Domain-Specific Languages

Mathieu Acher

@acherm

Maître de Conférences

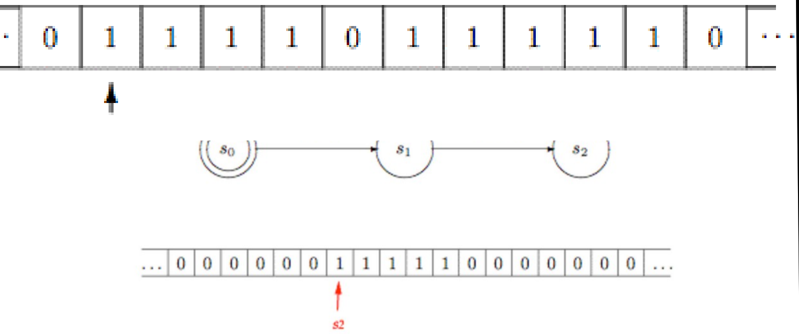
mathieu.acher@irisa.fr

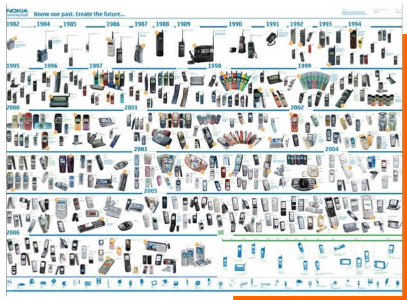
General Purpose Languages

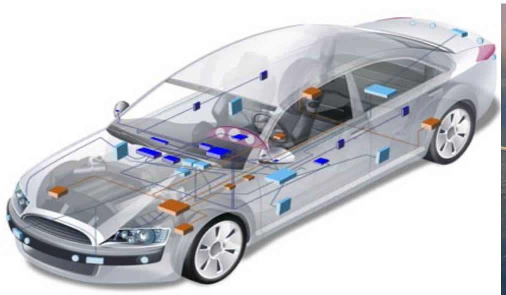
ssembly ?

OBOL ? LISP ? C ? C++ ?

ava? PHP ? C# ? Ruby ?



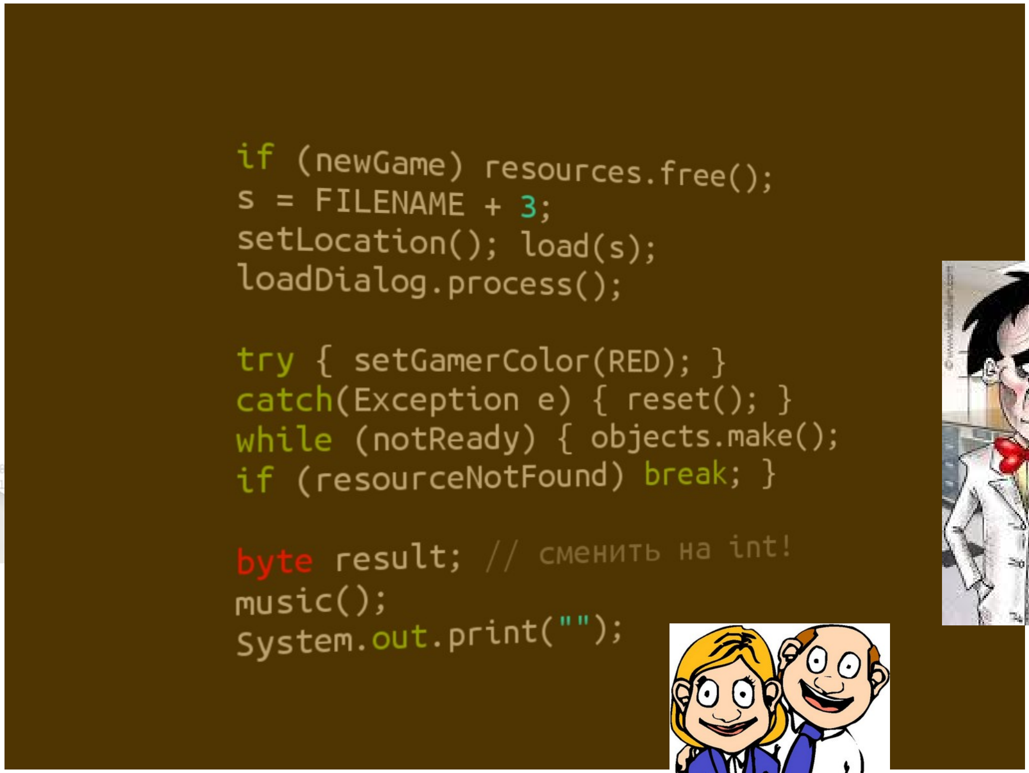




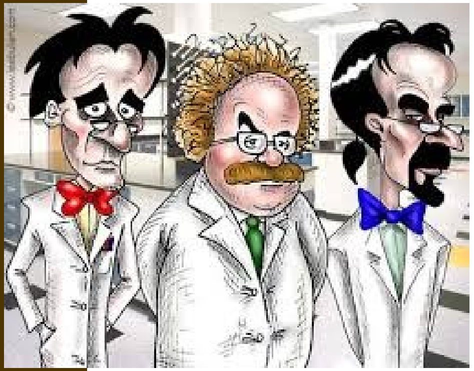
Limits of General Purpose Languages (1)

- **Abstractions** and **notations** used are not natural/suitable for the stakeholders

User Interface Designer with a passion for designing beautiful and functional user experiences. Minimalist who believes that less is more.



```
if (newGame) resources.free();
s = FILENAME + 3;
setLocation(); load(s);
loadDialog.process();

try { setGamerColor(RED); }
catch(Exception e) { reset(); }
while (notReady) { objects.make();
if (resourceNotFound) break; }

byte result; // сменить на int!
music();
System.out.print("");
```



Limits of General Purpose Languages (2)

- Not targeted to a **particular** kind of problem, but to any kinds of software problem.

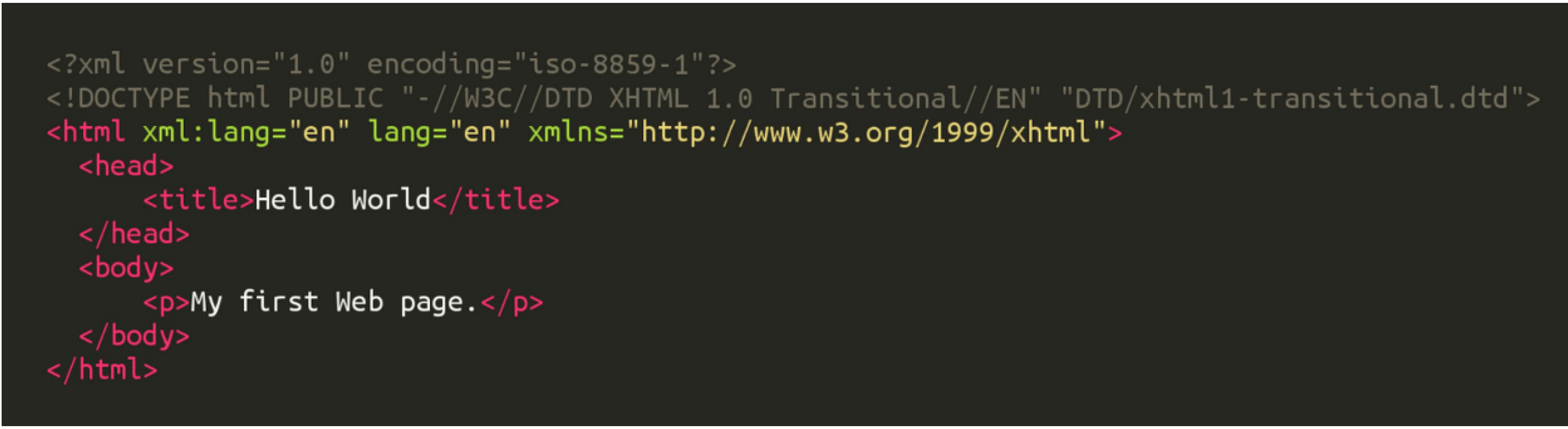
Domain Specific Languages

- Targeted to a **particular** kind of problem, with dedicated notations (textual or graphical), support (editor, checkers, etc.)
- Promises: more « efficient » languages for resolving a set of specific problems in a domain

Domain Specific Languages (DSLs)

- Long history: used for almost as long as computing has been done.
- You’re using DSLs in a daily basis
- You’ve learnt many DSLs in your curriculum
- Examples to come!

