## Adatbázis programozás Java nyelven – JDBC

## (3 -sql utik és relációs atatbáziskezelés, JDK része, mindennek az alapja)

## JDBC-vel szembeni elvárások

## (3 - perzisztencia, SQL nyelv kompatibilitás, platformfüggetlenség, amennyire lehet)

## A JDBC hogyan valósítja meg ezeket a követelményeket?

## (JDBC API = interfacek és metik; implementációk külön; Java.sql csomag, interface példák, magasabb szintűre is)

## JDBC Driver

## (=JDBC interface implementációk; ki szállítja; .jar állomány, feladatok: hálózatkezelés, csatlakozás, sql, eredmények fogadása és visszaadása, driver fajták, használat)

## JBC kapcsolat

## Java alkalmazás -> JDBC API -> Driver

## JDBC url

## (adatbázis hol található és mik a kapcsolódási paraméterek? host, port séma; majd user és password, és egyéb)

## Használat lépései adatmódosítás esetén

## (előfeltétel: JDBPC API és valamilyen Driver; majd DataSource/kapcsolódási paraméterek eltárolása - > Connection példány létehozása és csatlakozás - > PreparedStatement létrehozása, sql utasítással, paraméterezés ?, setXXX(), Execute, TWR, Exception handling

## properties állományban tárolás, nem beégetés; Connection pool, SQLE Kivételkezelés);

## Paraméterezett adatmódosítás

## (beégetés helyett property? PreparetStatementnek átadott Stringbe ? placeholderek majd setxxx(column, érték,)

## Egyszerű JDBC lekérdezés

## (Előfeltétel: Driver(classpath vagy pom.xml) + Connection és PreparedStatemenet példány SQL utival; executeQuery(), ResultSet(), if vagy while(több találat), +IllegalArgEx, plusz TWR ha paraméterezett a query)

## Paraméterezett lekérdezés

## (PreapredStatementbe placeholderek és setXXX()), Resultset(), plus+1TWR setxxx() miatt, , kiszervezés)

## Performancia

## (kisebb darab-> kisebb memória igény de lassabb; nagyobb darab- nagyobb mem igény de gyorsabb)

## Alkalmazás architektúra

## (perzisztens-, üzleti-, felhasználói rétegek + egyéb)

## = DAO (Data Access Object) osztályok

## (DAO szerep, rétegelt szerkezet előnyök, átláthatóság, high responsibility, datasource dependency injeciton high tech esetén és teszt adatbázis)

## Séma inicializálás

## (verziókezelés jelentése, alk-adatbázis verziókezelés kéz a kézben, séma létrehozása az alkalmazással, history table)

## Elvárások séma inicializáló eszközzel kapcsolatban

## (egységes nyelv(xml), platformfüggetlenség, lightweight, visszaállás, parancssoros indítás, clusterezés, placeholder használat, modularizáció, több séma)

## Séma inicializálás Flyway eszközzel

## flyway object, .setDataSource(), .migrate(), .clean; saját history tábla, elvárás: db/migration könyvtár, formátum: V1\_filenév.sql, *version history table, db/migration, V1\_name.sql)*

## Integrációs tesztelés

## (nem jó a unit tesztelés integrációs kell; adatelőkészítés és függetlenség biztosítás, beforeEach-el takarítás)

## Integrációs teszt példa – hogyan érdemes megírni

## (BeforeEach: dataSource példány, flyway példány, .clean(), .migrate(), DAO példány létrehozása konstruktorban átadva a dataSourcet =dependency injection; a konkrét tesztmetódus ezt a DAO példányt használja)

## Generált azonosító lekérdezése

## (cél\_ insert metódus visszatérési értéke legyen az id -> Statemenet.RETURN\_GENERATED\_KEYS paraméter a conn.preparedStatement()-, és ResultSet generálás PreparedStatement.getGeneratedKeys()-el; külön metódusba kiszervezés: executeAndGenerateKey(stmt))

