**Main**

public static void main(String[] args) {

}

**Értékek kiírása**

System.out.println("Hello World!");

**Értékek beolvasása**

Scanner scanner = new Scanner(System.in);

System.out.println("What's your name?");

String name = scanner.nextLine();

System.out.println("What's your year of birth?");

int yearOfBirth = scanner.nextInt();

Vigyázz a nextInt() metódussal, mert az nem olvassa be a sorvége jelet, és a következő nextLine() nem fog működni.

int age = scanner.nextInt();

scanner.nextLine(); //Ez fogja az ottmaradt sorvége jelet beolvasni

String name = scanner.nextLine();

Helyette jobb lehet a következő (NumberFormatException kivételt dob, ha nem szám):

String line = scanner.nextLine();

int age = Integer.parseInt(line); // Exception-t

Fájlból:

try (Scanner scanner = new Scanner(Path.of("books.txt"))) {

while(scanner.hasNextLine()) {

String line = scanner.nextLine();

System.out.println(line);

}

}

catch (IOException ioe) {

throw new IllegalStateException("Cannot read file", ioe);

}

try (Scanner scanner = new Scanner(FileScannerMain.class.getResourceAsStream("/books.txt"))) {

while(scanner.hasNextLine()) {

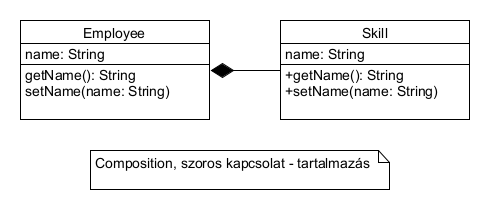
String line = scanner.nextLine();

System.out.println(line);

}

}

**UML**

[](https://github.com/Training360/strukturavalto-java-public/blob/master/cheet-sheet/uml-demo.png)

**Konstruktor**

**public** Trainer(String name, int yearOfBirth) {

**this**.name = name;

**this**.yearOfBirth = yearOfBirth;

}

private: senki számára nem látható és nem módosítható

default vagy package private: azonos csomagban lévő osztályok látják

protected: azonos csomagban lévő vagy a leszármazott osztályokból látható public: mindenki számára látható és módosítható

public Product(String name) {

this.name = name;

}

public Product(String name, int stock) {

this(name);

this.stock = stock;

}

super()

public class Employee extends Person {

private int salary;

public Employee(String name, int age, int salary) {

super(name, age);

this.salary = salary;

}

}

**Getter - Setter**

**public** String getName() {

**return** name;

}

**public** void setName(String name) {

**this**.name = name;

}

**Változók**

* **byte**: egész 8-bit -128 - 127 (inclusive).
* **short**: egész 16-bit -32,768 - 32,767 (inclusive).
* **int**: egész 32-bit -231 - 231-1
* **long**: egész 64-bit -263- 263-1
* **float**: 32-bit
* **double**: 64-bit
* **char**: 16-bit Unicode character. '\u0000' (or 0) - '\uffff' (or 65,535 inclusive).

i = (int) d; // típuskonverzió

boolean result = true;

char capitalC = 'C';

byte b = 100;

short s = 10000;

int i = 100000;

var sum = 1.5; // a típust a kezdőérték határozza meg

double d1 = 123.4; // same value as d1, but in scientific notation

double d2 = 1.234e2;

float f1 = 123.4f;

long socialSecurityNumber = 999\_99\_9999L;

public static final int NUMBER\_OF\_SEASONS = 4; //”konstans”

**Számok, műveletek**

Bináris, pl. 0b0011

Oktális, pl. 0377

Hexadecimális, pl. 0xff

Decimális, pl. 12

Integer.toBinaryString //binárisba

Integer.toString(100, 2) //más számrendszerbe (2, 8, 16)

Integer.parseInt(String, int) // más számrendszerből vissza

+, -, \*, /, % +=, -=, \*=, /=, %=, &=, |=, ^=

<, <=, >, >=, ==, != !, &, |, &&, ||

(? :) //háromoperandusú

boolean even = i % 2 == 0; // Páros?

