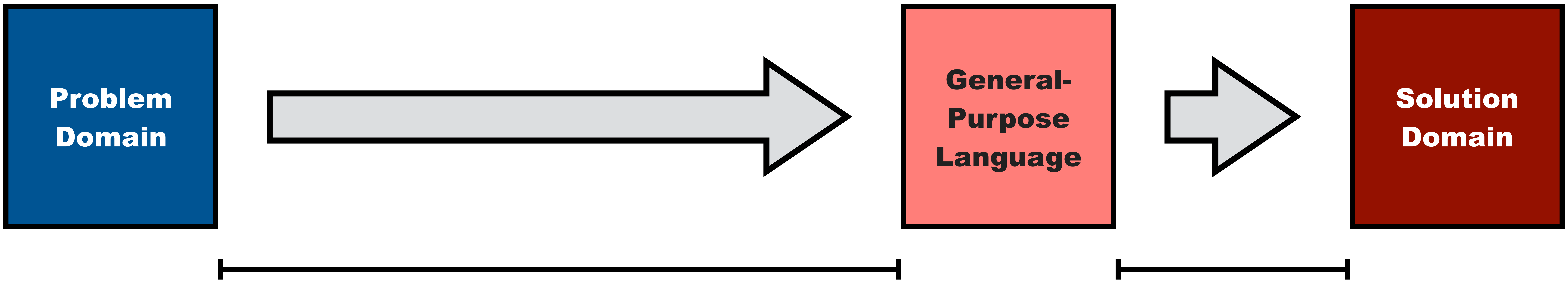
Separation of Concerns in Language Definition

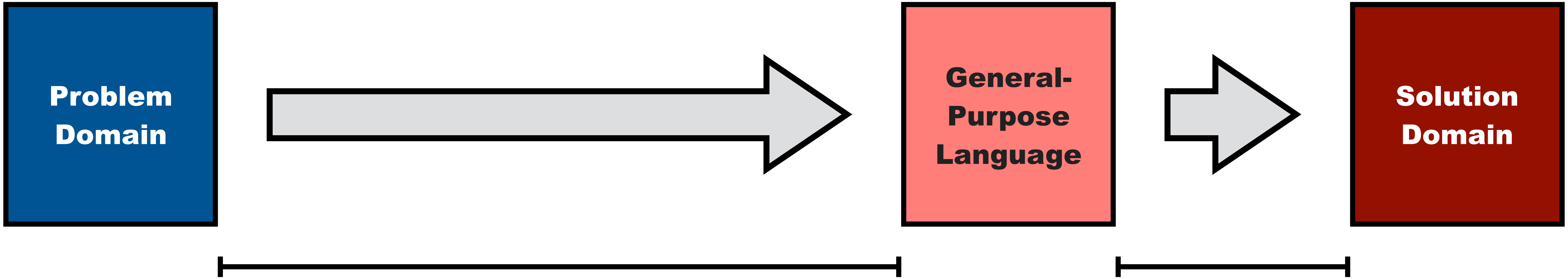
Eelco Visser
Delft University of Technology

TUDelft

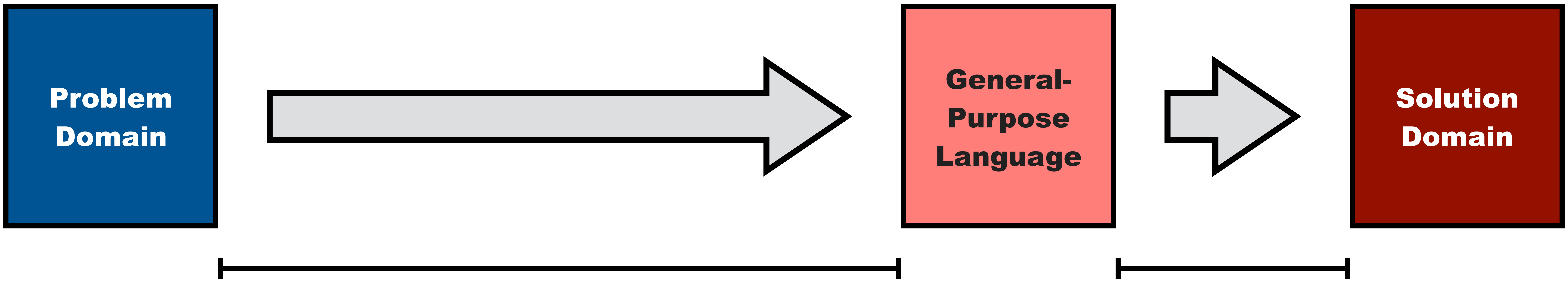
“99% of errors found by Semmle are due to bad language design”