## Tranzakciókezelés

## (tranzakció def, együttes végrehajtás vagy semmi; begin, comit, autocmit false, gond esetén rollback)

## ACID elvek (4)

## Tranzakciókezelés Javaban

## begin(maga a művelet), alapértelmezett autocommit helyett false; commit(), rollback()

**Adatbáziskezelés**

## Architektúra

## (SQL def, server – kliens kapcsolat, (R)DBMS, MariaDB(2), HeidiSQL(2), ACID elvek, SQL nyelv része(4 -> DML, DDL, DCL, TCL)

## A különböző kapcsolatok típusai

## (kapcsolat fajták (3), jelölések(5)

## Adattípusok + egyéb

## (adattípus példák (8), gyorsbillentyűk(4), elnevezési- jelölési konvenciók (4)

## Parancsok, Lekérdezések

## (AUTOINCREMENT (manapság + törlésnél visszaállítás), PK, FK, összetett kulcsok, JOIN+fajtái(4), Beágyazott lekérdezések(4), UNION(def + megkötés), Kapcsoló tábla(def), Constraintek(def)

## 

## CREATE – SELECT – WHERE – AND / OR – ORDER BY – ASC / DESC – LIMIT – COUNT – MIN/MAX/AVG/SUM – GROUP BY – LIKE, ’\_’, ’%’ – SELECT .. WHERE.. (SELECT…) – INSERT INTO – ALTER TABLE .. ADD.. – UPDATE.. SET.. – DELETE FROM – SELECT..FROM..JOIN..ON.. – ON DELETE/ON UPDATE CASCADE, ON DELETE SET DEFAULT/NULL, UNIUQ, NOT NULL, FOREIGN KEY..REFERENCES.., SELECT..WHERE..UNION SELECT..WHERE.. – SELECT.. GROUP BY .. HAVING..

## Java SE haladó - Kollekciók és osztálykönyvtárak

## Generikusok

## Generikusok előtt a kollekciók, hogy néztek ki és milyen hátrányokkal járt ez? (4)

## (bármi lehetett, tudni kellett; instanceof; típuskényszerítés és ClassCastException; ősostályokkal vagy interfacekkel korrlátozás, de ez nem jó megoldás

## Generikusok használata Java 5-től (6)

## (= típus paraméter, korlátozni lehet, fordítási idejű ellenőrzés, mik kerülhetnek be, relációs jelek, diamond operátor)

## Equals(), hashCode

## (Object ős oszátlytól örököklt és gyakran használt)

## Equals

## (Mit vizsgál, mivel több, mint a = operátor, eredeti viselkedés, saját osztályok használata -> hasznos attribútum; equals hívása null vizsgálat, null vizsgálat Stringeknél;, Objects.equals() stat metódus miért jó, ennek jellemzői(3))

## Szabályok equals() írásakor (5)

## refelxív, szimmetrikus, tranzivtív, konzisztens, + 1 nullra

**hashCode()**

**mit csinál/képez le; korlátosabb értékkészlet (info vesztés és nincs visszaképzés); mire hívható; visszatérési érték; nullra mit ad vissza; ugyanarra az objektumra hívva; két különbözőre kaphatjuk -e ugyanazt?; Object toStirng) metódusa**

**Kollekciók – hashcode viszonya**

**tárolás és keresés, kupac, keresési feltételen hash függvény hívása, gyorsaság**

**Szabályok hashCode() implementálásnál (5)**

**ha nem változik az állapot, két nem nullra ha az equals true, , két nem nullra ha az equals false, mely attribútumok használhatóak(2), Objecs.hash() mire jó? override, csak String esetén delegálás, lehet több paramétere, saját osztályokban felhasználás**

