Szoftverfejlesztés (INBPM0420E) – az 1. zárthelyi dolgozat témai

2022. március 11.

# Java technológiai áttekintés

* Java virtuális gép fogalma

Egy saját utasításkészlettel rendelkező és vezérelhető absztrakt számítógép, mely semmit sem tud a Java programozási nyelvről, hanem csak egy speciális bináris formátumot ismer, a class állományformátumot.

* Platform fogalom

A Java platform egy olyan szoftver platform, mely más hardver platformokon fut, és melynek az alábbi két fő komponense van: Java- **virtuális** gép és **API.**

* Java platformok: Java SE, Java EE/Jakarta EE

**Java SE** (Standard Edition) asztali- és szervergépekhez fejlesztéséhez használt.

**Két fajtája van**: **JDK** (Development Kit) **JRE** (Java Runtime Environment)

**Java EE**: API-t és futtató környezetet biztosít többrétegű, és biztonságos **szerver oldali** alkalmazások fejlesztéséhez. A Java SE-re épül.

**JAKARTA EE :** 2018. februárban bejelentésre került, hogy a korábban Java EE-ként ismert technológia Jakarta EE-re lett átnevezve. Visszafele nem kompatibilis.

* Java Runtime Environment (JRE)

**A Java programozási nyelven írt programok végrehajtásához szükséges állományokat tartalmazza**: *egy virtuális gépet, osztálykönyvtárakat és további kapcsolódó állományokat*.

* Java Development Kit (JDK)

**Ez kell ahhoz hogy Java kódot írj, és fejlessz.**

* OpenJDK és Oracle JDK, kapcsolatuk

**JDK11** előtt az **Oracle üzleti funkciói nem nyílt forráskódúak, míg az OpenJDK** nevéből adódóan **nyílt forráskódú 100%-ban.**

**JDK11 után a kettő azonos és felcserélhető**

* A Java SE kiadási modelljének változása

**– Fő kiadások félévente**: bármilyen kis kaki újdonságot tartalmazhatnak, márciusban és szeptemberben jönnek. Az első 2018-ban

– **Frissítések negyedévente**: Biztonsági problémák, bugok fixelése, és két frissítés egy főkiadás előtt

-**Hosszú távon támogatott (long term support, LTS) kiadások háromévente**: Ezekhez legalább 3 évig lesznek frissítések, nagyvállalatok számára van hogy ne ríjanak.

* Az Oracle JDK licencének változása

**Java SE 11** **előtt** a *JDK és a JRE* a **BCL** hatálya alatt került terjesztésre.

**Java SE 11 után** az Oracle ingyenes terjeszti, **kivéve** az üzletit, mert nem akarnak éhen halni gondolom. Subscription alapú, felhasználónként egyet veszel, vagy szerverenként magonként egyet.

* Kapcsolódó programozási nyelvek és platformok

**JavaFX**: GUI készítés (JDK 11-től OpenFX, ugyanaz, más a neve)

**Android:** Elsődleges nyelve a Kotlin, a többit had ne írjam le.

**Apache Groovy:** **OOP** **nyelv Java platformhoz.** Használható szkriptnyelvként, szakterület specifikus nyelvek, funkcionális programozás, meta-programozás.

# Rendszerépítés

* Mi a rendszerépítés?

**A rendszerépítés** az a folyamat, melynek során egy teljes végrehajtható rendszer kerül létrehozásra a rendszerkomponensek, külső könyvtárak, konfigurációs állományok lefordításával, és összeszerkesztésével.

* *Build* automatizálás, automatikus *build* eszközök

**Build automatizálás**: Egy eszköz ami összegyűjti a szoftverrel és működési környezetével kapcsolatos sok információkat egymaga, így egy paranccsal előállítható egy egész rendszer akár.

