Keretrendszer alapú programozás

Fazekas Csaba
fazekas.csaba@uni-eszterhazy.hu

Build Automatizáció

Build folyamat

Mi minden mondható el egy Java projektről:

- meghatározott könyvtár szerkezettel rendelkezik,
- tartalmaz Java forráskódot,
- a forráskódhoz szükség lehet külső könyvtárakra (függőség vagy dependency)
- tartalmazhat addicionális erőforrásokat
    - XML, JSON adatok,
    - grafikák, hangfájlok.

Mindezekre mint arvifact-okra (műalkotásokra) szoktak hivatkozni.

Build folyamat

Ahhoz, hogy futtatható alkalmazásunk legyen szükség lesz a következőkre:

- el kell készíteni a JVM által futtatható byte kódot,
- összecsomagolni mindent egy futtatható állományba,
- tesztek futtatása,
- dokumentációt készíteni például JavaDoc formátumban.

Ezeket fogják össze a build eszközök.

Java bájt kód

```
outer:
for (int i = 2; i < 1000; i++) {
  for (int j = 2; j < i; j++) {
    if (i % j == 0)
      continue outer;
  }
  System.out.println (i);
}
```

```
0: iconst_2
1: istore_1
2: iload_1
3: sipush 1000
6: if_icmpge 44
9: iconst_2
10: istore_2
11: iload_2
12: iload_1
13: if_icmpge 31
16: iload_1
17: iload_2
18: irem
19: ifne 25
22: goto 38
25: iinc 2, 1
28: goto 11
31: getstatic
#84; //Field java/
lang/
System.out:Ljava/
io/PrintStream;
34: iload_1
35: invokevirtual
#85; //Method java/
io/
PrintStream.println
:(I)V
38: iinc 1, 1
41: goto 2
44: return
```

Build eszközök fejlődése

Make

- 1976-ban hozta létre Stuart Feldman a Bell Labs munkatársa.
  (Későbbiekben több helyen is dolgozott, például az IBM-nél és a Google-nél is mint Vice President Engineering, East Coast).

    - 2003-ban ACM Software System Award díjat kapott érte. (Association for Computing Machinery)
      https://en.wikipedia.org/wiki/ACM_Software_System_Award

- Egy olyan probléma szülte, hogy hibát kerestek egy olyan programban aminek a forrásásban a hiba javítva lett, de nem lett a kód lefordítva.

- Futtatható programok és lib-ek készítésére hozták létre.

- Makefile vezérli.

- Unix rendszerekben mind a mai napig használatos.

Makefile

- Make az aktuális könyvtárban keres egy makefile-t.
- Ez sima szövegfájl.
- Érzékeli a projekt minden fájlján a változást, tehát nem csak forrásokon.
- Képes a fordításon kívül másra is:
  - kép átkonvertálása más formátumra,
  - e-mail küldése a feladat végén vagy hiba esetén …

https://en.wikipedia.org/wiki/Make_(software)#Other_elements

Make

- A fájlok módosítási időpontja alapján dönti el mit kell fordítani és mit nem.

- Ha valami módosult és arra épül más fájl, akkor azt is lefordítja.

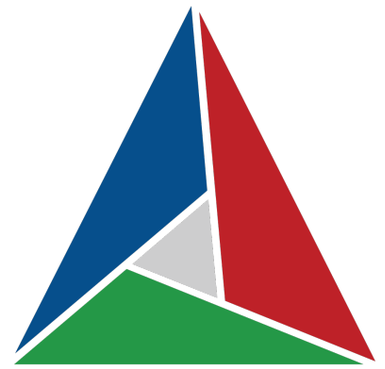
- Ahol nincs változtatás ahhoz nem nyúl.

Makefile példa

```
# Makefile for Writing Make Files Example

# *****************************************************
# Variables to control Makefile operation

CC = g++
CFLAGS = -Wall -g

# *****************************************************
# Targets needed to bring the executable up to date

main: main.o Point.o Square.o
    $(CC) $(CFLAGS) -o main main.o Point.o Square.o

# The main.o target can be written more simply

main.o: main.cpp Point.h Square.h
    $(CC) $(CFLAGS) -c main.cpp

Point.o: Point.h

Square.o: Square.h Point.h
```

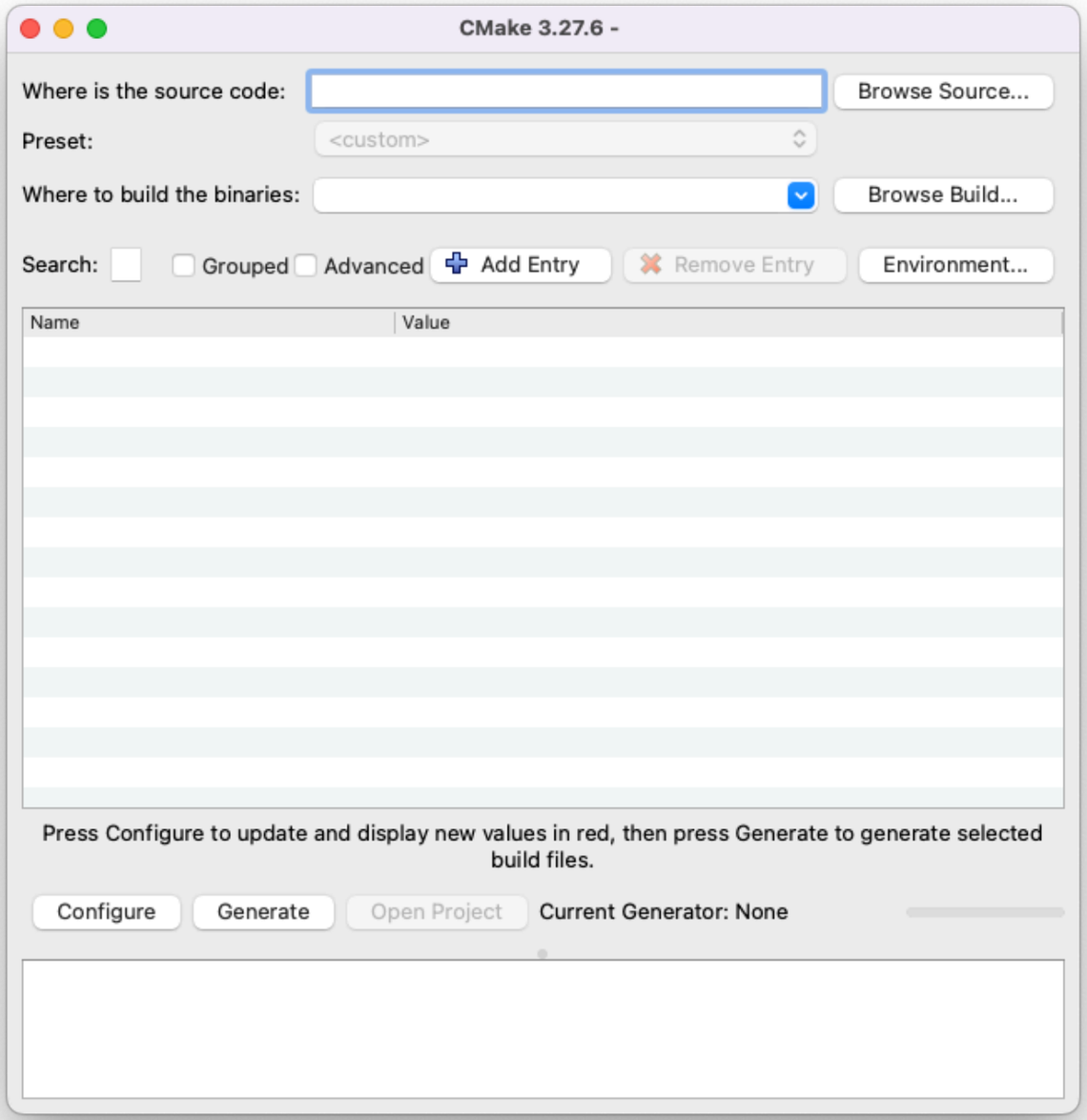
https://www.softwaretestinghelp.com/cpp-makefile-tutorial/#C_Makefile

