös öröklődés: vesszővel elválasztva
- nincs gyökér eleme
- specifikáció többszörösen örököltethető, az implementáció csak egyszeresen

Beágyazott osztályok

- vonatkozásai
- tagosztályok
- lokális osztályok
- névtelen osztályok

Beágyazott osztályok vonatkozásai

- statikus vonatkozás
  - további belső tagolás
  - hozzáférési szabályok
  - hatáskör
- dinamikus vonatkozás
  - rögzíti a környezetének példányait és változók tartalmát
  - azok élettartamán túl, a blokk futása után is megőrzi azokat
- külön class fájlba fordulnak le!

Tagosztályok

- hozzáférnek egymás privát tagjaihoz
- befoglaló osztály nevével minősítjük
- statikus
  - csak statikus vonatkozás
  - használati esetek
- nem statikus
  - dinamikus vonatkozás is
  - két aktuális példány, osztálynév.this

Lokális osztályok

- utasításblokkban definiált
- csak olyan változókra hivatkozhatunk, melyek final módosítóval vannak deklarálva
- példányhoz a környezete másolódik: zárvány/bezárás

Névtelen osztály

- alkalmazási köre hasonló a lokális osztályéhoz
- csak egyetlen helyen példányosítjuk
- osztály kiterjesztésekor
- interfész implementálásakor
- Swing eseménykezelés

- azonos nevű, eltérő fogalom: névtelen tömb, objektum
  – értékadáskor, egyszeri használatkor

Végleges (final) osztályok

- letiltjuk a metódus vagy osztály működésének megváltoztathatóságát
- kulcsszava: final
- metódus: nem definiálható felül
- nem származtatható belőle gyermek osztály

Absztrakt osztály

- kulcsszó: abstract
- tartalmazhat absztrakt metódust
- definíció adott, implementáció nem
- leszármazott implementál
- leszármazott nem implementál: absztrakt
- nem példányosítható
- ha interfészt implementál, de nem minden metódusát: absztrakt

Generikusok (Java megoldás)

- paraméterezett osztályok és problémák
  - template
  - generic - J2SE5 - <típus megadása obj tárolásnál>
  - autoboxing - dinamikus váltás a primitív és az object típus között

Generikus osztály

```
public class Box<T> {
      private T content;

      public T lookInto() {
            return contents;
      }

      public void pack(T content) {
            this.content = content;
      }
}
```


Generikus osztály használata

```
Box<Zebra> boxForZebra = new Box<>();
boxForZebra.pack(new Zebra());
Zebra zebra = boxForZebra.lookInto();

Box<Treasure> boxForTreasure = new Box<>();
```

diamond operator

Raw type és heap pollution

- Raw type: generikus nélkül használjuk az osztályt
- Heap polution: ha futásidőben egy paraméteres típussal deklarált változó a nem neki megfelelő típusú objektumra tart referenciát
  - Compile warning
  - Következménye ClassCastException

Korlátozások, wildcard

- Upper bounds
  Box<T extends CanMakeSound>
- Lower bounds
  Box<T super Zebra>
- Wildcard
  Box<?>
  Box<? extends CanMakeSound>
  Box<? super Zebra>

Generikus metódus

- Típus deklarálása metódus szinten
- Főleg statikus metódusoknál, hiszen ott nem lehet a példány típust elérni
- Csak `Object` metódusok elérhetőek

```
public static <T> Box<T> ship(T t){
      Box<T> box = new Box<>();
      box.pack(t);
      return box;
}
```

Referenciák

- Cay S. Horstmann, Gary Cornell: Core Java Vol. I. Fundamentals
- Matt Weisfeld: The Object-Oriented Thought Process

Java API

Gyakran használt osztályok

Object

- Minden osztály őse
- Metódusai:
  - `boolean equals(Object obj)`
  - `int hashCode()`
  - `String toString()`
  - `Class<?> getClass()`
- IDE segít ezek implementálásában

String I

- String s1 = „alma”
- String s2 = new String(„alma”);
- Immutable, String pool
- Metódusai:
  - `int length()`
  - `boolean isEmpty()`
  - `char charAt(int index)`
  - `String concat(String str)`
  - `boolean contains(CharSequence s)`

String II

- Metódusai:
  - `boolean startsWith(String prefix)`
  - `boolean endsWith(String suffix)`
  - `String replace(CharSequence old, CharSequence new)`
  - `String[] split(String regex [, int limit])`
  - `String trim()`
  - `String substring(int startIndex [, int endIndex])`
  - `String toLowerCase()`
  - `String toUpperCase()`
  - `char[] toCharArray()`

