# C# versus Java

## **Óbudai Egyetem**, Java Standard Edition
## Mérnök Informatikus szak, BSc
## Labor 3

Bedők Dávid
2016.10.01.
v0.4

# Azonosságok I.

- Curly-bracket nyelvek ({})
  - C, C++, PHP, JavaScript, stb.
  - **C**-re épülő **syntax**
- Erősen típusosak (**statically typed** és nem dynamic typing a jellemző)
- **Objektum-orientált**ak (és osztályokra épített eszközkészlettel rendelkeznek)
- **Just-In-Time** (JIT) **compilation** (semi-interpretation)
- **Garbage Collector** (GC), finalizer/destructor
- A String immutable és reference type mindkét nyelvben
- Referencia típusok és array-ek kezelése
- @Annotációk megléte és szintakszisa (Java-ban Annotációnak, míg C#-ban Attribútumnak hívjuk)

**ProgrammingTheorem** project

# Azonosságok II.

- Collection Framework hasonlóságok
  - A Java **Map** és C# **Dictionary**
  - Set(s), List(s), Vector(s), sorted versions
  - Queue/Stack
  - Párhuzamosságra optimalizált változatok
- Auto boxing/unboxing
- A Java **`final`** nagyon hasonló a C# **`readonly`**-hoz
  - a `final` közelebb áll a valósághoz (elnevezésben is)
    - Java: init blokkban vagy ctor-ban állítható, de pontosan egyetlen egy értékadásban szerepelhet a `final` változó (különben hiba)
    - C#: akármennyi értékadásban szerepelhet a "`readonly`" változó, de muszáj hogy legyen értéke a ctor(ok) lefutása után
  - apróbb különbségek vannak
- Try-with-resources megoldás
  - C# **`using`** `(){}`, Java **`try`** `(){}`
- String típusok switch-ben használhatóak

# Azonosságok III.

- **overloading**, **overriding**
  - Bár a Java virtualitás miatt vannak gyakorlati különbségek (de elméletben azonos)
- **varargs** (java: `String…` C#: `params String[]`)
- Java **`instanceof`** megfelel a C# **`is`**-nek (is/as operator)

**IsAsProject** project

- Java **package**, C# **namespace**
  - szintakszis és gyakorlati különbségek itt vannak
- Szálkezelés (thread, pools, semaphores, thread-local variables)
- Native metódusok

Java 8 óta:

- Metódus referencia, Closures és Lamba kifejezések vannak mindkét nyelvben

# Ami a Java-ban nincsen I.

- Előjelnélküli (**unsigned**) típusok
- 128 bites típusok
- Complex szám kezelés
  - 3rd party library természetesen létezik
- Mátrix típusú tömbök
  - Kiterített vektor (igény esetén hamar kódolható)
- Tulajdonságok (**property**), **indexer**-ek
  - Egyértelműséget sérti (elsősorban nem clean kódokban)
  - IDE getter/setter generálásával, e célt szolgáló 3rd party library-val (pl. AutoValue) elérhető hasonló (de pl. Groovy-Java konbinált projektben ez nem probléma)
  - Indexer-ek nagyon sérthetik az egyértelműséget,

# Ami a Java-ban nincsen II.

- **Tuples**-ek
  - Groovy-Java vagy Scala-Java kombinált projektben kiküszöbölhető ezeknek a hiánya, illetve itt is lehet találni 3rd party-t vagy készíteni egy apró generic típus-t e célra
- **Nem-virtuális metódusok**
  - Minden metódus virtuális, minden hívás címe a VMT-ből kereshető ki (C# def: nem-virtuális)
- Képviselők (**`delegates`**), **`event`**
  - Eseménykezelés megvalósítása klasszikus OO alapokon nyukszik Java-ban (egyértelműbb)
- **Lifted** (nullable) típusok (`?`, `Nullable<>`)
  - Java-ban a primitív wrapper típusok értéke lehet `null`, ez a gyakorlatban elegendő
- Dynamic type, pointers

**Delegate** project

# Ami a Java-ban nincsen III.

- **`foreach`**
  - for használható helyette...
- **`goto`**
  - de a szó reserved, de semmire sem használható
- **`const`**
  - `static final`-t használunk helyette, de a `const` reserved, hogy ne legyen zavaró (jelentése nincs)
- **`out`** és **`ref`** argumentumok
  - szerencsére... helyette objektum-orientáltság van “erőltetve”
- **partial class**-ok
- **operator overloading**
  - de Groovy vagy Scala használata esetén ezzel lehet élni