— Oege de Moor, CEO Semmle

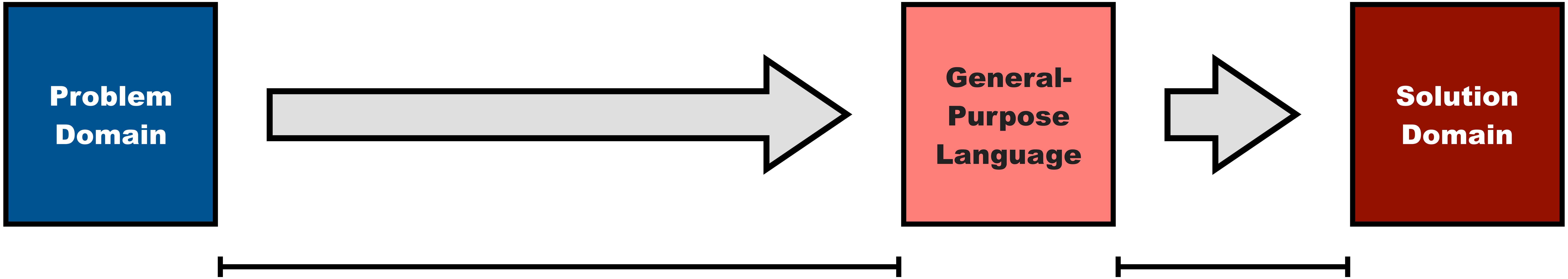

Problem Domain
General-Purpose Language
Solution Domain



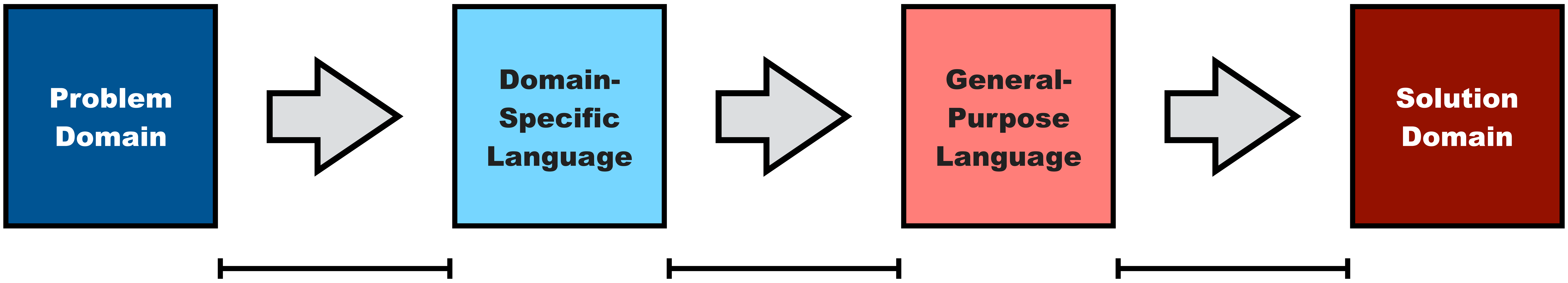
- Lack of safety



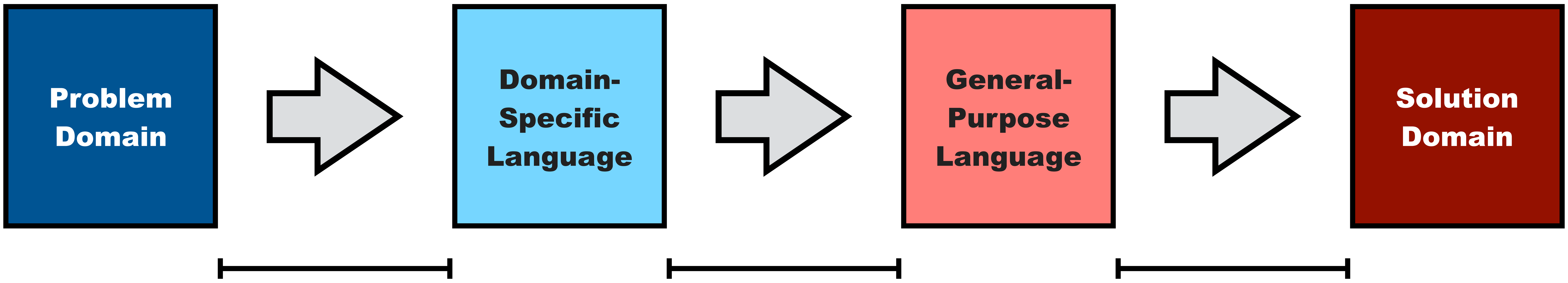
- Lack of safety
- Lack of abstraction



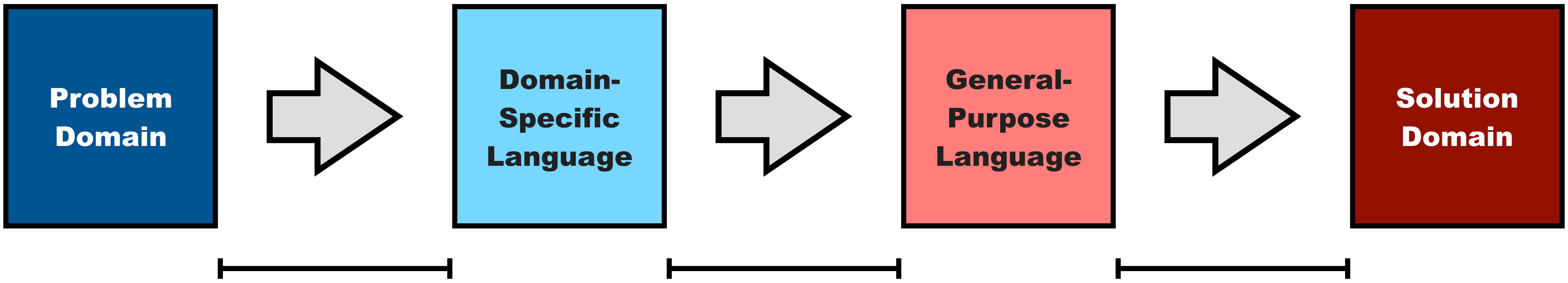
- Lack of safety
- Lack of abstraction
- Distance from domain



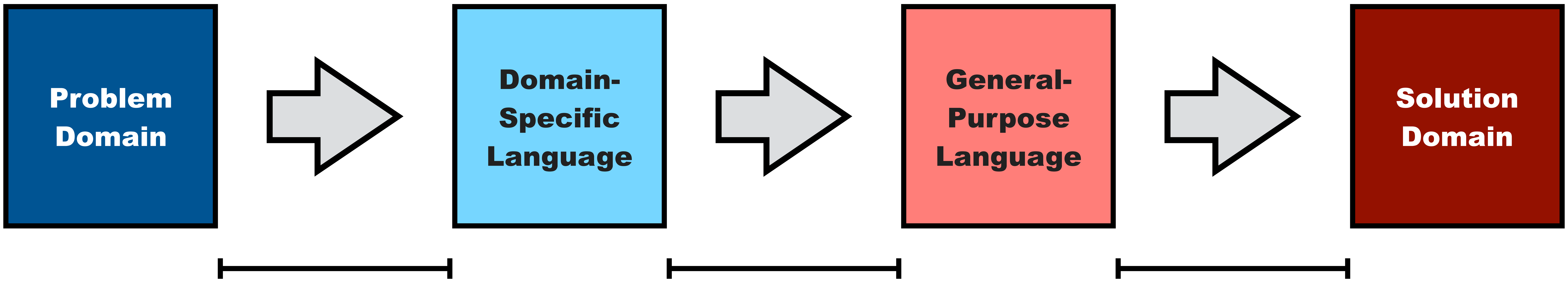
- Lack of safety
- Lack of abstraction
- Distance from domain



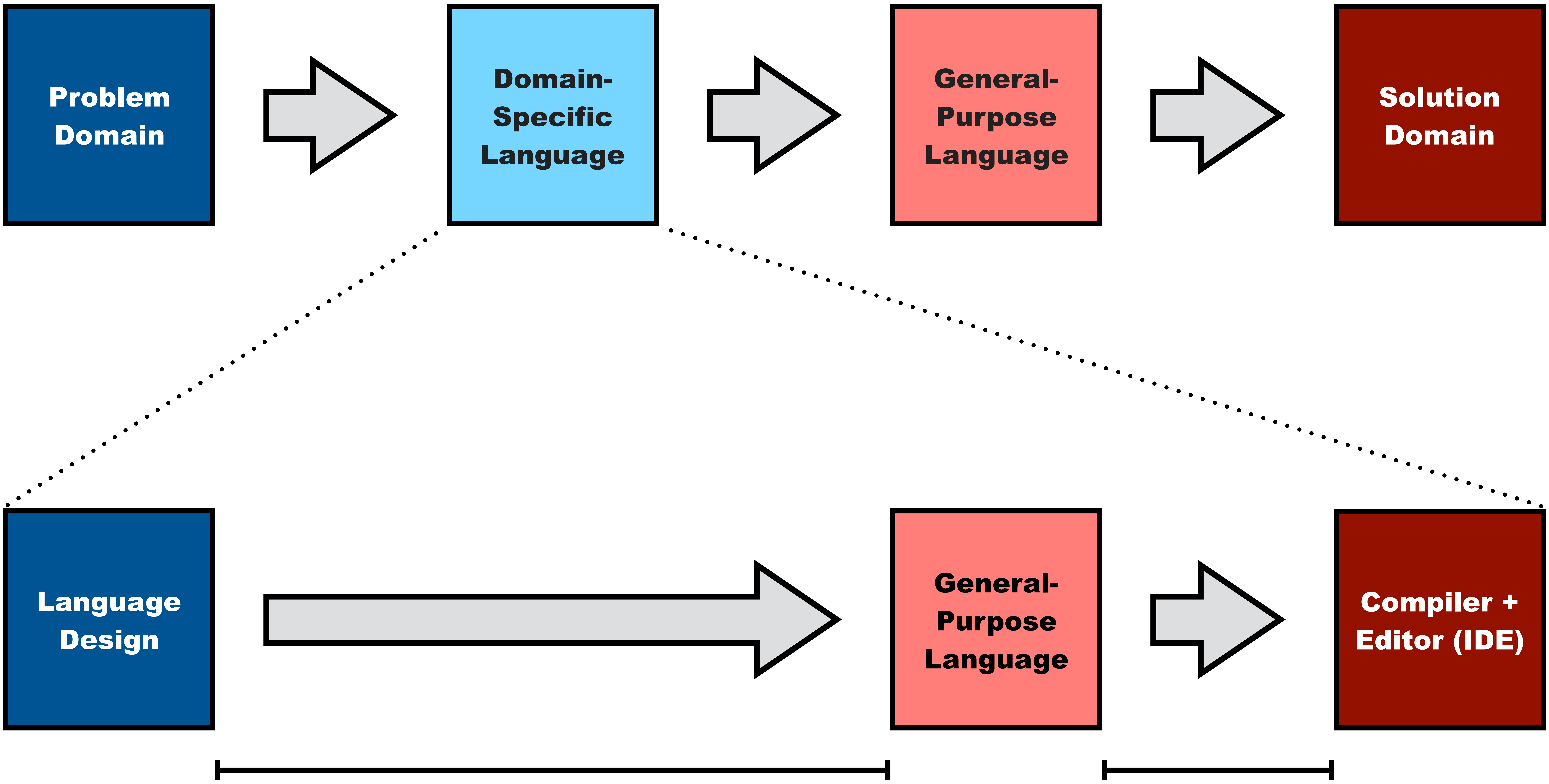
- Language-based safety and security
- Lack of abstraction
- Distance from domain