**Build eszközök**:

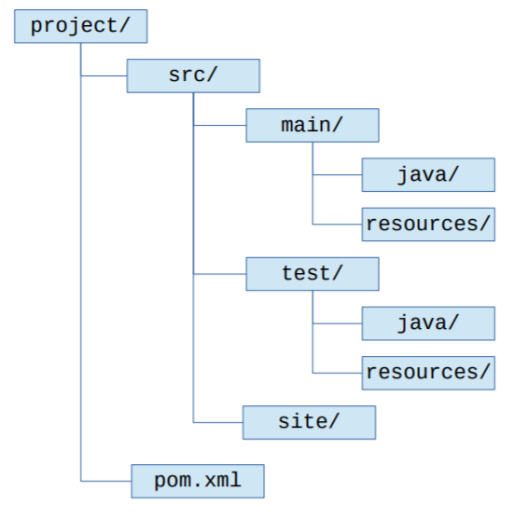
* **Build szkript generálás**: A rendszer elemzi az összeállítandó programot, és csinál egy build szkriptet, de ezt lehet kézzel is, ha nagyon akarod.
* **Verziókezelő rendszer integráció**: A rendszernek le kell tudni szedni a verziókezelő rendszerből a komponensek szükséges verzióit.
* **Minimális újrafordítás:** A rendszer meghatározza mit kell újrafordítani és ha talál valamit, akkor újra is fordítja ¯\\_(ツ)\_/¯
* **Végrehajtható rendszer létrehozása:** A rendszer összeszerkeszt lefordított tárgykódú állományokat egymással, és más állományokkal, aztán létrehozza a végrehajtó rendszert.
* **Teszt automatizálás:** Automatizálási eszközökkel a futtat teszteket, és megnézi hogy összeomlik-e a rendszer változás során, aztán megnézi mi a rossz. Ez volt pl a 3. gyak anyaga @Test JUnit annotációval
* **Jelentéskészítés:** Megmondja hogy jó-e vagy mehet a kukába a program, meg azt is megsúgja hogy az automatizált tesztek eredménye milyen lett.
* **Dokumentáció generálás:** Kézikönyv oldalak, vagy API dokumentációk készítése.
* **Függőségkezelés:** Betölti a szükséges komponensek verzióit a tárolóból.

# Apache Maven

* Mi az Apache Maven?

**Egy projektkezelő eszköz**, mindent megcsinál**, összeállítási folyamatot kezeli**, **projekt információkat szolgáltat, verziót és függőséget kezel, jelentést készít, webhelyet hozhat létre, illetve disztribúciót tesz közzé** és feneket töröl

* A szabványos könyvtárszerkezet



* Parancssori használat

**mvn** -el kezdődnek, **mvn help** manualt hoz be, **mvn site:run, mvn package,** vagy tároló és artifact elérése **mvn org.codehaus.mojo:versions-mavenplugin:2.9.0:help**

Alapfogalmak:

* **POM:** Egy projekt deklaratív leírását tartalmazó XML dokumentum (pom.xml).

**Szuper POM:** A szuper-POM a Maven által alapértelmezésben használt POM. Ha egy projektnek nincs explicit módon megadott szülője, akkor az úgynevezett szuper-POM beállításait örökli

**Effektív POM:** A projekthez tartozó POM, a felmenő ági projektekhez tartozó POM-ok és a szuper-POM kombinációja.  *A futás során a projekthez ténylegesen felhasználásra kerülő beállításokat szolgáltatja.*

**Maven koordináták:**

* **groupId**: csoportazonosító, melynél gyakori a fordított domain-nevek használata (pl org.apache.maven.plugins)
* **artifactId:** projektnév.
* **version:** verzió. na hát!
* groupId:artifactId:version a sorrend pl hu:videosbolt.unfield:2.1

**Tárolók:** A termékek, köztük a bővítmények elérése tárolókból történik, amelyeknek két fajtája van:

* **Távoli tároló:** A weben érhetők el, például HTTP vagy HTTPS protokollon keresztül. Van központi tárolója is a Mavennek.
* **Lokális tároló:** Lokális helyi letöltött termékeket tartalmaz és az *mvn install-*al telepített termékeket.
* Életciklusok, életciklus fázisok
* **Életciklusok:** Egy életciklus jól meghatározott életciklus fázisok egy sorozatát jelenti. **Minden életciklus fázist egy egyedi név azonosít**. – A fázisokhoz bővítmény-célokat lehet hozzárendelni, a **hozzárendelést kötésnek** nevezik.
* Az életciklus fázisok végrehajtása a hozzájuk tartozó bővítmény-célok végrehajtását jelenti. – **Adott fázis végrehajtása maga után vonja valamennyi, a sorrendben azt megelőző fázis végrehajtását**
* Három szabványos életciklus: ***clean***, ***default***, ***site***

A site életcíklusai: pre-site, site, post-site, site-deploy, általában ez is a séma, kivéve a defaultnál

* A default életciklus és főbb fázisai
* **generate-sources:** resources:resources
* **process-resources :** resources:resources
* **compile:** compiler:compile
* **package:** ejb:ejb / jar:jar / rar:rar / war:war
* **install:** install:install
* **deploy:** deploy:deploy
* Tulajdonságok, hivatkozás tulajdonságokra (sztm nem fontos)

A **${x}** formájú hivatkozások helyettesítése a POM-ban. **– ${env.név}** formájú hivatkozások helyettesítése a megfelelő nevű környezeti változó értékével.