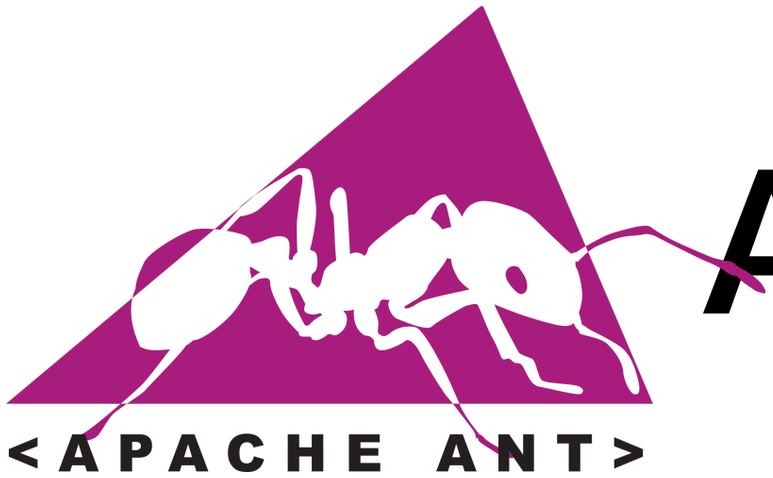
CMake



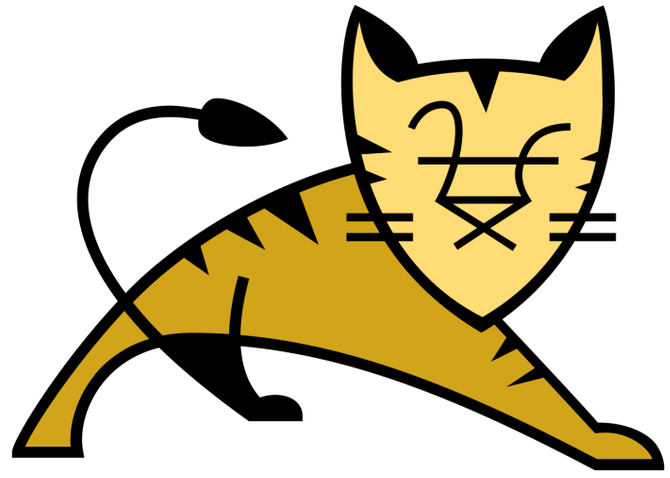
- Segítségünkre lehet a Makefile-ok szerkesztésében.

https://cmake.org/



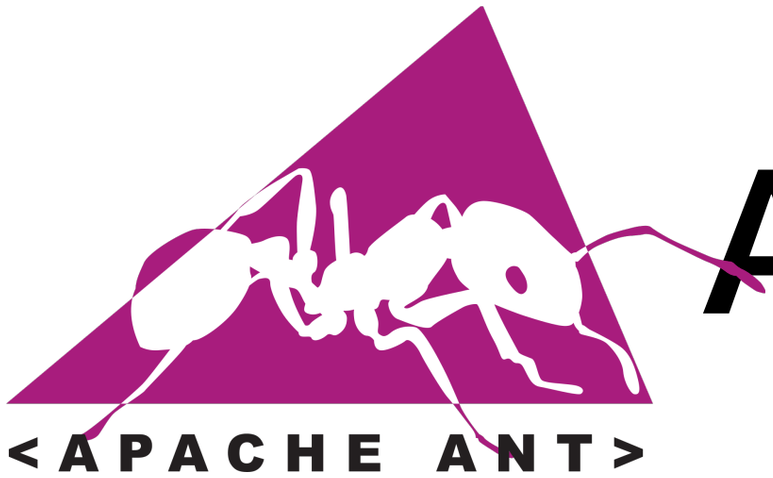


Apache Ant



- James Duncan Davidson fejlesztette ki az Apache Tomcat szerverrel együtt a Sun Microsystems -nél.
- 1999-ben indul el a Tomcat.
- A Make kiváltására hozták létre.
- Java nyelvhez készült és szükséges hozzá a Java platform.

https://en.wikipedia.org/wiki/Apache_Ant



Apache Ant

- Az volt a cél vele, hogy a Tomcat-et tetszőleges rendszeren le lehessen fordítani.

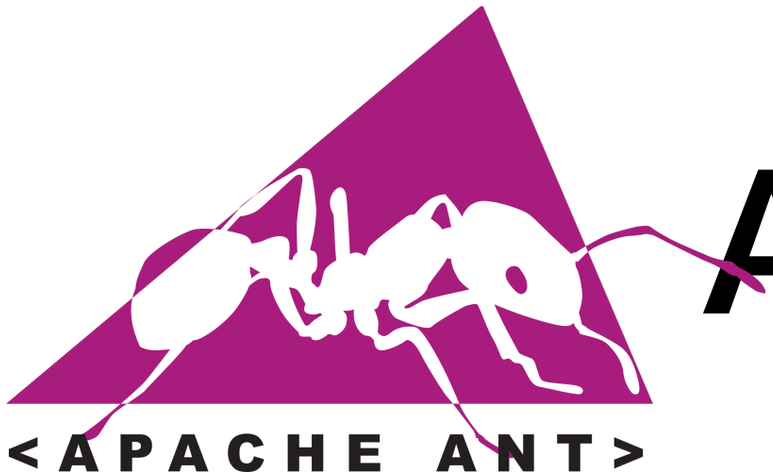
- XML leíró fájlokat használ.

- 2000 július 19-én indult el önálló termékként.

- 2002-től Java projektek build eszközévé vált. (Open source projektekben megjelent a build.xml fájl.)

- JUnit tesztek futtatását is segítette, így az Extrém Programozás és Test Driven Development terjedését segítette.

https://en.wikipedia.org/wiki/Apache_Ant



Apache Ant

```xml
<?xml version="1.0"?>
<project name="Hello" default="compile">
    <target name="clean" description="remove intermediate files">
        <delete dir="classes"/>
    </target>
    <target name="clobber" depends="clean" description="remove all artifact files">
        <delete file="hello.jar"/>
    </target>
    <target name="compile" description="compile the Java source code to class files">
        <mkdir dir="classes"/>
        <javac srcdir="." destdir="classes"/>
    </target>
    <target name="jar" depends="compile" description="create a Jar file for the application">
        <jar destfile="hello.jar">
            <fileset dir="classes" includes="**/*.class"/>
            <manifest>
                <attribute name="Main-Class" value="HelloProgram"/>
            </manifest>
        </jar>
    </target>
</project>
```

https://en.wikipedia.org/wiki/Apache_Ant

Apache Maven™

- 2002-ben az Apache Turbine framework részeként kezdte fejleszteni Jason van Zy.

  turbine

  - Apache Turbine™ is a servlet based framework that allows experienced Java developers to quickly build web applications. Turbine allows you to use personalize the web sites and to use user logins to restrict access to parts of your application.
    (https://turbine.apache.org/)

- Java mellett C#, Ruby, Scala, (plugin segítségével C/C++) és más nyelveket is támogat.

- Újdonság, hogy a build folyamat megállapodásokra (konvenciókra) alapul és csak az ehhez képesti eltéréseket kell definiálni. (Convention over configuration)

https://en.wikipedia.org/wiki/Apache_Maven

Convention over configuration

“Convention over configuration (also known as coding by convention) is a software design paradigm used by software frameworks that attempts to decrease the number of decisions that a developer using the framework is required to make without necessarily losing flexibility and don't repeat yourself (DRY) principles.[1]

The concept was introduced by David Heinemeier Hansson to describe the philosophy of the Ruby on Rails web framework, but is related to earlier ideas like the concept of "sensible defaults" and the principle of least astonishment in user interface design.

