**lokális változó scope**

**visszatérési érték**

**aktuális-formális paraméter egyezés szűkebb láthatóság**

**final, private, void**

**Java alkalmazás ↔ Adatbázis kapcsolat**

**Java alkalmazás**

**↓**

**JDBC API**

JDK része;

interfacek és metódusok

**↓**

**JDBC Driver**

JDBC implementációk

classpathon, mint jar állomány vagy Maven esetén depencencyként felvéve

**↓**

**Adatbázis**

server(maga az adatbázis = tárolás) és kliens (adatbázissal a kommunikáció)

MariDb – HeidiSQL

**ACTIVITIES** (pk, autoincrement) **id**, start\_time, activity\_desc, actvity\_type

**TRACKPOINTS** (pk, autoincrement) **id**, act:time, lat, lon, **(fk -> activities id) activity\_id**

**Java obketumok**

*Activity* **– két konstruktor\_ id-val (a visszanyert id-val vlaó példányosításhoz) és id nélkül (az insterhez való példányosításhoz)**

* > lesz egy List<TrackPoint > atrija;

*TrackPoint* – – két konstruktor\_ id-val (a visszanyert id-val vlaó példányosításhoz) és id nélkül (az insterhez való példányosításhoz)

kell a foreing key-s id atribútum bele, mert úgyis tárolja majd az Activity a saját lista atribútumában a trackpointot és össze fognak tartozni – adatbázisban activityk és trackpointok két külön táblan, ott kell a foreign ki az összekapcsolás miatt

**Kapcsolat létesítése az adatbázisal JDBC-vel**

**1.DataSource példány létrehozása,** és benne a kapcsolódási paraméterek eltárolása

* > JDBC URL: -setUrl(); host, port, séma, egyéb
* setUser(), setPassword()
* egyszer hozzuk létre!
* teszt BeforeEach-ben vagy main metódusban példányosítjuk és a DAO osztály attribútumaként tároljuk:

public class ActivityDao {  
  
 MariaDbDataSource dbDataSource;

…

class ActivityDaoTest {  
  
 private ActivityDao activityDao;  
  
 @BeforeEach  
 public void init () {

MariaDbDataSource dataSource;  
try {  
 dataSource = new MariaDbDataSource();  
 dataSource.setUrl("jdbc:mariadb://localhost:3306/activitytracker?useUnicode=true");  
 dataSource.setUser("activitytracker");  
 dataSource.setPassword("activitytracker");  
} catch (SQLException se) {  
 throw new IllegalStateException("Cannot create database", se);  
}

**2.Connection példány létrehozása**

* az adatbázissal való kapcsolat létrehozása
* a Datasource példány alapján választ implementációt
* magában a DAO osztály metódusaiban példányosítjuk (újra és újra)

Connection connection = dbDataSource.getConnection();

**3. Statement létrehozása**

* ez reprezentálja majd a kérést az adatbázis felé
* sima Statement: az SQL utasítást a .executaUpdate() paramétereként várja
* PreparedStatement : az SQL utasítás a példányosítás paramétere, és lehet bele tenni placeholdert, ezáltal paraméterezhető

PreparedStatement ps = connection.prepareStatement("insert into activities (start\_time, activity\_desc, activity\_type) values(?,?,?)")) {

* > oszlopok felsorolásával és placeholderekkel!

**4. PreparedStatement felparaméterezése**

setXXX() metódusokkal

első paraméter a placeholder száma, második az érték

ps.setTimestamp(1, Timestamp.*valueOf*(activity.getStartTime()));  
ps.setString(2, activity.getDesc());  
ps.setString(3, activity.getType().name());

**3. Statement végrehajtása**

**ha update:**

ps2.executeUpdate();

**4. Ha query – visszakapjuk a query eredményét egy ResultSet-ben:**

**Resultset** rs = ps.executeQuery(); //kurzoros!

*ResultSet metódusok:*

**next()**

**getXXX(columnindex);**

**5. Update & query id - ha az adatbázis által generált (auto\_increment) id-t akarjuk visszakapni:**

PreparedStatement ps = connection.prepareStatement("insert into activities (start\_time, activity\_desc, activity\_type) values(?,?,?)", Statement.*RETURN\_GENERATED\_KEYS*)) {