**A hivatkozásban megadható Java rendszertulajdonság neve**. Pl: ${java.home}, ${line.separator}

${project.x} formájú hivatkozások helyettesítése a POM megfelelő elemének értékével. Csak egyszerű típusú elemekhez használható!

${settings.x} formájú hivatkozások helyettesítése a settings.xml állomány megfelelő elemének értékéve

* Függőségek megadása

Ami kell:

* **groupId, artifactId, version**: Sima Maven koordináták
* **classifier**: az egy projekt által létrehozott termékek megkülönböztetésére szolgál. (javadoc, sources)
* **type**: a függőség típusát tartalmazza. A típus meghatározza a termék állománynév kiterjesztését (alapértelmezés: jar)
* **optional:** opcionális-e a függőség
* **scope:** a függőség hatáskörét tartalmazza

**Fontos elemei a scope-nak (zh kérdés)**

* **compile**: minden classpath tartalmazza a függőséget, a függő projekteknek is függősége lesz (ez az alapértelmezés)
* **provided**: a függőséget a futtató környezet (például a JDK) biztosítja, csak a fordításhoz használt classpath tartalmazza, nem tranzitív
* **runtime**: a függőség csak a végrehajtáshoz szükséges (a programtesztek végrehajtásánál is rendelkezésre áll)
* **system:** a függőséget nem egy tároló szolgáltatja, hanem a lokális állományrendszerben található
* **test:**a függőség csak a programtesztek fordításához és végrehajtásához áll rendelkezésre, nem tranzitív
* **import: kizárólag pom típusú függőségekhez adható meg** a dependencyManagement részben, egy ilyen függőség kicserélésére kerül a POM-ja dependencyManagement részének függőségeire
* Verziószámok

**A verziószámok p.q.r-s alakúak**

* p a főverzió
* q az alverzió
* r inkrementális verzió
* s build szám vagy minősítő
* Verzió követelmények

Függőségekben verziószám helyett megadható verziótartomány. ([3.8, 4.0])

Ha a <version> között csak egy verzió van akkor a Maven helyettesíti amivel akarja szal tökmind1 mit írsz oda, ez fossa le, de ha azt akarod hogy ugyanaz a verzió legyen akkor kapcsos zárójellel kell megadni [3.8] módon. **Ha két különálló verziót máshogy adunk meg, akkor az BUILD FAILURE-t fog eredményezni**.

* Tranzitív függőségek, tranzitív függőségek kizárása

Ha B függősége A-nak, C pedig B-nek, akkor azt mondjuk, hogy C tranzitív függősége A-nak. A Maven képes megoldani tranzitív függőségek által adódott problémát. Tranzitív függőségek kizárására szolgál az **exclusions** elem.

* *Snapshot*  és *release* termékek

Ha valami még nem jelent meg snapshot, ez hamar elavul, ha valami megjelent akkor az release, általában a 2 külön maven tárolót használ.

* Öröklés

**Olyan projekt lehet szülő, melynél a csomagolás módja pom.**

**A gyerek projekt a szülő projekthez tartozó POM-ból automatikusan örököl** bizonyos beállításokat az effektív POM előállítása során, néha csak akkor lesz a **szülőtől átvéve ha a gyerekbe nincs explicit módon megadva valami**

**Bizonyos elemek esetén a tartalom kombinálása történik**, ha a szülő és a gyerek POM-ban is szerepelnek (repositories, plugins>

* Többmodulos projektek

A többmodulos projektek, más néven aggregátor projektek moduloknak nevezett projektekből állnak. Csomaglás módja pom.

A modulok felsorolása a <modules> elemben történik. Az aggregátor projekt általában alkönyvtárként tartalmazza a modulokat.

