Compiler und Programmtransformation

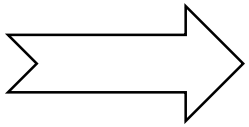
Henning Bordihn

Institut für Informatik und Computational Science

Universität Potsdam

Einige der Folien gehen auf B. Steffen und O. Rüthing (TU Dortmund) zurück.

Compilation



WAS-Beschreibung

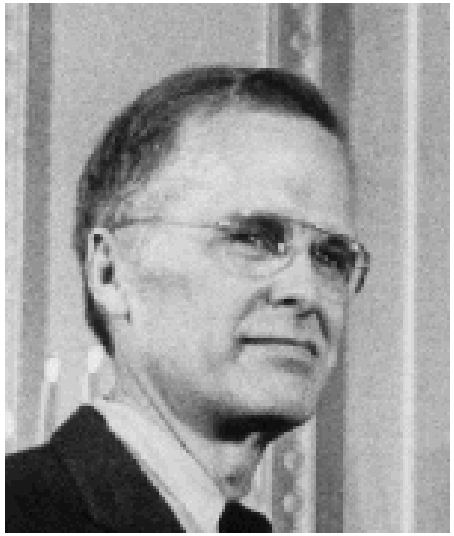
WIE-Beschreibung

Compilation

Ursprünge

50er Jahre: FORTRAN [1957]

John Backus: *We certainly had no idea that languages almost identical to the one we were working on would be used for more than one IBM computer, not to mention those of other manufactures.*



Compilation

Entwicklung

WAS-Beschreibungen:

- Höhere Programmiersprachen (deklarativ, objektorientiert, ...)
- Domänenspezifische Sprachen (z.B. **G**ame **P**rogramming **L**anguage)
- Verteilung (Module, Bibliotheken, Aspekte, ...)
- Modellierungssprachen (UML, BPEL, ...)
- …

Compilation

Entwicklung

WIE-Beschreibungen:

- Architekturen (Multicore, Threading, ..)
- Speicher (Register, Caches, Persistenz,..)
- Verteilung (Client-Server, Cluster, Internet, ..)
- …