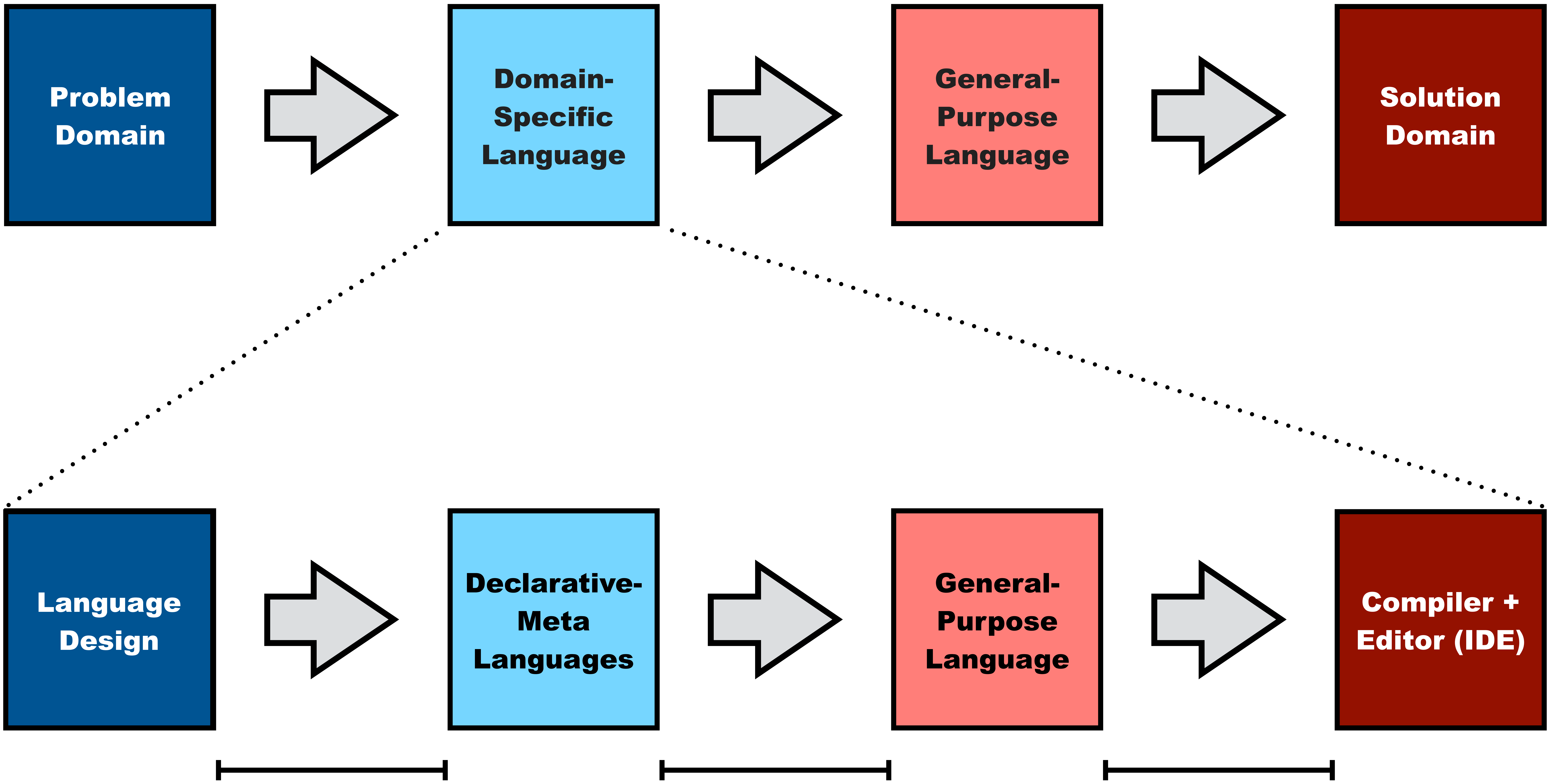
- Language-based safety and security
- High-level domain-specific abstraction
- Distance from domain



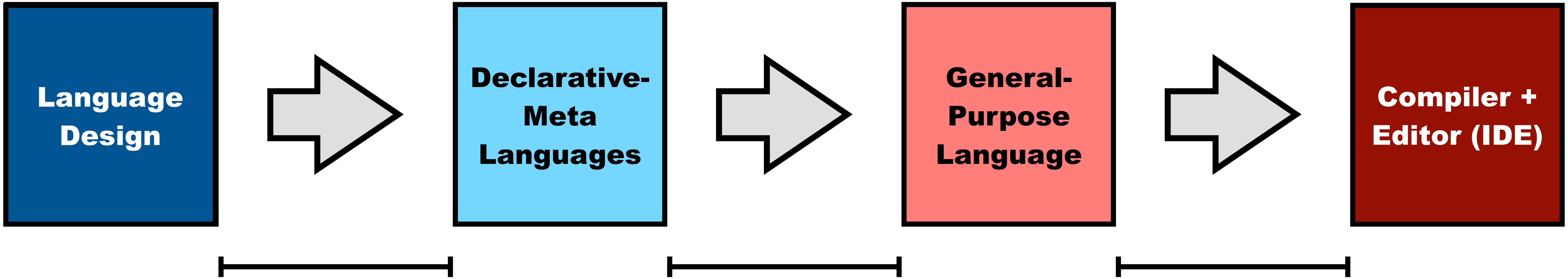
- Language-based safety and security
- High-level domain-specific abstraction
- Reduced distance from problem domain



Problem Domain
Domain-Specific Language
General-Purpose Language
Solution Domain
Language Design
General-Purpose Language
Compiler + Editor (IDE)

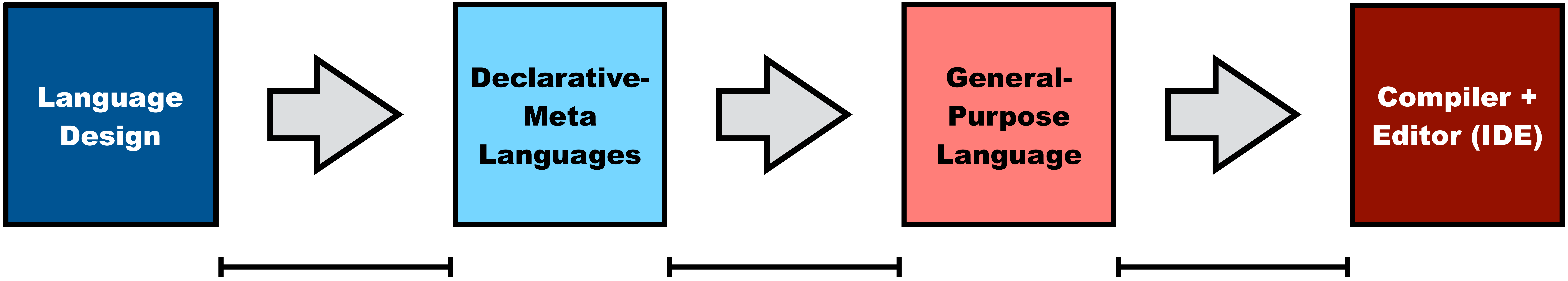

Problem Domain
Domain-Specific Language
General-Purpose Language
Solution Domain
Language Design
Declarative-Meta Languages
General-Purpose Language
Compiler + Editor (IDE)

Language workbench



Language workbench

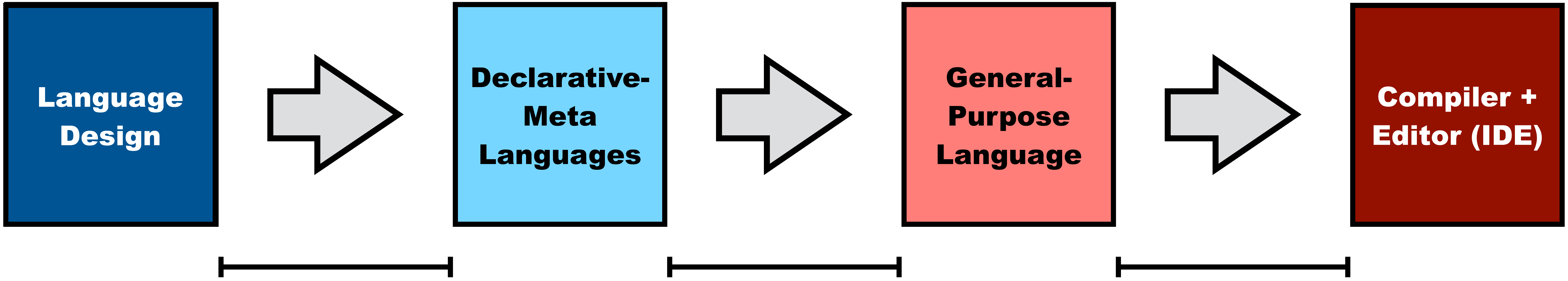
Declarative multi-purpose meta-languages