The phrase essentially means **a developer only needs to specify unconventional aspects of the application.** For example, if there is a class Sales in the model, the corresponding table in the database is called "sales" by default**.** It is only if one deviates from this convention, such as the table "product sales", that one needs to write code regarding these names**.**

When the convention implemented by the tool matches the desired behavior, it behaves as expected without having to write configuration files**. Only when the desired behavior deviates from the implemented convention is explicit configuration required.**”

(Wikipedia)

https://en.wikipedia.org/wiki/Convention_over_configuration

Apache Maven™

- XML konfigurációs fájlja van, ami a következőket tartalmazza:
  - projekt leírása
  - függőségek
  - külső modulok
  - fordítási sorrend
  - esetleges szükséges plugin-ok.

https://en.wikipedia.org/wiki/Apache_Maven

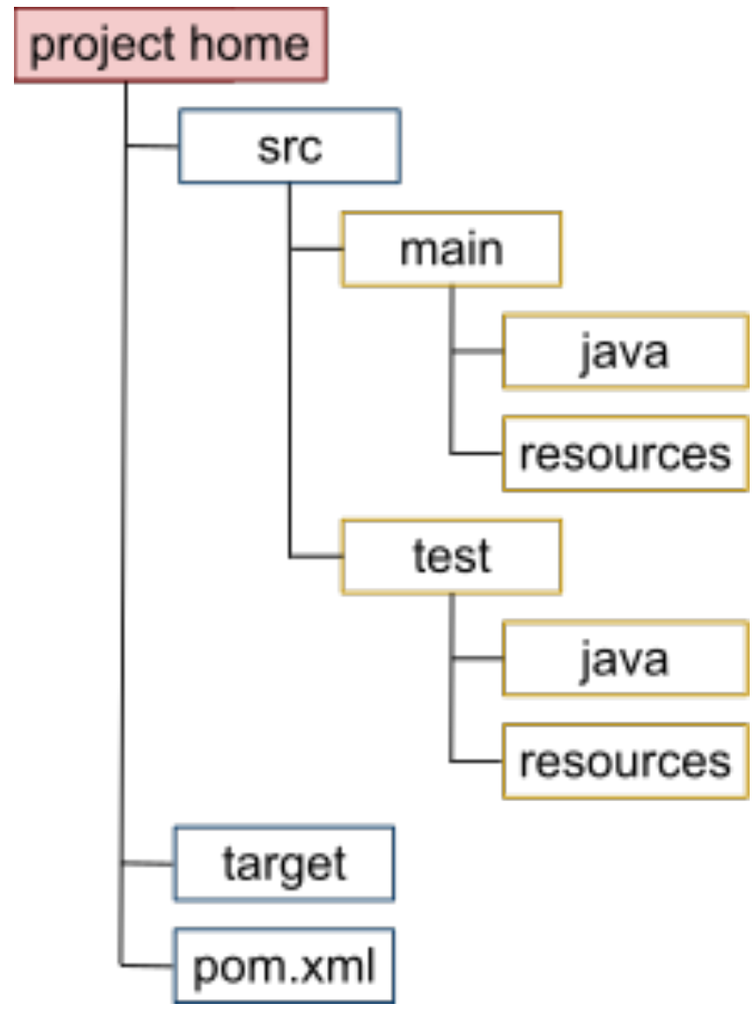
Apache Maven™

- Repository-kból automatikusan tölt le
  - Java forrásokat,
  - plugin-okat.
- Maven 2 Central -> remote repository
- Lokális repozitorit is tud használni.

https://en.wikipedia.org/wiki/Apache_Maven

Apache Maven™

Maven project



https://en.wikipedia.org/wiki/Apache_Maven

Apache Maven™

Project Object Model (POM) XML fájl

```xml
<project>
  <!-- model version is always 4.0.0 for Maven 2.x POMs -->
  <modelVersion>4.0.0</modelVersion>
  <!-- project coordinates, i.e. a group of values which uniquely identify this project -->
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0</version>
  <!-- library dependencies -->
  <dependencies>
    <dependency>
      <!-- coordinates of the required library -->
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <!-- this dependency is only used for running and compiling tests -->
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

https://en.wikipedia.org/wiki/Apache_Maven

Apache Maven™

- Függőség kezelés:
  - Ha a POM fájlban megadunk egy függőséget, akkor a Maven letölti azt automatikusan.
  - Letölti a transitive dependency-ket is, tehát a függőségek függőségeit.
  - Keresők segítenek a függőségek megtalálásában.

https://en.wikipedia.org/wiki/Apache_Maven

Apache Maven™

- Build lifecycle:
  - Goal-okkal határozzuk meg a tennivalók céljait.
  - Fordítási fázisok sorolhatóak fel a célok elkészítésére.
  - Maven mindig az adott Goal-hoz szükséges fázisokat futtatja.

Default lifecycle:

- validate
- generate-sources
- process-sources
- generate-resources
- process-resources
- compile
- process-test-sources
- process-test-resources
- test-compile
- test
- package
- install
- deploy

https://en.wikipedia.org/wiki/Apache_Maven

- 2008 április 21-én jelent meg.
- Neve a Cradle (bölcső) szóból ered a használt Groovy nyelv G kezdőbetűje miatt.
- Támogatott nyelvek: Java, Kotlin, Groovy, Scala, C/C++, JavaScript.
- Apache Ant és Apache Maven koncepciójára épít.
- XML helyett Groovy nyelvet használ, illetve ma már Kotlint is.
- Java Virtuális Gépen fut.
- Repozitorikat használ.

https://en.wikipedia.org/wiki/Gradle

- Convention over configuration módszert használja.
    - Emiatt a build leírások rövidek.
- Plugin-okkal bővíthető.
- Android ezt használja.
- Főbb IDE-k támogatják (IntelliJ, AndroidStudio, Eclipse, NetBeans, Visual Studio Code), de futtatható parancssorból is.
- **build.gradle** fájlok határozzák meg a projekt fordítását **taszk**okkal.
- **Önmaga állítja össze a futtatandó taszkok végrehajtási sorrendjét.**

https://en.wikipedia.org/wiki/Gradle

Build Taszkok (feladatok) futtatásának jellemzői:

- megfelelő sorrendben (mindennek meglegyen az előkészülete) hajtja végre ezeket,
- minden taszknak van bemenete és kimenete,
- elmenti minden taszk kimenetét és ha valaminél azt látja nem változott, akkor a ráépülőket nem futtatja, mivel korábban is ugyanazt az inputot kapta, így ismételten ugyanaz lenne az eredménye.

Függőségek (dependency) kezelését végzi. Ezek lehetnek:

- Java osztályok
- Lib-ek
- függőségek-függőségei (transitive dependency).

Mindenből a megfelelő verziójút fogja használni.

Telepítése

- Gradle-hez szükség van Java Development Kit-re.
- https://gradle.org/install

Projekt készítése Gradle segítségével:

```
csabafazekas@MacBook-Air-3 ~ % gradle
Starting a Gradle Daemon (subsequent builds will be
faster)

> Task :help

Welcome to Gradle 7.3.3.

Directory '/Users/csabafazekas' does not contain a
Gradle build.

To create a new build in this directory, run gradle init
. . .
```

Projekt létrehozása:

1. Keressünk egy könyvtárat az új projektünknek!
2. mkdir firstproject
3. cd firstproject
4. gradle init

```
Select type of project to generate:
  1: basic
  2: application
  3: library
  4: Gradle plugin
Enter selection (default: basic) [1..4] 2 -> Application

Select implementation language:
  1: C++
  2: Groovy
  3: Java
  4: Kotlin
  5: Scala
  6: Swift
Enter selection (default: Java) [1..6] 3 -> Java
```

Majd:

```
Split functionality across multiple subprojects?:
  1: no - only one application project
  2: yes - application and library projects
Enter selection (default: no - only one application
project) [1..2] 1
```

Ezt követően:

```
Select build script DSL:
  1: Groovy
  2: Kotlin
Enter selection (default: Groovy) [1..2] 1
```

Végezetül pedig:

```
Select test framework:
  1: JUnit 4
  2: TestNG
  3: Spock
  4: JUnit Jupiter
Enter selection (default: JUnit Jupiter) [1..4] 4

Project name (default: firstproject): first
Source package (default: first): com.tutorial.gradle
```

Az eredmény pedig:

```
> Task :init
Get more help with your project: https://
docs.gradle.org/7.3.3/samples/
sample_building_java_applications.html

BUILD SUCCESSFUL in 4m 52s
2 actionable tasks: 2 executed
```

A könyvtárunk a következőket tartalmazza:

```
app  gradle  gradlewgradlew.batsettings.gradle
```

Listázzuk ki milyen taszkok állnak rendelkezésre:

```
./gradlew tasks
```

```
> Task :tasks

------------------------------------------------------------
Tasks runnable from root project 'first'
------------------------------------------------------------

Application tasks
-----------------
run - Runs this project as a JVM application

Build tasks
-----------
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.

Build Setup tasks
-----------------
init - Initializes a new Gradle build.
wrapper - Generates Gradle wrapper files.