double quotient = 10 / 4; // 2.0 – egészosztás

Math.round() //egészre kerekít

Math.PI, Math.E

Math.abs

Math.sqrt Math.cbrt //gyök, köbgyök

Math.hypot // sqrt(x2 +y2)

Math.signum //előjel

Math.scalb //hatvány

**Típuskényszerítés, autoboxing, String műveletek**

double d = 12; // int -> double implicit

int i = (int) 3.14; // double -> int kényszerítéssel, értéke 3

Integer i = 12; // int -> Integer autoboxing

int j = i; // Integer -> int autoboxing

int i = Integer.parseInt("12"); // Parse

double d = Double.parseDouble("12.1") // Parse

String si = Integer.toString(12); // toString

String di = Double.toString(d); // toString

**Stringműveletek**

boolean eq = name.equals(anotherName); // Egyenlőségvizsgálat

s1.concat("def") vagy s1 + ("def") //összefűzés

String message = String.join("-", "Java", "is", "cool"); // "Java-is-cool"

String fruit = "Apple"; // keresés

int position = fruit.indexOf("pp"); // 1 (ha nem találja, -1)

int indexOf(String substring, int startIndex)

int length = fruit.length() // 5 (hossza)

String sub = fruit.substring(1, 4); // ppl (1. indexű bekerül, 4. indexű nem kerül bele)

String upper = fruit.toUpperCase(); // APPLE ldm. toLowerCase

boolean empty = fruit.isEmpty(); // false, üres string esetén true

boolean blank = fruit.isBlank(); // false, ha csak whitespace karaktereket tartalmaz, akkor true

boolean startsWith = fruit.startsWith("app"); // true

boolean endWith = fruit.endsWith("le"); // true

boolean containsDoubleP = fruit.contains("pp"); // true

String part = fruit.substring(0, 2).toLowerCase(); // ap – összefűzhetők

word.charAt(i) //String adott karaktere

word1.equalsIgnoreCase(word2) //Stringek összehasonlítása

trim() length()

String santa = "ho".repeat(3); // "hohoho"

Daraboljunk!

String s = "John;1980";

String[] parts = s.split(";"); // ; a separator karakter

String name = parts[0];

int yearOfBirth = Integer.parseInt(parts[1]);

Járjuk be karakterenként!

String s = "abcd";

for (char c: s.toCharArray()) {

System.out.println(Character.isLetter(c));

}

Hosszú string összefűzése:

StringBuilder sb = new StringBuilder();

for (int i = 0; i < 10; i++) {

sb.append(i).append(" \* ").append(i).append(" = ").append(i \* i);

}

toString(), charAt(), indexOf(), length(), substring(), insert(), delete() / deleteCharAt(), reverse()

nextXXX(), hasNextXXX() – boolean

next() / hasNext(), useDelimiter(), extLine() / hasNextLine()

Formázás:

String formatted = String.format("Name: %s, year of birth: %d", employee.getName(), employee.getYearOfBirth());

// Name: John Doe, year of birth: 1980

// Vagy azonnal kiírva:

System.out.printf("Name: %s, year of birth: %d", employee.getName(), employee.getYearOfBirth());

**Dátumok**

LocalDate fromDate = LocalDate.of(2015, Month.JANUARY, 30);

LocalDateTime fromDateTime = LocalDateTime.of(2015, 1, 20, 10, 15);

now()

**Elágazások**

if () {} else {}

switch(){

case "":

;

case "":

;

break;

default:

;

}

**Ciklusok**

Klasszikus for ciklus. Csak akkor használjuk ezt, ha szükségünk van a ciklusváltozóra.

for (int i = 0; i < 10; i++) {}

Un. for-each ciklus, használjuk mindig ezt, ha lehet!