Language workbench

Declarative multi-purpose meta-languages

Useable language implementations

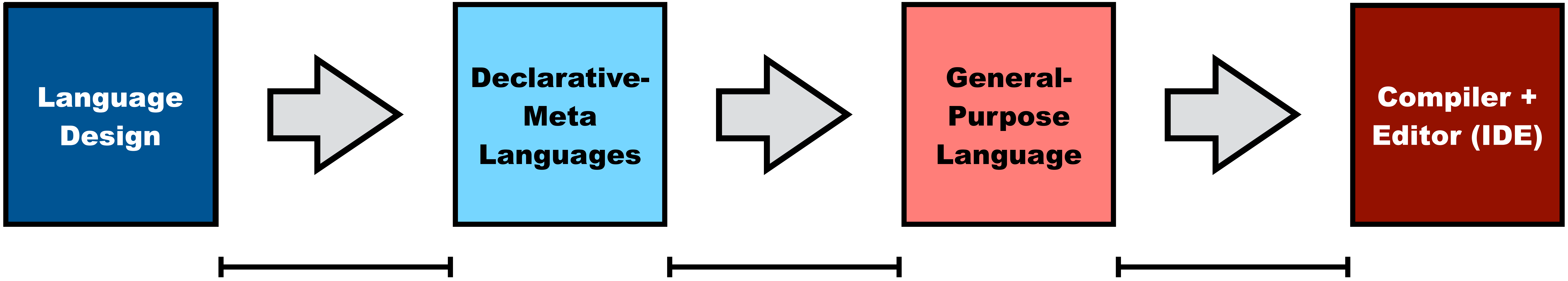


Language workbench

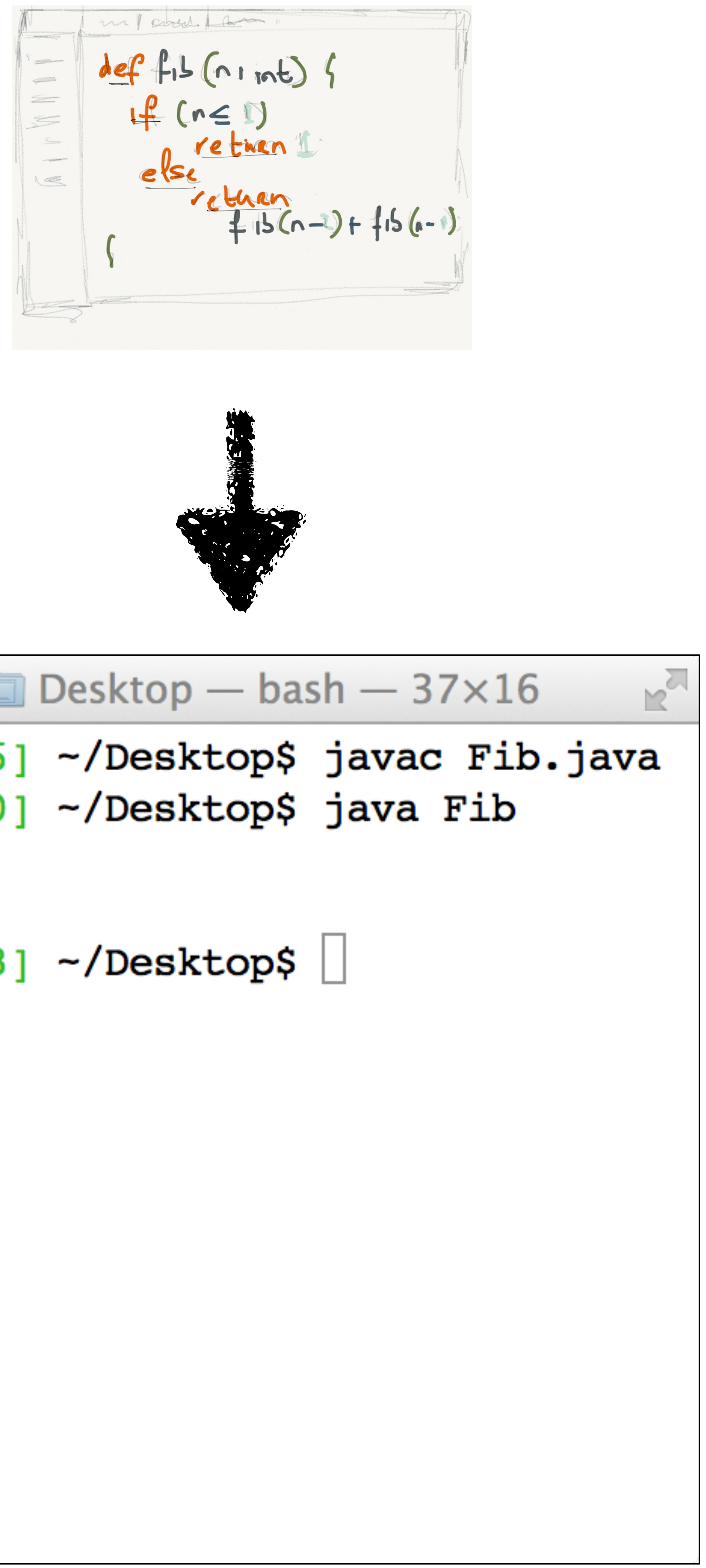
Declarative multi-purpose meta-languages

