Medii de proiectare și programare

2019-2020
Curs 1

Conținut

- Inversion of Control - Spring
- Aplicații client-server - Șablonul de proiectare Proxy
- Apelul metodelor la distanță (Remote Procedure Call) - Remoting/WCF, RMI, Spring Remoting
- Object Relational Mapping (Strategii, Hibernate, Entity Framework)
- Enterprise Application Integration - Protocol buffers, Thrift, ActiveMQ, RabbitMQ
- REST Services
- Dezvoltarea aplicațiilor web folosind frameworkuri (React)
- Web sockets
- Securitate Web - Acces bazat pe roluri

Bibliografie

- Joseph Albahari, Ben Albahari, *C# 6.0 in a Nutshell*, Sixth Edition, O'Reilley, 2015.
- Craig Larman, *Applying UML and Design Patterns: An Introduction to OO Analysis and Design and Unified Process*, Berlin, Prentice Hall, 2002.
- Martin Fowler, *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2002.
- Hohpe, G., Woolf, B., *Enterprise integration patterns*, Addison-Wesley, 2003.
- ***, Microsoft Developer Network, Microsoft Inc., http://msdn.microsoft.com/
- ***, The Java Tutorials, Oracle Java Documentation, Inc. https://docs.oracle.com/javase/tutorial/
- Craig Walls, *Spring in Action*, 4th Edition, Ed. O'Reilley, 2015.
- Documentație Spring http://spring.io/projects
- Alte tutoriale

Evaluare

- Întrebări în timpul cursului (QC) - 10%
- Examen scris și practic (NE) - 50%
- Teme în timpul laboratorului (TL) - 10%
- Laboratoare (LB) - 30 %
- Nota finala NF=0.1*QC+0.5*NE+0.1*TL+0.3*LB
- Condiții pentru promovare:
  - NE>=5, LB>=4.5
  - NF>=5

Laborator

- Asignarea și proiectarea aplicației. Studenții trebuie să proiecteze și să dezvolte o aplicație client-server.
- Configurarea aplicației folosind Gradle, IoC.
- Proiectarea și implementarea persistenței (baze de date relaționale, ORM)
- Proiectarea și implementarea serviciilor (Șablonul Proxy).
- RMI/ Remoting/WCF
- Enterprise Application Integration (Protobuff/Thrift/gRPC)
- Servicii REST
- Aplicații web.

Notare laborator

- Fiecare temă de laborator are un termen de predare.
- Predare completă a temei la termen: nota 10.
- Pentru fiecare saptămână întârziere în care nu s-a predat nimic din tema: penalizare 3 puncte/săpt.
- Pentru fiecare saptămână întârziere în care s-a predat ceva (este incompletă, erori): penalizare 1 punct/săpt, pâna la predarea completă a temei.
- Tema nepredată: nota 0.

Curs 1

- Instrumente pentru construire automată
  - Gradle
- Instrumente pentru jurnalizare
  - Log4j 2

Instrumente pentru construire automată*

- Procesul automatizat corespunzător construirii unui sistem soft și a proceselor asociate (compilarea codului sursă, rularea testelor automate, împachetarea codului binar, etc).
  - Makefile
- Două categorii de instrumente:
  - Utilitare pentru construire automată:
    - Generează artifactele corespunzătoare construirii în timpul compilării și/sau link-editării codului.
    - Exemple: Make, MS build, Ant, Gradle, Maven etc.
  - Servere pentru construire automată:
    - Sisteme soft care execută utilitarele de construire automată la perioade de timp predefinite sau în momentul apariției anumitor evenimente.
    - Exemple: TeamCity, Jenkins, CruiseControl, etc.

*Eng. *Build automation tools*

Utilitare pentru construire automată

- Permit automatizarea sarcinilor simple și repetabile.
- Utilitarul permite atingerea unui scop/obiectiv prin execuția fiecărei sarcini dintr-un set de sarcini specifice în ordinea corectă.
- Avantaje:
  - Îmbunătățesc calitatea produselor
  - Accelerează compilarea și link-editarea
  - Elimină sarcinile redundante
  - Elimină dependențele de angajați ‘*cheie*’
  - Oferă un jurnal al construcțiilor ce pot fi folosite când apar probleme
  - Reduce numărul construcțiilor eșuate
  - Economisesc timp și bani

- Gradle Build Tool, https://gradle.org/
- Tutoriale:
  - http://www.vogella.com/tutorials/Gradle/article.html#introduction-to-the-gradle-build-system
  - https://www.petrikainulainen.net/getting-started-with-gradle/

Gradle Build Tool

- Gradle este un utilitar ce permite gestiunea avansată a construirii aplicațiilor.