**Ha az aggregátor projekt főkönyvtárában kezdeményezzük életciklus fázisok vagy bővítmény-célok végrehajtását**, akkor a **végrehajtás minden egyes modulban megtörténik.**

* Profilok, profil aktiválás

A profilok a POM olyan opcionális beállításokat tartalmazó részei, amelyek csak aktiválás esetén kerülnek felhasználásra. Lehetővé teszik a POM futásidejű módosítását. Véld úgy hogy mint egy sablont



* Bővítmények használata

Lényegében pluginok a <pluginsban>

A plugin elemben rendelkezésre álló elemek:

* **groupId, artifactId, version**: a bővítmény Maven koordinátái
* **configuration**: konfigurációs paramétereket tartalmaz a célok végrehajtásához, amely az xml séma tartalomra nem tesz semmilyen megszórítást
* **dependencies**: a bővítményhez szükséges függőségeket tartalmazza
* **executions**: lehetővé teszi bővítmény-célok végrehajtásának hozzákötését életciklus fázisokhoz
* **id**: a végrehajtás egyedi azonosítója
* **phase**: az életciklus fázis neve, melyhez hozzá kell kötni a cél(ok) végrehajtását
* **goal**: a végrehajtandó bővítmény-célok nevei
* **inherited**: azt jelzi, hogy öröklés során át kell-e venni az execution elemet
* **configuration**: konfigurációs paramétereket tartalmaz a goal elemekben felsorolt célok végrehajtásához
* **extensions**: azt jelzi, hogy be kell-t tölteni a bővítmény kiterjesztéseit
* **inherited**: azt jelzi, hogy öröklés során át kell-e venni a bővítmény beállításait (true, false)

Egy bővítmény-célhoz tartozhat egy alapértelmezett életciklus fázis, ekkor az **execution** elemben nem szükséges megadni a **phase** elemet.

Ha nincs alapértelmezett életciklus fázis, akkor a **phase** elem hiányában a bővítmény-cél nem kerül végrehajtásra!

# Szoftver sebezhetőségek

* Sebezhetőség fogalma

Egy tökéletlenségből származó hiba egy szoftverben, firmware-ben, hardverben vagy szolgáltatás komponensben mely kiaknázható az érintett komponens vagy komponensek titkosságára, integritására vagy rendelkezésre állására negatív hatást gyakoroló módon.

* Sebezhetőségi adatbázis fogalma

Egy sebezhetőségi adatbázis (Vulnerability Database, VDB) egy olyan platform, melynek célja a felfedezett számítógépes biztonsági sebezhetőségek gyűjtése, és terjesztése.

* A CVE es NVD

**CVE:** A Gyakori Sebezhetőségek és Kitettségek (Common Vulnerabilities and Exposures, CVE) program, melynek célja a sebezhetőségek összegyűjtése, és katalogizálása. Egy JSON séma definiálja.

**NVD:** A Nemzeti Sebezhetőségi Adatbázis (National Vulnerability Database, NVD) a NIST által karbantartott, a **CVE** **listára épülő és azzal teljesen szinkronizált sebezhetőségi adatbázis.**

# Annotációk a Java programozási nyelvben

* Annotáció fogalma

Egy olyan deklaráció a kódban, amely metaadatokat szolgáltat egy program konstrukcióhoz, de nincs közvetlen hatása a program végrehajtására.

* Lehetséges felhasználások
* **Információk szolgáltatása a fordítónak**: például tekintsen el bizonyos figyelmeztetésektől, jelezzen bizonyos hibákat.
* **Kódgenerálás:** az annotációk alapján kód generálható
* **Futásidejű feldolgozás**: bizonyos annotációkhoz hozzá lehet férni végrehajtási időben. (pl JUnit egység tesztelés)
* Annotációk szintaxisa
* **Egy annotációt a következők alkotnak**: Egy annotáció interfész neve vagy opcionálisan egy olyan lista, melyet vesszővel elválasztott elem-érték párok alkotnak.
* **Az annotáció interfész határozza meg a használható elem-érték párokat**.
* **Az elem-érték párok sorrendje** **nem lényeges**.
* **Az annotációt az annotáció interfész annotációjának mondjuk.**
* Annotációk fajtai
* **Közönséges annotáció: @XMLElement** (name = "creator", namespace = "http://purl.org/dc/terms/", required = true)
* **Egyelemű annotáció: @SuppressWarnings**(value = "unchecked") **vagy pl: @Target**({ElementType.FIELD, ElementType.METHOD})
* **Jelölő annotáció: @NotNull(), NotNull**
* Hol alkalmazható annotáció?
* **Deklarációkra mindenhol:** Deklaráció annotációknak nevezzük a deklaráció kontextusban előforduló annotációkat.
* **Típus annotációknak** nevezzük a típus kontextusban előforduló annotációkat.
* Előre definiált annotáció interfészek: @SuppressWarnings, @Override, @FunctionalInterface

**@Deprecated:** Az annotációval ellátott elem használata kerülendő, mert például veszélyes vagy jobb alternatíva létezik helyette.

