Medii de proiectare și programare

2024-2025
Curs 2

Conținut curs 2

- Gradle (cont. curs 1)
- Jurnalizare
- SQLite, SQLiteStudio, MySQL
- Accesul la baze de date relaționale
    - Java: JDBC
    - C#: ADO.NET - curs 3
- Configurarea (Java properties, C# app.config - curs 3)
- Ierarhia repository - curs 3

Gradle Build Tool

- Structura unui proiect Java:
  - *src/main/java* - directorul corespunzător codului sursă.
  - *src/main/resources* - directorul corespunzător resurselor folosite în proiect (fișiere de proprietăți, imagini, fxml etc.).
  - *src/test/java* - directorul corespunzător testelor automate.
  - *src/test/resources* - directorul corespunzător resurselor necesare testelor automate.
  - build - directorul ce conține toate artefactele construite folosind Gradle.
    - *classes* - conține fișierele .class.
    - *libs* - conține fișierele jar, war, ear create folosind Gradle.
    - etc.

Gradle Build Tool

- Crearea unui proiect Java

  În fișierul build.gradle:

apply plugin: ‘java'	*plugins{* *id ‘java’* *}*

- Crearea unui proiect Java cu structura implicită***:

  gradle init --type 'java-library'

  gradle init --type 'java-application'

***în directorul rădăcină al proiectului

Gradle Build Tool

- Sarcinile asociate unui proiect Java (plugin java):
  - *assemble* - compilează codul sursă al aplicației și creează fișierul jar. Nu rulează testele automate.
  - *build* - construiește toate artefactele asociate proiectului.
  - *clean* - șterge directorul build asociat proiectului.
  - *compileJava* - compilează codul sursă al aplicației.
  - etc.

Gradle Build Tool

- gradle tasks - afișează lista completă a sarcinilor ce pot fi executate pentru un proiect și descrierea acestora:
  - *assemble - Assembles the outputs of this project.*
  - *build - Assembles and tests this project.*
  - *buildDependents - Assembles and tests this project and all projects that depend on it.*
  - *buildNeeded - Assembles and tests this project and all projects it depends on.*
  - *classes - Assembles main classes.*
  - *clean - Deletes the build directory.*
  - *jar - Assembles a jar archive containing the main classes.*
  - *testClasses - Assembles test classes.*
  - *check - Runs all checks.*
  - *test - Runs the unit tests*.
  - etc.

Gradle Build Tool

- Împachetarea aplicației (obținerea artefactelor):
  - *gradle assemble*

    :compileJava

    :processResources

    :classes

    :jar

    :assemble

Gradle Build Tool

- Împachetarea aplicației (obținerea artefactelor și rularea testelor automate):
  - *gradle build*