- Oferă suport pentru configurarea și descărcarea automată a bibliotecilor și a altor dependențe.

- Oferă suport pentru depozitarele (eng. *repositories*) Maven și Ivy pentru refacerea dependențelor.

- Permite refolosirea artifactelor construite folosind alte utilitare.

- Oferă suport pentru construiri multi-proiect și multi-artifacte.

Gradle Build Tool

- Gradle folosește două concepte de bază: proiecte (eng. *project*) și sarcini (eng. *tasks*).
- Un *proiect* este:

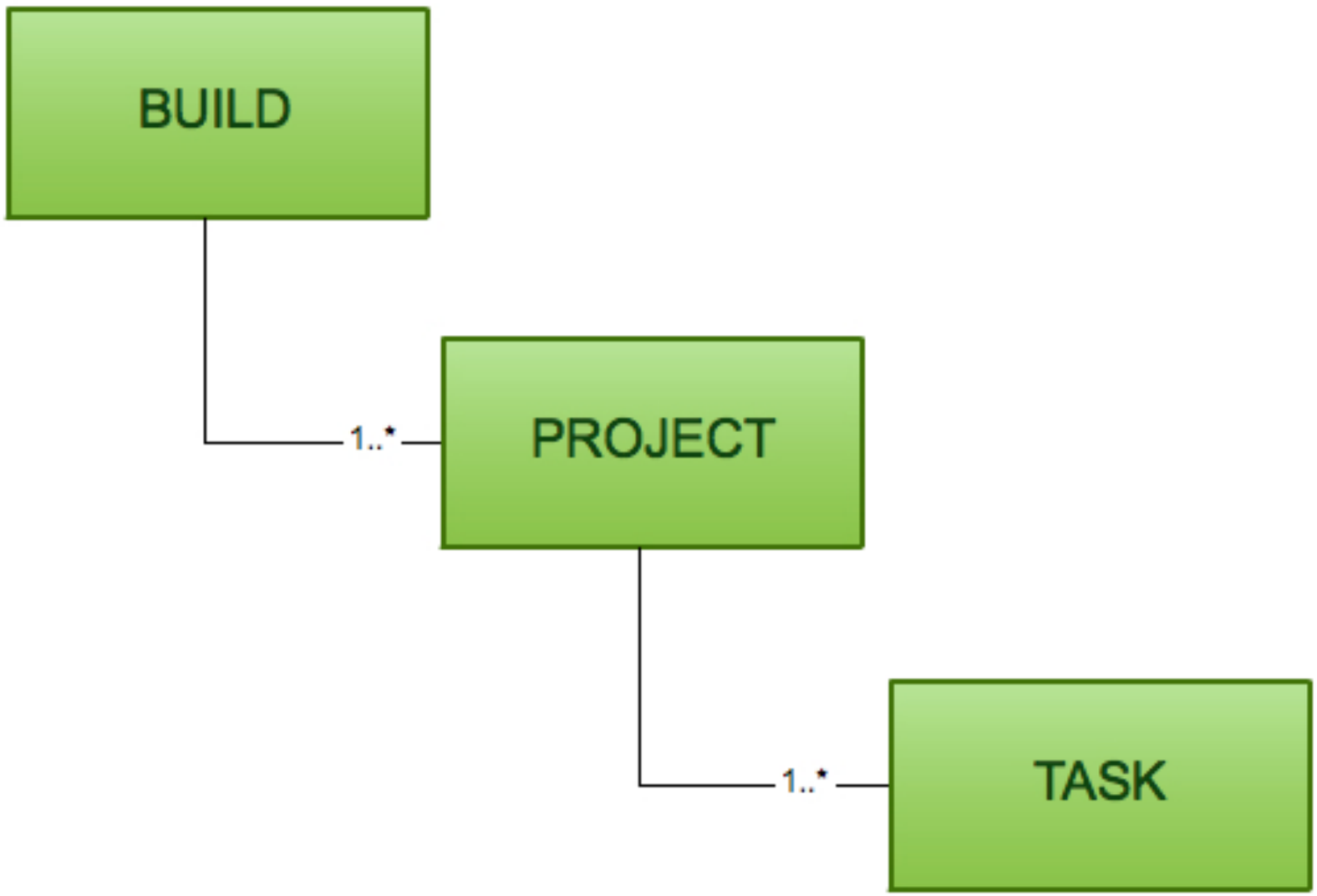
  a. ceva ce se dorește a se construi (ex. un fișier jar),

  b. ceva ce se dorește a se face (instalarea unui aplicatii pentru folosirea de către utilizatori).