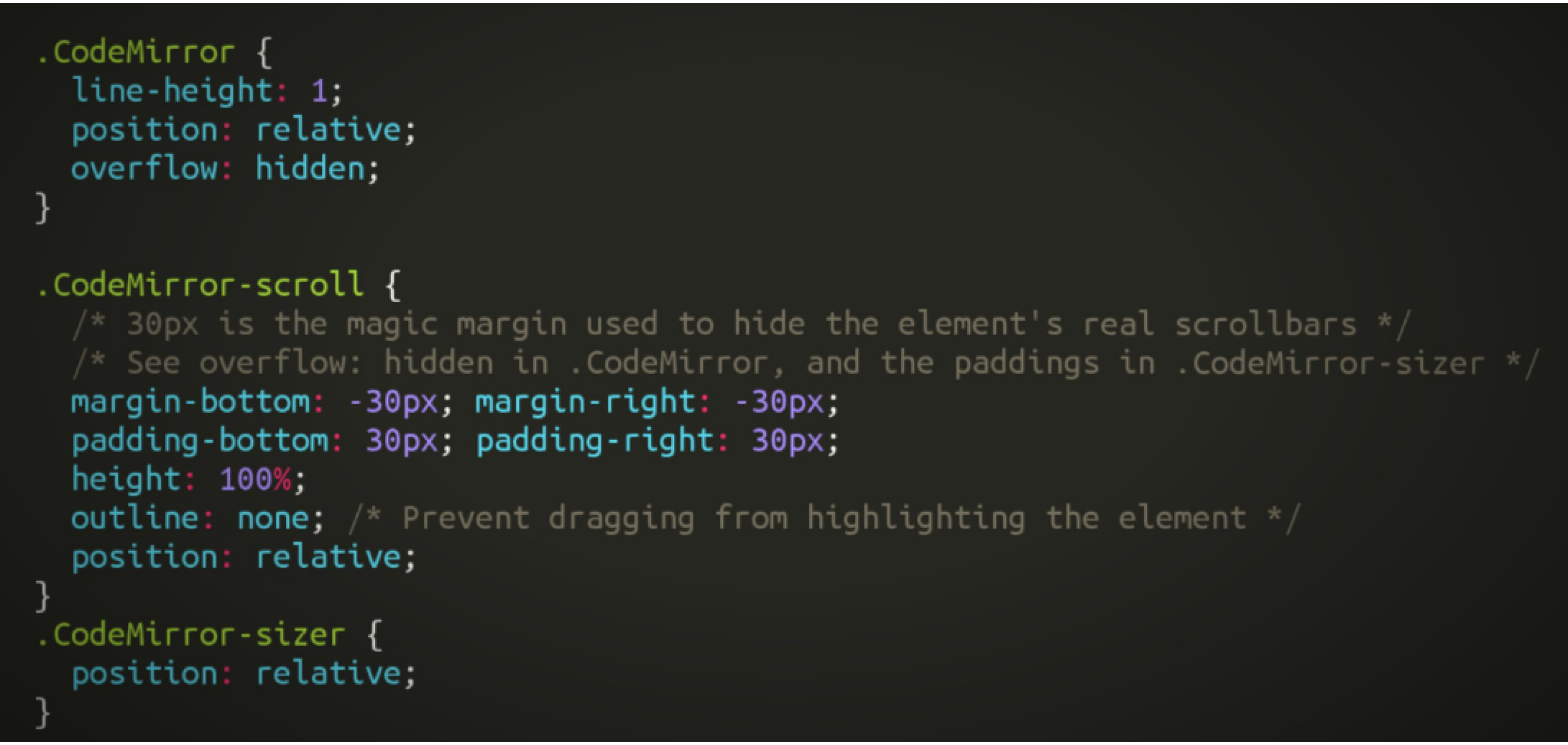
HTML



```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "DTD/xhtml1-transitional.dtd">
<html xml:lang="en" lang="en" xmlns="http://www.w3.org/1999/xhtml">
  <head>
      <title>Hello World</title>
  </head>
  <body>
      <p>My first Web page.</p>
  </body>
</html>
```

Domain: web (markup)

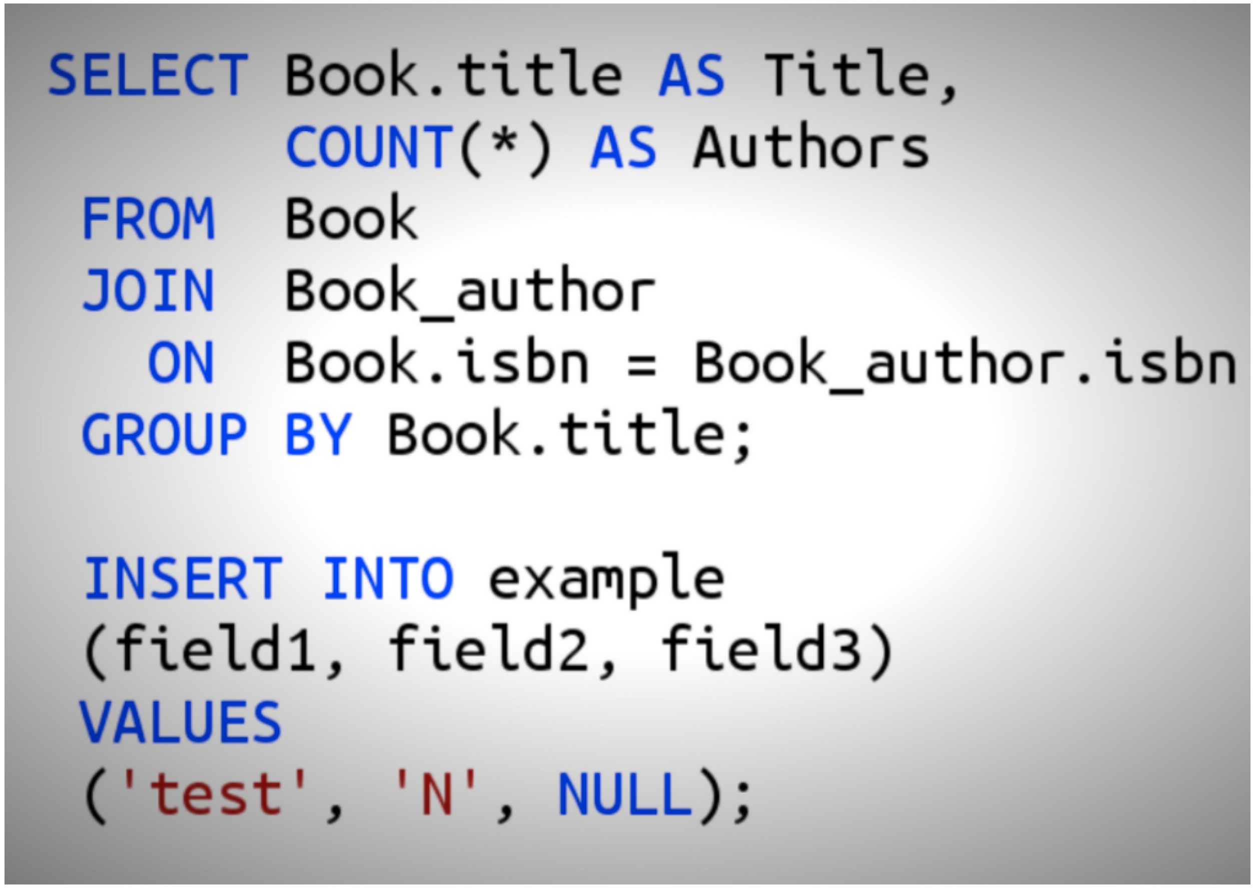
CSS



```
.CodeMirror {
  line-height: 1;
  position: relative;
  overflow: hidden;
}

.CodeMirror-scroll {
  /* 30px is the magic margin used to hide the element's real scrollbars */
  /* See overflow: hidden in .CodeMirror, and the paddings in .CodeMirror-sizer */
  margin-bottom: -30px; margin-right: -30px;
  padding-bottom: 30px; padding-right: 30px;
  height: 100%;
  outline: none; /* Prevent dragging from highlighting the element */
  position: relative;
}
.CodeMirror-sizer {
  position: relative;
}
```

Domain: web (styling)