Distribution tasks
------------------
assembleDist - Assembles the main distributions
distTar - Bundles the project as a distribution.
distZip - Bundles the project as a distribution.
installDist - Installs the project as a distribution as-is.
```

Documentation tasks
----
javadoc - Generates Javadoc API documentation for the main source code.

Help tasks
----
buildEnvironment - Displays all buildscript dependencies declared in root project 'first'.
dependencies - Displays all dependencies declared in root project 'first'.
dependencyInsight - Displays the insight into a specific dependency in root project 'first'.
help - Displays a help message.
javaToolchains - Displays the detected java toolchains.
outgoingVariants - Displays the outgoing variants of root project 'first'.
projects - Displays the sub-projects of root project 'first'.
properties - Displays the properties of root project 'first'.
tasks - Displays the tasks runnable from root project 'first' (some of the displayed tasks may belong to subprojects).

Verification tasks
----
check - Runs all checks.
test - Runs the test suite.

To see all tasks and more detail, run **gradlew tasks --all**

To see more detail about a task, run **gradlew help --task <task>**

BUILD SUCCESSFUL in 382ms
1 actionable task: 1 executed

- Gradle build végrehajtásának legjobb módja a Gradle Wrapper futtatása

  - Ez egy szkript ami **meghatározott Gradle verziót** hív meg.
  - Ha szükséges, le is tölti azt. Tehát **átadhatjuk** a projektünket másnak és ha kell a Gradle-t telepíti a Wrapper neki!
  - **gradle wrapper** paranccsal tudjuk hozzáadni egy Gradle projekthez ha az még nem tartalmazza.
  - A gradle/wrapper/gradle-wrapper.properties tartalmazza a beállításait.


```
.
├── a-subproject
│   └── build.gradle
├── settings.gradle
├── gradle
│   └── wrapper
│       ├── gradle-wrapper.jar
│       └── gradle-wrapper.properties
├── gradlew
└── gradlew.bat
```

A gradle/wrapper/**gradle-wrapper.properties** tartalma:

```
distributionBase=GRADLE_USER_HOME
distributionPath=wrapper/dists
distributionUrl=https\://services.gradle.org/distributions/gradle-7.3.3-bin.zip
distributionUrl=https\://services.gradle.org/distributions/gradle-8.3-bin.zip
zipStoreBase=GRADLE_USER_HOME
zipStorePath=wrapper/dists
```


```
.
├── a-subproject
│   └── build.gradle
├── settings.gradle
├── gradle
│   └── wrapper
│       ├── gradle-wrapper.jar
│       └── gradle-wrapper.properties
├── gradlew
└── gradlew.bat
```

- **gradle-wrapper.jar** - A gradle-wrapper.properties fájlban megadott gradle verzió letöltését végzi.
- **gradlew** - Gradle taszk futtatását végzi Gradle Wrapper segítségével.
- **gradlew.bat** - Gradle taszk futtatását végzi Gradle Wrapper segítségével Windows alatt.

```
./gradlew -version

------------------------------------------------------------
Gradle 8.3
------------------------------------------------------------

Build time:   2023-08-17 07:06:47 UTC
Revision:     8afbf24b469158b714b36e84c6f4d4976c86fcd5

Kotlin:       1.9.0
Groovy:       3.0.17
Ant:          Apache Ant(TM) version 1.10.13 compiled on January 4 2023
JVM:          19.0.2 (Oracle Corporation 19.0.2+7-44)
OS:           Mac OS X 12.6.3 aarch64
```

build.gradle

```
plugins {
    // Apply the application plugin to add support for building a CLI application in Java.
    application
}

repositories {
    // Use Maven Central for resolving dependencies.
    mavenCentral()
}

dependencies {
    // Use JUnit Jupiter for testing.
    testImplementation("org.junit.jupiter:junit-jupiter:5.9.1")

    // This dependency is used by the application.
    implementation("com.google.guava:guava:31.1-jre")
}
```

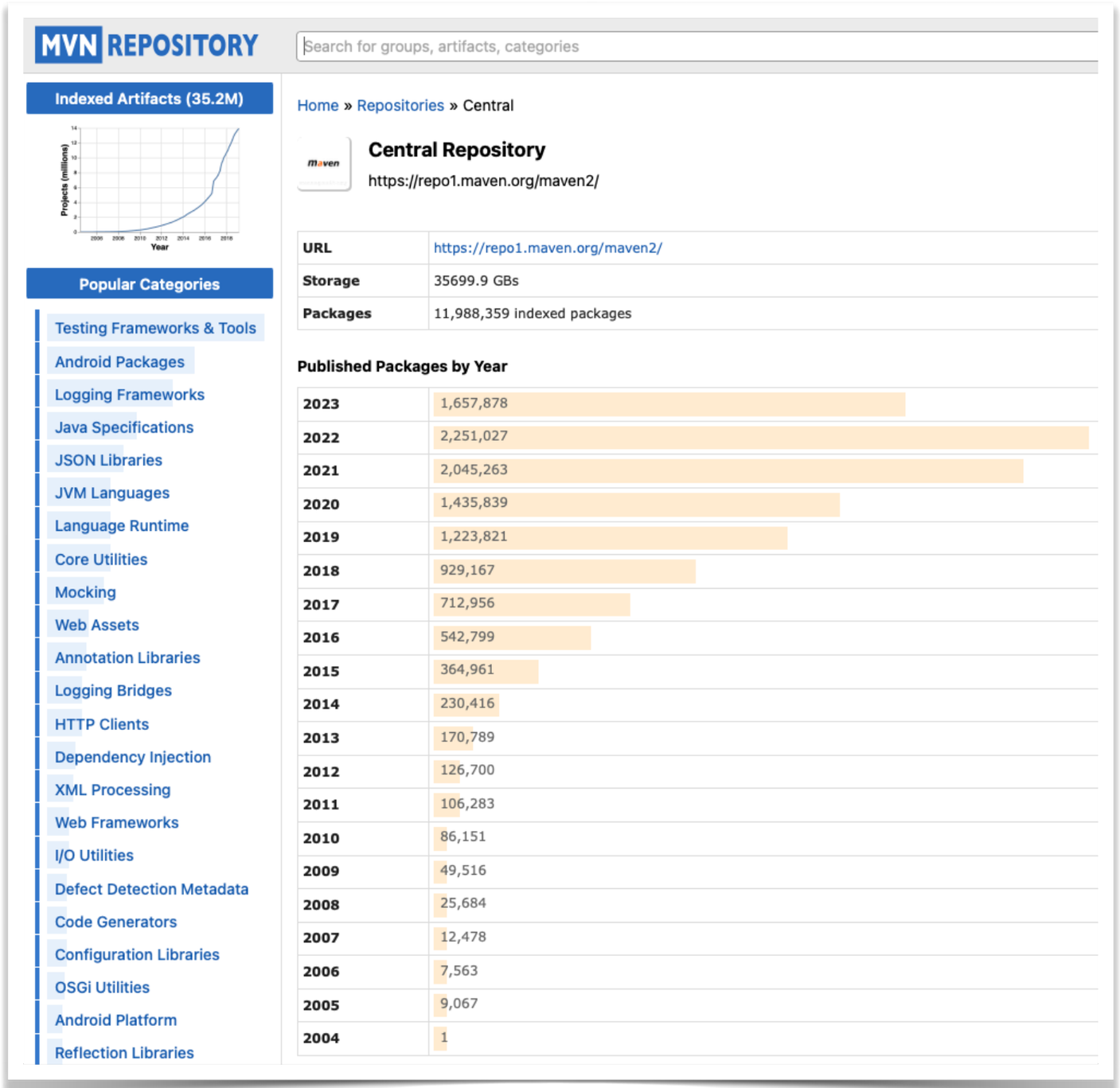
```
// Apply a specific Java toolchain to ease working on different environments.
java {
    toolchain {
        languageVersion.set(JavaLanguageVersion.of(17))
    }
}

application {
    // Define the main class for the application.
    mainClass.set("com.gradle.tutorial.App")
}

tasks.named<Test>("test") {
    // Use JUnit Platform for unit tests.
    useJUnitPlatform()
}
```

(https://docs.gradle.org/8.3/userguide/part1_gradle_init.html#part1_begin)

Gradle

Dependencies - függőségek

https://mvnrepository.com/repos/central



Taszkok

- Keressük meg a build.gradle fájlt!
- Adjuk hozzá a következő taszkot!

```
tasks.register("hello"){
  println("Hello World from Gradle!")
}
```

Taszkok

- Bővíthetjük további taszkokkal!