Virtualisierung

Fazit: WAS-WIE-Graben ist in den letzten Jahrzehnten stark gewachsen

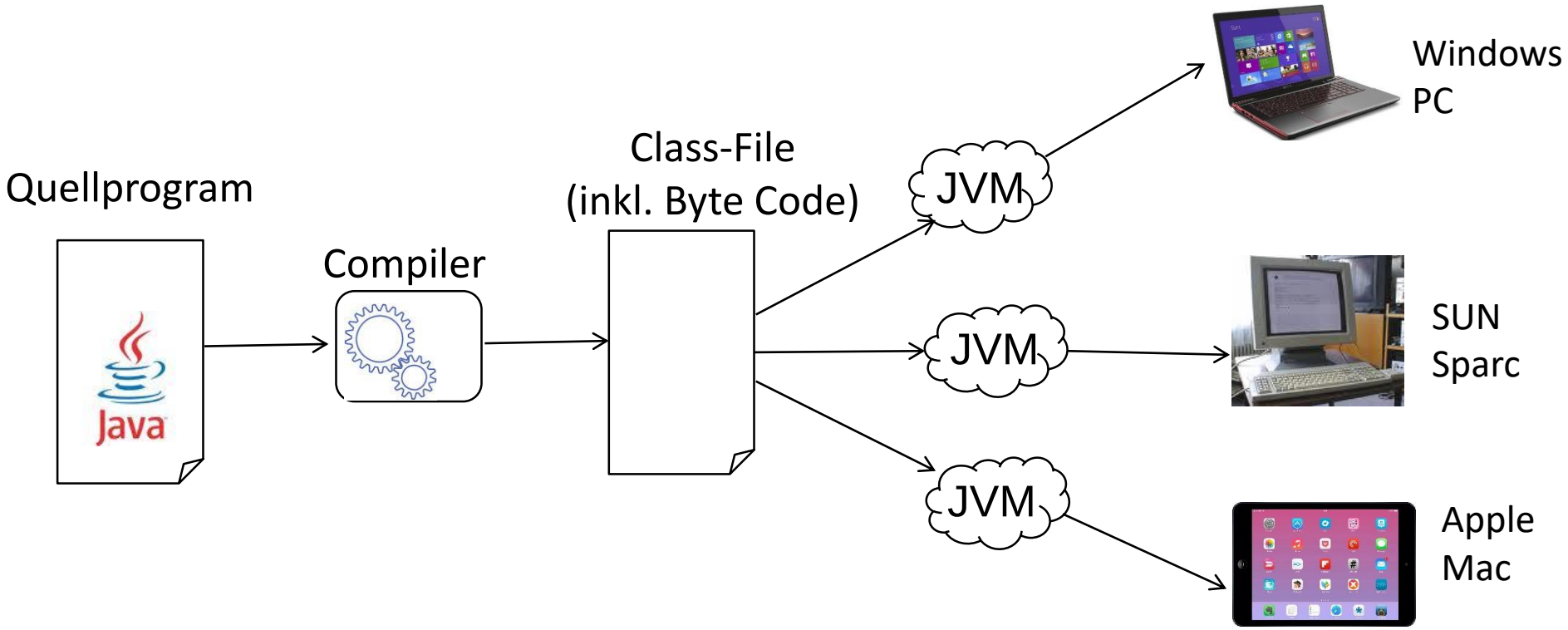
Lösung: Virtualisierungsebenen

- Abstraktion von spezifischen Details der Plattformen
- Unterstützung der automatischen Codeerzeugung aus den WAS-Beschreibungen (Compilation)

  → *modellgetriebene Softwareentwicklung (MDD / MDSD)*

Virtualisierung

Beispiel: Java Virtual Machine



Vorlesungsplan

Teil 1: Compiler (eher klassisch)

- Lexikalische Analyse, Syntaxanalyse
- Statisch semantische Analyse
- Codegenerierung, Optimierung

Teil 2: Neue Herausforderungen

- Neue Sprachniveaus, neue Plattformen
- Definition und Compilation domänenspezifischer Sprachen

Begleitend:

Praktische Projekte mit Tooleinsatz

AntLR, Xtext

Organisation

- Vorlesung (2 SWS) + Übungen (2 SWS)

- **Termine:**

  1. Termin/“Vorlesung“: donnerstags, 12:15 – 13:45 (Haus 70, 0.09)
  2. Termin/“Übung“: freitags, 14:15 – 15:45 (Haus 70, 0.11)

- Vorlesungen/Übungen wechseln nach Bedarf

- Material (Folien & Übungsaufgaben) → Webseite (cupt / tpuc!)
  - Folien: englisch
  - Übungsaufgaben, Vortrag: deutsch

Modalitäten

- **Credits:** 6
- **Modulprüfung:** mündliche Prüfung (30 Minuten)
- **Leistungsanforderung:**
  - regelmäßige, aktive Teilnahme an der Übung
  - Übungsaufgaben zu Hause vorbereiten, in der Übung durchsprechen
  - Anfertigung der beiden Praxisprojekte (Pflicht, ggf. PNL) in Teams
- **Voraussetzungen:**
  - etwas Erfahrung im Umgang mit Compilern (Programmierung, IDEs)
  - Basiskenntnisse aus der Theoretischen Informatik
    (endliche und Pushdown-Automaten, reg. Ausdrücke, CFG)

Motivation Part 1

Compilation

Why Study Compilers?

- understand existing languages
- appreciate current limitations
- talk intelligently about language design
- implement your very own general-purpose language
- implement lots of useful domain-specific languages
- see a great example of theory in practice
- process/translate/manage information
  - all aspects of CS

Issues of Compilation

- Understanding **implementation issues**:
  - How is this implemented?
  - Why does this run so slow?
  - Why does the compiler/runtime system report this error message?

- Understand **language design issues**:
  - Why does/doesn‘t the language definition offer this feature?

- Background for **language implementation**:
  - How to write or modify compilers/interpreters?
  - How to improve code quality?

Myth?

"People are better at optimizing their programs than compilers."

```
for (i = 0; i < N; i++)
{
   a[i] = a[i] * 2000;
   a[i] = a[i] / 10000;
}
```

```
b = a;
for (i = 0; i < N; i++)
{
   *b = *b * 2000;
   *b = *b / 10000;
   b++;
}
```

Which loop runs faster?