**@SuppressWarnings:** Azt jelzi a fordító számára, hogy el kell tekinteni az annotált elemen (és a benne tartalmazott programelemeknél) az adott figyelmeztetésektől

**@Override:** Azt jelzi, hogy a megjelölt metódus felülír egy olyan metódust, mely egy ősosztályban került deklarálásra. Nem kötelező megadni.

**@Functional Interface:** Annak jelzésére szolgál, hogy egy interfész funkcionális. A funkcionális interfészeknek pontosan egy absztrakt metódusa van.

* Meta-annotációk:

Meta-annotációnak nevezünk egy annotáció interfész deklaráción megjelenő annotációt. A meta-annotációk metaadatokat szolgáltatnak az annotáció interfészekről.

* **@Documented:** Azt jelzi, hogy a megjelölt annotáció interfész annotációinak használata meg kell, hogy jelenjen az API dokumentációban
* **@Inherited:** Azt jelzi, hogy egy annotáció interfész automatikusan öröklődik.
* **@Repeatable:** A Java SE 8-ban jelent meg, azt jelzi, hogy a megjelölt annotáció interfész annotációi akár többször is alkalmazhatók ugyanarra a deklarációra vagy típus használatra
* **@Retention:** Meghatározza a megjelölt annotáció interfész annotációihoz a tárolás módját, az alábbi lehetőségek választhatóak.
* **@Target:** Meghatározza, hogy az annotáció mely elemekre használható, az alábbi lehetőségek állnak rendelkezésre:
* Annotáció interfészek deklarálása

Új annotáció interfész létrehozása az alábbi annotáció interfész deklarációval történik: – módosítók @interface név { deklarációk }

Az annotáció típus deklaráció törzsében elhelyezett metódus deklarációk mindegyike egy elemet deklarál.

A deklaráció törzsében az alábbi deklarációk megengedettek:

* Osztálydeklaráció
* Interfész deklaráció
* Konstans deklaráció, mint például: int MIN = 0; int MAX = 10;
* Speciális metódus deklaráció

A metódus deklarációkban nem megengedettek **formális** **paraméterek**, **típusparaméterek** és **throws** kulcsszó sem.

# Moduláris programozás Java-ban

* Modul fogalma

Egy modul kód és adatok egy névvel rendelkező, önleíró gyűjteménye. A kódja csomagokba van szervezve, melyek típusokat tartalmaznak. Adatként erőforrásokat és más típusú statikus információkat tartalmaz

Egy modul exportáltként jelölheti meg néhány vagy minden csomagját, ami azt jelenti, hogy a típusaik hozzáférhetők a modulon kívüli kódból.

* Egy felső szintű típus akkor, és csak akkor hozzáférhető a deklaráló modulon kívülről, ha nyilvánosnak deklarált és egy exportált csomag tagja.
* Ha egy csomagot nem exportál egy modul, akkor csak a modulon belüli kód számára hozzáférhetők a típusai.

Ha egy modul egy másik modul által exportált csomagokhoz szeretne hozzáférni, akkor explicit módon függenie kell tőle.

* Modul deklarációk

Egy modul deklaráció egy **module-info.java** nevű állományban kerül megadásra a modul forráskódját tartalmazó könyvtárszerkezet gyökérkönyvtárában

Egy module-info.class állományt, azaz egy bájtkódra fordított modul **deklarációt modul leírónak** (**module descriptor**) nevezünk

A modul deklarációt megelőzhetik import deklarációk

* Olvashatóság

**Ha az A modul függ a B modultól, akkor azt mondjuk, hogy A olvassa (reads) B-t, és hogy B olvasható (readable) A által.**

Minden modul olvassa önmagát.

Az olvashatóság reflexív, nem tranzitív.

* Névtelen modulok

Egy speciális, úgynevezett névtelen modulhoz (unnamed module) tartozik minden olyan típus, mely nem tartozik valamely nevesített modulhoz. Egy névtelen modul minden más modult olvas, és minden csomagját exportálja, tehát nyilvánossá teszi.

Az automatikus modulokat kivéve semmilyen nevesített modul sem tud olvasni névtelen modult.

* Moduláris JDK, szabványos és nem szabványos modulok

**Moduláris JDK:** A JDK olyan modulokra történő felosztása, melyek fordítási időben, összeállítási időben és futási időben különféle konfigurációkká kombinálhatók össze, ideértve többek között: A teljes Java SE platformnak, a teljes JRE-nek és a teljes JDK-nak megfelelő konfigurációkat.

**Szabványos modulok:**

* Olyan modulok, melyek specifikációit a JCP szabályozza.
* A szabványos modulok neve a java. karakterlánccal kezdődik.