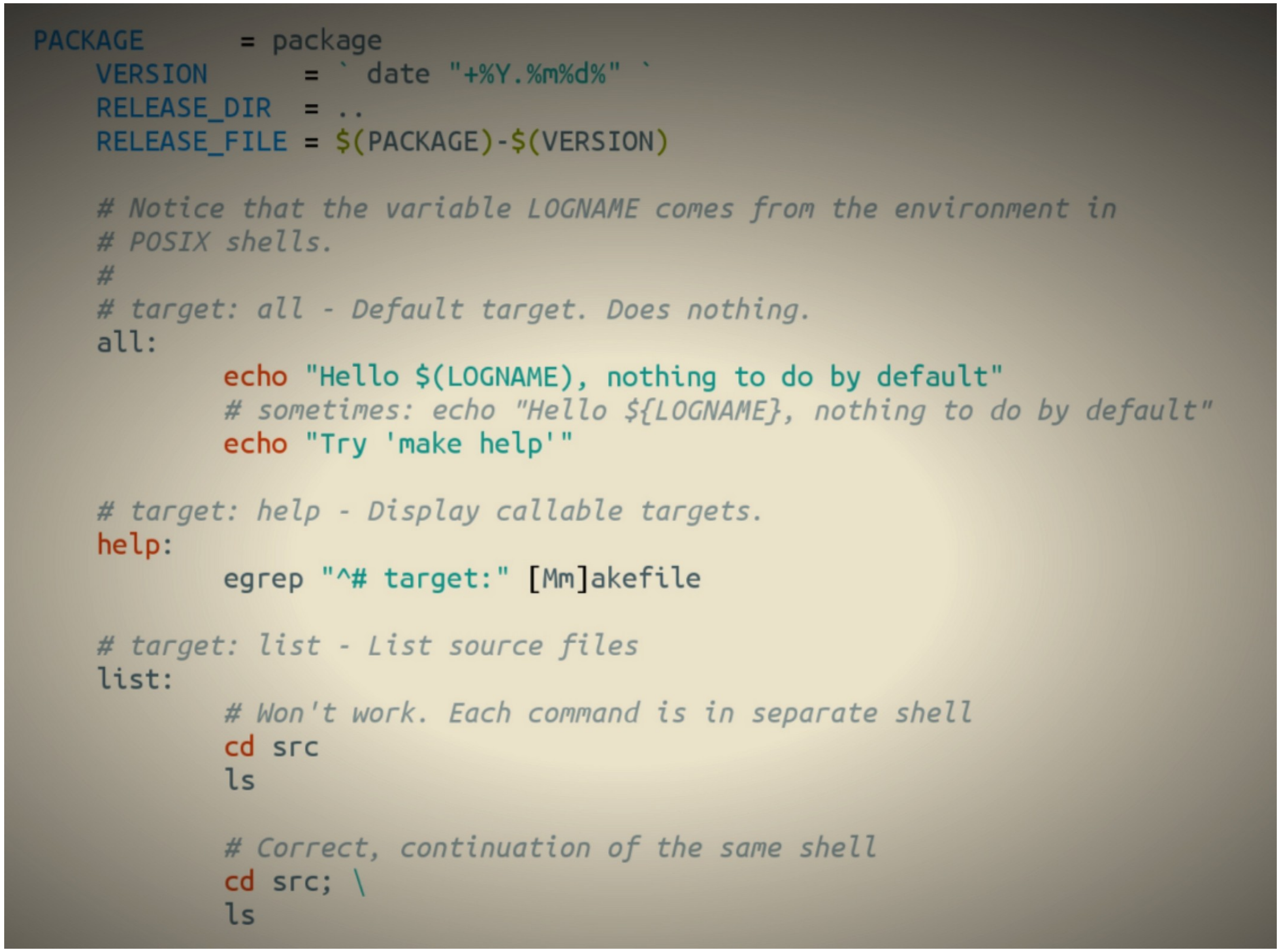
SQL



```sql
SELECT Book.title AS Title,
       COUNT(*) AS Authors
 FROM  Book
 JOIN  Book_author
   ON  Book.isbn = Book_author.isbn
 GROUP BY Book.title;

 INSERT INTO example
 (field1, field2, field3)
 VALUES
 ('test', 'N', NULL);
```

Domain: database (query)

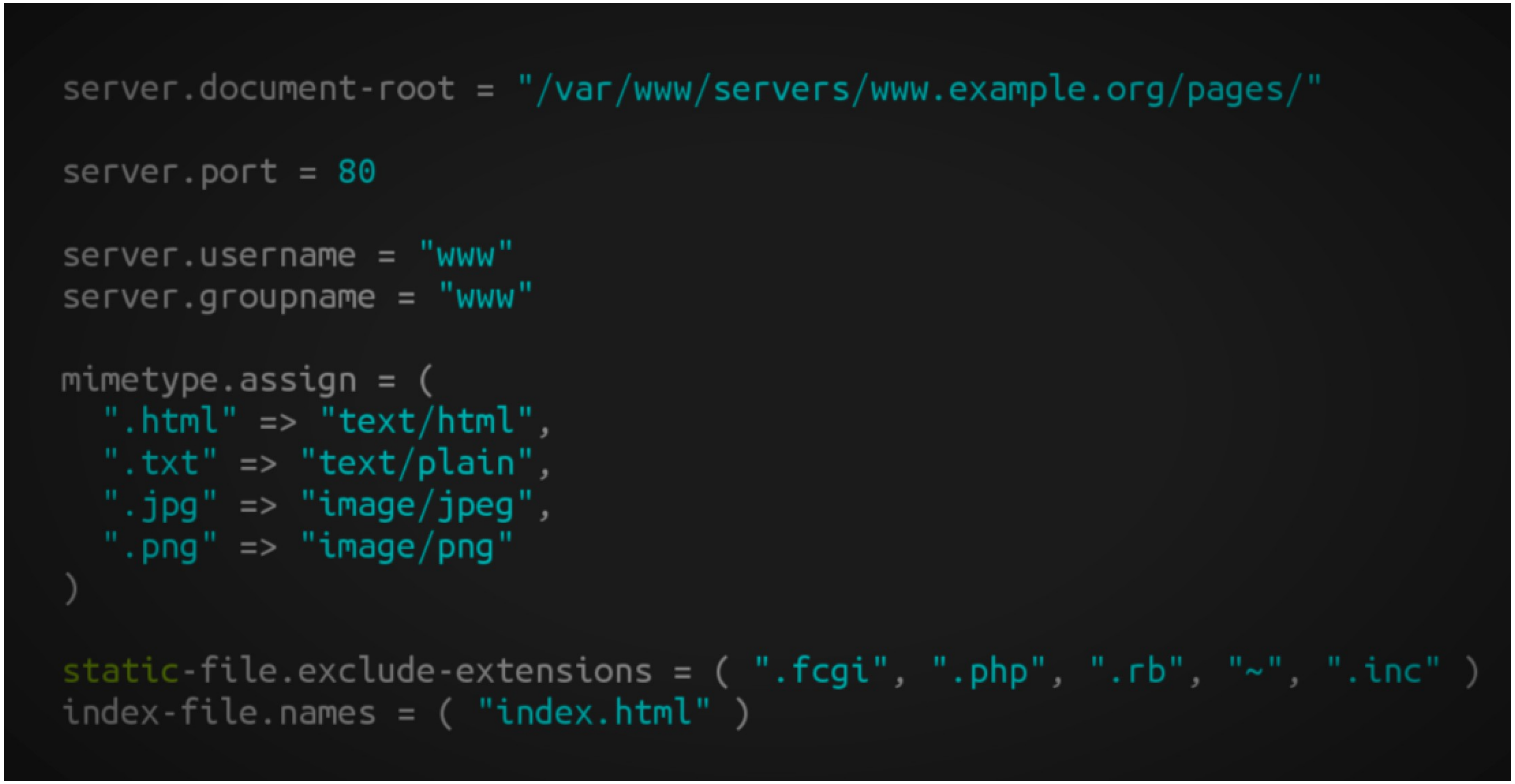
Makefile



```
PACKAGE        = package
    VERSION        = ` date "+%Y.%m%d%" `
    RELEASE_DIR  = ..
    RELEASE_FILE = $(PACKAGE)-$(VERSION)

    # Notice that the variable LOGNAME comes from the environment in
    # POSIX shells.
    #
    # target: all - Default target. Does nothing.
    all:
            echo "Hello $(LOGNAME), nothing to do by default"
            # sometimes: echo "Hello ${LOGNAME}, nothing to do by default"
            echo "Try 'make help'"

    # target: help - Display callable targets.
    help:
            egrep "^# target:" [Mm]akefile

    # target: list - List source files
    list:
            # Won't work. Each command is in separate shell
            cd src
            ls

            # Correct, continuation of the same shell
            cd src; \
            ls
```