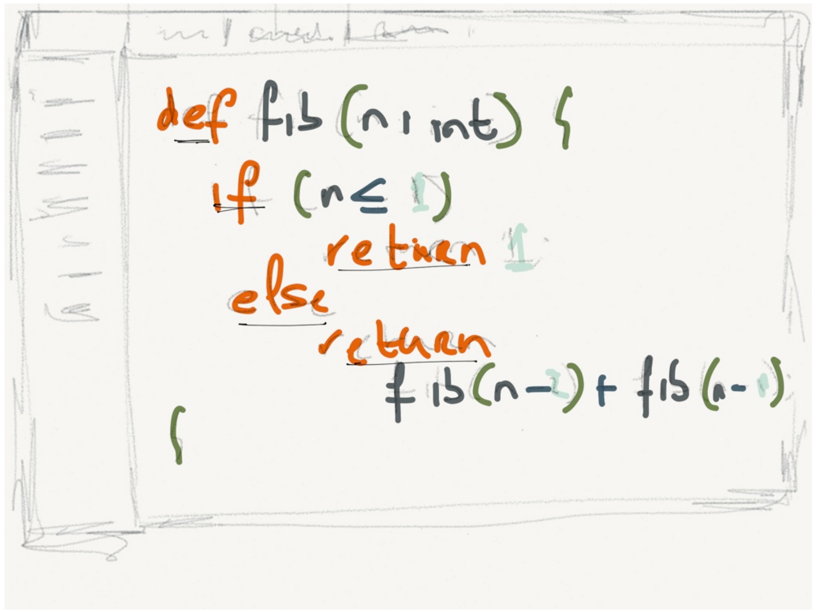
Useable language implementations

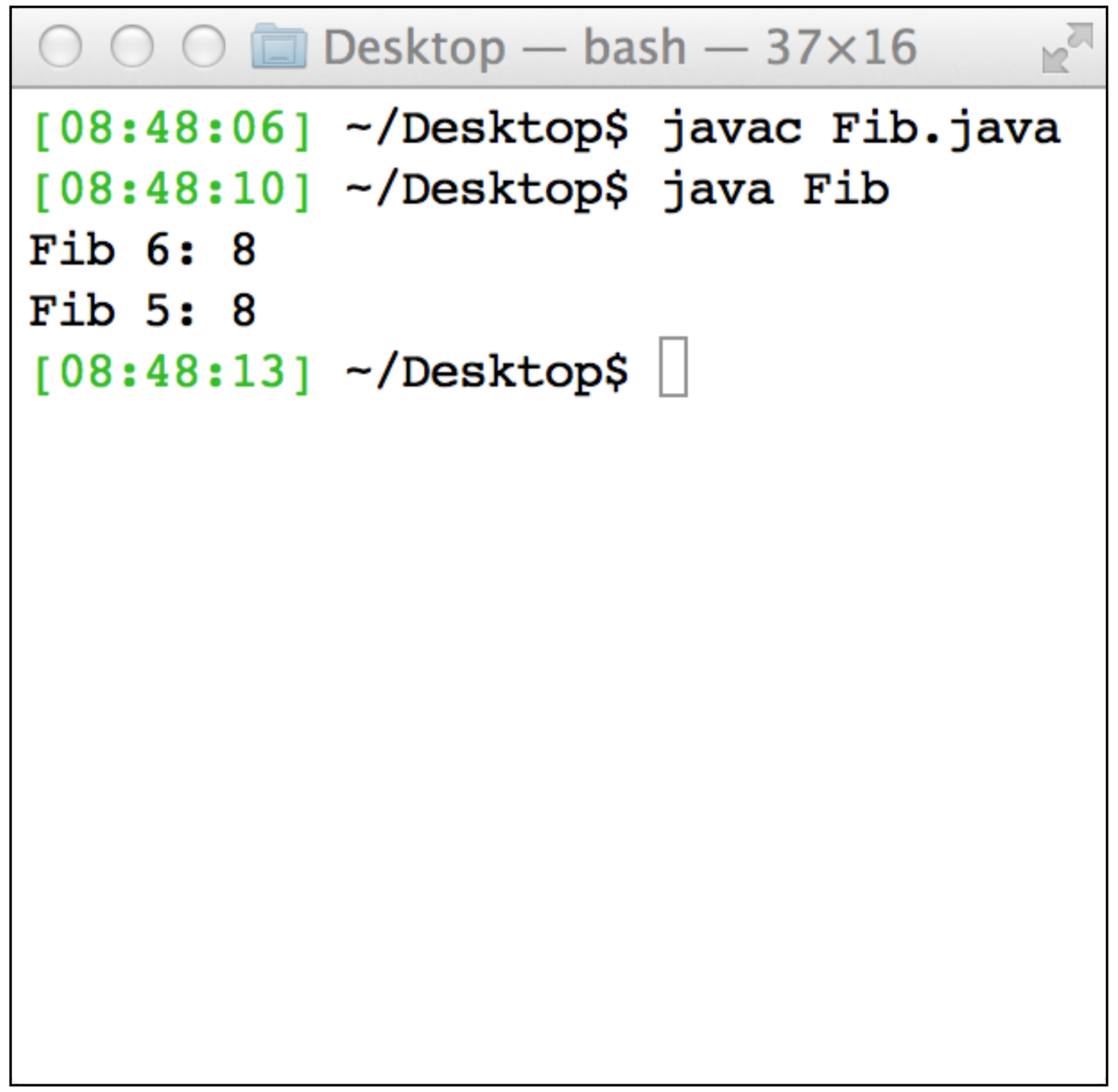
High quality language designs

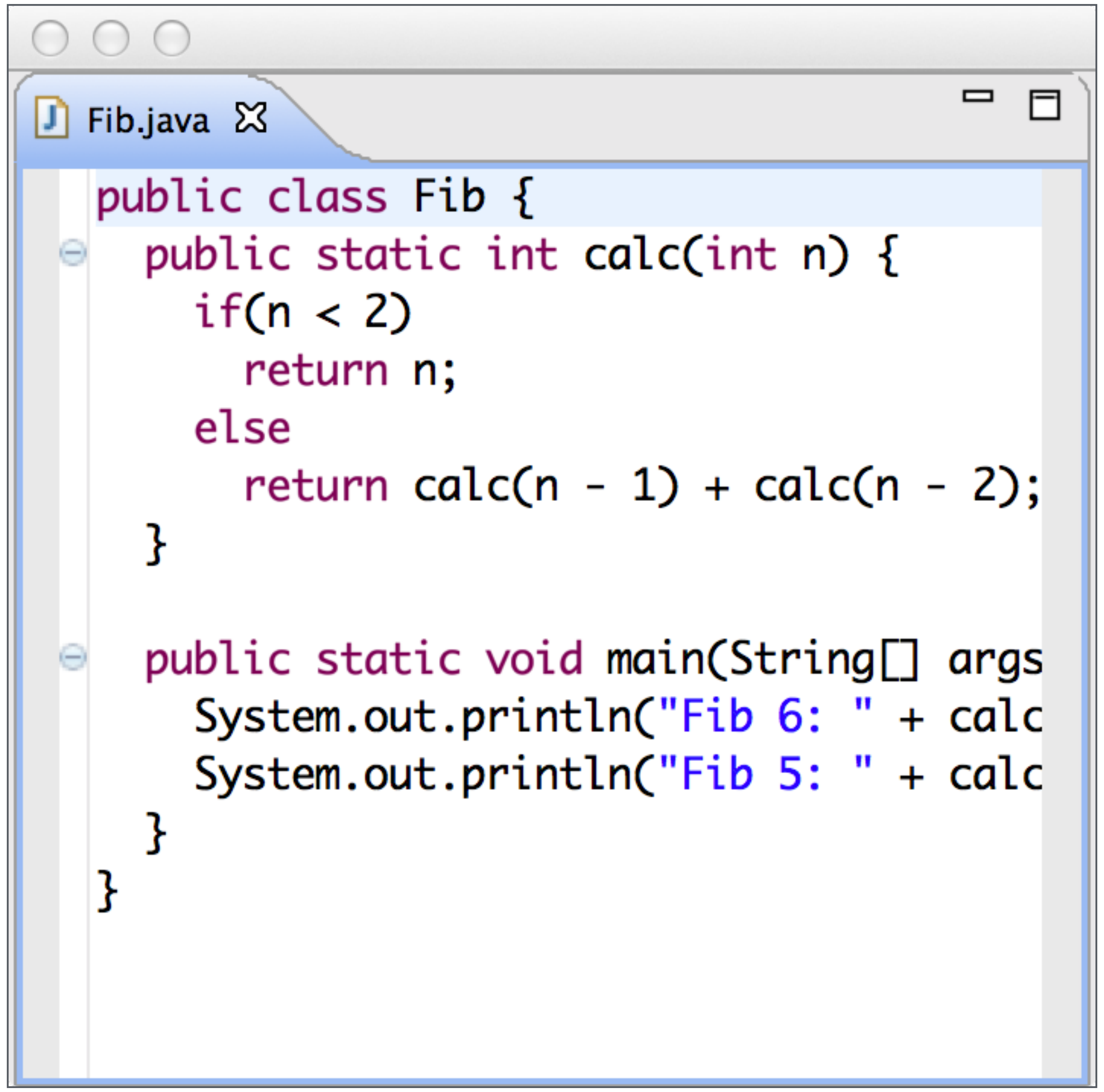


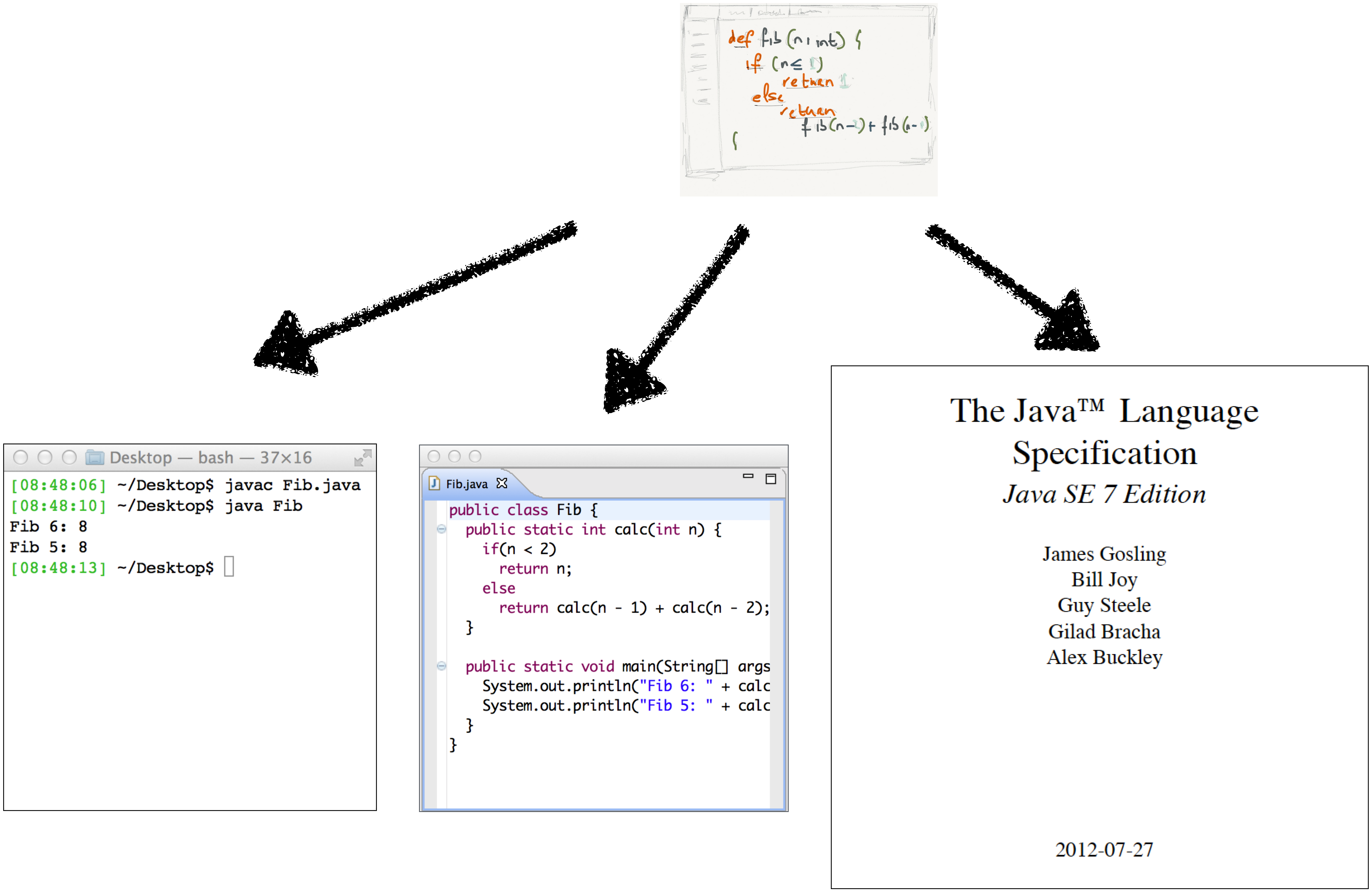
```
def fib(n: int) {
  if (n ≤ 1)
      return 1
  else
      return
          fib(n-2) + fib(n-1)
}
```


def fib (n, int) {
if
return
else
return
Desktop — bash — 37×16
[08:48:06] ~/Desktop$ javac Fib.java
[08:48:10] ~/Desktop$ java Fib
Fib 6: 8
Fib 5: 8
[08:48:13] ~/Desktop$


def fib (n, int) {
if (n≤
return
else
return
fib(n-
) + fib(n-
}


Desktop — bash — 37×16
[08:48:06] ~/Desktop$ javac Fib.java
[08:48:10] ~/Desktop$ java Fib
Fib 6: 8
Fib 5: 8
[08:48:13] ~/Desktop$


Fib.java
public class Fib {
public static int calc(int n) {
if(n < 2)
return n;
else
return calc(n - 1) + calc(n - 2);
}
public static void main(String[] args
System.out.println("Fib 6: " + calc
System.out.println("Fib 5: " + calc
}
}



def fib (n: int) {
if (n ≤
return
else
return
fib(n-
Desktop — bash — 37×16
[08:48:06] ~/Desktop$ javac Fib.java
[08:48:10] ~/Desktop$ java Fib
Fib 6: 8
Fib 5: 8
[08:48:13] ~/Desktop$
Fib.java
public class Fib {
public static int calc(int n) {
if(n < 2)
return n;
else
return calc(n - 1) + calc(n - 2);
}
public static void main(String[] args
System.out.println("Fib 6: " + calc
System.out.println("Fib 5: " + calc
}
}
The Java™ Language Specification
Java SE 7 Edition
James Gosling
Bill Joy
Guy Steele
Gilad Bracha
Alex Buckley
2012-07-27

Desktop — bash — 37×16
[08:48:06] ~/Desktop$ javac Fib.java
[08:48:10] ~/Desktop$ java Fib
Fib 6: 8
Fib 5: 8
[08:48:13] ~/Desktop$
Fib.java
public class Fib {
public static int calc(int n) {
if(n < 2)
return n;
else
return calc(n - 1) + calc(n - 2);
}
public static void main(String[] args
System.out.println("Fib 6: " + calc
System.out.println("Fib 5: " + calc
}
}
The Java™ Language Specification
Java SE 7 Edition
James Gosling
Bill Joy
Guy Steele
Gilad Bracha
Alex Buckley
2012-07-27
Describing the Semantics of Java and Proving Type Soundness
Sophia Drossopoulou and Susan Eisenbach
Department of Computing
Imperial College of Science, Technology and Medicine
1 Introduction
Java combines the experience from the development of several object oriented languages, such as C++, Smalltalk and CLOS. The philosophy of the language designers was to include only features with already known semantics, and to provide a small and simple language.