**és**

ResultSet rs = ps.getGeneratedKeys();

long id = rs.getLong(1);

**Tranzakció kezeléshez kiegészítések:**

.setAutoCommit(false,) .executeUpdate(), .commit() vagy .rollback()

…

conn.setAutoCommit(false);

…

if (name.startsWith("x")) {  
 throw new IllegalArgumentException("");

…

conn.commit();  
} catch (IllegalArgumentException iae) {  
 conn.rollback();  
}

…

**Architektúra**

**DAO osztály**

public class EmployeesDao {  
  
 private DataSource dataSource;

**…..**

**metóduskban új Conneciton és PreparedStatement példányosítás**

**TestOsztály**

**/1/ xxxDAO attribútum**

public class EmployeesDaoTest {  
  
 private EmployeesDao employeesDao;  
  
 **/2/** **@BeforeEach**

**/2a/ - Datasource példányosítás és SQLException handling** public void init() {  
 MariaDbDataSource dataSource;  
 try {  
 dataSource = new MariaDbDataSource();

**…**

**/2b/** **Flway = séma inicializálás**

Flyway flyway = Flyway.*configure*().dataSource(dataSource).load();  
flyway.clean();  
flyway.migrate();

/2c/ **DataSource dependency injection**

employeesDao = new EmployeesDao(dataSource);

/3/ **@Test metódusok**

update + lekérdezés kombináció

.createEmployee()

..ListEmployeeNames() /assertEquals/

**SQL utasítások / séma tárolása**

Resources/db/migration/V1\_name.sql;

**Gyakorlat – architektúra felépítése az Activitytracker2-es feladathoz**

1. activitytracker2 séma futtatása Heidiben
2. Java -a activitytracker2 package
3. Activity osztály létrehozása két konstruktorral
4. Type enum létrehozása
5. activities tábla léttrehozása Heidiben: id = pk, auto\_inc! Timestamp!
6. ActivityTrackerMain osztály létrehozása
7. main metódus és Activityk példányosítása
8. savetActivity(Activity) metódus megírása
9. findActivityById(id) metódus megírása
10. listAllActivities() metódus létrehozása
11. DAO osztály létrehozása és az ActivityTrackerMainből a metódusok kiszervezése
12. Adadtbázisban az activties tábla kitörlése, sql kód kiszervezése V1\_Activities fileba Javaba
13. Integrációs teszt megírása, flyway-el
14. saveActivity(Activity) metódus módosítása úgy, hogy visszaadjon egy Activityt a kinyert id-val
15. Trackpoiint osztály létrehozása
16. Activity osztály kiegészytése List<Trackpoint> attribútummal
17. track\_point tábla létrehozásának felvétele a sémába; fk = activity\_id(=activitytracker id)
18. saveAcvitiy() módosítása, hogy a trackpointokat lementse a track\_point táblába
19. findActivityById() módosítása, hogy betöltse a Trackpointokat is
20. tesztesetek módosítása
21. setautocommit(false), és rollback, ha a la tés lon nem valid
22. teszteset erre is

Először **másold át magadhoz a teszteseteket**, majd **commitolj azonnal!**A két feladatra **1,5 órád** van összesen!Old meg a feladatokat, minden feladat megoldását egy **külön commitban add be**!  
Ha letelik a másfél óra az első commitodhoz képest és nem fejezted be, megint **commitolj,** akkor is,  
ha nem vagy kész! Utána nyugodtan folytathatod a megoldást, akár több órát is  
ülhetsz felette, ha kész vagy, commitolj!

**netes tesztek**

olyan nincs, hogy NULL, csak null

private class-t nem lehet extendelni

anonymus inner class

They can be declared to extend another class or implement a single interface

public static void main(String[] args)

{

Object obj = new Object()

{

public int hashCode()

{

return 42;

}

};

System.out.println(obj.hashCode());

}

hashcode lehet negatív int is!