# Ami a Java-ban nincsen IV.

- **Extension methods**
  - Szerencsére :)
- **Query language** (C# LINQ)

# Ami a C#-ban nincsen

- Soft és Phantom referencia
  - De Weak referencia mindkét nyelvben van
- Checked exceptions

**Exceptions** project

  - C#-ban csak unchecked exception-ök vannak
  - Java **`throws`** kulcsszó
- Instance level inner classes (fa-gyöngy)

**ClassType** project
`hu.qwaevisz.innerclass.tree.Tree`

- Immutable (**`final`**) metódus paraméter

# Java final kulcsszó - véglegesség

- `final` **field**
  - C# `readonly`
- `final` **method**
  - nem lehet override-olni a leszármazottakban
  - Abstract metódus értelemszerűen nem lehet final
- `final` **class**
  - nem lehet leszármaztatni
  - C# `sealed` “zárt”
- `final` **arguments**
  - adott metódusban nem szerepelhet értékadásban (elegáns megoldás automatikusan minden paramétert final-re beállítani)
- `final` **variable**
  - adott blokkban kizárólag egyszer szerepelhet értékadásban

# Java static kulcsszó - osztályszintűség

- `static` **field**
  - Azonos a C# `static` field-el
- `static` **method**
  - Azonos a C# `static` method-al
- `static` **import**
  - Adott statikus metódust úgy importál az adott állományba (class-ba) hogy nem kell az osztályt mindig kiírni (pl. `Math.sin()` helyett elég `sin()`-t írni).
  - C# 6 óta van `using static`
- `static` **class**
  - Kizárólag static inner class létezik. Az inner class az outer class osztályszintű "eleme" lesz. Az static inner class memóriába betöltése (pl. saját static változóinak inicializálása) akkor történik meg, amikor az osztályra az első hivatkozás lefut. Ekkor az osztály top-level szintre kerül!
  - C# `static` class: olyan class aminek csak static metódusai vannak → teljes képzavar, a static kulcsszó a metódusnál osztályszintűséget jelöl, míg a class esetén ez valamilyen marker annotáció szerepét látja el...

# Különbségek

- C# **value type**-ot Java-ban **primitív**nek hívjuk (a Java primitív fix lista, nem bővíthető)
- Java-ban a Date/time referencia típus, míg C#-ban value
- Java **enum** referencia típus, míg C#-ban value (scalar)
- C# Unified type system-je nem található meg a Java-ban (a primitívek teljesen különálló lények), de primitív wrapper típusok léteznek Java-ban is

# Java enum

- Java enum egy korlátozott lehetőséggel rendelkező osztály. Az esetek túlnyomó részében az őse az `Enum` (de **`extends`** `Enum` nem írható ki). Létezik egy un. **`abstract`** enum, ez esetben anonymous leszármazottak vannak, és ennek a közvetlen őse az általunk írt enum típus lesz, és nem az Enum.

**EnumDemo** project

- Az Enum felsorolt értékei az Enum saját típusának megfelelő **`static final`** mezők.
- Lehetnek instance és static mezői, instance és static metódusai.
  - C# extension method segítségével (ami rendelkezik a “static” kucslszóval mint valami annotációval) lehet egy C# value type enum-hoz is instance szintű metódusokat “hozzácsapni”.
- Kizárólag private ctor-a lehet, azonban abból akár több is.