```
tasks.register("hello"){
    println("Hello World from Gradle!")
}

tasks.register("first"){
    println("Let's do something first ...")
    dependsOn("hello")
}
```

Build-en kívül mire lehet még használni?

```
gradle javadoc
```

vagy

```
./gradlew javadoc
```

- Javadoc formátumú dokumentáció készítés

Javadoc

“**Javadoc is a documentation generator** created by Sun Microsystems for the Java language (now owned by Oracle Corporation) for generating API documentation in HTML format from Java source code.

The **HTML format is used for adding the convenience of being able to hyperlink related documents together**.[1] The "doc comments" format[2] used by Javadoc is the de facto industry standard for documenting Java classes. Some IDEs,[3] like IntelliJ IDEA, NetBeans and Eclipse, automatically generate Javadoc templates.

Many file editors assist the user in producing Javadoc source and use the Javadoc info as internal references for the programmer.”

(https://en.wikipedia.org/wiki/Javadoc)

Javadoc

- Tehát a Javadoc :

  - alapvetően egy dokumentáció generátor, ami a Java forrásba elhelyezett megfelelő megjegyzéseket dolgozza fel hierarchikus formában,

  - megjegyzés a /** nyitó elemmel indul és */ zárja,

  - már magához az adott fájlhoz is adhatunk leírást, de osztályonként, metódusonként és változóként is készíthetünk továbbiakat.

Javadoc

- Példa:

```
/**
 * This Java source file was generated by the Gradle 'init' task.
 * @author Csaba Fazekas
 * @version 0.1
 */
package com.tutorial.gradle;

/**
 * Hello World printing application.
 */
public class App {
    /**
     * Creates the famous Hello World text.
     * @return predefined constans string.
     */
    public String getGreeting() {
        return "Hello World!";
    }
```

Javadoc

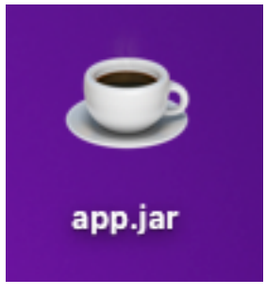
Metódusok megjegyzései a következőkből épülhetnek fel:

1. Első bekezdés az adott metódus leírása.
2. Ezt követhetik a következők:
    1. @param után felsorolva a metódus paraméterei.
    2. @return a metódus visszatérő értéke.
    3. @throws után ha valamilyen exception-t kezelni kell.

Tag & Parameter	Usage	Applies to
@author *John Smith*	Describes an author.	Class, Interface, Enum
{**@docRoot**}	Represents the relative path to the generated document's root directory from any generated page.	Class, Interface, Enum, Field, Method
@version *version*	Provides software version entry. Max one per Class or Interface.	Class, Interface, Enum
@since *since-text*	Describes when this functionality has first existed.	Class, Interface, Enum, Field, Method
@see *reference*	Provides a link to other element of documentation.	Class, Interface, Enum, Field, Method
@param *name description*	Describes a method parameter.	Method
@return *description*	Describes the return value.	Method
@exception *classname description* **@throws** *classname description*	Describes an exception that may be thrown from this method.	Method
@deprecated *description*	Describes an outdated method.	Class, Interface, Enum, Field, Method
{**@inheritDoc**}	Copies the description from the overridden method.	Overriding Method
{**@link** *reference*}	Link to other symbol.	Class, Interface, Enum, Field, Method
{**@linkplain** *reference*}	Identical to {@link}, except the link's label is displayed in plain text than code font.	Class, Interface, Enum, Field, Method
{**@value** *#STATIC_FIELD*}	Return the value of a static field.	Static Field
{**@code** *literal*}	Formats literal text in the code font. It is equivalent to <code>{@literal}</code>.	Class, Interface, Enum, Field, Method
{**@literal** *literal*}	Denotes literal text. The enclosed text is interpreted as not containing HTML markup or nested javadoc tags.	Class, Interface, Enum, Field, Method
{**@serial** *literal*}	Used in the doc comment for a default serializable field.	Field
{**@serialData** *literal*}	Documents the data written by the writeObject( ) or writeExternal( ) methods.	Field, Method
{**@serialField** *literal*}	Documents an ObjectStreamField component.	Field

Gradle tutorial:

- https://docs.gradle.org/8.3/userguide/introduction.html#getting_started



Jar fájl

app

Név	Módosítás dátuma	Méret	Fajta
com	ma 19:03	--	Mappa
tutorial	ma 20:03	--	Mappa
gradle	ma 19:03	--	Mappa
App.class	ma 19:03	672 bájt	Java-osztály fájl
META-INF	ma 20:03	--	Mappa
MANIFEST.MF	ma 20:03	↑ 2 bájt	Dokumentum

XML

?????

What is XML?

- XML is a simplified subset of Standard Generalized Markup Language (SGML).
- SGML is based on GML from 1960.
- The original SGML was accepted in October 1986.
- SGML was originally designed to enable the sharing of machine-readable large-project documents in government, law, and industry.
- SGML was reworked in 1998 into XML.
- It was developed by the World Wide Web Consortium.

What is XML?

- The name 'XML' is an acronym for ‘E**x**tensible **M**arkup **L**anguage' (with 'X' replacing 'E' for aesthetic impact).
- The XML data format is always supported by an underlying text format, like ASCII, or **UTF-8** what is the near-universal format for text-based information storage and transfer.

What is XML?

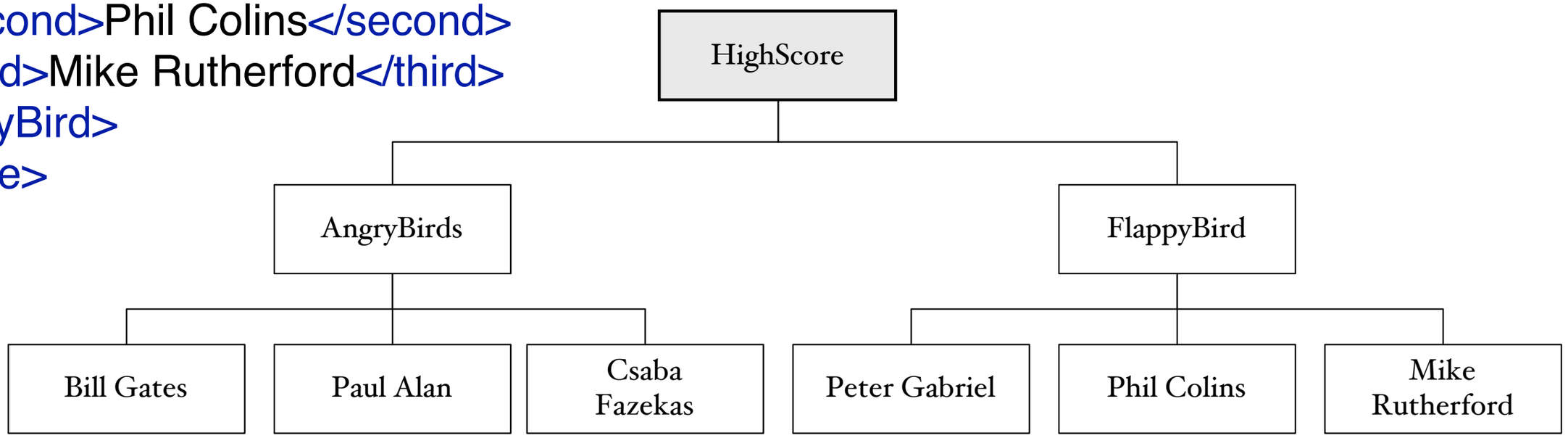
- It is similar in some ways to HTML, but its **focus is on describing structured data**,
- **rather than on presentation of data** via a web browser to a human user.
- The primary purpose of XML is to facilitate the **sharing of data** across different information systems.

What is XML?

- XML uses a **hierarchical model** for the representation of data;
- data is represented as a **tree** in which an element encloses other elements.

An example

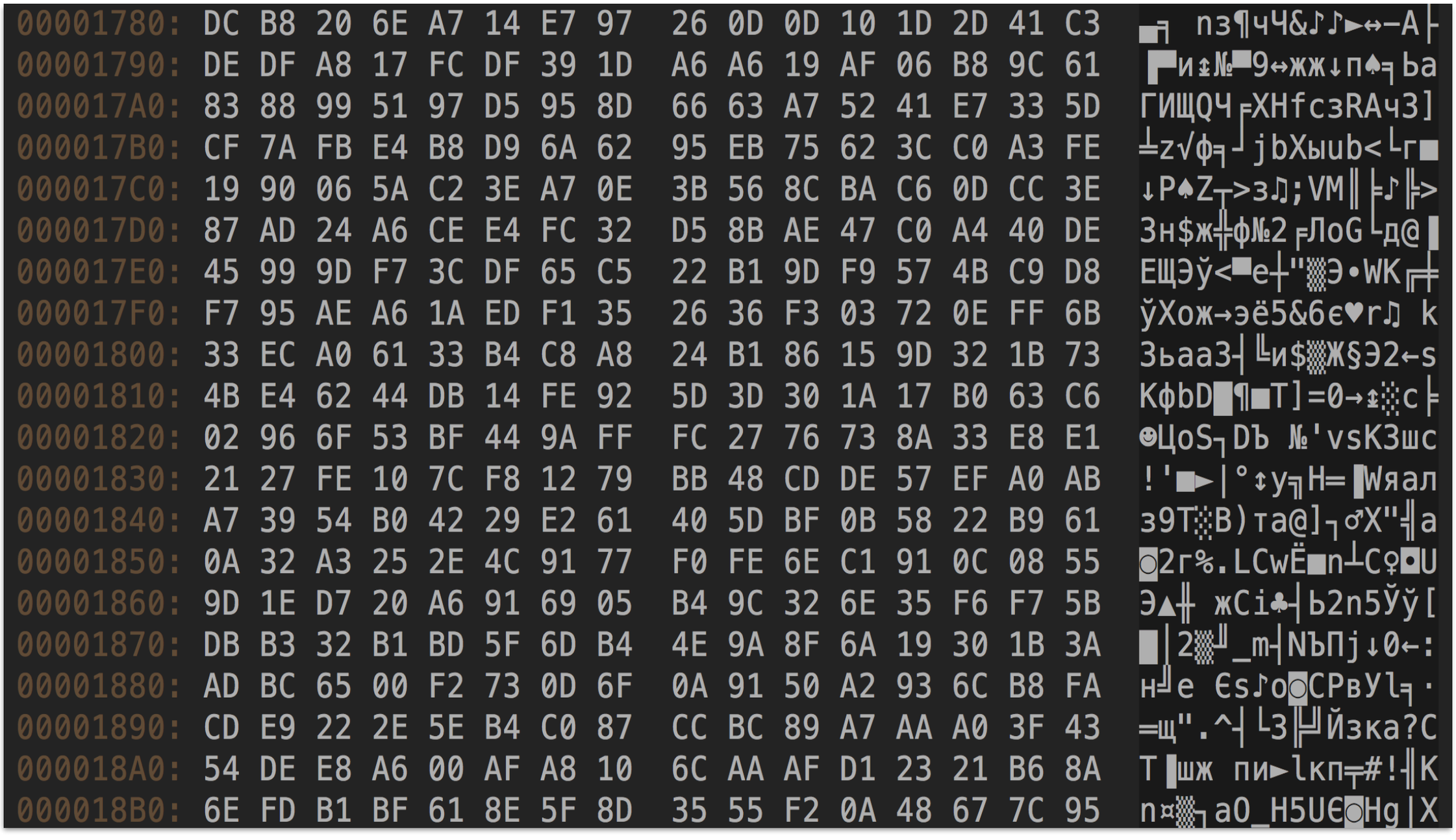
```
<?xml version="1.0" encoding="UTF-8"?>
<HighScore>
    <AngryBirds>
        <first>Bill Gates</first>
        <second>Paul Alen</second>
        <third>Csaba Fazekas</third>
    </AngryBirds>
    <FlappyBird>
        <first>Peter Gabriel</first>
        <second>Phil Colins</second>
        <third>Mike Rutherford</third>
    </FlappyBird>
</HighScore>
```



What is XML good for?

- XML has almost unlimited application.
- The two primary uses are:
  - **data exchange** format
  - **document publishing** format.
- But also used for configuring applications, used as a database, etc.

Why we need it?



Why we need it?

- At the beginning, there were different character codings per country.
- The first 128 characters of ASCII (7 bit) was standard, the upper half was subject to change.
- UTF-8, UNICODE helps a lot.

Simple Markup

- ASCII text can be made smarter, simply by adding on an extra layer of meaning to certain characters or sequences of characters in the text.
- For example:
    - the **comma** and **line-end codes** are deemed significant in the CSV format.