Domain: software building

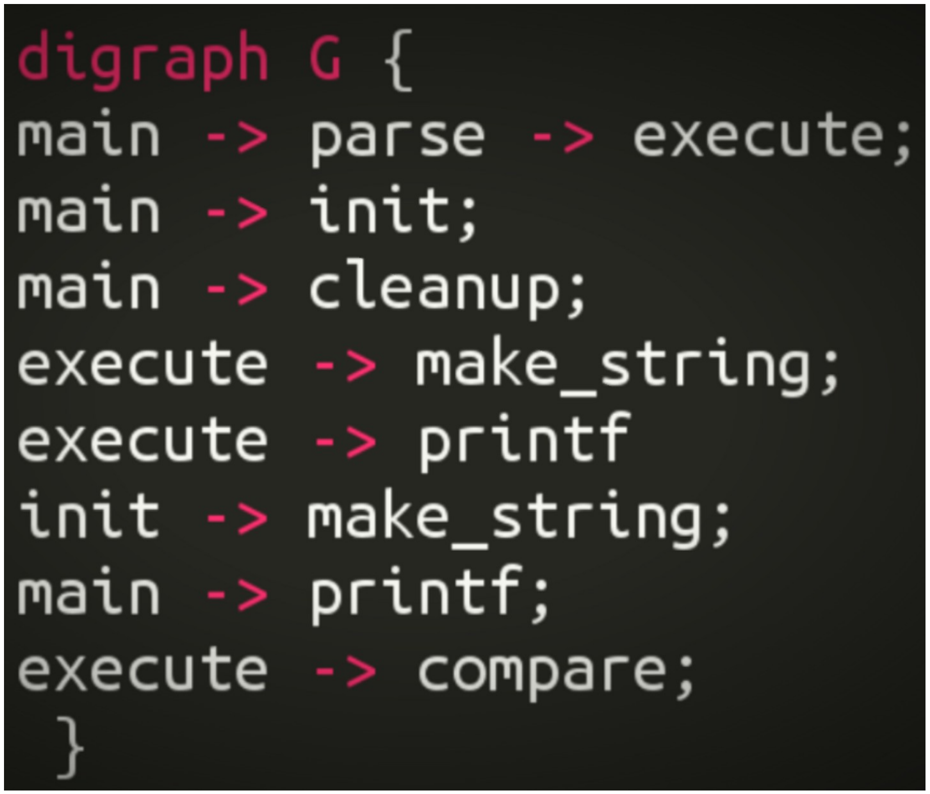
Lighthttpd configuration file



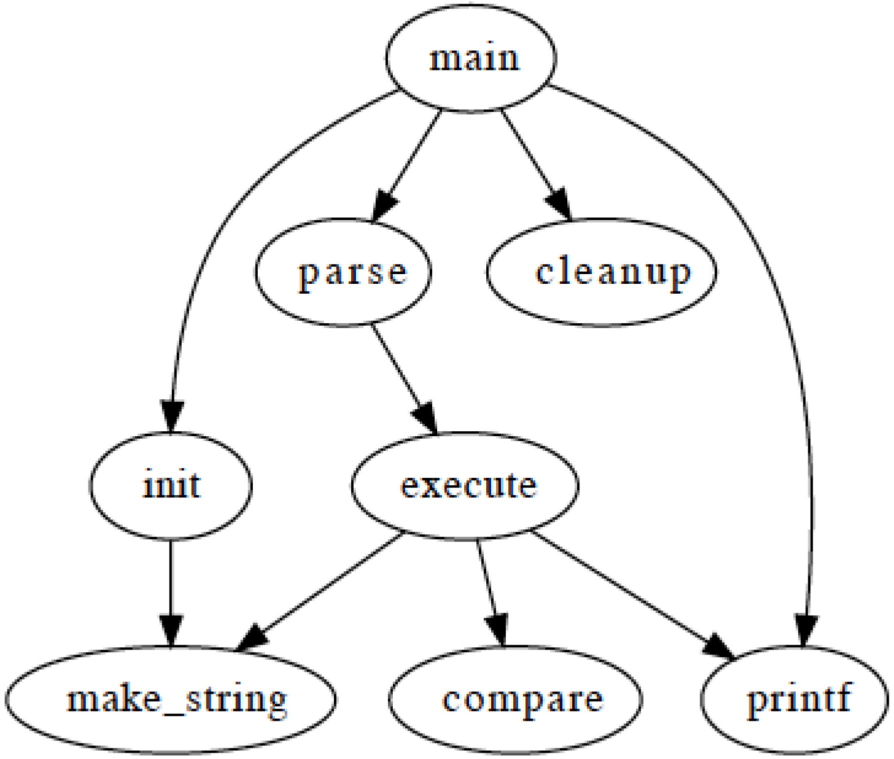
```
server.document-root = "/var/www/servers/www.example.org/pages/"

server.port = 80

server.username = "www"
server.groupname = "www"

mimetype.assign = (
  ".html" => "text/html",
  ".txt" => "text/plain",
  ".jpg" => "image/jpeg",
  ".png" => "image/png"
)

static-file.exclude-extensions = ( ".fcgi", ".php", ".rb", "~", ".inc" )
index-file.names = ( "index.html" )
```

Domain: web server (configuration)

Graphviz

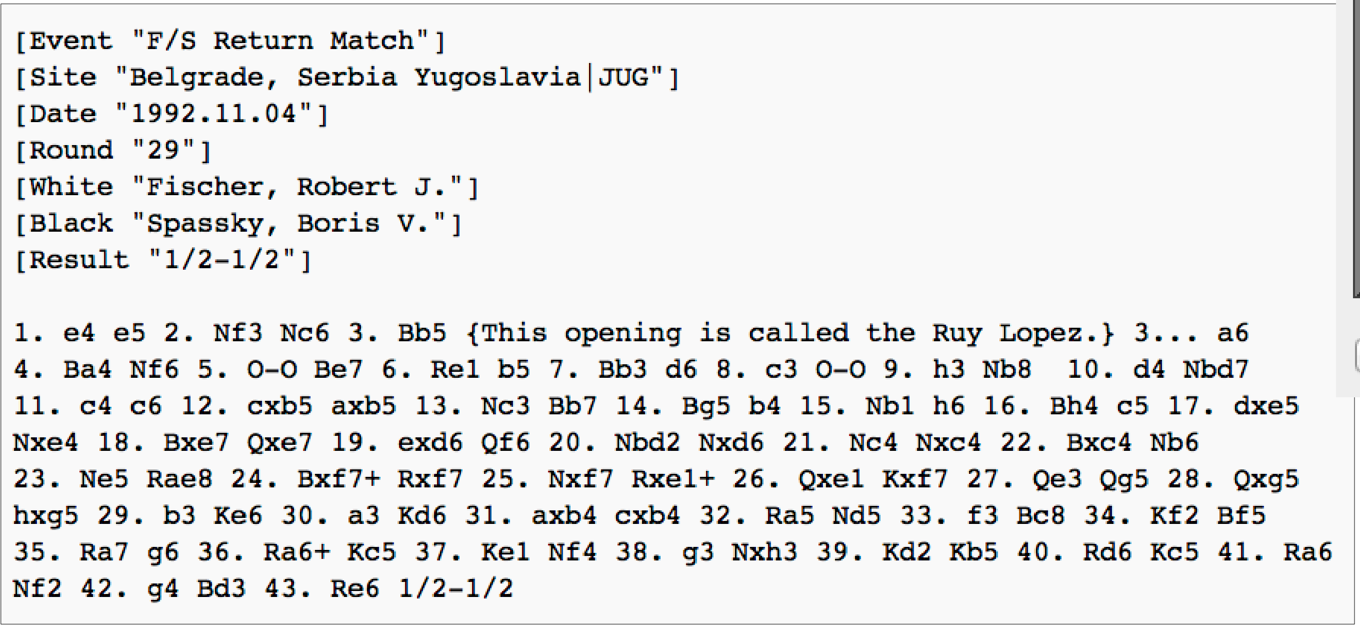


```
digraph G {
main -> parse -> execute;
main -> init;
main -> cleanup;
execute -> make_string;
execute -> printf
init -> make_string;
main -> printf;
execute -> compare;
 }
```

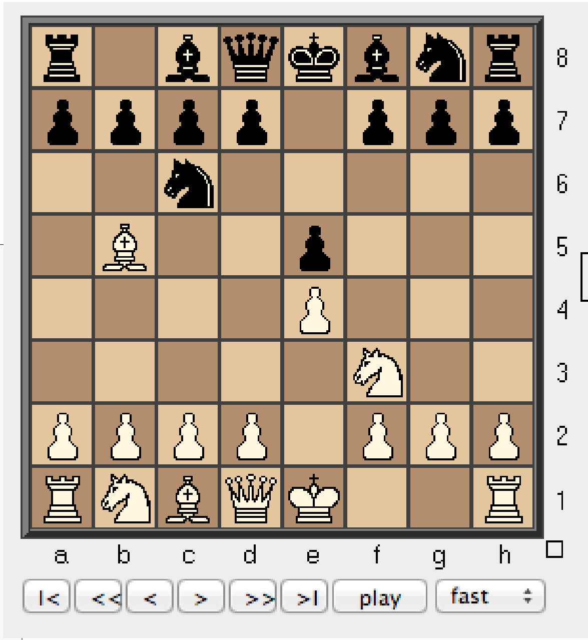


Domain: graph (drawing)

PGN (Portable Game Notation)

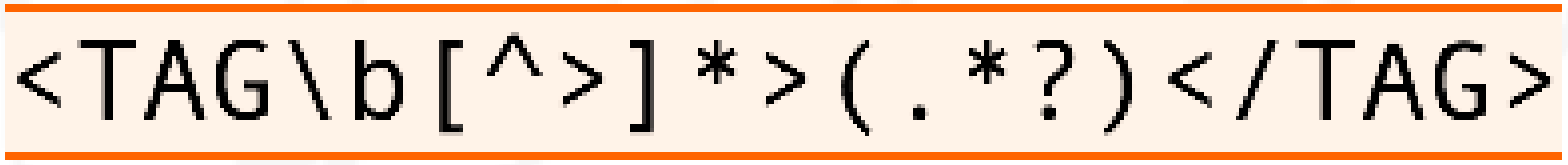


```
[Event "F/S Return Match"]
[Site "Belgrade, Serbia Yugoslavia|JUG"]
[Date "1992.11.04"]
[Round "29"]
[White "Fischer, Robert J."]
[Black "Spassky, Boris V."]
[Result "1/2-1/2"]

1. e4 e5 2. Nf3 Nc6 3. Bb5 {This opening is called the Ruy Lopez.} 3... a6
4. Ba4 Nf6 5. O-O Be7 6. Re1 b5 7. Bb3 d6 8. c3 O-O 9. h3 Nb8  10. d4 Nbd7
11. c4 c6 12. cxb5 axb5 13. Nc3 Bb7 14. Bg5 b4 15. Nb1 h6 16. Bh4 c5 17. dxe5
Nxe4 18. Bxe7 Qxe7 19. exd6 Qf6 20. Nbd2 Nxd6 21. Nc4 Nxc4 22. Bxc4 Nb6
23. Ne5 Rae8 24. Bxf7+ Rxf7 25. Nxf7 Rxe1+ 26. Qxe1 Kxf7 27. Qe3 Qg5 28. Qxg5
hxg5 29. b3 Ke6 30. a3 Kd6 31. axb4 cxb4 32. Ra5 Nd5 33. f3 Bc8 34. Kf2 Bf5
35. Ra7 g6 36. Ra6+ Kc5 37. Ke1 Nf4 38. g3 Nxh3 39. Kd2 Kb5 40. Rd6 Kc5 41. Ra6
Nf2 42. g4 Bd3 43. Re6 1/2-1/2
```