Stringben van hashcode implementáció karakterek alapján

iterator nem az Iterator interface metódusa, hanem a Collectioné

finally ág akkor is lefut ha a try[] ban return van

ha error van, lefut a finally de aztán lehal

ha exception van és el van kapva, akkor utána fut tovább, ha nincs új dobás

a finally ágon a close() kötelező IO-t dob! vagy fejlécben eldobatni vagy finally ágra is try-catch kell; ez így nem elég:

finally

{

out.close();

}

try lezárás után csak catch jöhet, semmi más, egy sout se!

tryhoz nem kötelező catch! de ha nincs, akkor kell finally.

tömb deklaráció – közvetlen váltózó előtt és után állhat a [], máshol nem ->

~~public [ ] int a~~ nem oké!

vigyázni! short és Short

Java is case sensitive – While lehet osztálynév, mert nem while

case-ben csak egy érték lehet!

~~case 0, 1: j = 1;~~

(b2 = true) nem ugyanaz, mint( b2==true)!

swtich - Case expressions must be constant expressions., Since x is marked final, lines 12 and 13 are legal; however y is not a final so the compiler will fail at line 11.

## compliation fails! ’i’ is delcared inside the loop:

for (int i = 0; i < 4; i += 2)

{

System.out.print(i + " ");

}

System.out.println(i);

**swtichben a break** a swtichből lép ki,; ha a swtich egy ciklusban van, akkor a ciklus fut tovább!

ahol belép, onnan mindenre lefut, amíg nincs break!!

**default ágba** nem fut belefeltétlen, csak ha az utsó ág?

continue and break statement can only occur in a looping construct.

char c6 = '\uface'; - ok, unicode representation

**interfaceben** nem static metódus nem lehet protected se private

float f1 = -343; //ok, integer és float is 32 bit, nem kell kitenni, hogy F

~~int f1 = -343.4F;~~ fordítva nem ok! Explicit casting kell

float f2 = 3.14; . nem ok, jobb oldali double, explicit konv kéne vagy kitenni, hogy F

class y implements x { void method() {} n – nem lesz ok, az overridenál ez így package private láthatóság, ami szűkebb, mint az interfaceben deklarált public metódus

default értéke az interface atirnak static final, ezt egy implementáló osztály nem változtathatja meg:

interface A

{

    int i = 111;

class B implements A

{

    void methodB()

    {

~~i = 222;~~

}

Interfaces can’t have initializers.

i =+ i / i; - nem +=! csak a + értékét veszi

ha az osztály örököl más osztálytól metódust, amit egyébként egy implementált interface számára előír, akkor az implementációnak minősül és nem kell override

 when a function is static, runtime polymorphism doesn't happen.**Ha az ős statikus metódusa ha uaz, mint a leszármazotté,** ősét fogja hívni, ha a statikus típus az ős, ilyenkor nincs polimorfizmus; ha a statikus is a leszámrazott, akkor fogja a leszármazottét

a statikus a gyermekben nem írható felül nem statikussal, és fordítva sem lehet!

Base b = new Derived();;

        b.show();

    }

**örökölt metódus láhatósága nem lehet szűkebb**! ha public volt, az override nem lehet private

**In Java, it is not allowed to do super.super.**

**ilyet nem lehet:**

interface Nameable {

default void setName(String name) {

this.name = name;

}

**objects.equals(Ob1, Pbj2)**

nem száll el nullpointerrel, ha egyik null - > false; mindkettő null -> true

ha egyik sem null obj1 equals()át fogja használni

**Lehet e tömbre contains?** **NEM**, csak kollekciókra.



**Típuskonverziók**

long k = 1000000000; / literált int-en értelmezi, de amíg elbírja az int tartomány, implicit típuskonverzió lesz és ok (10 számjegyig)

**nagyobb tartományon értelmezett változót akarom belerakni a kisebbe, úgy, hogy egyébként az érték beleférne - explicit konverzió kell**

long k = 10000000;

int i = k; /nem engedi, bár beleférne, de kell az explicit:

long k = 10000000;

int i = (int)k; / így kell

**nagyobb tartományon értelmezett változót akarom belerakni a kisebbe, úgy, hogy egyébként az érték nem fér bele - explicit konverzió kell és értékvesztés, fura szám, ne használjuk!**

int k = 100000000;  
short i = (short)k /engedi, de fura szám - > ha nem fér bele és explicit, fura szám lesz

double k = 1000.44;  
int i = (int)k; / engedi, nem fura szám és értékvesztés -> belefért azért nem fura a szám

double k = 1000;  
int i = k; /nem fordul le!