    :compileJava

    :processResources

    :classes

    :jar

    :assemble

    :compileTestJava

    :processTestResources

    :testClasses

    :test

    :check

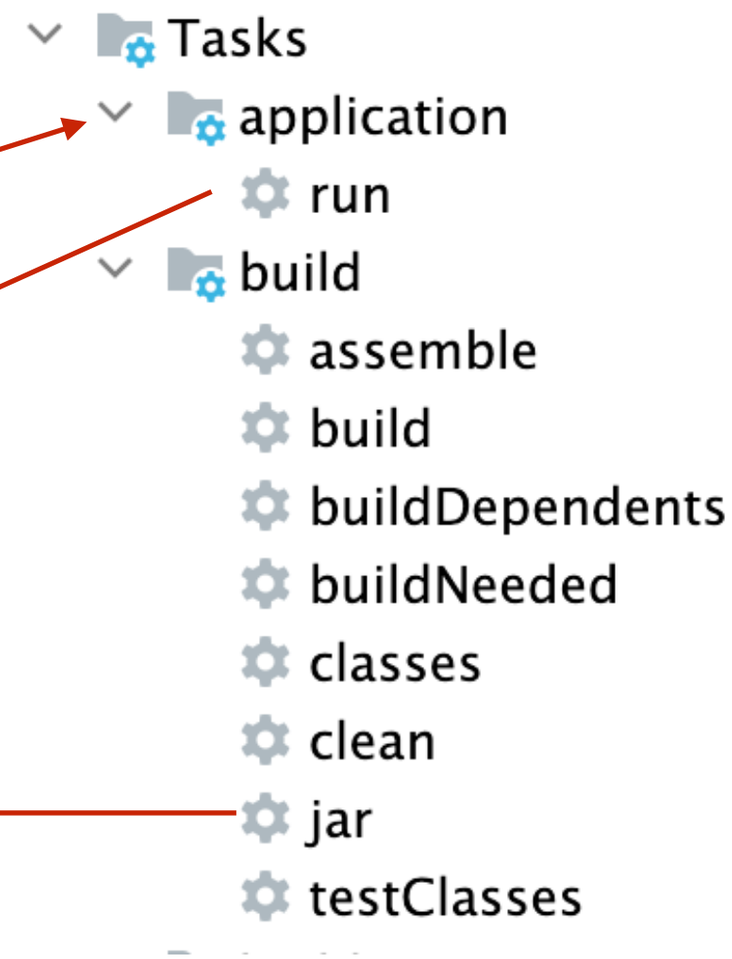
    :build

Gradle Build Tool

- Fișiere *jar executabile*:
  - *build.gradle*

```
plugins{
        id 'java'
        id 'application'
        }
application{
   mainClass='NumeClasaCuMain'
}
jar {
    manifest {
        attributes('Main-Class':'NumeClasaCuMain')
    }
    from {
        configurations.runtimeClasspath.collect { it.isDirectory() ? it : zipTree(it) }
    }
   duplicatesStrategy = DuplicatesStrategy.EXCLUDE

}
```



Gradle Build Tool

- Proiecte multiple:
  - Fiecare proiect (subproiect) are aceeași structură - corespunzătoare proiectelor Gradle Java.
  - Fiecare proiect (subproiect) va conține fișierul build.gradle propriu, cu configurările specifice.
  - Proiectul rădăcină (root) conține obligatoriu și fișierul settings.gradle:

```
include 'A'
include 'B'
```

Gradle Build Tool

- Dependențe între (sub)proiecte: Subproiectul B depinde de subproiectul A:
  - build.gradle corespunzător subproiectului B:

```
dependencies {
    implementation project(':A')
}
```

Gradle Build Tool

- Proiectul A: build.gradle

```
plugins{
    id 'java'
}
repositories {
    mavenCentral()
}
dependencies {
   implementation 'com.google.guava:guava:20.0'
    testImplementation 'junit:junit:4.11'
}
```

Gradle Build Tool

- Proiectul B: build.gradle

```
plugins{
    id ‘java'
    id ‘application'
}
repositories {
    mavenCentral()
}
application{
    mainClass=‘StartApp’
}
dependencies {
    testImplementation ‘junit:junit:4.11'
    implementation project(‘:A')
}
```

Gradle Build Tool

- Proiectul Root: build.gradle

```
allprojects {
        plugins{
            id ‘java’
        }
        repositories {
            mavenCentral()
        }
        dependencies {
            testImplementation ‘junit:junit:4.11’
        }
}
```

Gradle Build Tool

- Proiectul A: build.gradle modificat

```
dependencies {
    implementation 'com.google.guava:guava:20.0'
    }
```

- Proiectul B: build.gradle modificat

```
plugins{
    id ‘application’
}
application{
    mainClass=‘StartApp’
}
dependencies {
      implementation project(‘:A')
}
```

Gradle Build Tool

- Proiectul Root: build.gradle

```
subprojects {
    //Configurări comune tuturor subproiectelor
}
project(':A') {
    //Configurări specifice proiectului A
}

project(':B') {
    //Configurări specifice proiectului B
}
```

Instrumente pentru jurnalizare

- Un ***instrument pentru jurnalizare*** permite programatorilor să înregistreze diferite tipuri de mesaje din codul sursă cu diverse scopuri: depanare, analiza ulterioară, etc.