Domain: chess (games)

Regular expression



```
<TAG\b[^>]*>(.*?)</TAG>
```

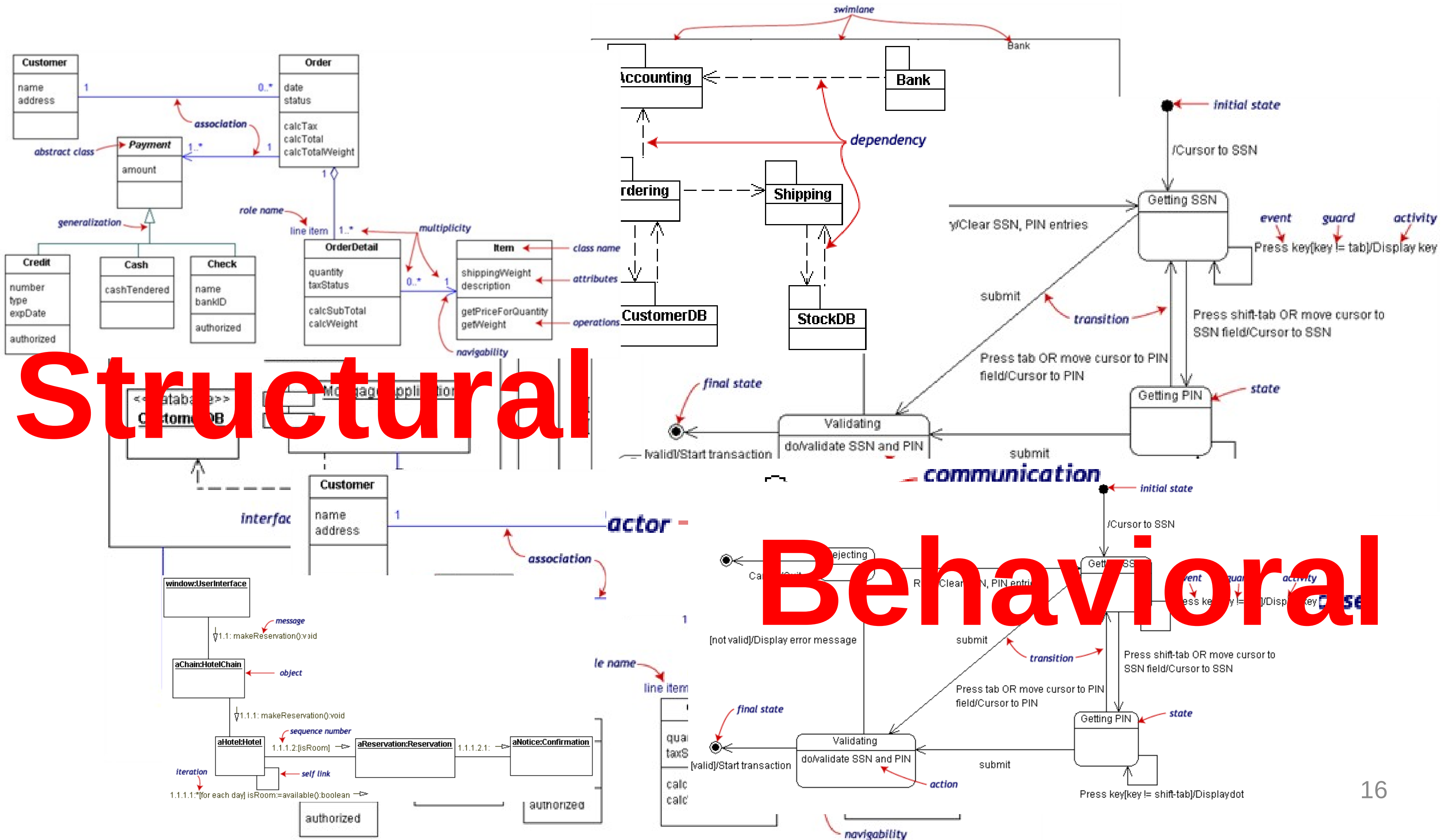
Domain: strings (pattern matching)

OCL

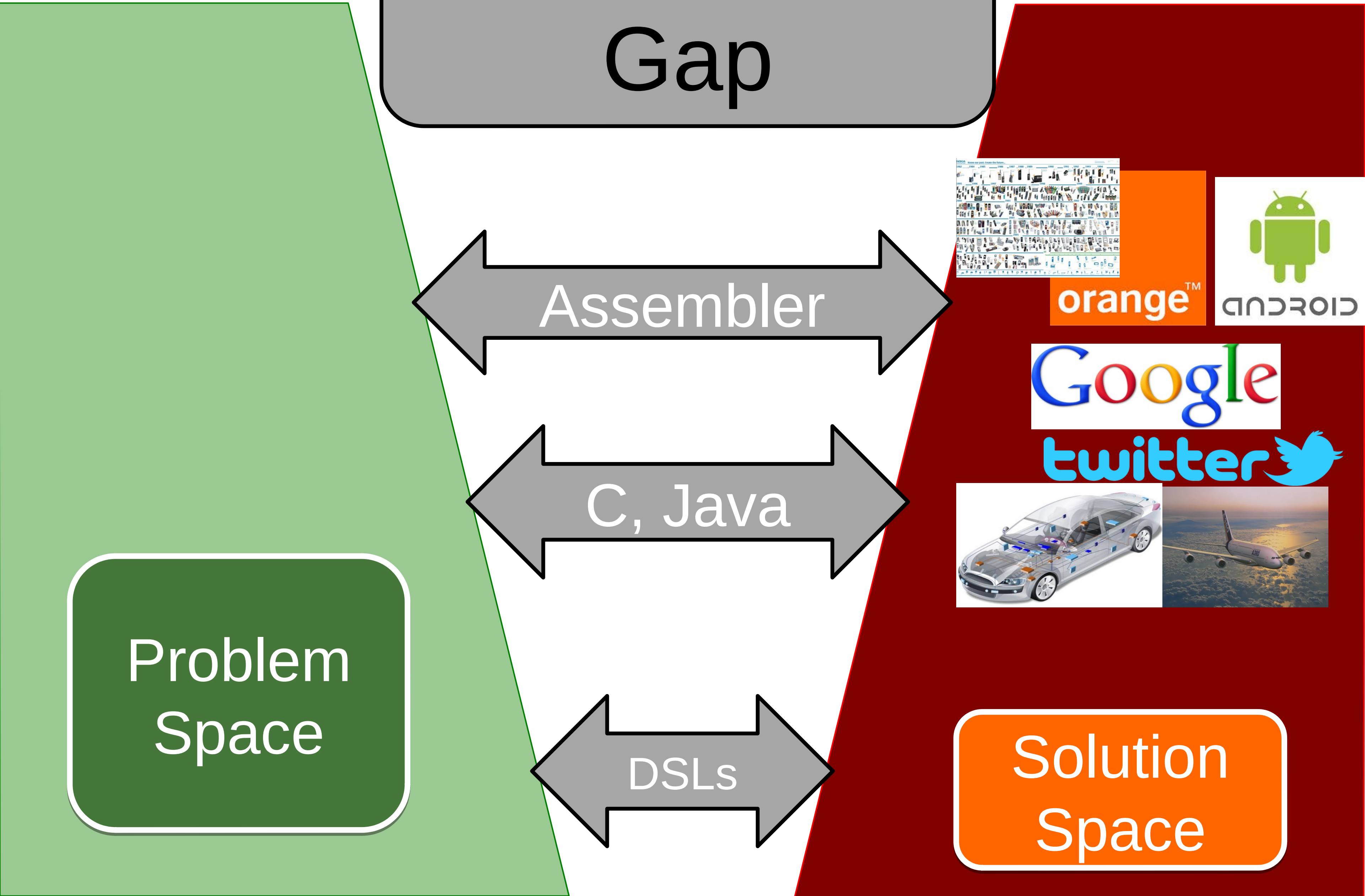
```
self.questions->size
self.employer->size
self.employee->select (v | v.wages>10000 )->size
Student.allInstances
  ->forAll( p1, p2 |
            p1 <> p2 implies p1.name <> p2.name )
```