Nevertheless, we feel that the introduction of some new features in Java, as well as the specific combination of features, justifies a study of the Java formal semantics. The use of interfaces, reminiscent of [11,5] is a simplification of the signatures extension for C++ [4] and is – to the best of our knowledge – novel. The mechanism for dynamic method binding is that of C++, but we know of no formal definition. Java adopts the Smalltalk [15] approach whereby all object variables are implicitly pointers.
Furthermore, although there are a large number of studies of the semantics of isolated programming language features or of minimal programming languages [1], [31], [34], there have not been many studies of the formal semantics of actual programming languages. In addition, the interplay of features which are very well understood in isolation, might introduce unexpected effects.

Desktop — bash — 37×16
[08:48:06] ~/Desktop$ javac Fib.java
[08:48:10] ~/Desktop$ java Fib
Fib 6: 8
Fib 5: 8
[08:48:13] ~/Desktop$
Fib.java
public class Fib {
public static int calc(int n) {
if(n < 2)
return n;
else
return calc(n - 1) + calc(n - 2);
}
public static void main(String[] args
System.out.println("Fib 6: " + calc
System.out.println("Fib 5: " + calc
}
}
The Java™ Language Specification
Java SE 7 Edition
James Gosling
Bill Joy
Guy Steele
Gilad Bracha
Alex Buckley
2012-07-27
Describing the Semantics of Java and Proving Type Soundness
Sophia Drossopoulou and Susan Eisenbach
Department of Computing
Imperial College of Science, Technology and Medicine
1 Introduction
Java combines the experience from the development of several object oriented languages, such as C++, Smalltalk and Clos. The philosophy of the language designers was to include only features with already known semantics, and to provide a small and simple language.
Nevertheless, we feel that the introduction of some new features in Java, as well as the specific combination of features, justifies a study of the Java formal semantics. The use of interfaces, reminiscent of [11,6] is a simplification of the signatures extension for C++ [4] and is – to the best of our knowledge – novel. The mechanism for dynamic method binding is that of C++, but we know of no formal definition. Java adopts the Smalltalk [13] approach whereby all object variables are implicitly pointers.
Furthermore, although there are a large number of studies of the semantics of isolated programming language features or of minimal programming languages [1], [31], [34], there have not been many studies of the formal semantics of actual programming languages. In addition, the interplay of features which are very well understood in isolation, might introduce unexpected effects.