```
Name;Neptun code
Kortjohann Jonathan;QN3W1P
Shorbila Seif Eldeen;UWRLPC
Salama Ahmed;YLAYHF
Ben el Heni Bilel;F62NON
Bejaoui Ines;HV4IEZ
Akari Mohamed Sabri;L3BH99
Labidi Najdi;IL971U
```

Simple Markup

- Weaknesses:
    - Only **tabular information** can be represented, so **every row must hold the same kind of information**.
    - **Each column must have a pre-defined purpose**, so if repeatable items are required then the maximum number of occurrences must be defined in advance.

Elements

- XML allows documents to be decomposed into smaller, meaningful **elements** that can be recognized.
- Every XML document must contain at least **one element**, and the first element always identifies and encloses the entire document.

Container Element

- The term **container element** is used to describe an element that *encloses* the data it identifies.
- Three parts of a container element:
    - start-tag
    - data
    - end-tag
- There are no pre-defined element names.

Start and End tags

- The **start-tag** is identified by the surrounding markup characters '<' and ‘>’.
- The **end-tag** begins with the sequence ‘</' and ends with ‘>’.

Limitations in element names

- Element names are **case sensitive**,
- an element name must **begin with a letter**,
- an underscore character, '_',
- or a colon, '**:**' (though there are restrictions on the usage of the colon),
- and may additionally contain digits and some other punctuation characters ('**.**' and '**-**').
- For example:

  <p>, <X:123> and <ALongElementName>.

Empty Element

- Elements do not have to be containers, but can act as **place holders**, to implement specific document features:

<pageBreak></pageBreak>

or

<pageBreak/>

Special Characters

<code>if (x **<** y) { ... } </code>

- '**<**' (less than) code represents the '**<**' character
- '**>**' (greater than) code represents the '**>**' character

note the semicolon (;) at the end

<code>if (x **<** y) { ... } </code>

Special Characters

The **<name>** tag identifies a name.

The **<name>** tag identifies a name.

Special Characters

- **<** becomes **<**
- **>** becomes **>**
- **&** becomes **&**
- **"** becomes **"**
- **'** becomes **'**

Comments

- Comments begin with '**<!—**' and end with '**-->**'. For example:

```
<?xml version='1.0' encoding='UTF-8'?>
<!-- Based on an example in Wikipedia, March 2008 -->
<recipe name='bread' prep_time='5 mins' cook_time='3 hours'>
    <title>Basic bread</title>
    ....
</recipe>
```

Attributes

- Elements can have attributes; these include information about the element, additional to its content.
- Attributes occur inside the opening tag, and take the form

name=‘value’

- For example two attributes (amount and unit):

```
<ingredient amount='1' unit='teaspoon'>Salt</ingredient>
```

- An attribute name can only occur once in an element. (Note that this is not the case for elements: an element can contain several child elements with the same name.)

Elements or attributes?

<match league="Barclays Premiership" date="2006-10-21" homeTeam = "Barcelona" awayTeam = "Manchester City" > .... </match>

OR

It does not show any relationship. These are just data.