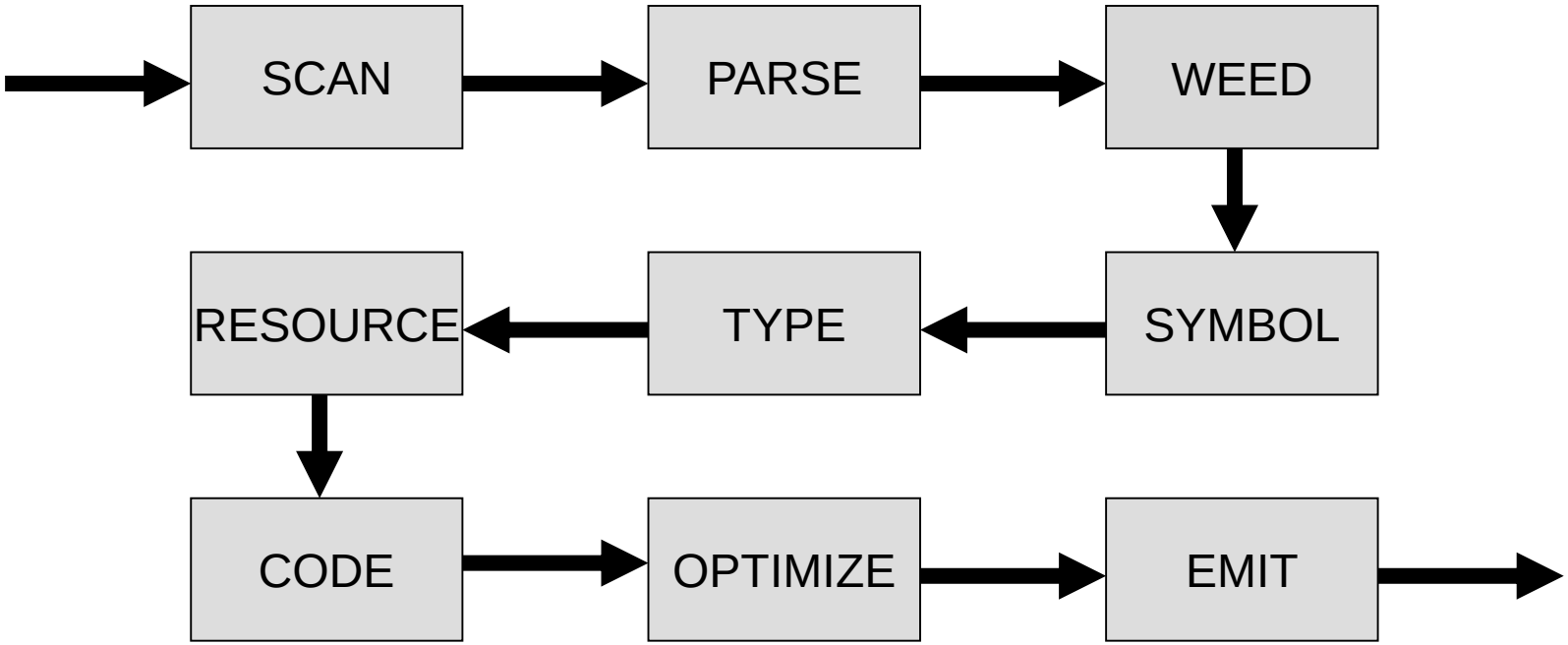
The Answer is …

Loop	Optimization	SPARC	MIPS	Alpha
array	none	20.5	21.6	7.85
array	opt	8.8	12.3	3.26
array	super	7.9	11.2	2.96
pointer	none	19.5	17.6	7.55
pointer	opt	12.4	15.4	4.09
pointer	super	10.7	12.9	3.94

Why?