## List, ArrayList kontra LinkedList

## hash kontra linked; kollekciók mire és tömbhhöz képest-i hatékonyság

## Collections Framework (1+2)

## tartalma, azok miért felelősek, mit segítenek, miközben mit biztosítanak

**4 fő interface : List -> sorrend, index, dupli; Set-> index, dupli; Queue - > használat mikor és fajták; Map -> elemei, egyediség, leszámrazás**

## Közös tulajdonságaik, viszonyuk (2)

**sok implementáció, egyezőség, eltérés miben, mit ír mindegyik felül**

## Collection interfészben deklarált (az 5 fő interface által örökölt) metódusok (6)

## add, remove, size, isEmpty, clear, contains (boolean!)

## A List interface és implementációi

**implementációk közös tulajdonságai(3), különbségei(1), összehasonlítás alapja**

**ArrayList osztály – (4+1)**

**mi van a háttérben, elemek elérése és hatékonysága, törlés és beszúrás esetén mi történik, kapacitás, erőforrásigény**

**LinkedList osztály (2. List interface implementáció) (4+1)**

**mi van a háttérben, az elemk mit tudnak magukról és hol vannak, eléréshez mi szükséges, beszúrás és törlés, erőforrásigények és gyorsaság, mit implementál ég**

**List metódusok (6)**

**add(void!), remove(void!), get, indexof, lastindexof, set**

**Set (5)**

**halmaz, inex van-e? dupli lehet-e? használat mikor? 2 implementácója**

**HashSet (5)**

**hashcode->klupcaok/bucketek , keresés hatékonysága, sorrend megmarad-e, equals() és hashcode() szinkronizáció és szabályok, lehet-e null és hányszor**

**TreeSet**

**keresőfa, rendezettség, mit implementál még, elemeknek minek kell lenniük, rendezettség mi mentén, null engedett-e,**

**LinkedHashSet**

**mit őríz meg?**

**Queue**

**elemekkel való műveletek sorrendje - > beszúrás, fordított (+nevek) rendezettség esetén prioritási sor= ?; lehet-e dupli, mi definiált (sima queue?)**

**Queue implementácók**

**LinkedList**

**láncolt lista, megjegyzi a sorrendet**

**PrioirityQueue**

**mi alapján rendez és mit vesz ki először**

**Queue metódusok**

**sikertelen művelet -> kivétel vagy speciális érétk(null)**

**add(), remove(), element()= első elem kivétele / offer()=hozzáadás a végére, poll()=eltávolítás az elejéről, peek() – első elem lekérése**

**Deque interfész (3)**

**kiterjeszt; mi definiált, neveik, mit tesz lehetővé, mely tipikus módszerre jó**

**Deque implementáció – ArrayDeque**

**tömb alapú**

**Deque metódusok**

**void push()=hozzáadás sor elejére; pop(), elem kivétele sor elejéről; addfirst addLast(E), getFirst, getLast()**

## Autoboxing használata kollekcióknál (4)

## primitívet hogy lehet tárolni - > csomi osztály; hozzáadás -> autoboxing; kivett elem primitív typusba mentése = autounboxing

**A null érték használata kollekciókban(3)**

**engedi mind kivéve: Treeset és Treemap - > miért? +1 problem autounboxingnál**

**Map interface (8)**

**kulcs-érték párok, generikusok, visszakeresés, egyedi kulcs, érték lehet akár?, Map.Entry interface, ezke metódusaival mit lehet**

**I\_Mplementációi (3)**

**metódusai:get, put, remove, containsKey, containsValue, entrySet - > Map entryk, ezekre még: getkey és getvalue;, keyset, values, visszaadás mögött mi van az utsó háromnál?**

**ALAPVETŐ ALGORITMUSOK**

**Bejárás**

**Iterator**

**interface, paraméterezés, kurzor, hozzáférés .iterator()-al; metódusai: hasNext()+ NoSuchElementException, next(), remove(); sima for ciklusban miért nem módosítunk? - > ConcurrentModificationException**