List<String> names = List.of("John Doe", "Jack Doe");

for (String name: names) {

System.out.println(name);

}

While:

int count = 1;

while(count < 11){

System.out.println("Count is: " + count);

count++;

}

Do-While:

int count = 1;

**do** {

System.out.println("Count is: " + count);

count++;

} **while**(count < 11);

break; //kilép a cilklusból

continue; //a cilklus következő értékére lép

**Tömbök**

Lehetőleg kerüljük, helyette használjunk listát! Néhány helyen elkerülhetetlen, pl. varargs, split() vagy ha a feladat így kéri.

int[] numbers = new int[10];

int[][] numbers = {{1}, {1, 2}, {1, 2, 3}};

String[] names = new String[]{"John", "Jack"};

String toString() String deepToString()

boolean equals() boolean deepEquals()

void sort() //sorba rendez

T[] copyOf() T[] copyOfRange() //másol

int length = names.length;

for (String name: names) {

System.out.println(name);

}

**Listák**

List<String> names = List.of("John", "Jack"); // Módosíthatatlan lista

List<String> moreNumbers = new ArrayList<>(); // Módosítható lista, diamond operátor, primitív típus nem lehet

List<String> copy = new ArrayList<>(numbers); // Módosítható másolat

E get(int index) //adott elem vissza

void clear() //listát törli

boolean contains(Object o) //tartalmazza-e

int indexOf(Object o) //hol van

boolean remove(Object o) //törli az elemet

int size() //elemszám

copy.add("Jane"); // ["John", "Jack", "Jane"] - hozzáadás

copy.remove("John"); // ["Jack", "Jane"] - eltávolítás

boolean containsJohn = names.contains("John"); // false - nincs benne

int size = names.size(); // 2 - hossza

int indexOfJane = names.indexOf("Jane") // 1 - 1. indexen

for(String name: names){

System.out.println(name);

}

**Véletlenszám**

Random rnd = new Random();

int randomNumberTo10 = rnd.nextInt(10); // 0 - 9-ig generálhat számokat, 10-et sosem generál

**Felsorolásos típus**

public enum Coin {

TWOHUNDRED, HUNDRED, TWENTY, TEN, FIVE

}

public enum Coin {

TWOHUNDRED(200), HUNDRED(100), TWENTY(20), TEN(10), FIVE(5);

private final int value;

Coin(int value){this.value=value;}

public int getValue(){return value;}

for(Coin i : Coin.values()){} // bejárás

values() //értékek tömbje

Coin c = Coin.valueOf("HUNDRED"); *//Coin.HUNDRED*

String nameOfCoin = c.name(); //"HUNDRED"

int index = c.ordinal(); //1, elem sorszáma

public enum TransactionState { //absztrakt metódussal

SUCCESS {

@Override

public boolean isComplete() {

return true;

}

};

public abstract boolean isComplete();

}

public enum TrafficLight { //állapotgép

RED {

TrafficLight next() {

return TrafficLight.RED\_YELLOW;

}

};

abstract TrafficLight next();

}

**JUnit**

public class TestCalculator {

void testAdd() {

// Given

Calculator calculator = new Calculator();

// When

int result = calculator.add(2, 3);

// Then

assertEquals(5, result);

}

}

**Kivételkezelés**

try {

int result = a / b;

// Ha nincs hiba

System.out.println("A hányados alsó egészrésze: " + result);

} catch (ArithmeticException ae) {

// Ha van hiba

throw new IllegalArgumentException("Arithmetic error", ae);

}

finally {

// lezárni az erőforrásokat! pl.

if (scanner != null) {

scanner.close();

}

}

Multicatch:

public class TrainerParser {

public static final String SEPARATOR =";";

public Trainer parse(String line) {

try {

String[] fields = line.split(SEPARATOR);

Trainer trainer = new Trainer(fields[0], Integer.parseInt(field[1]));

} catch (NullPointerException | IllegalArgumentException | IndexOutOfBoundsException e) {

throw new ParseException("Invalid line = " + line, 0);