**Nem szabványos modulok:**

* A JDK részét alkotó egyéb modulok.
* A nem szabványos modulok neve a *jdk*. karakterlánccal kezdődik.

A legalsó *java.base* modul olyan alapvető osztályokat tartalmaz, mint például a *java.lang.Object* és a *java.lang.String.* Ezek **nem függenek egyetlen modultól se, de minden modul függ tőlük.** Akárcsak a ””független nők””.

# A Java SE/JDK új lehetőségei

**Egy előzetes lehetőség (preview feature)** a Java nyelv, a virtuális gép vagy a Java SE API egy új lehetősége, mely pontosan meghatározott, teljesen implementált, de még nem végleges

**Java SE 10 / JDK 10:**

**Lokális változó típus megadása nélkül deklárhatóak**, a ***var***azonosítóval. A típusát fordítási időben következteti ki. *pl: for-each ciklus indexei, for ciklus, deklarált változók, try-with-resources (var i = 0, var szex = „VAJON MI??”)*

**Java SE 11 / JDK 11:**

**HTTP2 Web socketet implementáló kliens** (java.net.http). pl httprequest.newbuilder(), httprequest.client(), response.valami..stb

**Lokális változó szintaxis lambda paraméterekhez,** implicit módon, mint például ( (*var x, var y) -> x.process(y))*

Vagy annotációkkal: (@NonNull varx, @Nullable var y ) -> x.process(y)

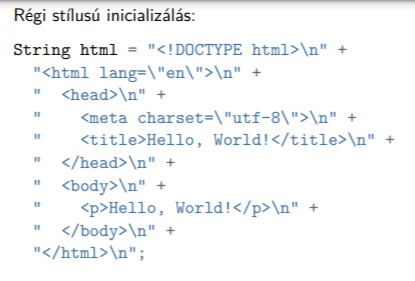
**Java SE 12 – 13 /** Preview featureok, szövegblokkok, switchek

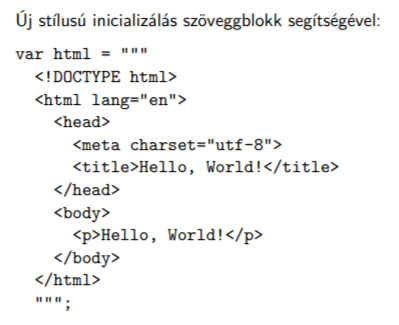
**Java SE 14 / JDK 14:**

**Switch kifejezések (nézz utána mert egyszerű),** minden lehetséges esethez kell egy illeszkedő címke, de nem muszáj, ha van defaultunk megadva pl

switch (kifejezés) { case 1 -> asd, case 2 -> szuka, case 3 -> ingyombingyom, default -> { int k = jozsef\_attila }}

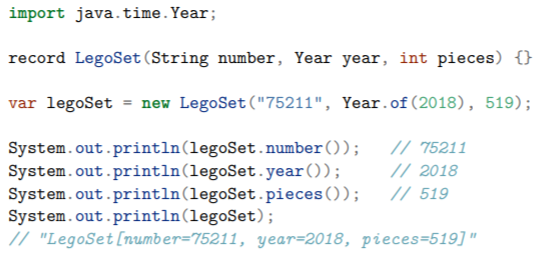
**Java SE 15 / JDK 15:**

Szövegblokkok: Egy szövegblokk egy többsoros sztring literál, mely bárhol használható, ahol egy közönséges sztring literál.



**Java SE 16 / JDK 16:**

**Mintaillesztés instanceof operátorhoz**

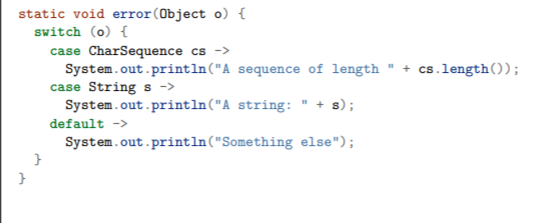
**Rekordok**:

A rekord osztályok nem módosítható adatokat becsomagoló újfajta osztályok. A rekord példányok rekord komponenseknek nevezett rögzített értékek egy halmazát ábrázolják. A rekord osztályok a **java.lang.Record** osztály alosztályai.

Egy rekord osztálynak minden egyes komponenséhez van egy implicit módon deklarált lekérdező metódusa. Van implicit módon deklarált konstruktora, equals(), hashCode() és toString() metódusa is.