- Un proiect are asociate una sau mai multe sarcini.
- O sarcină este o unitate de lucru care este efectuată pentru construirea automată:
  - compilarea unui proiect
  - rularea testelor automate

Gradle Build Tool

- Fiecare construire Gradle conține unul sau mai multe proiecte.



Gradle Build Tool

- O construire Gradle se configurează folosind fișierele:
  - build.gradle (*Gradle build script*) - specifică un proiect și sarcinile sale asociate.
  - gradle.properties (*Gradle properties file*) - este folosit pentru configurarea proprietăților construirii.
- settings.gradle (*Gradle Settings file*) - opțional la construirile ce conțin un singur proiect.
  - Dacă construirea este formată din mai multe proiecte Gradle, fișierul este obligatoriu și descrie proiectele conținute.
  - Fiecare construire multi-proiect trebuie să aibă un fișier settings.gradle în directorul rădăcină al proiectului.

Gradle Build Tool

- build.gradle

```
apply plugin: 'java'
repositories {
//pentru rezolvarea dependențelor: Maven, JCenter, Ivy
      mavenCentral()
       //jcenter()
}
dependencies {
    //format GAV  Grup:Artefact:Versiune
    compile 'com.google.guava:guava:20.0'
    testCompile group: 'junit', name: 'junit', version: ‘4.+'
    //alte dependențe
 }
```

Gradle Build Tool

- settings.gradle

```
rootProject.name = 'NumeProiect'
```

Gradle Build Tool

- Plugin-uri Gradle
  - Toate caracteristicile/facilitățile utile unui proiect sunt furnizate de pluginuri.
  - Un plugin Gradle poate să:
    - adauge sarcini noi la proiect.
    - ofere o configurație implicită la sarcinile adăugate. Configurația implicită poate să adauge noi convenții proiectului (ex. locația codului sursă).
    - adauge noi proprietăți care sunt folosite pentru a redefini configurația implicită a proiectului.
    - adauge noi dependențe proiectului.

Gradle Build Tool

- Un plugin poate fi adăugat unui proiect folosind numele sau tipul acestuia:

- Exemplu folosind numele:

  În fișierul build.gradle:

  ```
  apply plugin: 'foo'
  ```

- Exemplu folosind tipul:

  În fișierul build.gradle:

  ```
  apply plugin: 'com.bar.foo'
  ```

- Blocul plugins (versiuni mai noi Gradle):

```
plugins {
    id 'java'
    id 'application'
}
```

- Se poate adauga si versiunea unui plugin

```
id "xyz" version "1.0.0"
```

Gradle Build Tool

- Pluginuri Gradle standard:
  - java: adaugă sarcini pentru compilarea, testarea și împachetarea unui proiect Java. Este un plugin de bază pentru alte pluginuri.
  - groovy: adaugă suport pentru proiecte Groovy.
  - cpp: adaugă sarcini pentru compilarea codului unui proiect C++.
  - application: adaugă sarcini pentru rularea și împachetarea unui proiect Java ca și aplicație executabilă.
  - distribution: adaugă suport pentru construirea distribuțiilor de tip ZIP și TAR.
  - signing: adaugă abilitatea de a semna digital fișierele și artefactelor construite.
  - etc.

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, 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'

- 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 crează 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*

```
apply plugin: 'java'
apply plugin: ‘application'
mainClassName=‘ClasaCuMain'

jar {
    manifest {
        attributes('Main-Class':'ClasaCuMain')
    }

    from {
        configurations.compile.collect { it.isDirectory() ? it : zipTree(it) }
    }
}
```

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 {
    compile project(':A')
}
```

Gradle Build Tool

- Proiectul A: build.gradle

```
apply plugin: 'java'
repositories {
    mavenCentral()
}
dependencies {
   compile 'com.google.guava:guava:20.0'
    testCompile 'junit:junit:4.11'
}
```

Gradle Build Tool

- Proiectul B: build.gradle

```
apply plugin: 'application'
apply plugin: 'java'
repositories {
    mavenCentral()
}
mainClassName=‘StartApp’
dependencies {
    testCompile ‘junit:junit:4.11'
    compile project(‘:A')
}
```

Gradle Build Tool

- Proiectul Root: build.gradle

```
allprojects {
        apply plugin: 'java'
        repositories {
            mavenCentral()
        }
        dependencies {
            testCompile ‘junit:junit:4.11’
        }
}
```

Gradle Build Tool

- Proiectul A: build.gradle modificat

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

- Proiectul B: build.gradle modificat

```
apply plugin: 'application'
mainClassName=‘StartApp’
dependencies {
        compile project(‘:A')
}
```

Gradle Build Tool

- Proiectul Root: build.gradle

```
subprojects {
    //Configurări specifice 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.