Domain: model management

UML can be seen as a collection of domain-specific modeling languages



Abstraction Gap



« Another lesson we should have learned from the recent past is that the development of 'richer' or 'more powerful' programming languages was a mistake in the sense that these baroque monstrosities, these conglomerations of idiosyncrasies, are really unmanageable, both mechanically and mentally.

aka **General-Purpose Languages**

I see a great future for very systematic and very modest programming languages »

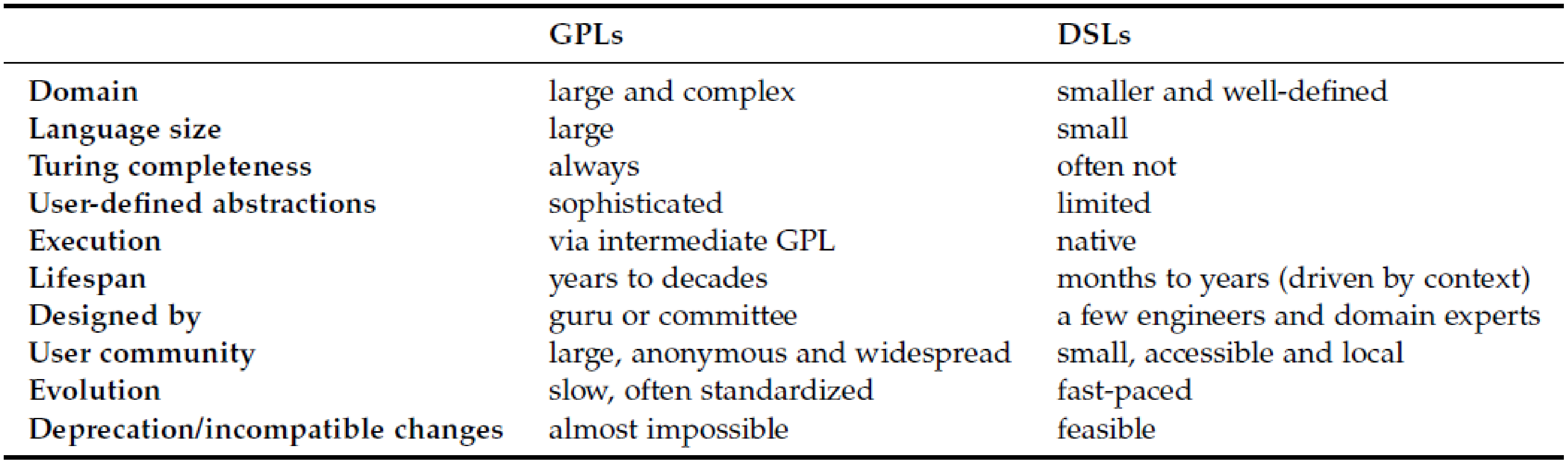
aka **Domain-Specific Languages**

1972

ACM Turing Lecture, « The Humble Programmer »
Edsger W. Dijkstra

GeneralPL vs DomainSL

The boundary isn't as clear as it could be. Domain-specificity is not black-and-white, but instead gradual: a language is more or less domain specific



	GPLs	DSLs
Domain	large and complex	smaller and well-defined
Language size	large	small
Turing completeness	always	often not
User-defined abstractions	sophisticated	limited
Execution	via intermediate GPL	native
Lifespan	years to decades	months to years (driven by context)
Designed by	guru or committee	a few engineers and domain experts
User community	large, anonymous and widespread	small, accessible and local
Evolution	slow, often standardized	fast-paced
Deprecation/incompatible changes	almost impossible	feasible

Specializing syntax and environment pays off?

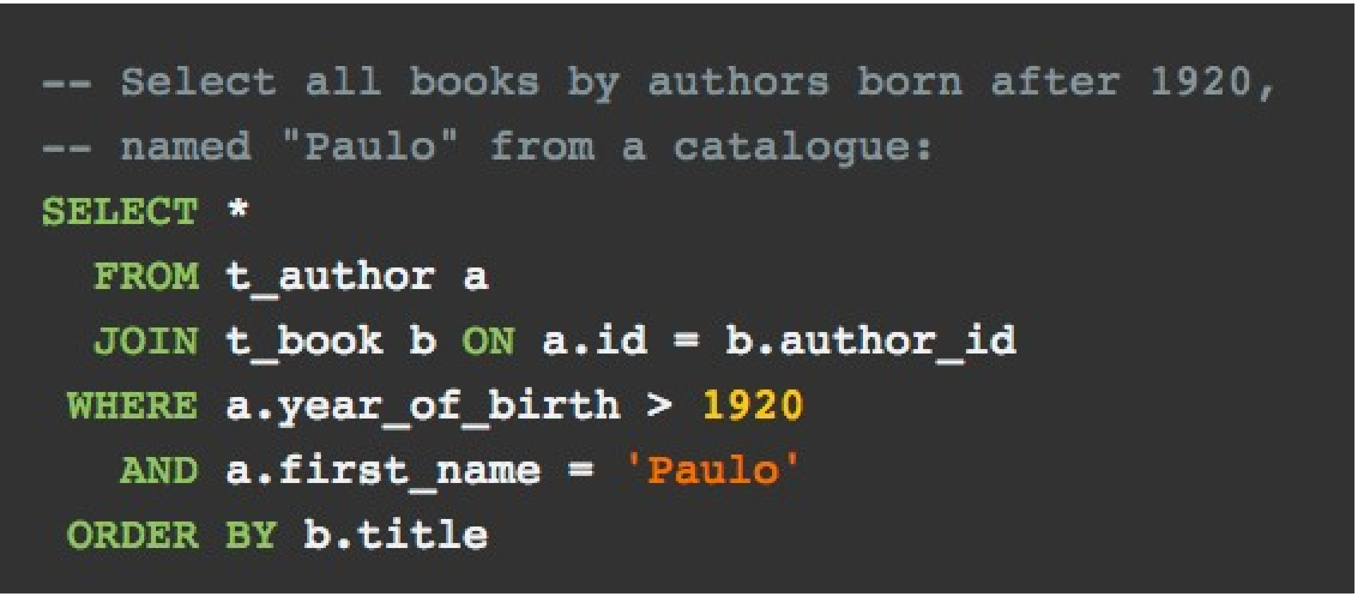
- Promises of DSL« improvement » in terms of
  - usability, learnability, expressiveness, reusability, etc.
- Empirical study on the role of syntax
  - C-style syntax induces problems in terms of usability for novices; language more or less intuitive for (non-)programmers (Stefik et al. 2014)
  - Syntax issues with Java for students (Denny et al. 2011)
  - PL usability: method namings/placement, use of identifiers, API design (Ellis et al., Styllos et al., Clarke, Montperrus et al., etc.)
- More specialized/sophicated tools/IDE can be derived from a DSL
  - editors, compilers, debuggers

External DSLs vs Internal DSLs

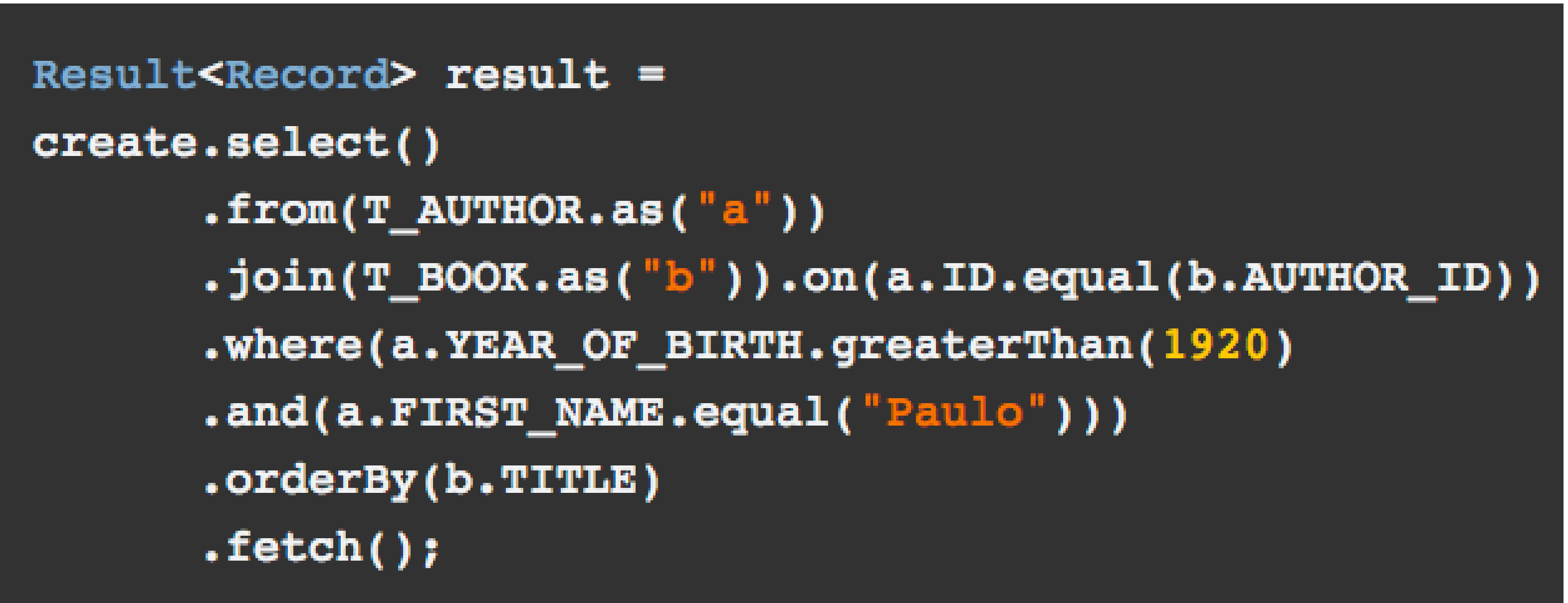
- An **external** DSL is a completely separate language and has its own custom syntax/tooling support (e.g., editor)