}

}

}

Saját kivétel:

public class InvalideAgeException extends RuntimeException {

private int parameterAge;

private int minAge;

public InvalideAgeException(String message, int parameterAge, int minAge){

super(message);

this.minAge = minAge;

this.parameterAge = parameterAge;

}

public int getParameterAge() {

return parameterAge;

}

public int getMinAge() {

return minAge;

}

}

**Összegzés tétele**

Listában szereplő számok összege

public int sum(List<Integer> numbers) {

int sum = 0;

for (Integer n: numbers) {

sum += n;

}

return sum;

}

**Számlálás tétele**

A stringben szereplő b betűk száma:

public int countLetterB(String s) {

int count = 0;

for (char c: s.toCharArray()) {

if (c == 'b') {

count++;

}

}

return count;

}

**Szélsőérték keresés tétele**

Listában szereplő legnagyobb szám:

public int max(List<Integer> numbers) {

int max = Integer.MIN\_VALUE;

for (Integer n: numbers) {

if (n > max) {

max = n;

}

}

return max;

}

**Eldöntés tétele**

Csak 100-nál nagyobb számokat tartalmaz?

public boolean containsGreaterThanHundred(List<Integer> numbers) {

for (Integer i : numbers) {

if (i > 100) {

return true;

}

}

return false;

}

**Szűrés**

Csak a száznál nagyobb számok listáját adja vissza:

public List<Integer> greaterThanHundred(List<Integer> numbers) {

List<Integer> filtered = new ArrayList<>();

for (Integer i : numbers) {

if (i > 100) {

filtered.add(i);

}

}

return filtered;

}

**Transzformáció**

Csak a neveket adja vissza:

public List<String> getNames(List<Employee> employees) {

List<String> names = new ArrayList<>();

for (Employee employee: employees) {

names.add(employee.getName());

}

return names;

}

**Rekurzió**

public long getFactorial(int n) {

if(n > 1) {

long solution = getFactorial(n - 1);

return n \* solution;

} else {

return 1;

}

}

**Bináris fájl olvasása, írása**

Path file = Path.of("data.dat");

try {

byte[] bytes = Files.readAllBytes(file);

System.out.println(Arrays.toString(bytes));

}

catch (IOException ioe) {

throw new IllegalStateException("Can not read file", ioe);

}

Path file = Path.of("data.dat");

try {

Files.write(file, new byte[]{97, 98, 99, 100, 101});

}

catch (IOException ioe) {

throw new IllegalStateException("Can not write file", ioe);

}

**Szöveges fájl beolvasása soronként**

Path fileInRootDirectory = Path.of("C:\\employees.txt");

Path fileInActualDirectory = Path.of("employees.txt");

Path path = Path.of("employees", "john-doe.txt")

public void readIdsFromFile(String file) {  
 Path source = Path.*of*("src/main/resources/" + file);  
 try (BufferedReader reader = Files.*newBufferedReader*(source)) {  
 String line;  
 while((line = reader.readLine()) != null) {  
 ids.add(line);  
 }  
 }  
 catch (IOException ioe) {  
 throw new IllegalStateException("Can not read file", ioe);  
 }  
}

Ugyanez classpathról:

**try** (BufferedReader reader = **new** BufferedReader(**new** InputStreamReader(

EmployeeService.class.getResourceAsStream("/employees.txt")))) {

String line;

**while**((line = reader.readLine()) != **null**) {

System.out.println(line);

}

}

**catch** (IOException ioe) {

**throw** **new** IllegalStateException("Can not read file", ioe);

}

**Szöveges állomány kiírása soronként**

public class FileWriter {

public void writeLines(List<String> employees, BufferedWriter writer) {

try {

for (String employee : employees) {

writer.write(employee.getName() + "," + employee.getYearOfBirth());

}

} catch (IOException ioe) {

throw new IllegalStateException("Can not read file", ioe);

}

}

public static void main(String[] args) {

try (BufferedWriter writer = Files.newBufferedWriter(Path.of("data.csv"))) {

new FileWriter().writeLines(writer);

} catch (IOException ioe) {

throw new IllegalStateException("Can not write file", ioe);