Desktop — bash — 37×16

```
[08:48:06] ~/Desktop$ javac Fib.java
[08:48:10] ~/Desktop$ java Fib
Fib 6: 8
Fib 5: 8
[08:48:13] ~/Desktop$
```

Fib.java

```
public class Fib {
  public static int calc(int n) {
    if(n < 2)
      return n;
    else
      return calc(n - 1) + calc(n - 2);
  }

  public static void main(String[] args
    System.out.println("Fib 6: " + calc
    System.out.println("Fib 5: " + calc
  }
}
```

The Java™ Language Specification

Java SE 7 Edition

James Gosling
Bill Joy
Guy Steele
Gilad Bracha
Alex Buckley

2012-07-27

Describing the Semantics of Java and Proving Type Soundness

Sophia Drossopoulou and Susan Eisenbach

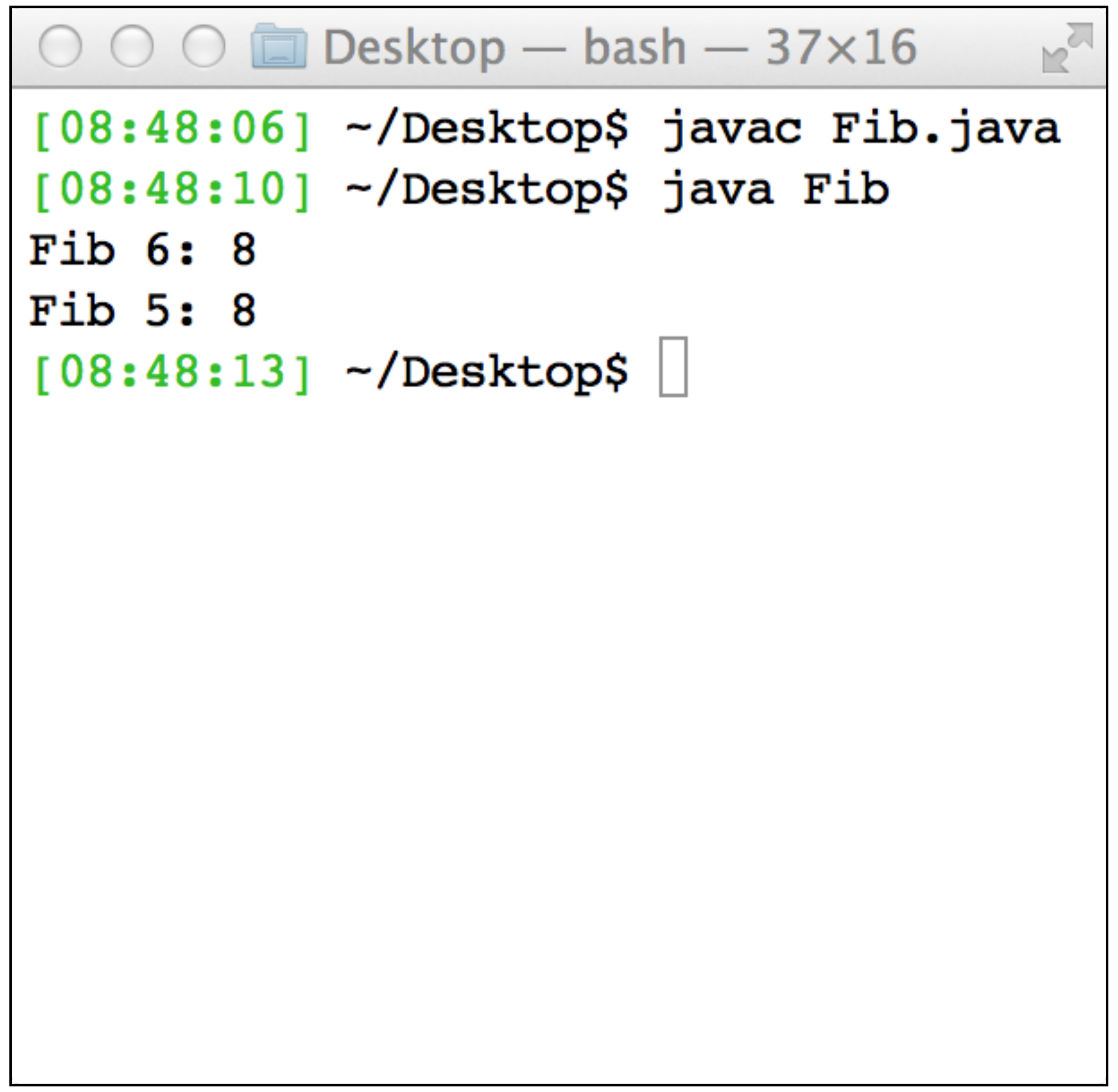
Department of Computing
Imperial College of Science, Technology and Medicine