```
<match>
    <league>Barclays Premiership</league>
    <date>2017-01-16</date>
    <teams>
        <home>Barcelona</home>
        <away>Manchester City</away>
    </teams>
</match>
```

The corresponding elements are grouped.

Elements or attributes?

- No agreed principles
- However, there are some restrictions on the kind of data that can be represented in an attribute:
  - attribute values are just short strings
  - attribute names must be unique within an element
  - the order of attributes within an element is not significant

Elements or attributes?

- So an attribute is only appropriate for modeling a single-valued property of an object, with a simple data type and no internal structure.

Elements or attributes?

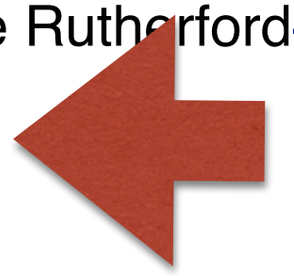
- If the data:
    - has structure (it is composed of simpler items)
    - has repeated components (like an array)
    - has subcomponents in a particular order
    - is a simple string that might need to be "cut up" into multiple pieces in the future
    - has text content which is very long

then attributes are not suitable, and **we have to use elements**.

One more thing

- The closing element should enclose its content

```
<?xml version="1.0" encoding="UTF-8"?>
<HighScore>
    <AngryBirds>
        <first>Bill Gates</first>
        <second>Paul Alen</second>
        <third>Csaba Fazekas</third>
    </AngryBirds>
    <FlappyBird>
        <first>Peter Gabriel</first>
        <second>Phil Colins</second>
        <third>Mike Rutherford</third>
    </HighScore>
</FlappyBird>
```



```
<?xml version="1.0" encoding="UTF-8"?>
<HighScore>
    <AngryBirds>
        <first>Bill Gates</first>
        <second>Paul Alen</second>
        <third>Csaba Fazekas</third>
    </AngryBirds>
    <FlappyBird>
        <first>Peter Gabriel</first>
        <second>Phil Colins</second>
        <third>Mike Rutherford</third>
    </FlappyBird>
</HighScore>
```

Exercise 1

```
<?xml version="1.0" encoding="UTF-8"?>
<!— Document : db.xml -->
<Person>
      <name>
            <first>Bill</first>
            <family>Gates</family>
      </name>
      <PersonalDetails gender="male" personellNo="123456">
            <dob>1955-10-23</dob>
            <Address>
                  <houseNumber>4</houseNumber>
                  <street>Leanyka</street>
                  <town>Eger</town>
                  <postCode>3300</PostCode>
                  </Address>
            <email type=home>info@ektf.hu</email>
            <PhoneNumbers>
                        <home>003636123456</home>
                        <mobile>003610123456<mobile>
                        <work></work>
            </PersonalDetails>
      </PhoneNumbers>
</Person>
```

Exercise 2

- Produce an XML document capturing the following information about a book:
  - It is called “UML 2.0 in a nutshell”.
  - Its authors are Dan Pilone and Neil Pitman.
  - It is published by O’Reilly.
  - The year of publication is 2005.
  - ISBN is 0-596-00795-7.
  - It has a preface, 12 chapters, two appendices.
    - The chapters and appendices are: Chapter 1, Chapter 2,… Chapter 12
    - Appendices: Appendix A, Appendix B

```
<?xml version="1.0" encoding="UTF-8"?>
<book isbn="0-596-00795-7">
  <title>UML 2.0 in a nutshell</title>
  <authors>
     <author>Dan Pilone</author>
     <author>Neil Pitman</author>
  </authors>
  <publisher>O’Reilly</publisher>
  <year>2005</year>
  <content>
     <preface/>
     <chapters>
        <chapter>Chapter 1</chapter>
        <chapter>Chapter 2</chapter>
        <chapter>Chapter 12</chapter>
     </chapters>
     <appendices>
        <A> Appendix A</A>
        <B> Appendix B</B>
     </appendices>
  </content>
</book>
```

Exercise 3

- Produce a well-formed XML document to describe yourself.
- The root element should be <Person>.
- You might include personal information, your likes, dislikes and hobbies, educational achievements, etc.
- Use elements.

Document Type Definition (DTD)

Validation and Document Type Definitions

- A **document model** describes a structure in terms of the allowed elements and attributes, and the relationships between them.
- An XML file is **valid** if it conforms to the rules laid down in its document model.

What is a document model?

- We can make up **whatever element (tag) and attribute names that we like, and use** them in whatever way is convenient in our XML file.
- The result may be **human-readable** and easy to understand.
- If we want such documents to be processed by programs, however, we will need to be strict about **what elements occur, how these elements are structured, the order in which they occur, what attributes they have, etc**.
- A program will need to know this information to be able to extract information from the document.

What is a document model?

- Some rules need to established **to define the structure of** a class of **similar documents**.
- **Defining such structure is equals to the definition of a new language**: this might be a very simple language for describing the contents of an email, or it might be a complex language for defining figures produced in two-dimensional graphics.
- This set of rules for a class of similar documents is called a **document model**.
- We can **think of these rules as the syntax or grammar** for the language.

What is a document model?

- Each document of the class concerned is an instance of the document model.
- This is similar to the notions of **object** and **class** in **object-oriented programming** languages:
  - The class is a description of the properties (fields and methods) of its objects.
  - The class lays down the rules that all of its instances must follow.
  - A document model defines the structure of a number of similar documents, which differ in the data that they contain.

What is a document model?

- There are several ways of defining a document model:
  - Document Type Definitions (DTDs)
  - XML Schemas
  - etc.

An example XML file

```
<?xml version="1.0" encoding=“UTF-8"?>
<email priority="High">
    <sender>fcsaba.eger@gmail.com</sender>
    <date>2017-02-01T11:11:11</date>
    <recipients>
        <recipient>info@youtube.com</recipient>
        <copy>admin@youtube.com</copy>
    </recipients>
    <subject>Bug report</subject>
    <body>I have found a bug … </body>
</email>
```

The DTD file for the XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- root definition for email -->
<!ELEMENT email (sender, date, recipients, subject?, body?)>
<!ATTLIST email priority CDATA #REQUIRED>
<!-- definitions for level 1 elements -->
<!ELEMENT sender (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT recipients (recipient+, copy*, blind-copy*) >
<!ELEMENT subject (#PCDATA)>
<!ELEMENT body (#PCDATA)>
<!ELEMENT recipient (#PCDATA)>
<!ELEMENT copy (#PCDATA)>
<!ELEMENT blind-copy (#PCDATA)>
```

DTD structure

- Each line of the DTD begins with the symbols '**<!**' and ends with a '**>**'.
- Each line beginning **<!ELEMENT defines** (inside round brackets) **the content model for the element** named immediately after <!ELEMENT.
- **A bracketed, comma-separated list indicates composition.** (So an email is composed of sender, date, recipients, etc., elements in that order. This is how parent-child element relationships are shown.)

Elements

- Format of Elements

```
<!ELEMENT element-name category>
or
<!ELEMENT element-name (element-content)>
```

- Empty elements

```
<!ELEMENT element-name EMPTY>
```

Example:

```
<!ELEMENT br EMPTY>
```

XML example:

```
<br />
```

- Elements **that just contain simple character content** (i.e. no children) have a content model which is (**#PCDATA**) (parsed character data).

<!ELEMENT from (#PCDATA)>

- Elements with children

```
<!ELEMENT element-name (child1)>
or
<!ELEMENT element-name (child1,child2,...)>
```

Example:

```
<!ELEMENT note (to,from,heading,body)>
```

What else is in the DTD?

- **<!ATTLIST** is followed by the name of an element, and then a list of information about the attributes of that element.

  <!ATTLIST email priority CDATA #REQUIRED>

  Where the email element has one attribute called priority. The type of this attribute is **CDATA** (character data) and it must be present (**#REQUIRED**).

- For the data type of an attribute, we can also specify a list of possible values, for example:

  <!ATTLIST email priority (Low|Medium|High) #REQUIRED>

- As well as #REQUIRED, we can say that an attribute is optional (**#IMPLIED**),

- or we can give an attribute a default value (a value that the attribute will have if it is not explicitly specified in the element).

  <!ATTLIST email priority (Low|Medium|High) "Low">

Declaring the number of occurrence of Elements

- An Element occurs only once:

```
<!ELEMENT element-name (child-name)>
```

Example:

```
<!ELEMENT note (message)>
```

- Special characters are used to indicate the multiplicities of child elements.
  - **‘*’** indicates 0 or more (i.e. many),
  - **‘+’** indicates 1 or more (i.e. minimum once),
  - **‘?’** indicates 0 or 1 (i.e. an optional element).

Declaring the number of occurrence of Elements

- Declaring either/or Content:

```
<!ELEMENT note (to,from,header,(message|body))>
```

It declares that the "note" element must contain a "to" element, a "from" element, a "header" element, and either a "message" or a "body" element.

- Declaring Mixed Content:

```
<!ELEMENT note (#PCDATA|to|from|header|message)*>
```

It declares that the "note" element can contain zero or more occurrences of parsed character data, "to", "from", "header", or "message" elements.