- **Pointers** confuse most compilers
  - difficult to optimize code using pointers
  - use arrays whenever possible
- Compilers use sophisticated **register allocation** algorithms
  - languages don't give enough control
  - it is too expensive to compute a good assignment by hand
- High-level languages are for **people**
  - write clear code and let the compiler optimize it

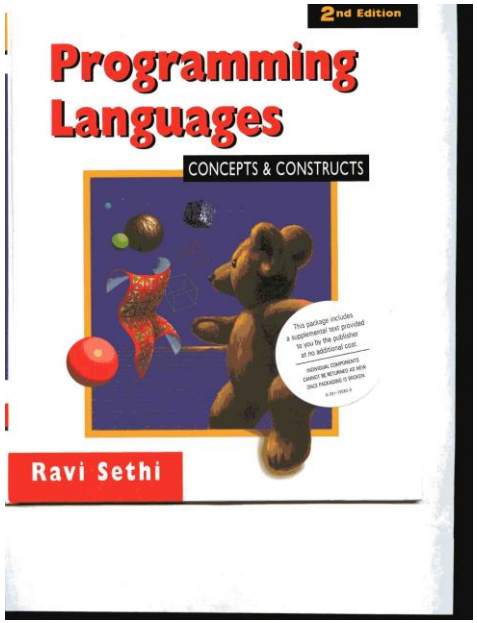
Compilation Phases

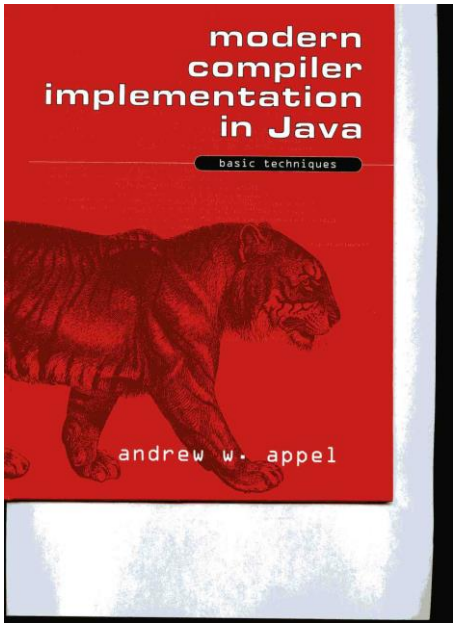


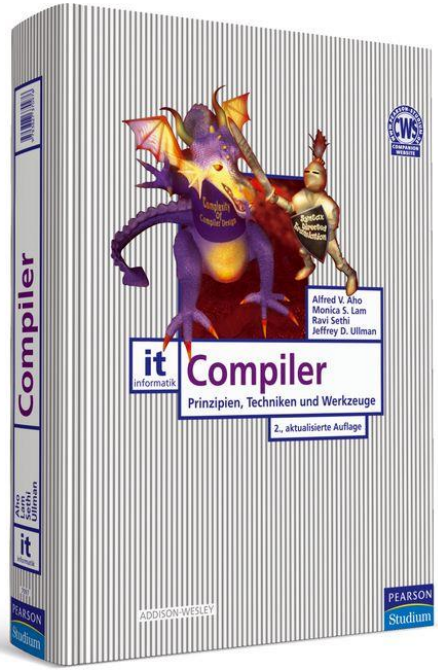
Theory in Practice

- in the 60s *compilation* was **art**
- in the 70s *compilation* was **studied by theoreticians**
- in the 80s *compilation* was studied as a **software product line**, and
- it is probably the most **mature** *software domain* you will ever work in.

Literatur


2nd Edition
Programming Languages
CONCEPTS & CONSTRUCTS
Ravi Sethi


modern compiler implementation in Java
basic techniques
andrew w. appel


Compiler
Alfred V. Aho
Monica S. Lam
Ravi Sethi
Jeffrey D. Ullman
it informatik
Compiler
Prinzipien, Techniken und Werkzeuge
2., aktualisierte Auflage
ADDISON-WESLEY
PEARSON Studium


Reinhard Wilhelm
Helmut Seidl
Compiler Design
Virtual Machines
Springer

Motivation Part 2

New Challenges

Edsger Dijkstra (1972):

"[The major cause of the

software crisis

is] that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem."

Answers …

- Global Libraries
- Millions of Testers
- Enhanced Development Processes
- Better Tool Support
- Lightweight Formal Methods
  - Typing,
  - Runtime Checking,
  - Model Checking,
  - Model-Based Testing.

But: Rarely Formal Verification (except safety critical systems, ...)