}

}

}

try (BufferedWriter writer = Files.*newBufferedWriter*(file, StandardOpenOption.*APPEND*)) {  
 writer.write(name + "\n");  
} catch (IOException e) {  
 throw new IllegalStateException("Can't open file", e);  
}

Files.writeString(file, "John Doe\nJane Doe", StandardOpenOption.APPEND); //létezőhöz

**Printwriter – eltérő típusokhoz**

List<String> employees = List.of("John Doe", "Jane Doe", "Jack Doe");

Path file = Path.of("employees.txt");

try (PrintWriter writer = new PrintWriter(Files.newBufferedWriter(file))) {

for (String employee: employees) {

writer.print(employee);

writer.print(",");

writer.println(200\_000);

}

}

catch (IOException ioe) {

throw new IllegalStateException("Can not read file", ioe);

}

**InputStream**

Path file = Path.of("data.dat");

try (InputStream inputStream = new BufferedInputStream(Files.newInputStream(file))) {

byte[] bytes = new byte[1000];

int size;

while ((size = inputStream.read(bytes)) > 0) {

System.out.println(size);

}

}

catch (IOException ioe) {

throw new IllegalStateException("Can not read file", ioe);

}

**OutputStream**

Path file = Path.of("data.dat");

try (OutputStream outputStream = new

BufferedOutputStream(Files.newOutputStream(file))) {

for (int i = 0; i < 1100; i++) {

outputStream.write("abcde".getBytes());

}

} catch (IOException ioe) {

throw new IllegalStateException("Cannot write file", ioe);

}

Path file = Path.of("data.dat");

try (DataOutputStream outputStream = new DataOutputStream(

new BufferedOutputStream(Files.newOutputStream(file)))) {

outputStream.writeUTF("John Doe");

outputStream.writeInt(200\_000);

}

catch (IOException ioe) {

throw new IllegalStateException("Can not write file", ioe);

}

exists(Path): létezik-e a Path által reprezentált fájl vagy könyvtár

isDirectory(Path): az átadott Path könyvtár-e

isRegularFile(Path): az átadott Path normál fájl-e

size(Path): a fájl méretét adja vissza bájtban mérve

createDirectory(Path): létrehozza a Path által reprezentált könyvtárat feltéve, hogy annak minden szülőkönyvtára már létezik

createDirectories(Path): létrehozza a Path által reprezentált könyvtárat, és a hiányzó szülőkönyvtárakat is

copy(Path source, Path target, CopyOption... options): fájl másolása

copy(InputStream in, Path target, CopyOption... options): InputStream tartalmának fájlba írása

copy(Path source, OutputStream out): fájl tartalmának OutputStream-re helyezése

move(Path source, Path target, CopyOption... options): fájl vagy könyvtár mozgatása

delete(Path): könyvtár vagy fájl törlése, ha nem létezik kivételt dob

deleteIfExists(Path): könyvtár vagy fájl törlése, ha létezik

**Collection**

List: elemeknek sorrendje, index alapján és duplikált is

Set-ben nincs sorrend és duplikációt sem tartalmazhat.

Queue elemek egy megadott sorrendben

Map elemei kulcs-érték párok, kulcsok mindig egyediek

boolean add(E element): új elem hozzáadása.

boolean remove(Object o): elem eltávolítása; igazzal ha volt

int size(): az elemek számát adja meg.

boolean isEmpty(): igazzal tér vissza, ha a kollekció elemszáma nulla.

void clear(): eltávolítja az összes elemet.

boolean contains(Object o): megadja, hogy megtalálható-e a kollekcióban.

boolean hasNext() E next()

**List**

ArrayList: osztály a háttérben tömbben tárolja az elemeket

LinkedList: háttérben kétirányú láncolt lista áll

Queue elemek egy megadott sorrendben

Map elemei kulcs-érték párok, kulcsok mindig egyediek

void add(int index, E element): új elemet szúr be a meghatározott pozícióra.