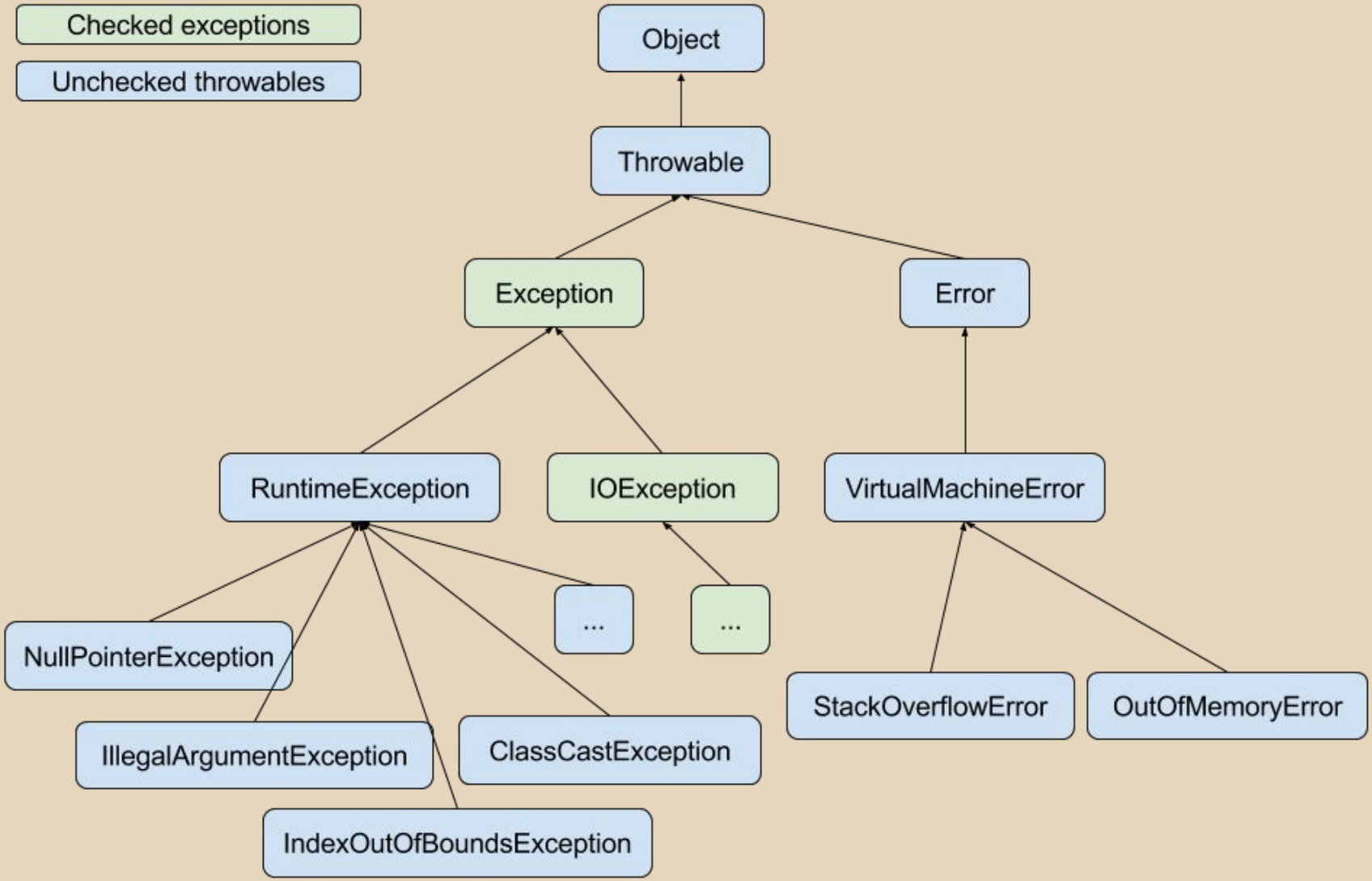
# Láthatóság (access modifier)

Java: `public`, package (default), `protected`, `private`

C#: `public`, `internal`, `protected`, `private`, `protected internal`

A Java csomagolással oldja meg azt, amit az internal láthatóság biztosítana. Itt a logika az, hogy nem keveri az OO és a business láthatóságot.

# Kivételkezelés



# Generics

- Nagyon hasonlónak tűnik, de nem az!
- C# **runtime** kezeli, míg a Java **compile time**
  - A Java generic típusról nem tud a JVM, a byte-code nem hivatkozik semmilyen generic típusra
- Java csak referencia típusra értelmezi
- C#-ban lehet ctor megszorítást is adni
  - E miatt lehet létrehozni új példányt (Java-ban nem)
- Subtype megszorítás mindkét nyelven van, de supertype csak C#-ban
- Kapcsolat a nem generikus világgal a Java-ban megoldott, C#-ban nem-igen (pl.: a Java Collection API-val kompatibilisek a generikus típusok)
- Java-ban az `instanceof` nem használható generic típusra (C# `is`/`as` használható)

**Generic** project