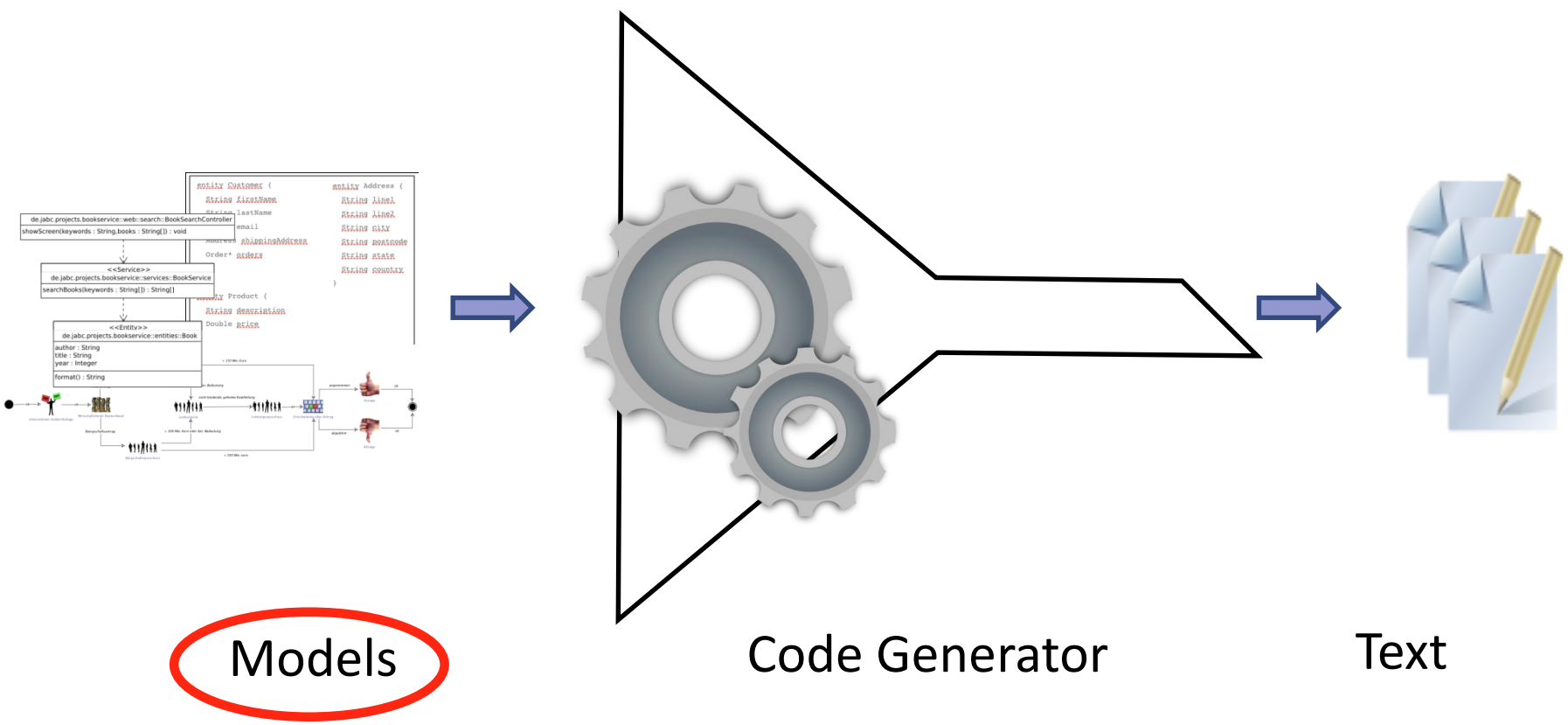
Answers …

Model-Driven Software Development (MDSD)

- Global Libraries
- Millions of Testers
- Enhanced Development Processes
- Better Tool Support
- Lightweight Formal Methods
  - Typing,
  - Runtime Checking,
  - Model Checking,
  - Model-Based Testing.

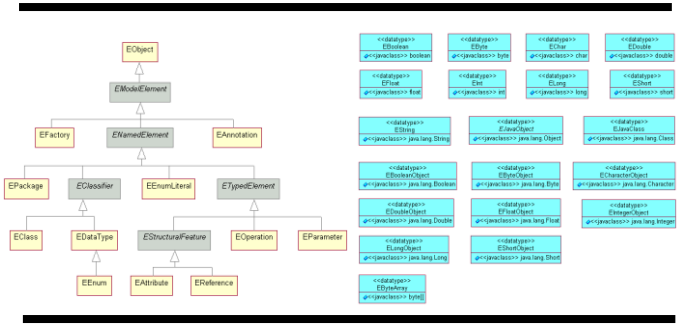
But: Rarely Formal Verification (except safety critical systems, ..)