**Java SE 17 / JDK 17:**

**Mintaillesztés switch-hez:** A switch utasítások és kifejezések továbbfejlesztése a **case** címkéknél a konstansokon túl lehetővé téve minták használatát. A mintás case címkéknél a kiválasztást mintaillesztés határozza meg egyenlőség vizsgálat helyett.



**Lezárt osztályok:** A lezárt osztályok és interfészek megszabhatják, hogy mely osztályok vagy interfészek terjeszthetik ki vagy implementálhatják őket.

Lehetővé teszik olyan osztályhierarchia deklarálását, mely nem nyitott a tetszőleges osztályok általi kiterjesztésre. Egy lezárt osztályt vagy interfészt csak azok az osztályok terjeszthetnek ki vagy implementálhatnak, melyek számára ez megengedett.

* **final**: Nem terjeszthető ki
* **sealed**: Csak a megengedett alosztályok terjeszthetik ki
* **non**-**sealed**: Kiterjeszthetik nem ismert alosztályok, egy lezárt osztály nem akadályozhatja meg ebben a megengedett alosztályait

# A Java haladó szintű lehetőségei

* Nem absztrakt (alapértelmezett, statikus, privát) interfész metódusok

Egy nem absztrakt interfész metódus egy, a **default**, **static** vagy **private** módosítók valamelyikével deklarált interfész metódus, melynek metódustörzse is van.

**Az abstract, default és static módosítók kölcsönösen kizárják egymást** interfész deklarációknál

**Egy alapértelmezett metódus** egy interfészben a ***default*** módosítóval deklarált példánymetódus.

Fordítási hiba, ha egy alapértelmezett metódus a java.lang.Object osztály egy metódusát írja felül.

**Egy statikus interfész** metódus egy interfészben a ***static*** módosítóval deklarált metódus.

A statikus interfész metódusokat nem öröklik az alinterfészek

A statikus interfész metódusok lehetővé teszik egy interfészhez kötődő konkrét segédmetódusok hozzáadását közvetlenül magához az interfészhez

**Egy privát interfész** metódus egy interfészben a ***private*** módosítóval deklarált metódus

Az alapértelmezett metódusok és a statikus interfész metódusok közötti kódmegosztásra szolgálnak.

* Beépített funkcionális interfészek:

**Consumer**: Egy olyan műveletet ábrázol, mely egyetlen input argumentumot vár és nem ad vissza eredményt.

**Function**: Egy eredményt létrehozó egyargumentumú függvényt ábrázol.

**Predicate**: Egy egyargumentumú predikátumot (logikai értékű függvényt) ábrázol.

**Supplier:** Egy eredményeket szolgáltató objektumot ábrázol.

* Lambda kifejezések

Egy funkcionális interfészt implementáló névtelen belső osztály egy példányát ábrázolják nagyon tömören

A lambda kifejezések névtelen függvényeket ábrázolnak.

*változótípus(opcionális) változónév* -> *érték* alakúak, lehetnek zárójelezettek is

* Metódus referenciák

Egy metódus referencia arra szolgál, hogy egy metódushívásra hivatkozzunk anélkül, hogy ténylegesen hívás történne, és egy funkcionális interfész típus példányát hozza létre

**Referencia egy statikus metódusra:**

* System::currentTimeMillis
* Collections::singletonList

**Referencia egy bizonyos objektum egy példánymetódusára**

* "Hello, World!"::length
* System.out::println

**Referencia egy konstruktorra:**

* int[]::new
* ArrayList::new
* Streamek

Egy stream elemek egy sorozata, melyen műveletek végezhetők, nincs mögöttük tároló, csak továbbitanak adatot.

**A köztes műveletek:** egy új streamet adnak vissza. filter (), map(), sorted()

**A terminális műveletek** egy streamtől különböző eredményt hoznak létre vagy mellékhatást eredményeznek. count(), max(), forEach ()

A köztes műveletek további két típusba sorolhatók:

**Az állapotmentes műveletek** (stateless operations) nem őriznek a korábban látott elemekből állapotot, amikor egy új elemet dolgoznak fel. filter(), map()

**Az állapotőrző műveletek** felhasználhatnak a korábban látott elemekből állapotot. distinct(), sorted()

Egy köztes művelet rövidzár, ha végtelen bemenetből is létrehozhat eredményként egy véges streamet. limit(), takeWhile()

* Streamek: viselkedési paraméterek

**Interferencia-mentesség** (non-interference): A viselkedési paraméterek nem szabad, hogy módosítsák a stream adatforrását.