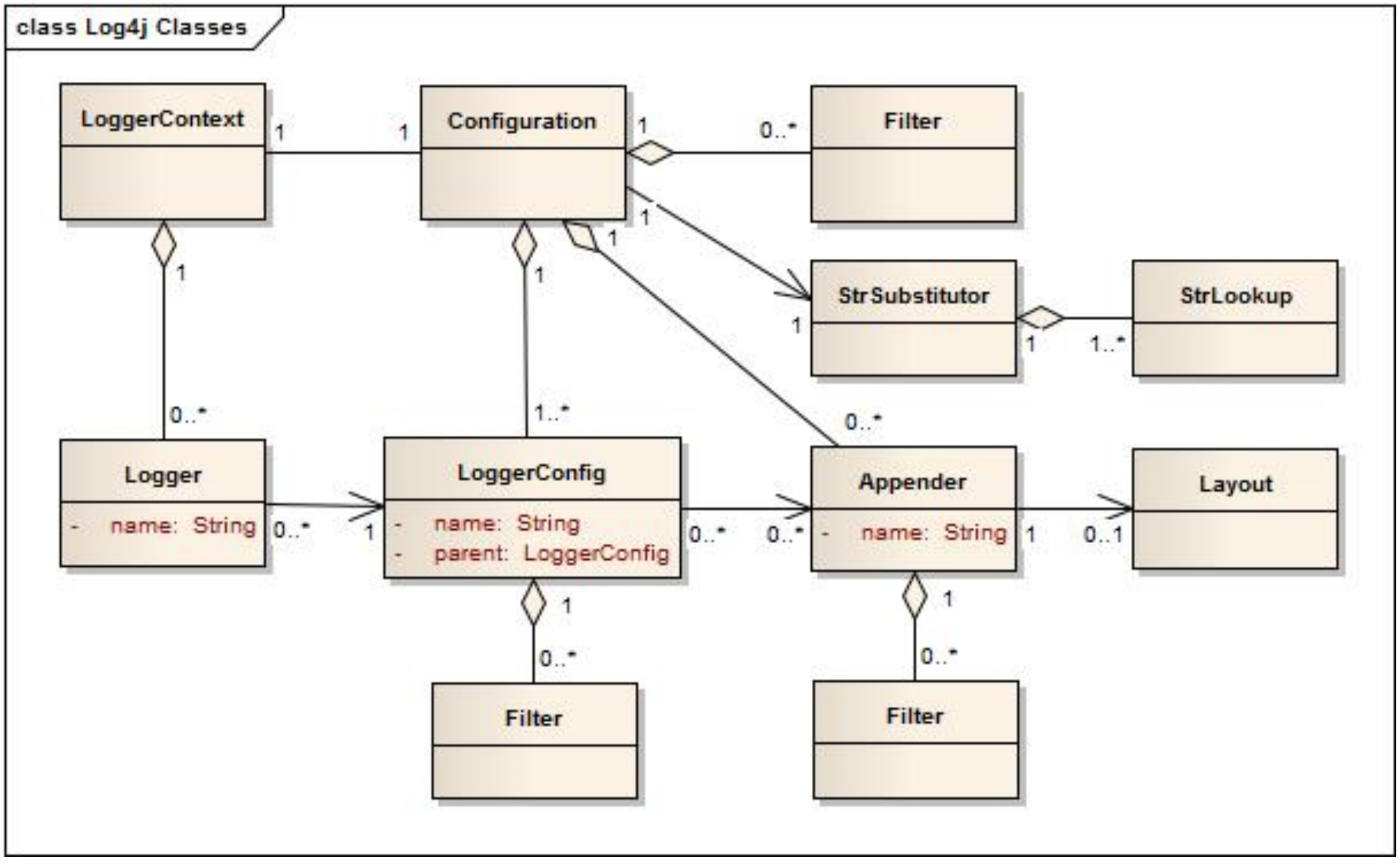
- Apache Log4j, Logging SDK, slf4j, etc.

- 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).

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 acelasi nume ca și pachetul părinte,
  - este rădăcina LoggerConfig.
- Obiectele de tip LoggerConfig sunt create folosind declarațiile din fișierul de configurare.
- Fiecărui LoggerConfig îi este asociat 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>
```