Explanation of child elements

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- root definition for email -->
<!ELEMENT email (sender, date, recipients, subject?, body?)>
<!ATTLIST email priority CDATA #REQUIRED>
<!-- definitions for level 1 elements -->
<!ELEMENT sender (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT recipients (recipient+, copy*, blind-copy*) >
<!ELEMENT subject (#PCDATA)>
<!ELEMENT body (#PCDATA)>
<!ELEMENT recipient (#PCDATA)>
<!ELEMENT copy (#PCDATA)>
<!ELEMENT blind-copy (#PCDATA)>
```

Optional 0 or 1

Optional 0 or 1

0 or more

0 or more

1 or more.

There has to be at least one!

Attributes

- Attributes are declared with an ATTLIST declaration.

Syntax:

```
<!ATTLIST element-name attribute-name attribute-type attribute-value>
```

DTD example:

```
<!ATTLIST payment type CDATA "check">
```

XML example:

```
<payment type="check" />
```

attribute-type

Type	Description
CDATA	The value is character data
(*en1*|*en2*|..)	The value must be one from an enumerated list
ID	The value is a unique id
IDREF	The value is the id of another element
IDREFS	The value is a list of other ids
NMTOKEN	The value is a valid XML name
NMTOKENS	The value is a list of valid XML names
ENTITY	The value is an entity
ENTITIES	The value is a list of entities
NOTATION	The value is a name of a notation
xml:	The value is a predefined xml value

attribute-value

Value	Explanation
value	The default value of the attribute
#REQUIRED	The attribute is required
#IMPLIED	The attribute is optional
#FIXED *value*	The attribute value is fixed

#REQUIRED

Use the #REQUIRED keyword if you don't have an option for a default value, but still want to force the attribute to be present.

Syntax

```
<!ATTLIST element-name attribute-name attribute-type #REQUIRED>
```

Example

DTD:

```
<!ATTLIST person number CDATA #REQUIRED>
```

Valid XML:

```
<person number="5677" />
```

Invalid XML:

```
<person />
```

#IMPLIED

Use the #IMPLIED keyword if you don't want to force the author to include an attribute, and you don't have an option for a default value.

Syntax

```
<!ATTLIST element-name attribute-name attribute-type
#IMPLIED>
```

Example

DTD:

```
<!ATTLIST contact fax CDATA #IMPLIED>
```

Valid XML:

```
<contact fax="555-667788" />
```

Valid XML:

```
<contact />
```

#FIXED

Use the #FIXED keyword when you want an attribute to have a fixed value without allowing the author to change it. If an author includes another value, the XML parser will return an error.

Syntax

```
<!ATTLIST element-name attribute-name attribute-type #FIXED “value">
```

Example

DTD:

```
<!ATTLIST sender company CDATA #FIXED "Microsoft">
```

Valid XML:

```
<sender company="Microsoft" />
```

Invalid XML:

```
<sender company="W3Schools" />
```

ENUMERATED VALUES

Use enumerated attribute values when you want the attribute value to be one of a fixed set of legal values.

Syntax

```
<!ATTLIST element-name attribute-name (en1|en2|..) default-value>
```

Example

DTD:

```
<!ATTLIST payment type (check|cash) "cash">
```

XML example:

```
<payment type="check" />
or
<payment type="cash" />
```

How to associate the DTD to an XML file?

- The DTD can be inserted in-line inside the XML document itself in a special <!DOCTYPE section. This declaration is not very flexible, as it involves repeating the DTD for each document.
- We can put a link into the XML document, indicating where the parser can find the DTD, giving an external declaration.

XML with internal DTD

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
  <to>Csaba Fazekas</to>
  <from>Istvan Fazekas</from>
  <heading>Reminder</heading>
  <body>Don't forget the meeting.</body>
</note>
```

!DOCTYPE note defines that the root element of this document is note

!ELEMENT note defines that the note element must contain four elements: "to,from,heading,body"
!ELEMENT to defines the to element to be of type "#PCDATA"
!ELEMENT from defines the from element to be of type "#PCDATA"
!ELEMENT heading defines the heading element to be of type "#PCDATA"
!ELEMENT body defines the body element to be of type "#PCDATA"

How to associate the DTD to an XML file?

```
<?xml version="1.0" encoding=“UTF-8"?>
<!DOCTYPE email SYSTEM "email.dtd">
<email priority="High">
    <sender>fcsaba.eger@gmail.com</sender>
    <date>2017-02-01T11:11:11</date>
    <recipients>
        <recipient>info@youtube.com</recipient>
        <copy>admin@youtube.com</copy>
    </recipients>
    <subject>Bug report</subject>
    <body>I have found a bug … </body>
</email>
```

Exercise 1

- Create an e-mail XML file with a DTD.

Create an XML file called *email.xml*

```
<?xml version="1.0" encoding=“UTF-8”?>
<!DOCTYPE email SYSTEM "email.dtd">
<email priority="High">
    <sender>fcsaba.eger@gmail.com</sender>
    <date>2017-02-01T11:11:11</date>
    <recipients>
        <recipient>info@youtube.com</recipient>
        <copy>admin@youtube.com</copy>
    </recipients>
    <subject>Bug report</subject>
    <body>I have found a bug … </body>
</email>
```

Create a DTD file called *email.dtd*

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- root definition for email -->
<!ELEMENT email (sender, date, recipients, subject?,
body?)>
<!ATTLIST email priority CDATA #REQUIRED>
<!-- definitions for level 1 elements -->
<!ELEMENT sender (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT recipients (recipient+, copy*, blind-copy*) >
<!ELEMENT subject (#PCDATA)>
<!ELEMENT body (#PCDATA)>
<!ELEMENT recipient (#PCDATA)>
<!ELEMENT copy (#PCDATA)>
<!ELEMENT blind-copy (#PCDATA)>
```

Exercise 2

- Create a DTD for the recipe.xml file.

Create an XML file called *recipe.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE recipe SYSTEM "recipe.dtd" >
<recipe name="bread" prep_time="5 mins" cook_time="3 hours">
    <title>Basic bread</title>
    <ingredient amount="3" unit="cups">Flour</ingredient>
    <ingredient amount="0.25" unit="ounce">Yeast</ingredient>
    <ingredient amount="1.5" unit="cups" state="warm">Water</ingredient>
    <ingredient amount="1" unit="teaspoon">Salt</ingredient>
    <instructions>
        <step>Mix all ingredients together, and knead thoroughly.</step>
        <step>Cover with a cloth, and leave for one hour in warm room.</step>
        <step>Knead again, place in a tin, and then bake in the oven.</step>
    </instructions>
</recipe>
```

```
<!ELEMENT recipe (title, ingredient+, instructions+) >
<!ATTLIST recipe name CDATA #REQUIRED>
<!ATTLIST recipe prep_time CDATA #REQUIRED>
<!ATTLIST recipe cook_time CDATA #REQUIRED>
<!ELEMENT title (#PCDATA) >
<!ELEMENT ingredient (#PCDATA) >
<!ATTLIST ingredient
    amount CDATA #REQUIRED
    unit CDATA #REQUIRED
    state (warm|cold) #IMPLIED >
<!ELEMENT instructions (step*) >
<!ELEMENT step (#PCDATA) >
```

Exercise 3

- Create a DTD for the match.xml file.

Create an XML file called *match.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE match SYSTEM "match.dtd">
<match league="Barclays Premiership" date="2017-03-06">
    <teams>
        <home goals="2">Chelsea</home>
        <away goals="1">Portsmouth</away>
    </teams>
    <scorers>
        <homeScorer>Drogba</homeScorer>
        <homeScorer>Schevchenko</homeScorer>
        <awayScorer>Benjani</awayScorer>
    </scorers>
    <referee>Dermot Gallagher</referee>
    <attendance>45326</attendance>
</match>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT match (teams, scorers, referee, attendance)>
<!ATTLIST match
date CDATA #REQUIRED
league CDATA "Barclays Premiership">

<!ELEMENT teams (home, away)>
<!ELEMENT scorers (homeScorer*, awayScorer*)>
<!ELEMENT referee (#PCDATA)>
<!ELEMENT attendance (#PCDATA)>

<!ELEMENT home (#PCDATA)>
<!ATTLIST home
goals CDATA "0">

<!ELEMENT away (#PCDATA)>
<!ATTLIST away
goals CDATA "0">

<!ELEMENT homeScorer (#PCDATA)>
<!ELEMENT awayScorer (#PCDATA)>
```