MDSD: The big picture

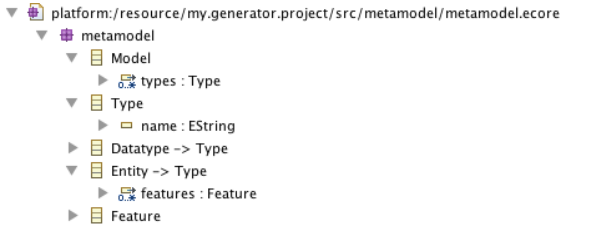


MDSD: Example with ECORE

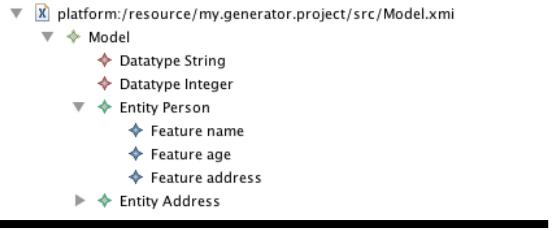
Model Hierarchies



Metametamodel



Metamodel



Model

```
Person bob = new Person();
bob.setName("Bob");
bob.setAge(35);
Address address = new Address();
address.setStreet("Mulholland Drive");
address.setZip("8079");
address.setCity("Los Angeles");
```

Model Instance

MDSD with Xtext

Model-based development of DSL

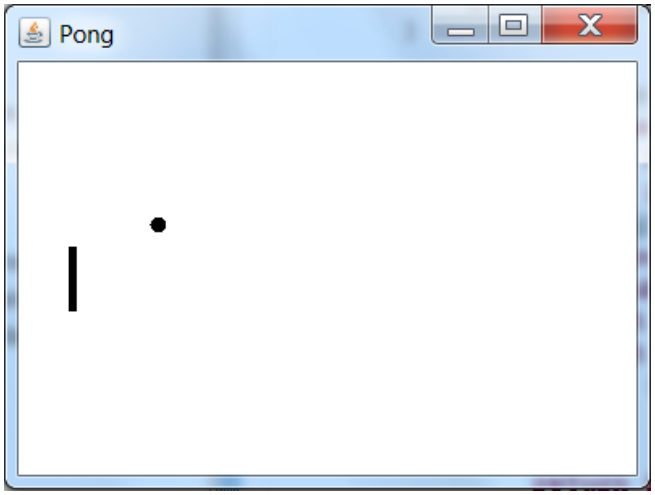
Xtext:

- Eclipse tool
- Meta models: various formats (BNF, UML, ...)
- Models: DSL programs
- Covers all aspects of language infrastructure
  - parsers
  - linkers, compilers (template engine)
  - Eclipse-editors
  - static analyses

Project MGPL
(Mini game programming language)

MGPL specification

```
int paddle_increment =  10;
int ball_x_increment = 5;
int ball_y_increment = 2;
int paddle_width = 5;
int paddle_height = 40;
int ball_size = 10;

// create  a rectangle that will be used for the paddle

rectangle paddle(x = Pong.width/10, y = Pong.height/2, w = paddle_width, h = paddle_hei

// create a rectangle that will be used for the ball
// note that the ball has an animation handler

circle ball(x = Pong.width/2, y = Pong.height/2, radius = ball_size/2, animation_block

// empty initialization block

{ }

// animation handler for the ball
// this block is called at regular intervals for the ball object

animation ball_animate(circle cur_ball)
{
    // if ball has reached either the left or right, reverse its direction
    if  (cur_ball.x < 0  &&  ball_x_increment < 0 || Pong.width - ball_size < cur_ball.
        {ball_x_increment = -ball_x_increment;}


    // if ball has reached either the top or bottom, reverse its direction
    if (cur_ball.y < 0 || Pong.height - ball_size < cur_ball.y)
       {ball_y_increment = -ball_y_increment;}
```



Running acarde game