## Comparator és Comparable (4 + 2)

**rendezettséghez mit kell tudni? számok esetén mi van?, Stringek esetén mi van?(problémák), hogy tudatjuk a futtatókörnyezettel(2)**

**Comparable interface**

**paraméterezés, ha implementálja az osztály, abból mi lesz? hányféle rendezettséget lehet így, compareTo() metódusa - > hívás és paraméter, visszatérési érték, hívás delegálása hasznos attribútumra, ha az equals() true?**

**Comparator interface**

**mikor használjuk?(2), compare() metódusa -> paraméterezés, működtetés, visszatérési érték és szabályok, implementálás külön osztályban generikussal, delegáció, többlépcsős rendezés (ha az első egyenlő); Használata(2sort és konstruktor TreeXXX-nek)**

## Collator osztály

## absztrak o, mire megoldás, Unicode alapján hogy néz ki a rendezettség, mit implementál, használat: Collections.sort(Collator.getInstance(new Locale(”hu”, ”HU” )));

**Keresés**

**lináris-bináris különbség, előny-hátrány; bináris mit feltételez**

**Keresés tömbben – Arrays.binarySearch()**

**rendezettség(természetes vagy Comparator), 2+1 paraméter, visszatérési érték, visszatérési érték mit jelent, - mit jelent**

**Keresés kollekciókban**

**boolean visszaadás: contains() és containsAll() - > bármilyen kollekcióra**

**index visszaadás lineáirs kereséssel : indexOf()-al -> csak listára, visszatérési értékei, működése mivel;**

**index visszaadás bináris kereséssel : Collections.binarySearch() -> csak milyen listára lehet? paraméterek 2+1**

**min és max vizsgálat: Collecitons.min() és Collecitons.max()**

**Rendezés**

## Rendezési algoritmusok

## sokféle; hatékonyság mitől függ? rendezési elvek (2)

## Beszúrásos rendezés

## menet(egyenként helykeresés), rendezett sor, kártyapaklik

## Buborékos rendezés

## egymás melletti elemek vizsgálatával, cserélés, legnagyobb elem mindig a sor vége felé vándorol, sorozat vége rendezett

## Gyorsrendezés

## kiemelt elem - > két részsorozat elé vagy utána, részhalmazokon ismétlés, amíg van rendezetlen

## Rendezés Javaban

## tömbök rendezése: Arrays.sort() - > comparator vagy természetes; TreeSet és TreeMap -y eleve rendezett; természetes vagy konstruktorban Comparator; Collecitons.sort() vagy list.sort(); Timsort algoritmus

## Collections osztály (keresésen, rendezésen felüli metódusok)

## Collections egy utility -y stat metódusok kollekciókhoz; emptyXXX() – üres immutable kollekciók; singletonXXX() ->egyeleű immutable; syncronizedXXX(9 -> szálbiztos burkoló osztállyíal tér vissza, nézet, kapcsolat az eredetivel!; unmodifiableXXX() - -> csak olvasható nézet; önmagán nem hívhat módosítást, de tükrözi az eredeti módosítását; list.shuffle() + random, és list.reverse()

## A clone() metódus, deep clone (3)

## cél: állapotegyezés; Clonable marker interface és Object clone() metódusa nem jó gyakorlat effektív Java alapján; helyette copy konstruktor - > példányt várt paraméterként, állapotot másol át; shallow copy/értékmásolás primitíveknél, ez nem elég objektumoknál mert akkor osztozás lesz az objektumon, helyette new vagy az atrinak is kell copykonstruktor

## Properties állományok

## Properties osztály (4)

## kizárólag mit tartalmaz és ezért mihez hasonlít? általában mi van benne; metódusai: getProperty(), setProperty(), load(), store(), túlterhelések streamre és stringre(reader/writer); XML preferencia vagy .properties; forámtum: egy sorba mi, figyelembe véve mi van? csak egyfajta elválasztó; Xmlből Propertiesbe, speciális adatszerkezet kell -> gyökérelem, entry tagek s közte értékek, atrik, comment