E element(int index): elem lekérdezése az indexe alapján.

int indexOf(Object o): átadott elem első előfordulás indexével tér vissza, vagy -1

int lastIndexOf(Object o): átadott elem utolsó indexével, vagy -1.

void remove(int index): törli a listából a megadott indexű elemet.

E set(int index, E element): lecseréli elemet átadottra, régi tér vissza.

**Set**

Set<String> names = new HashSet<>();

names.add("John");

names.add("Jack");

names.add("John");

System.out.println(names); // [John, Jack]

names.remove("John");

boolean contains = names.contains("Jack"); // true

for (String name: names) {

System.out.println(name);

}

HashSet elemeket hashCode-juk alapján bucket-ekbe (equals() és hashCode())

TreeSet elemeket keresőfában, képes rendezetten visszaadni

**Queue**

|  | **Kivételt dob** | **Speciális érték** |
| --- | --- | --- |
| **Elem hozzáadása végére** | boolean add(E e) | boolean offer(E e) |
| **Elem kivétele elejéről** | E remove() | E poll() |
| **Fej lekérdezése** | E element() | E peek() |

void push(E e): elem hozzáadás a verem elejéhez

E pop(): elem kivétele a verem elejéről

addFirst(E e), peekLast()

**Map**

Map<String, String> words = new HashMap<>();

words.put("apple", "alma");

words.put("peer", "körte");

System.out.println(words.get("apple")); // alma

words.put("apple", "Apfel"); // Felülírja az értéket

System.out.println(words.get("apple")); // Apfel

Set<String> keys = words.keySet(); // Kulcsok

Collection<String> values = words.values(); // Értékek

for (Map.Entry entry: words.entrySet()) {

System.out.println(entry.getKey() + " - " + entry.getValue());

}

**Számlálás Map-pel**

Számoljuk meg, melyik betűből mennyi van egy Stringben!

String s = "alma mater";

Map<Character, Integer> countOfLetters = new HashMap<>();

for (char c: s.toCharArray()) {

if (countOfLetters.containsKey(c)) {

countOfLetters.put(c, countOfLetters.get(c) + 1);

}

else {

countOfLetters.put(c, 1);

}

}

System.out.println(countOfLetters); // { =1, a=3, r=1, t=1, e=1, l=1, m=2}

int numberOfM = countOfLetters.get('m'); // 2

**Lista rendezése**

List<Integer> numbers = new ArrayList<>(List.of(6, 5, 8, 3));

Collections.sort(numbers);

// Figyelembe veszi az ékezeteket

List<String> names = new ArrayList<>(List.of("Benjámin", "Áron", "József", "Arnold"));

Collections.sort(names, Collator.getInstance(new Locale("hu", "HU")));

List<Employee> employees = List.of(

new Employee("John Doe", 1980),

new Employee("Jack Doe", 1970));

// Év vagy név szeretnénk rendezni

Collections.sort(employees, new Comparator<Employee>() {

@Override

public int compare(Employee o1, Employee o2) {

// return o1.getYearOfBirth() - o2.getYearOfBirth();

return o1.getName().compareTo(o2.getName());

}

});

*/\*  
 private Path myFile;  
 private List<Human> humans = new ArrayList<>();  
  
 public FileManager(Path myFile) {  
 this.myFile = Path.of(myFile);  
 }  
  
 public void readFromFile(BufferedReader reader) {  
 String line;  
 while ((line = reader.readLine()) != null) {  
 String[] parts = line.split(" ");  
 humans.add(new Human(parts[0], parts[1]));  
 }  
 }  
  
 public static void main(String[] args) {  
 try (BufferedReader reader = Files.newBufferedReader(Path.of("names.txt"))) {  
 new FileReader().readLines(reader);  
 } catch (IOException ioe) {  
 throw new IllegalStateException("Can't reach file!", ioe);  
 }  
 }  
  
 public Path getMyFile() {  
 return myFile;  
 }  
  
 public List<Human> getHumans() {  
 return new ArrayList<>(humans);  
 }  
}\*/*

**SQL**

create schema if not exists activitytracker default character set utf8 collate utf8\_hungarian\_ci;

create user 'activitytracker'@'localhost' identified by 'activitytracker';

grant all on \*.\* to 'activitytracker'@'localhost';