- Majoritatea instrumentelor definesc diferite nivele pentru mesaje: *debug*, *warning*, *error*, *information*, *sever*, etc.

- Configurarea instrumentelor se face folosind fișiere text de configurare și pot fi oprite sau pornite la rulare.

- Apache Log4j, Logging SDK (inclus JDK), slf4j, etc.

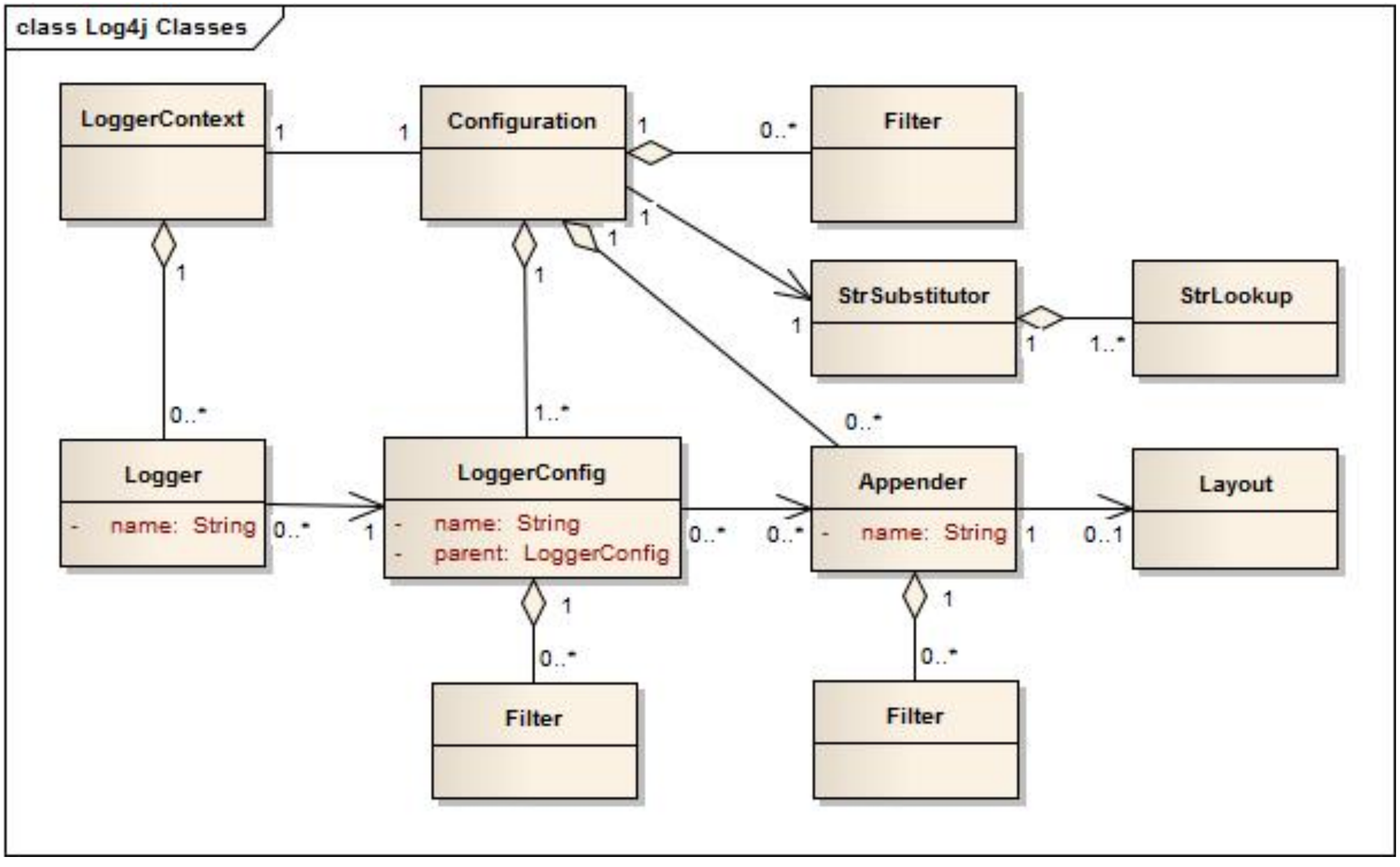
- Proiect open source dezvoltat de Apache Foundation.