- An internal DSL is more or less a set of APIs written on top of a host language (e.g., Java).
  - Fluent interfaces

External vs Internal DSL (SQL example)



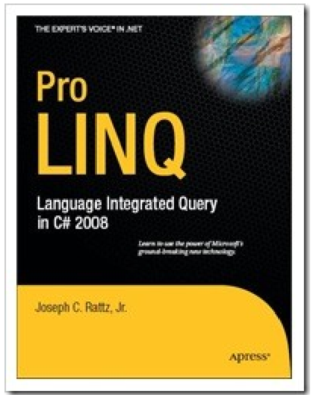
```
-- Select all books by authors born after 1920,
-- named "Paulo" from a catalogue:
SELECT *
  FROM t_author a
  JOIN t_book b ON a.id = b.author_id
 WHERE a.year_of_birth > 1920
   AND a.first_name = 'Paulo'
 ORDER BY b.title
```



```
Result<Record> result =
create.select()
      .from(T_AUTHOR.as("a"))
      .join(T_BOOK.as("b")).on(a.ID.equal(b.AUTHOR_ID))
      .where(a.YEAR_OF_BIRTH.greaterThan(1920)
      .and(a.FIRST_NAME.equal("Paulo")))
      .orderBy(b.TITLE)
      .fetch();
```

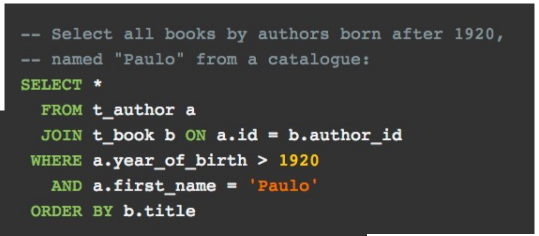
Internal DSL (LINQ/C# example)

```
// DataContext takes a connection string
DataContext db = new   DataContext("c:\\northwind\\northwnd.mdf");
// Get a typed table to run queries
Table<Customer> Customers = db.GetTable<Customer>();
// Query for customers from London
var q =
   from c in Customers
   where c.City == "London"
   select c;
foreach (var cust in q)
   Console.WriteLine("id = {0}, City = {1}", cust.CustomerID, cust.City);
```

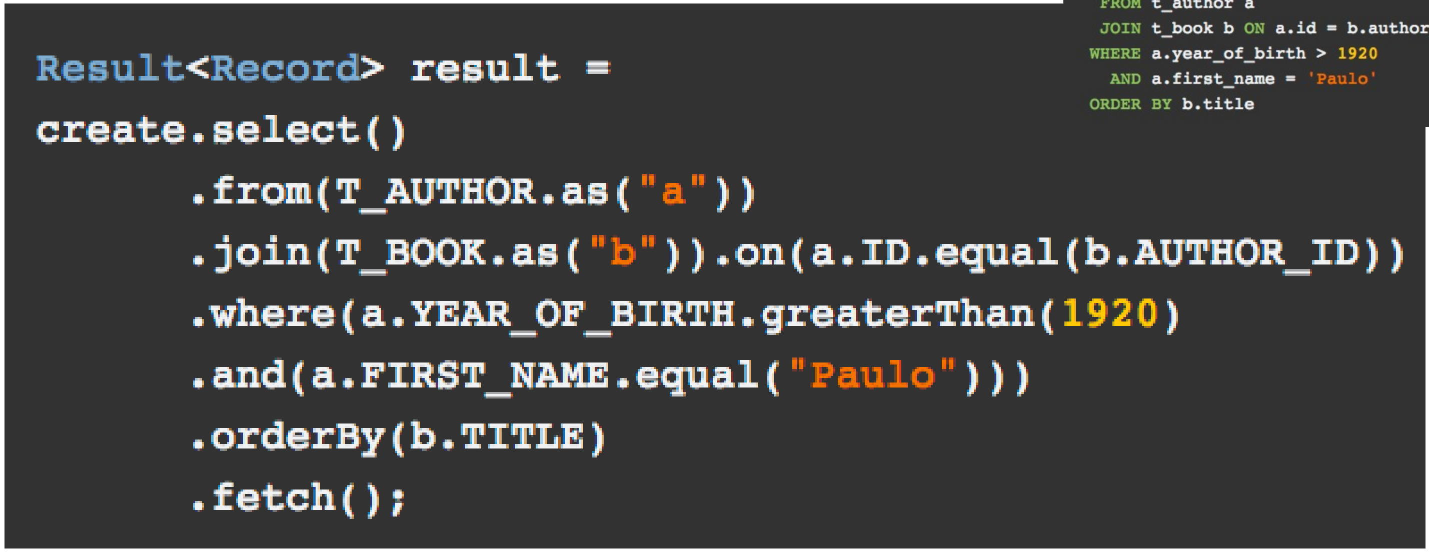

THE EXPERT'S VOICE® IN .NET
Pro
LINQ
Language Integrated Query in C# 2008
Joseph C. Rattz, Jr.
Apress®

Internal DSL

- « Using a host language (e.g., Java) to give the host language the feel of a particular language. »
- **Fluent** Interfaces
  - « The more the use of the API has that language like flow, the more fluent it is »



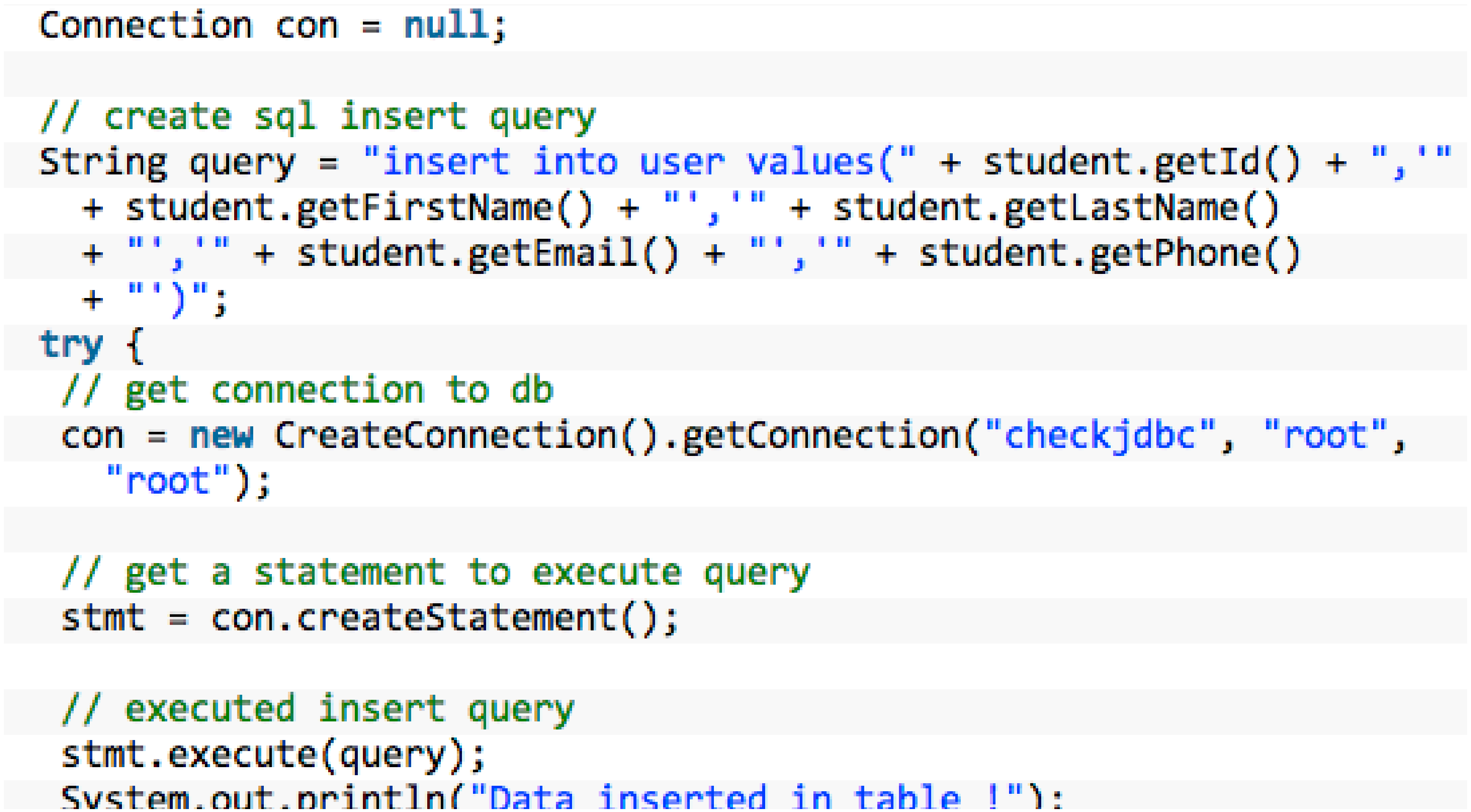
```sql
-- Select all books by authors born after 1920,
-- named "Paulo" from a catalogue:
SELECT *
  FROM t_author a
  JOIN t_book b ON a.id = b.author_id
 WHERE a.year_of_birth > 1920
   AND a.first_name = 'Paulo'
 ORDER BY b.title
```



```java
Result<Record> result =
create.select()
      .from(T_AUTHOR.as("a"))
      .join(T_BOOK.as("b")).on(a.ID.equal(b.AUTHOR_ID))
      .where(a.YEAR_OF_BIRTH.greaterThan(1920)
      .and(a.FIRST_NAME.equal("Paulo")))
      .orderBy(b.TITLE)
      .fetch();
```