StringBuilder I

- Szövegek dinamikus változtatására
- Metódusai:
  - `int length()`
  - `StringBuilder append(bármi)`
  - `char charAt(int index)`
  - `int indexOf(String str [, int fromIndex])`
  - `int lastIndexOf(String str [, int fromIndex])`
  - `StringBuilder delete(int start, int end)`

StringBuilder II

- Metódusai:
  - `StringBuilder deleteCharAt(int index)`
  - `StringBuilder insert(int offset, bármi)`
  - `String substring(int startIndex [, int endIndex])`
  - `StringBuilder replace(int start, int end, String str)`
  - `StringBuilder reverse()`
  - `String toString()`

Primitív típusok csomagoló osztályai I

- Byte
- Short
- Integer
- Long
- Character
- Boolean
- Float
- Double

Primitív típusok csomagoló osztályai II

- Közös statikus metódusok:
  - `valueOf(primitive)` → Wrapper
  - `valueOf(String)` → Wrapper
  - `parseXXX(String)` → primitive
- Autoboxing, autounboxing

Scanner

- Primitív típusok és szövegek reguláris kifejezések alapján történő olvasása
- Metódusai:
  - `boolean hasNext()`
  - `String next()`
  - `String nextLine()`
  - `boolean hasNextXXX()`
  - `XXX nextXXX()`
  - `close() //AutoClosable`

Math I

- Csupa statikus metódus (utility osztály)
- Metódusai:
  - `abs(value)`
  - `ceil(double value)`
  - `floor(double value)`
  - `round(double a)`
  - `min(a, b)`
  - `max(a, b)`

Math II

- Metódusai:
  - `log(value), log10(value)`
  - `exp(double n)` → e^n
  - `pow(double a, double b)` → a^b
  - `random()` → `[0,1)`
  - `sqrt(value)`
  - trigonometrikus függvények

Random

- Létrehozása: seed-del is lehet
- Metódusai:
  - `nextBoolean()`
  - `nextInt([int bound])`
  - `nextLong()`
  - `nextDouble()`
  - `nextGaussian()`

Dátum- és időkezelés I

- java.time csomag
- LocalDate, LocalTime, LocalDateTime
- ZonedDateTime
- immutable

Dátum- és időkezelés II

- Factory-t használ:
  - `LocalDate.of(int year, int/Month month, int day)`
  - `LocalTime.of(int hour, int minute [, int second])`
  - `LocalDateTime(year, month, day, hour, minute [, second])`
  - `LocalDateTime(LocalDate date, LocalTime time)`
  - `XXX.now()`

Dátum- és időkezelés III

- Metódusai:
  - `parse(String text [, DateTimeFormatter formatter])`
  - `String format(DatetimeFormatter formatter)`
  - `plusXXX(long amount)`
  - `minusXXX(long amount)`
  - `plus(long amount, TemporalUnit unit)`
  - `minus(long amount, TemporalUnit unit)`

Dátum- és időkezelés IV

- Metódusai:
  - `withXXX(int part)`
  - `boolean isAfter(other)`
  - `boolean isBefore(other)`
  - `int getXXX() //esetleg Month, DayOfWeek lehet a visszatérési érték`

Időszak I

- Period: év, hónap, nap
- Duration: nap, óra, perc, másodperc
- Factory:
  - Period.of(int years, int months, int days)
  - Period.ofXXX(int amount)
  - Duration.ofXXX(long amount)

Időszak II

- Metódusai:
  - between(one, another)
  - int/long getXXX()
  - long toXXX()
  - minusXXX(long)
  - plusXXX(long)
  - withXXX(int)

Kollekciók

A Collections keretrendszer

Kollekció keretrendszer

- alkalmazása
  - adattárolás
  - adatvisszanyerés
  - adatmanipulálás
  - adattovábbítás
- interfész és implementáció
- polimorfizmus kiaknázása
- algoritmusok
  - keresés
  - rendezés

Interfészek hierarchiája

- Collection
  - Set
    - SortedSet
  - List
  - Queue
- Map
  - SortedMap

Osztályok I.

- ArrayList
- LinkedList

- HashSet
- TreeSet
- LinkedHashSet

Osztályok II.

- PriorityQueue

- HashMap
- TreeMap
- LinkedHashMap

Közös műveletek kollekciókon

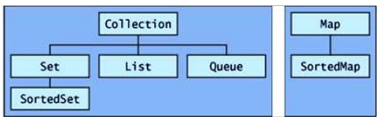
- iteráció
- keresés
- rendezés, feltétele az összehasonlíthatóság
- speciális műveletek

Korábbi Java verziók kollekciói

- Collection keretrendszer létrehozása
- ArrayList és HashMap nem szálbiztos
- szinkronizációs burokkal bármely Collection szálbiztossá tehető

Interfészek kapcsolata

Interfészek hierarchiája



Interfészek és osztályok viszonyai

- interfészek biztosítják a közös viselkedést
- Iterator és Iterable
- Collection interfész és Collections osztály
- ArrayList mint példa
- Map: kulcs – érték párok