  http://logging.apache.org/log4j/2.0/

- Log4j 2 are 3 componente principale:
  - *loggers*,
  - *appenders* (pentru stocare),
  - *layouts* (pentru formatare).

https://logging.apache.org/log4j/2.x/index.html

Arhitectura



- Aplicațiile care folosesc Log4j 2 cer o referință către un obiect de tip *Logger* cu un anumit nume de la *LogManager*.
- LogManager va localiza obiectul LoggerContext corespunzător numelui și va obține referința către obiectul Logger de la el.
- Dacă obiectul de tip Logger corespunzător încă nu a fost creat, se va crea unul nou și va fi asociat cu un obiect de tip LoggerConfig care fie:
  - are același nume ca și Logger,
  - are același nume ca și pachetul părinte,
  - este rădăcina LoggerConfig.
- Obiectele de tip LoggerConfig sunt create folosind declarațiile din fișierul de configurare.
- Fiecărui LoggerConfig îi sunt asociate unul sau mai multe obiecte de tip Appender.

- Fișierul de configurare (XML, JSON, proprietăți Java, yaml)
- Dacă nu este configurat, log4j 2 afișează doar mesajele de tip *error* la consolă

Exemplu fișier de configurare în format XML: ***log4j2.xml***

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="TRACE">
  <Appenders>
    <Console name="Console" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
    </Console>
  </Appenders>
  <Loggers>
    <Root level="TRACE">
      <AppenderRef ref="Console"/>
    </Root>
  </Loggers>
</Configuration>
```

- Log level - fiecare mesaj are asociat un anumit nivel
  - TRACE, DEBUG, INFO, WARN, ERROR și FATAL

Event Level	LoggerConfig Level						
	TRACE	DEBUG	INFO	WARN	ERROR	FATAL	OFF
ALL	YES	YES	YES	YES	YES	YES	NO
TRACE	YES	NO	NO	NO	NO	NO	NO
DEBUG	YES	YES	NO	NO	NO	NO	NO
INFO	YES	YES	YES	NO	NO	NO	NO
WARN	YES	YES	YES	YES	NO	NO	NO
ERROR	YES	YES	YES	YES	YES	NO	NO
FATAL	YES	YES	YES	YES	YES	YES	NO
OFF	NO	NO	NO	NO	NO	NO	NO

- Numele asociat unui logger: structura ierarhică asemănătoare structurii pachetelor Java.

```
public class LogTest {
//a
private static final Logger logger = LogManager.getLogger(LogTest.class);
 //b
  private static final Logger logger =
                                  LogManager.getLogger(LogTest.class.getName());
//c
private static final Logger logger = LogManager.getLogger();
}
```

- Clasa Logger conține metode ce permit urmărirea fluxului execuției unei aplicații.
  - entry(…) - 0 ..4 parametrii
  - traceEntry(String, …) - String și o lista variabilă de parametri
  - exit(…), traceExit(String, …)
  - throwing (…) - când se aruncă o excepție
  - catching(…) când se prinde o excepție
  - trace(…)
  - error(…)
  - log(…)
  - etc.
- **Exemplu**

- Salvarea mesajelor: fișier, consolă, baze de date, etc.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="TRACE">
  <Appenders>
    <File name="FisierLog" fileName="logs/app.log">
      <PatternLayout pattern="%d{DATE} [%t] %class{36} %L %M - %msg%xEx%n"/>
    </File>
  </Appenders>
  <Loggers>
    <Root level="TRACE">
      <AppenderRef ref="FisierLog"/>
    </Root>
  </Loggers>
</Configuration>
```