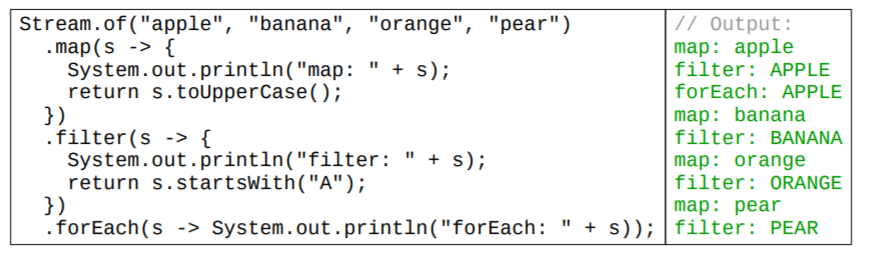
**Állapotmentesség:** Egy lambda kifejezés (vagy a megfelelő funkcionális interfészt implementáló más objektum) állapotőrző, ha eredménye olyan állapottól függ, mely a stream csővezeték végrehajtása során megváltozhat.

A stream csővezeték eredménye nemdeterminisztikus vagy rossz lehet, ha a stream műveletek viselkedési paraméterei állapotőrzők.

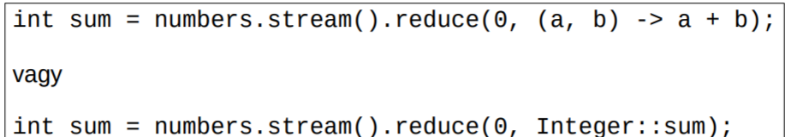
* Stream csővezetékek, műveletek kiértékelése, csővezeték végrehajtás

**Stream műveletek egy csővezetékké láncolhatók össze**. Egy stream csővezeték egy forrásból áll, melyet nulla vagy több köztes művelet és egy terminális művelet követ.

**Egy köztes művelet végrehajtása ténylegesen nem eredményez semmiféle műveletvégzést**. **A csővezeték forrásának bejárása nem kezdődik el, míg a terminális művelet végrehajtásra nem kerül.**



* Streamek: redukciós műveletek (reduce, collect)

A képen szöveg látható

Automatikusan generált leírás**A reduce()** egy terminális művelet, mely elemek egy sorozatát egyetlen elemre redukálja.



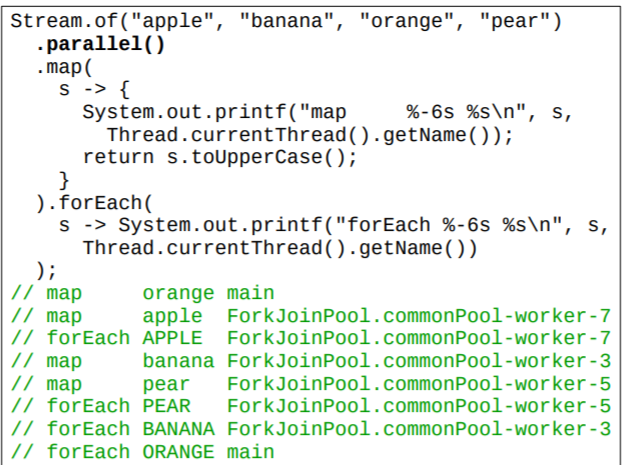
A módosítható redukciós művelet neve **collect().**

* Streamek: párhuzamosság

Minden stream művelet végrehajtható szekvenciálisan és párhuzamosan is.

Az alábbi módszerek valamelyikével **kapható egy párhuzamos stream:**

* Egy kollekció **parallelStream()** metódusának meghívásával.
* Egy már létező szekvenciális stream **parallel()** metódusának meghívásával.



# Git

* Tároló létrehozása (**git init**)
* Állományok figyelmen kívül hagyása**.gitignore állománnyal**
* A munkakönyvtár állapotának vizsgálata (**git status**), nem végleges módosítások megjelenítése **(git diff)**
* Változások véglegesítése (**git commit**), állományok hozzáadása tárolóhoz **(git add**), állományok törlésé tárolóból (**git rm**)
* Verziótörténet megjelenítése **(git log)**
* Ágak használata: ág létrehozása, váltás ágak között, ágak összefésülése, ágak törlése

**git branch \*ágnév\* ; git checkout \*ágnév\***

* Módosítások visszacsinálása: Az utolsó *commit* üzenet megváltoztatása **(git commit --amend)**, nem

véglegesített módosítások elvetése (**git reset**), véglegesített módosítások visszacsinálása (**git revert**) [commit után].