SQL in... Java

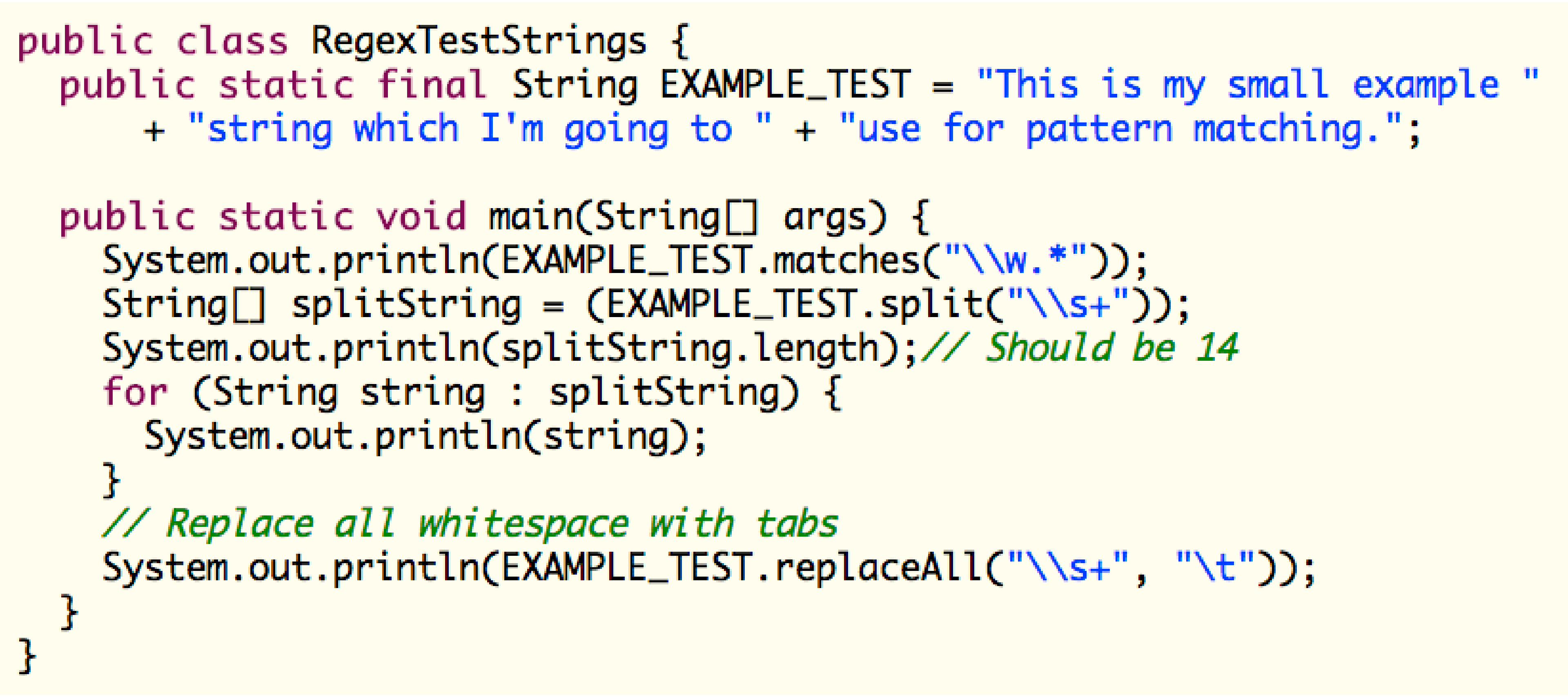
DSL in GPL



```
Connection con = null;

// create sql insert query
String query = "insert into user values(" + student.getId() + ",'"
   + student.getFirstName() + "','" + student.getLastName()
   + "','" + student.getEmail() + "','" + student.getPhone()
   + "')";
try {
 // get connection to db
 con = new CreateConnection().getConnection("checkjdbc", "root",
   "root");

 // get a statement to execute query
 stmt = con.createStatement();

 // executed insert query
 stmt.execute(query);
 System.out.println("Data inserted in table !");
```

Regular expression in... Java

DSL in GPL



```java
public class RegexTestStrings {
  public static final String EXAMPLE_TEST = "This is my small example "
      + "string which I'm going to " + "use for pattern matching.";

  public static void main(String[] args) {
    System.out.println(EXAMPLE_TEST.matches("\\w.*"));
    String[] splitString = (EXAMPLE_TEST.split("\\s+"));
    System.out.println(splitString.length);// Should be 14
    for (String string : splitString) {
      System.out.println(string);
    }
    // Replace all whitespace with tabs
    System.out.println(EXAMPLE_TEST.replaceAll("\\s+", "\t"));
  }
}
```

Internal DSLs vs External DSL

- Both internal and external DSLs have strengths and weaknesses
  - learning curve,
  - cost of building,
  - programmer familiarity,
  - communication with domain experts,
  - mixing in the host language,
  - strong expressiveness boundary
- Focus of the course
  - **external DSL** a completely separate language with its own custom syntax and tooling support (e.g., editor)

References

- Martin Fowler. Domain Specific Languages. Addison-Wesley Professional, 2010.
- Markus Voelter et al. “DSL Engineering: Designing, Implementing and Using Domain-Specific Languages.” dslbook.org, 2013.
- Kramer “Abstraction and Modelling - A Complementary Partnership” MODELS’08
- Tarr et al. “N Degrees of Separation: Multi-Dimensional Separation of Concerns” ICSE’99
- Benoit Combemale, Julien Deantoni, Benoit Baudry, Robert France, Jean-Marc Jézéquel, and Jeff Gray. « Globalizing Modeling Languages.” Computer, 2014.

References

- Leo A Meyerovich and Ariel S Rabkin. “Empirical analysis of programming language adoption” OOPSLA’13
- Felienne Hermans, Martin Pinzger, and Arie van Deursen. “Domain-Specific languages in practice: A user study on the success factors.“ MODELS’09
- Paul Denny, Andrew Luxton-Reilly, Ewan Tempero, and Jacob Hendrickx. “Understanding the syntax barrier for novices.” ITiCSE ’11
- Tiark Rompf et al . “Optimizing Data Structures in High-Level Programs: New Directions for Extensible Compilers based on Staging” POPL’13

References

- Mathieu Acher, Benoît Combemale, Philippe Collet: “Metamorphic Domain-Specific Languages: A Journey into the Shapes of a Language.” Onward! 2014
- Jeffrey Stylos and Brad A. Myers. “The implications of method placement on api learnability” FSE’08
- Martin Monperrus, Michael Eichberg, Elif Tekes, and Mira Mezini. “What Should Developers Be Aware Of? An Empirical Study on the Directives of API Documentation”. Empirical Software Engineering, 17(6):703–737, 2012.

Plan

- Domain-Specific Languages (DSLs)
  - Languages and abstraction gap
  - Examples and rationale
  - DSLs vs General purpose languages, taxonomy
- External DSLs
  - Grammar and parsing
  - Xtext
- DSLs, DSMLs, and (meta-)modeling

Contract

- Better understanding/source of inspiration of software languages and DSLs
  - Revisit of history and existing languages

- Foundations and practice of Xtext
  - State-of-the-art language workbench (Most Innovative Eclipse Project in 2010, mature and used in a variety of industries)

- Models and Languages
  - Perhaps a more concrete way to see models, metamodels and MDE (IDM in french)