## FILEKEZELÉS

## olvasunk vagy írunk?

## karaktertömbökkel=szöveggel v. bytesorozatokkal dolgozunk? (az adatfolyam a csőben mi lesz, byte vagy karakter)

## kis vagy nagy fileokkal dolgozunk, kell egy pufferelés?

## kell-e konvertálni az adattípusok között? streamem van, de karakteres adattal akarok dolgozni explicit konvertálás nélkül(azt akarok kiolvasni), vagy streamem van, de más típusú adatot akarok beleírni konvertálás explicit nélkül

## *Kis file, pufferelés nélkül, karakteres*

## olvasok: Files.readstring()

## írok: Files.writeString()

## *Kis file, pufferelés nélkül, byteos*

## Files.readAllBytes();

## Files.write()

## *Nagy file, puffereléssel, karakteres*: Reader és Writer

## olvasok: file > filesReader(alacsonyabb szintű; karakterenként olvas) > BufferedReader(magasabb szintű, darabonként olvas) > programom

## írok: file < Writer < BufferedWriter <programom – előny: gyorsabb, nem fogy el a mem, forrás cserélhető

## Files.newBufferedReader/Writer

## *Nagy file, puffereléssel, byteos : stream*

## olvasok: file > inputstream(byteonként) >BufferedInputStream (több byteot egyszerre, pufferelve)> programom

## írok: file < outputstream < bufferedoutputstream < programom

## new BufferedOutputsream(Files.newOutputstream)

## *konvertálás*

## PrintWriter – kül. típusok írása *szöveges állományba* explicit konverzió nélkül

## *különböző típusok írása streamre:* DataOutstream vagy PrintStream

## *különböző típusok olvasása streamből:* DataInputstream (write-/readUTF stb.)

## *Inputstreamem van, de szöveges adattal akarok dolgozni; jellemző példa:* String olvasás classPathról

## file < Inputstream < Inputstreamreader < Bufferedreader

## this.getclass().class.getresourceStream – inputstreamet ad vissza

## *Outputstreamem van, de tetszőleges típusú adatot akarok a fileba írni (pl. dat fileba akarok írni Stringet, doublet stb.)*

## file < Outputstream < Bufferedoutputstream> Printstream

## new Printsztream = new Pritstream(new Buffered(files.newputputstream)

## egybe ír ki, sout is prontstream

## *+ byteokkal dolgozom,* bufferedre DataoutputStream vagy DataInputStream

## különöbző típusokat írhatok bele vagy olvashatok ki, de kiolvasasnál tudni kell mi után milyen típus jön writeUTF, writeDouble, readUTF

## *írás zip fileba*

## Zipoutputstream csatlakoztatása BufferedOutputStreamre

## stream.putnExtEntry(new Zip Entry() String fileName-et vár, ha könyvtár, akkor

## /el”

## stream,write, byteokat vár! kell a getbytes();

## Files osztály

**File helyett Files és Path, Path objektum, konvertálás, gyakran haszánált műveletek: eixsts(), isdirectory(), size() , createdirectory(s(), copy(source, target, CopyOption.option), delete(), deleteifExists(); StandarCopyOption.REPLACE-EXISTING**

**Filekezelés tesztelése**

**@TempDir annotációval Path tempDir atri, majd Path path = tempDir.resolve(”employees.txt”) és ennek a pathnak a használata, asserteléskor ideiglenes file tartalmánka kiolvasása és összehasonlítás**

**Elmélet**

mire emlékszem? vázlat -> ráolvasás, kimaradtak megjelölése -> újra

közben felmerülő kérdések -> lista és utánanézés

Kivételkezelés \ Filekezelés

Generikusok \Kollekciók\ Alapvető algok

eddigi vizsga tesztek

netes tesztek

**Gyakorlat**

lecke vidik

lecke feladatok

konzi feladatok

**Filekezelés**

Döntések:

* írunk, vagy olvasunk
* karakteres adat, vagy byte állomány, amit írunk vagy olvasunk
* egybe vagy darabokban akarjuk-e végezni a z olvasást-írást (az adat/file kicsi vagy nagy-e?)
* szükség van-e konverzióra az olvasás-írás során (nem szövegest szöveges állományba írunk, nem szöveges állományt szövegesbe olvasunk be)
* a file az alkalmazásunk részét képezi e vagy sem
* szükség van-e tömörítésre
* fileból vagy meróiából olvasok? fileba vagy memóriába írok (stringwriter)?

1. Egybe / olvasás / karakteres adat

Files.readString(Path, karakterkódolás ->

**JAVA OO (objektum orientáltság) ELEMEK (5)**

**Attribútumok és metódusok (9)**

**Immutable objektumok**

def, létrehozás (3)

**JavaBeans objektumok**

def, szabályok(3+1), property, eredeti felhasználás

**Metódusok**

fej részei (5), törzs szabályai (5)

**Változó hosszúságú paraméterlista (varargs)**

leírás (4), szabályok(2)

**Metódus hívások láncolása**

def, feltétel, Builder osztályok (3)

**Metódus túlterhelés**

def, paraméter szignatúra def, használat, sorrend, visszatérési típus problem

**Metódus paraméterek és visszatérési értékek**

paraméterek (3), visszatérési értékek (2)

**Statikus attribútumok és metódusok**

hovatartozás – elérés – használat (2-2-2)

**Értékmásolás szerinti paraméterátadás**

def, primitív és osztálytípus; hatókör, visszahatás(2); memória

## Konstruktorok és inicializátorok (3)

## Default és paraméter nélküli konstruktor (3)

## Konstruktor túlterhelés

## def, par. egyeztetés, chaining, telescoping, private konstruktor, builder osztály

## Inicializátor

## def, típusok (2), mikor futnak le, használat, elhelyezés

## Öröklődés (5)

## Öröklődés

## mikor, 2 típus, nevek, jelölés, egyszeri öröklődés, object ősosztály, isa, hasa, melyiket?

## Konstruktorok és az öröklődés viszonya (4 + 2)

## Object ősosztály

## (3), metódusai, override

## Attribútumok öröklődése

## information hiding, elfedés/felülírás, super, láthatósági módosítók

## Metódusok öröklődése

## elérhetőség, override, override szabályai (6), super, elfedés névazonosság esetén, dinamikus kötés szerinti meghívás, override annotáció, statikus szabályok (3)

## Absztrakt osztályok és interfészek (9)

## Absztrakt osztályok

## def, használat, létrehozás, tartalom, implementáció/ override, referencia típus, változó statikus típus

**Interfészek**

jelentés, javaban(2->def), jellemzők(5) - deklarálás, elnevezés, használat, példányosítás, létrehozás, feladat; implementáló osztály, konstans, többszörös öröklődés, uml jelölés, marker, statikus típus, névütközés

**Interfészekkel kapcsolatos szabályok (8)**

**Dependency Inversion**

kompozíció, d.i. def, hasznosság, implementáció cserélés

## Default interface metódusok

## mikor, jelzés, tartalom, csak hol, öröklődés(3), többszörös öröklődésnél névütközés szabályai, private

## Interfészek és az öröklődés viszonya

## interfészek közötti többszörös, hálós kapcsolat, egy osztály, hogy osztály maradjon.., osztályok közötti öröklődésnél ? öröklődik az interface metódus; névütközés több interface implementációnál