**a kisebb tartományon értelmezett változót akarom belerakni a nagyobba - implicit konverzió lesz:**

short k = 100;  
int i = k; /ok

char k = 'a';  
int i = k; /ok

char k = 'a';  
short i = k; /nem fordul le!

int k = 1000;  
double i = k; /ok

**Összegzés**

ha kisebb tartományba akarok nagyobbat, mindenképp kell explicit konverzió és értékvesztéssel járhat! Ha nagyobba kisebbet, implicit konverzió lesz, de pl. osztásnál két int hányadosa is int lesz, erre kell vigyázni!

short k = 100;  
int i = k; /ok

char k = 'a';  
int i = k; /ok

char k = 'a';  
short i = k; /nem fordul le!

int k = 1000;  
double i = k; /ok

long k = 1000000000000L; /10 számjegy fölött kiakad az int tartomány és ki kell írni, hogy L, ez így már nem konverzió; long 19 számjegyig

long k = 5000000000000000000L;  
int i = (int)k; /engedte, de nagyon fura, teljesen más értéket ír ki; értékvesztés?

double k = 5000000.56;  
int i = k; /nem engedi

double k = 5000000.56;  
int i = (int)k; /explicit konverzió, értékvesztéssel

**SWITCH**

csak short, char, int és String lehet!

**NÉZETEK**

eredeti kollekcióval vannak kapcsolatban

*lehet csak olvasható*, pl Collecitons. unmodifiableList() – metódus rajta nem hívható (nincs visszahatással az eredetire) de minden eredetin történő változást tükrözni fog

*lehet írható* – pl. Collecitons.SyncronizedList() - hívható rajta metódus és oda-vissza hatással vannak egymásra az eredetivel

**ARRAYS.ASLIST() – mit enged?**

**fix elemszámú, de nem immutable!**

List<Integer> nums = Arrays.*asList*(1,2,3,4,5);

reverse, shuffle – ok

Collections.*reverse*(nums); / ok

művelet, de elemszám nem változik

nums.set(2, 5); /ok

elemszám módsító művelet (törlés, beszúrás) – nem ok!

nums.add(2, 5); /lefordul de nem fut le = kivételt dob: UnsupportedOp – AbstractList

nums.remove(2); /lefordul de nem fut le = kivételt dob: UnsupportedOp –

**LIST.OF() – mit enged?**

**Immutable!**

List<Integer> nums = List.*of*(1,2,3,4,5);  
  
Collections.*shuffle*(nums); - /lefordul de nem fut le = kivételt dob: UnsupportedOp - Immutablecollection

nums.add(2, 5); - /lefordul de nem fut le = kivételt dob: UnsupportedOp - Immutablecollection

**COMPARETO() – PRIMITÍVRE!**

**compareto () intet vár ! ezért int típusú atrira a kivonás oké, doublere nem; compareto() csak objektumra hívható!**

…double int s;

@Override  
public int compareTo(Employee o) {  
 return Double.*valueOf*(s).compareTo(Double.*valueOf*(o.s));  
} / OK

..private int s;

@Override  
public int compareTo(Employee o) {  
 return Integer.*valueOf*(s).compareTo(Integer.*valueOf*(o.s));  
} / ok

@Override  
public int compareTo(Employee o) {  
 return s - o.s;  
}

**collecitons.sort(Set)**

nem ment! fordítási idejű hiba; vagy egyből Treeset, vagy LinkedHashSet, ami megjegyzi aberakás sorrendjét

**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