SGBD

- Sqlite
    - SQLiteStudio
- MySQL/MariaDB

- https://www.sqlite.org/
- Bază de date relațională
- Nu necesită configurări adiționale
- Nu necesită pornirea unui proces separat
- Toate informațiile sunt păstrate într-un singur fișier
- Formatul fișierului este independent de platformă
- Open source, gratuit.

```
sqllite_dir> sqlite3
```

- https://sqlitestudio.pl/
- Sistem de gestiune a unei baze de date Sqlite
- Interfață grafică ușor de folosit
- Independent de platformă
- Gratuit
- Open source

- https://www.mysql.com/ https://mariadb.com/
- Sistem de gestiune a bazelor de date relaționale
- Rapid, scalabil, ușor de folosit
- Sistem de tip client-server/ embedded
- Gratuit
- Open source (MariaDB)

JDBC

- Java Database Connectivity (JDBC) API conține o mulțime de clase ce asigură accesul la date.
- Se pot accesa orice surse de date: baze de date relaționale, foi de calcul (*spreadsheets*) sau fișiere.
- JDBC oferă și o serie de interfețe ce pot fi folosite pentru construirea instrumentelor specializate.
- Pachete:
  - **`java.sql`** conține clase și interfețe pentru accesarea și procesarea datelor stocate într-o sursă de date (de obicei bază de date relațională).
  - **`javax.sql`** - adaugă noi funcționalități pentru partea de server.

Stabilirea unei conexiuni

- Conectarea se poate face în două moduri:
  - Clasa **`DriverManager`**: Conexiunea se creează folosind un URL specific.
    - Necesita încărcarea unui driver specific bazei de date (JDBC<4).
    - Incepând cu JDBC 4.0 nu mai este necesară încărcarea driverului.
  - Interfața **`DataSource`**: Este recomandată folosirea interfeței pentru aplicații complexe, deoarece permite configurarea sursei de date într-un mod transparent.
- Stabilirea unei conexiuni se realizează astfel:
  - Încărcarea driverului (versiuni JDBC <4.0)

```
Class.forName(<DriverClassName>);
```

    - **`Class.forName`** creează automat o instanță a driverului și o înregistrează la **`DriverManager.`**
    - Nu este necesară crearea unei instanțe a clasei.
  - Crearea conexiunii.

Crearea unei conexiuni

- Folosind clasa **`DriverManager`** :
  - Colaborează cu interfața Driver pentru gestiunea driverelor disponibile unui client JDBC.
  - Când clientul cere o conexiune și furnizează un URL, clasa DriverManager este responsabilă cu găsirea driverului care recunoaște URL și cu folosirea lui pentru a se conecta la sursa de date.
  - Sintaxa URL-ului corespunzător unei conexiuni este:

```
jdbc:subprotocol:<numeBazaDate>[listaProprietati]
```

```
Connection conn = DriverManager.getConnection("jdbc:sqlite:users.db");

String url = "jdbc:mysql:Test";

String url = "jdbc:mariadb://localhost:3306/Test"

Connection conn = DriverManager.getConnection(url, <user>, <passwd>);
```

Crearea unei conexiuni

- Folosind interfața **`DataSource`**:

```
InitialContext ic = new InitialContext()
 //a)
 DataSource ds = ic.lookup("java:comp/env/jdbc/myDB");
 Connection con = ds.getConnection();
//b)
 DataSource ds =
    (DataSource)org.apache.derby.jdbc.ClientDataSource()
 ds.setPort(1527);
 ds.setHost("localhost");
 ds.setUser("APP")
 ds.setPassword("APP");

Connection con = ds.getConnection();
```

Clasa Connection

- Reprezintă o sesiune cu o bază de date specifică.
- Orice instrucțiune SQL este executată și rezultatele sunt transmise folosind contextul unei conexiuni.
- Metode:
  - `close(), isClosed():boolean`
  - `createStatement(…):Statement  //overloaded`
  - `prepareCall(…):CallableStatement //overloaded`
  - `prepareStatement(…):PreparedStatement //overloaded`
  - `rollback()`
  - `setAutoCommit(boolean)   //tranzactii`
  - `getAutoCommit():boolean`
  - `commit()`