## Statikus interfész metódusok

## csak public és no default, öröklődés?, hívás, törzs, cél, list.of, java9

## Absztrakt metódusok implementálása felsorolásos típusokban

## nem maradhat.., ezért absztrakt metódusait..!, absztrakt metódus vagy kiszervezés interfacebe

## Állapotgép

## def, új állapot ? határozza meg és hány lehet; UML-irányított kapcsolat, enum használat, JVM és fordítók

## 

## Haladó OO elvek (3)

## Polimorfizmus

## def, stat és dinamikus típus, variációk, típuskényszerítés

## Dinamikus kötés

## mit határoz meg, mikor dől el, virtuális metódus, overrideolt metódus láthatósága

## Is-a has-a kapcsolatok

## isa(3-statikus/fordítási, példánya.., instanceof)

## hasa – kompozíció, dinamikus/ futási

## számosság, kötelezőség, irány

## preferálás - >tartalmazási/leszármazotti viszony - >újrafelhasználhatóság

## imlementáció kicserélése setterrel

## Kivételkezelés

## Throwable osztály és ábra

## keletkezhet(2) és kötelezően kezelendő kezelés (2)

## try catch – sorrend, finally; gyakorlati megoldások

## Saját kivétel

## implementálás előtt; létrehozás – cause és message már az ősben

## TWR

## mikor, szerkezet, autoclosable,

## Multicatch

## ne kapcsolódjanak, pipe

## Generikusok

## előtt; nem volt forditási idejű hiba

## típus paraméter, fordításkor végbemenő ellenőrzés, berakható elemek típusai

## diamond operátor, explicit dinamikusnál nincs polimorfizmus, de..

## Equals, hashcode

## equals

## eredeti működés ==;

## állapotra kell, és hasznos atrira delegáljuk

## objects.equals – nincs exception!

## szabályok: reflexív, szimmetrikus, tranzitív, konzisztens

## hashcode

## bármilyen adat leképzése..lehet az int - is

## kupacokba, gyors keresés

## szabályok

## objects.hash(Object, autob!)

## Collections

## több obj együttes kezelés, különböző módokon..miközben a méret dinamkus!

## metódusok add/remove/clear/size/isEmpty/contains

## List index, dupli ok, null ok

## ArrayList()

## gyors elérés indexxel, tömb a háttérben, kapacitás, műveletek lassúsága (törlés, beszúrás, csere

## Linked – láncolt, csak azt tudja magáról..mi gyors, mi lassú

## get, set, indexof, lastindex,of

## Set

## halmaz, no index, no dupli, null ok (kivéve treeSet)

## HashSet

## kupac gyors keresés; equals és hashcode szinkronban legyen!

## LinkedHashSet

## megjegyzi a berakás sorrendjét

## Treeset

## nem ok a null, rendezettség (természetes – compareto() vagy Comparator konstruktoran)

## Queue

## sor

## amikor a az elemekkell való művelet sorrendje számít(kivétel)

## lehet dupli!

## FIFO,LIFO

## sima queue – csak fifo; implementáció : priorityqueue, rendezettség szerint rakja be; add és poll

## Deque, kétvégű sor, fej és farok, mindkét véghez hozzáférünk, ArraxDeque, LIFO FIFOt is tud; push, pop, getfirst.removefirst és last