Collection interfész

- Collection hierarchia gyökere
- objektumok egy csoportját reprezentálja
- akkor használjuk, ha maximális általánosítás szükséges
- minden leszármazott tud kollekciót fogadni
- kötelező és opcionális műveletek

Kollekciók alapműveletei

- int size()
- boolean isEmpty()
- boolean contains(Object elem)
- add(Object elem)
- remove(Object elem)
- Iterator iterator()

Iteráció

- Iterator interfész az elemek bejárására
- az aktuális elemre hivatkozik az iterátor:
  - hasNext()
  - next()
- lehetőséget ad az aktuális elem törlésére:
  - remove()

Kollekciók tömegműveletei

- boolean containsAll(Collection c)
- boolean addAll(Collection c)
- boolean removeAll(Collection c)
- boolean retainAll(Collection c)
- void clear()
- Object[] toArray()

Kollekció típusok és alkalmazásuk

- Set - csak egyedi elemek legyenek benne
- List - megkötés nélkül, általános esetre, indexelt
- LinkedList - gyors elem mozgatásokhoz, beillesztés és eltávolítás könnyű
- Map - név-érték párok tárolásához

Set I.

- halmaz
- minden elem csak egyszer szerepelhet
- nem ad plusz metódusokat a Collection interfészhez képest
- halmazműveletek
  - tartalmazás
  - unió
  - metszet
  - kivonás

Set II.

- kapacitás, telítettség fogalma
- HashSet - elemeit hash táblában tárolja
- TreeSet - elemeit faszerkezetben tárolja
- EnumSet
- LinkedHashSet

HashSet

- rendezetlen, optimálisabb
- a hash kód szerepe:
  - gyors keresés rendezetlen táblában
  - előállítása az objektum adatmezők alapján
- hash tábla = láncolt listák tömbje, egy-egy lista az ún. bucket (indexelve)

TreeSet

- HashSet, rendezve
- berakás tetszés szerint
- lassabb az elemek hozzáadása
- átlagban $\log_2 n$ összehasonlítás kell
- elemek implementálják a Comparable interfészt!

Priority Queue

- tetszőleges berakás, rendezett kivétel
- a heap adatszerkezet alkalmazásával
  - heap - önrendező bináris fa, legkisebb elem a gyökérhez "húzódik"
- alkalmazása speciális esetekben
  - egyszerű prioritás elvek alkalmazhatók

List

- rendezett Collection (szekvencia), akkor, ha lényeges a sorrend
- az örököltek mellett a speciális List műveletek:
  - elemek pozíció szerinti elérése
  - keresés
  - részlista nézet
- ListIterator
- ArrayList
- LinkedList

Map

- kulcs – érték párokat tárol
- nincs iterátora
- HashMap
- TreeMap
- speciális Map típusok:
  - EnumMap
  - LinkedHashMap

Rendezés

- az implementáló osztály rendezett
- rendezetlenből rendezés segítségével rendezettet készítünk
  - Comparable interfész
  - Comparator interfész
  - Collections.sort(List)
  - Collections.sort(List, Comparator)

Comparable és Comparator

```
public interface Comparable<T> {
    public int compareTo(T o);
}

public interface Comparator<T> {
    public int compare(T o1, T o2);
}
```

Alap algoritmusok

- rendezés (Collections.sort)
- rendezés megszüntetése (Collections.shuffle)
- megfordítás, feltöltés, másolás
- bináris keresés
- maximum, minimum keresés

Kényelmi lehetőségek

- Arrays.asList(T… a)
- Collections.EMPTY_SET, -.EMPTY_LIST, -.EMPTY_MAP - konstans, szerializálható

Generikus típus

- befogadott objektum a legmagasabb típus, Object
- nehézkes, nem biztonságos
- megoldás a generics használata
  - gyakorlatilag kötelező
  - osztály, interfész és metódus paraméterezhető típussal

Nézet és burok (view and wrapper) I.

- új kollekció nézet formájában
- kollekció "csomagoló" – egyszerű tömb, mint kollekció
- egyszeres tárolás, többféle nézet
  - módosíthatatlansági nézet
  - szinkronizálás megoldható

Nézet és burok (view and wrapper) II.