CREATE TABLE activities (id BIGINT AUTO\_INCREMENT, activity\_desc VARCHAR(255),

activity\_type VARCHAR(255), PRIMARY KEY (id));;

INSERT INTO tábla (mező\_1,mező\_n) VALUES (v1,vn)

SELECT \* (mező AS álnév) FROM tábla WHERE mező LIKE ’%xyz%’ AND mező NOT LIKE ’xyz%’ ORDER BY mező DESC

Függvények: COUNT, SUM, AVG, MIN, MAX

SELECT \* FROM tábla1 AS e INNER/LEFT/RIGHT JOIN tábla2 AS k ON e.mező = k.mező

**JDBC**

public class ActivityTrackerMain {  
 public static void main(String[] args) {  
 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("Can not create data source", se);  
 }  
  
 Flyway flyway = Flyway.*configure*().dataSource(dataSource).load();  
 flyway.clean();  
 flyway.migrate(); // resources/db/migration/V0\_\_név.sql

// V0\_\_név.sql:

CREATE TABLE activities (id BIGINT AUTO\_INCREMENT,  
 start\_time TIMESTAMP,  
 activity\_desc VARCHAR(255),  
 activity\_type VARCHAR(255),  
 PRIMARY KEY (id));

// ARCHITEKTÚRA!

public class ActivityDao {  
 private DataSource dataSource;  
  
 public ActivityDao(DataSource dataSource) {  
 this.dataSource = dataSource;  
 }

// BESZÚRÁS

public void insertActivity(Activity activity) {  
 try(Connection conn = dataSource.getConnection();  
 PreparedStatement stmt = conn.prepareStatement("insert into activities(start\_time,activity\_desc,activity\_type) values (?,?,?)")) {  
 stmt.setTimestamp(1, Timestamp.*valueOf*(activity.getStartTime()));  
 stmt.setString(2, activity.getDesc());  
 stmt.setString(3, activity.getType().toString());  
 stmt.executeUpdate();  
 } catch (SQLException sqle) {  
 throw new IllegalStateException("Connection failed!", sqle);  
 }  
 }

// RESULTSET

private List<Activity> selectActivityByPreparedStatement(PreparedStatement stmt) {  
 List<Activity> result = new ArrayList<>();  
 try (ResultSet rs = stmt.executeQuery()) {  
 while(rs.next()) {  
 Activity activity = new Activity(  
 rs.getLong("id"),  
 rs.getTimestamp("start\_time").toLocalDateTime(),  
 rs.getString("activity\_desc"),  
 ActivityType.*valueOf*(rs.getString("activity\_type")));  
 result.add(activity);  
 }  
 return result;  
 } catch (SQLException sqle) {  
 throw new IllegalStateException("Cannot execute!", sqle);  
 }  
 }

// CONNECTION

public Activity selectActivityById(long id) {  
 try(Connection conn = dataSource.getConnection();  
 PreparedStatement stmt = conn.prepareStatement("select \* from activities where id =?")) {  
 stmt.setLong(1, id);  
 List<Activity> result = selectActivityByPreparedStatement(stmt); //RESULTSET  
 if(result.size()==1) {  
 return result.get(0);  
 }  
 throw new IllegalArgumentException("Not found!");  
 } catch (SQLException sqle) {  
 throw new IllegalStateException("Connection failed!", sqle);  
 }  
  
 }  
  
 public List<Activity> selectAllActivities() {  
 try(Connection conn = dataSource.getConnection();  
 PreparedStatement stmt = conn.prepareStatement("select \* from activities");) {  
 return selectActivityByPreparedStatement(stmt); //RESULTSET  
 } catch (SQLException sqle) {  
 throw new IllegalStateException("Connection failed!", sqle);  
 }  
 }