## Collections – Autoboxing

## csak objektumot; primityv autoboxingja csomaogló osztályba hozzáadásnál

## kivételnél autounboxing

## null érték probléma kicsomaoglásnál

## Map

## egyedi, kulcs-érték, nem collection

## map.entry interface

## .entriset(), .getkey(), .getvalue()

## .get(key), .put(key, value), contaiskey, containsvalue, .keyset, .values(Collection!)

## HashMap, treeMap, LinkedHashMap

## Bejárás

## Iterátor

## Colleciton.iterator, hasnext, next, remove, kurzor

## Comparable, Comparator, Collator

## natural ordering(egy) vagy Comparable(több)

## Comparable – generikus, compareto() – int, hasznos atrira; compareot és equals összhangja

## Comparator - mikor, compare(), int, 2 példányt kap, használat .sort() és Tree- konstruktor

## Collator – nicode, ékezetek stb.

## Collator.getIstance(new Locale, hu, HU)

## Keresés

## lineáris – bináris(rendezettség! - gyorsabb)

## Array.sBinarySearch() – natural ordering vagy Comparator, mínusz és info!

## Kollekcióban

## contains, contains all – boolean, indexof (lineáirs!)

## listára Collecitons.binarysearch(), rendezettség!

## .min és max()

## Rendezés

## újba vagy helyben

## Beszúrásos-kártya paklik, Buborékos, Gyors

## Arrays.sort(), Tree-, Collectins.sort()

## +metódusok

## emptyxxx() és singleton() – immutable (add-al futási idejű hiba!!)

## snycronized () – visszahat! de nézet

## unmodifiable() – ez csak nézet és nincs visszahatása, de tükrözi..

## reverse() és shuffle() – csak lista!

## Clone()

## helyett:copykonstruktor

## shallos éd deep, valahogy legyne egy new!

## Properties

## maphez hasonló, kizárólag stirng-string pár

## konfig adatok, : vagy =, trim

## new Propert()

## load, store Reader Writer, .properties pe vagy .propertiesból vagy xmlből

## geproperty(key), default, setproperty

## xml: entry tagek, comment, <properties>

**Parancssoros dolgok, Maiven, Github, IDEA-ban új projekt létrehozása és beállítások**

**JDK = JVM + osztálykönyvtár (ez volt régen a JRE) + parancssori eszközök (pl. a fordító)**

**Java file létrehozása és futtatása IDE-t nem használva:**

**NotePad -> java kód - > mentés mint filenév.java - > javac filenév.java (= fordítás; ezzel alakul bytekóddá és lesz belőle filenév.class) -> futtatás: java filenév (JVM futtatja a kódot)**

**Java filenévnek és osztálynévnek egyeznie kell!**

**Maven**

**szabványos eszköz a build folyamat megvalósítására + kezeli a függőségeket a pom.xmlbe tett konfigurációkkal; softver management and comprehension tool; convention over configuration elv (megfelelő struktúra legyen az első!)**

**Maven működésének előfeltétele -> standard projekstruktúra megléte; ezt az IDEA biztosítja**

**Új projekt létrehozása és konfigurációja IDEA-ban**

**File -> New project - > Maven felkapja; JDK és name, location beállítása - > létrejön a struktúra (src/main és src/test + java és resources; .idea (idea saját dolgai és gitignore!), pom.xml**

**pom.xmlbe a gépen található jdk verzió(az elején kiválasztottuk), a karakterkódolás és függőségek felvétele, amennyiben szükséges (jUnit, Flyway, MariaDB)**

**Maven build folyamat leírás**

1. **Java classokat bytekóddá alakítja és eltárolja . class állományokként itt: target/classes**
2. **erőforrás fileokat a resources könyvtárból átmásolja szintén ide: target/classes**
3. **classok + erőforrások becsomagolása .jar állományba artifactid+version.jar néven és eltárolás itt: target**

**Build indítása IDEA-ban**

**Build -> Build Project**

**Build indítása parancssorosan**

**mvn clean package parancs kiadása a projekt gyökerében - > újra létrehoz mindent a targetben**

**. jar állomány futtatása parancssorosan**

**java -classpath target\speedrent-1.0-SNAPSHOT.jar userinterface.CustomerMain**

**Ha a pom.xmlbe be van állítva a main metódus (maven-jar-plugin):**

**java -jar target\distjar-1.0-SNAPSHOT.jar;**

**GitHub repository ráállítása java projektre**

**1.GitHub remote repository létrehozása**

Github -> your repositories -> new -> repository name -> **NEM KELL SEHOVA TICK** -> create repository

**2.Projekt ráállítása a githubos repora:**

a) IDEA

Git – Enable Version Control – url megadása (ki kell másolni githubról a repo linket)

b) Parancssoros

projekt mappában:

git init

git remote add origin repourl

git add . , git commit -m „comment”, git push, git fetch