- szinkronizációs burok
  - szálbiztossá teszi
- módosíthatatlansági burok
  - nem adható hozzá és belőle nem vehető ki elem

Szinkronizációs burok

- synchronizedCollection(Collection<T> c)
- synchronizedList(List<T> list)
- synchronizedMap(Map<K,V> m)
- synchronizedSet(Set<T> s)
- synchronizedSortedMap(SortedMap<K,V> m)
- synchronizedSortedSet(SortedSet<T> s)

Módosíthatatlansági nézet

- unmodifiableCollection(Collection<? extends T> c)
- unmodifiableList(List<? extends T> list)
- unmodifiableMap(Map<? extends K, ? extends V> m)
- unmodifiableSet(Set<? extends T> s)
- unmodifiableSortedMap(SortedMap<K, ? extends V> m)
- unmodifiableSortedSet(SortedSet<T> s)

Referenciák

- Cay S. Horstmann, Gary Cornell
  Core Java Volume I. Fundamentals
- The Java Tutorial:
  http://java.sun.com/docs/books/tutorial/collections/index.html

Kivételkezelés

Elvek és gyakorlati alkalmazások
Kapcsolódó monitorozó eszközök

Tematika

- programfutás
- hibák és kivételek
- kivételosztályok hierarchiája
- kivételek használata
- asszerciók
- loggolás

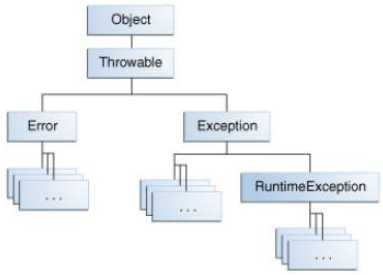
Mit várunk el a programtól?

- értesítés a hibáról, kivételről
- adatok, folyatban levő munka mentése
- nyitott erőforrások lezárása
- biztonságos kilépés a programból

Hibák és kivételek

- objektum alapú hibakezelő rendszer
- felhasználói beviteli hibák
- eszközök okozta hibák
- fizikai korlátok elérése
- kódhibák

Kivételosztályok hierarchiája



Kivételosztályok

- java.lang.Throwable
  - java.lang.Error
    kezelésük nem lehetséges
  - java.lang.Exception
    kezelhető esetek
- java.lang.Exception
  - java.lang.RuntimeException (nem ellenőrzött)
  - Pl. java.lang.IOException (ellenőrzött)

Saját kivételek definiálása

- Exception osztály kiterjesztésével történik
- tetszőleges plusz információt is hordozhat
- használata azonos a "hivatalos" osztályokéval

```
abstract class VeremKivetel extends Exception {}

class VeremMegteltKivetel extends VeremKivetel {}
class VeremUresKivetel extends VeremKivetel {}
```

Mikor használjuk?

- deklarálás metódushoz kapcsoltan
- minden olyan esetben, ahol az erőforrás megléte kritikus
- nem illik dobni RuntimeException esetén!
- NEM pótolja a gondos programozást!

Kivétel specifikálása és kiváltása

- kulcsszó: throws metódusfejben, throw kódban
- a kiváltott kivétel egy osztály példánya
- Kivétel típusú objektum jön létre

```
void egyMetodus() throws Kivétel {
        …
        throw new Kivétel(param1, param2);
        …
}
```


Kivétel elkapása

- kulcsszavak: try, catch, finally
- több catch ág is lehet, a sorrend lényeges
- a finally rész elhagyható

```
try {
        utasítások blokkja;
}
catch(Kivetel k1) {
        utasítások blokkja;
}
finally { utasítások blokkja; }
```

Több kivétel elkapása

- catch ágak sokszorozása bővített sorrendben
- kivétel dobás a catch ágban - kivétel típus váltás

```
catch (SQLException ex)
{
        Throwable forras = new ServletException("adatbázis
hiba");
        forras.initCause(ex);
        throw forras;
}
```


finally szerepe, használata

- erőforrások lezárása
- maga is dobhat kivételt!
- try/catch és try/finally szétkapcsolása
  - finally ág lezárja az erőforrásokat
  - catch ág elkapja a kivételeket

Tippek a kivételkezeléshez

- ne csak elkapjuk, kezeljük is le
- nem helyettesíti az egyszerű teszteléseket
- ne bontsuk szét a blokkot apróbb elemekre
- használjuk ki a kivétel hierarchiát
- nem szégyen a kivétel lehetőségek továbbdobása!

Stack trace analízis

ha már kivételt fogtunk, vizsgáljuk meg…

- throwable.getStackTrace() - StackTraceElement tömb
- Thread.getAllStackTraces() - szálakhoz tartozó StackTraceElement tömbök

Asszerciók

- ki-be kapcsolható hibakereső eszköz
- kulcsszó: assert
- szerkezete: assert feltétel : kifejezés
- asszerció: feltételezett "végzetes" probléma
- üzenetet kérek róla!
- bekapcsolás: java -ea BelepesiOsztaly

Logging

- logging API
- log rekordok állíthatók
- log rekordok átirányíthatók
- a rekordok szűrhetőek, formázhatók
- több logger használható
- konfigurálás fájl alapú

Referenciák

- Java loggolás
- Log4J alkalmazása