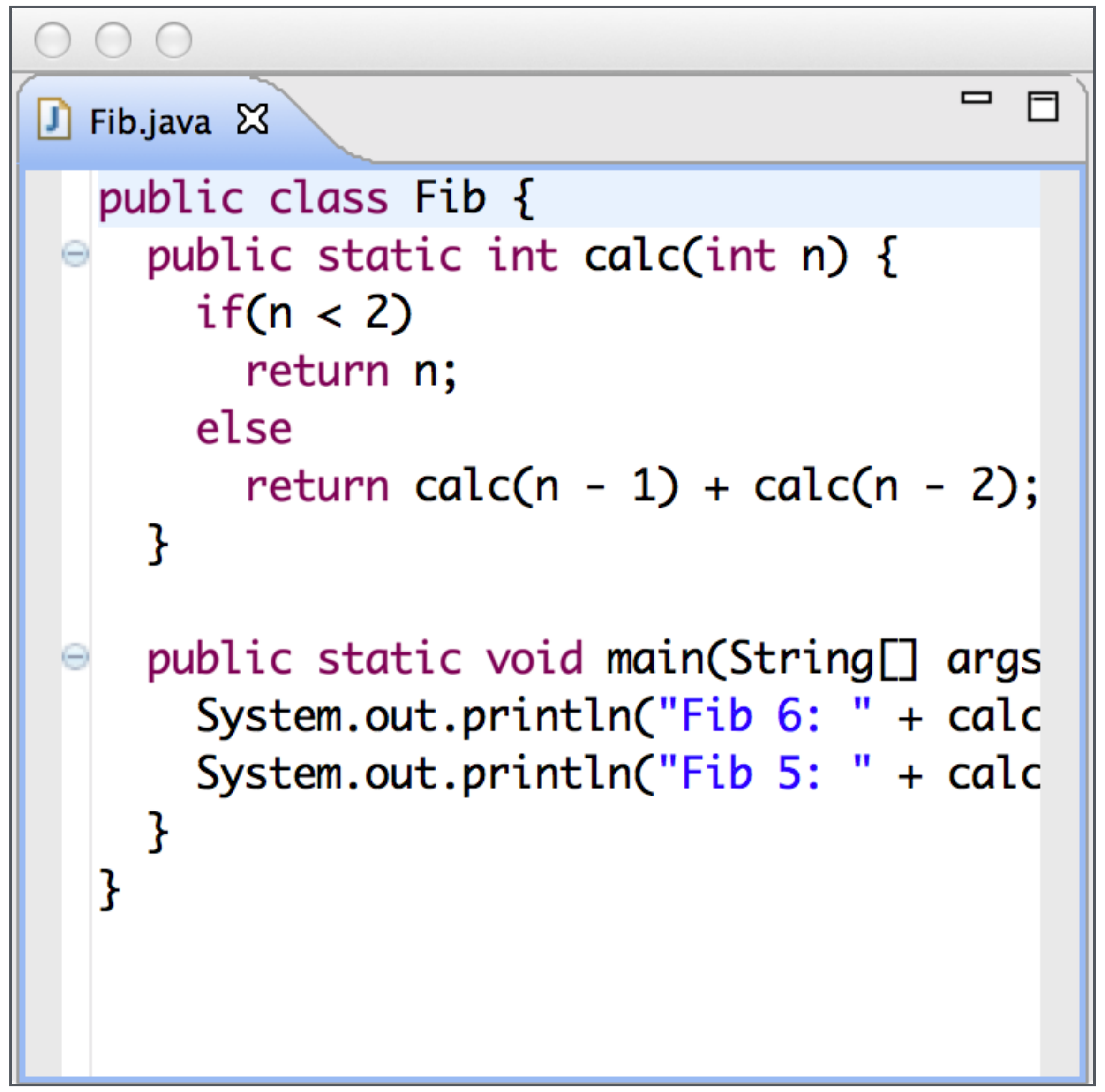
1 Introduction

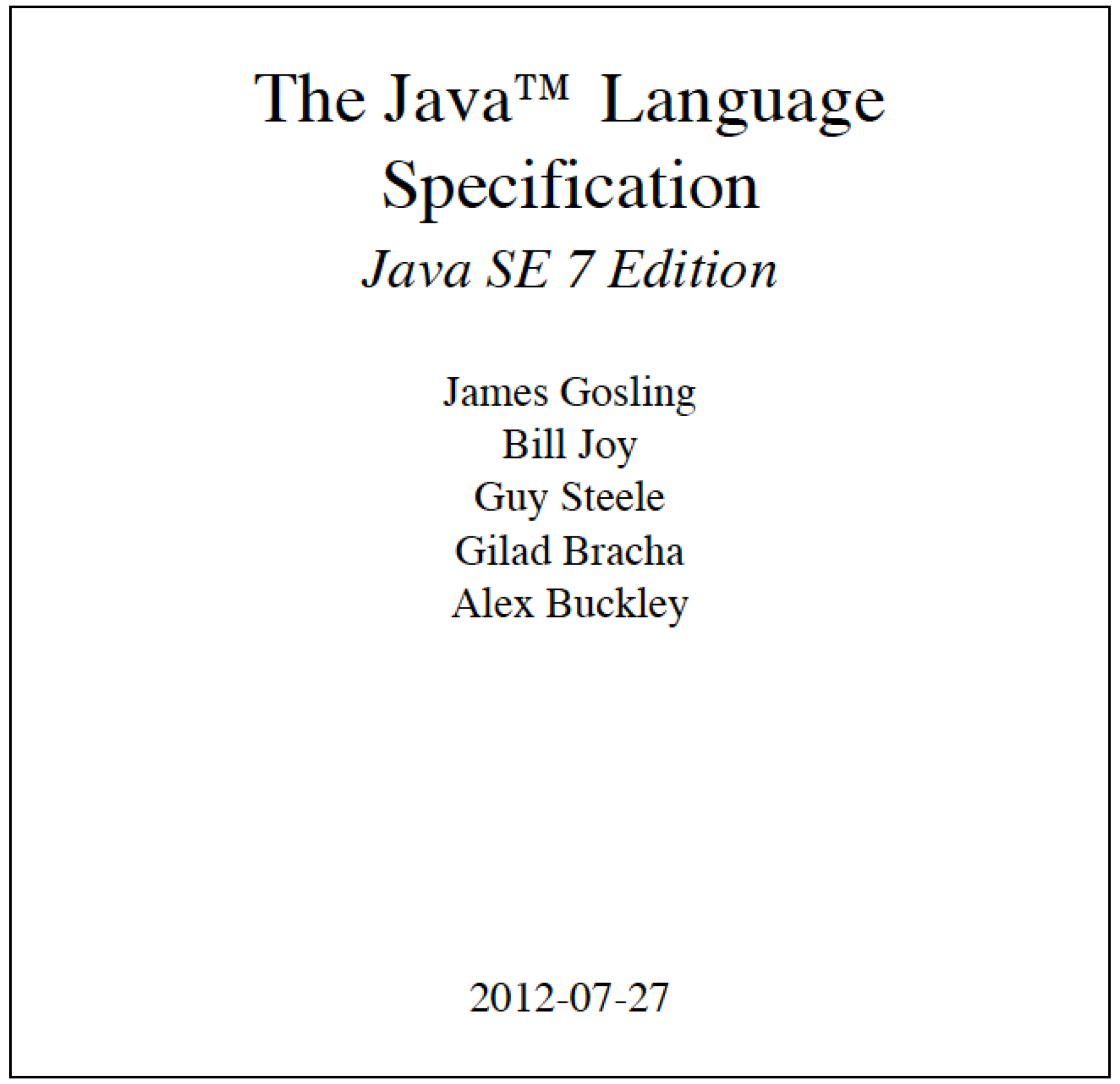
Java combines the experience from the development of several object oriented languages, such as C++, Smalltalk and CLOS. The philosophy of the language designers was to include only features with already known semantics, and to provide a small and simple language.

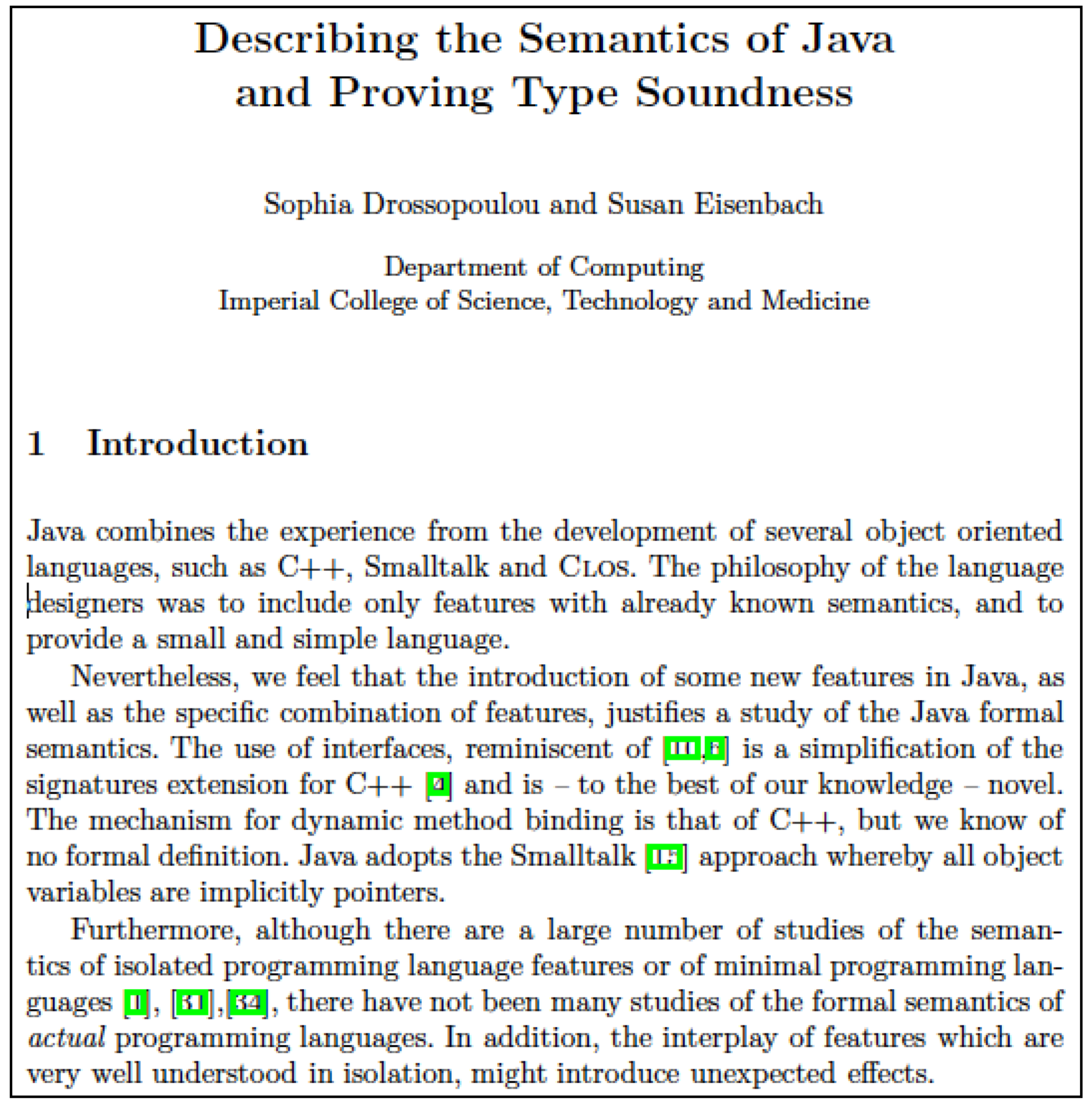
Nevertheless, we feel that the introduction of some new features in Java, as well as the specific combination of features, justifies a study of the Java formal semantics. The use of interfaces, reminiscent of [11,5] is a simplification of the signatures extension for C++ [4] and is – to the best of our knowledge – novel. The mechanism for dynamic method binding is that of C++, but we know of no formal definition. Java adopts the Smalltalk [13] approach whereby all object variables are implicitly pointers.

Furthermore, although there are a large number of studies of the semantics of isolated programming language features or of minimal programming languages [1], [31],[32], there have not been many studies of the formal semantics of *actual* programming languages. In addition, the interplay of features which are very well understood in isolation, might introduce unexpected effects.

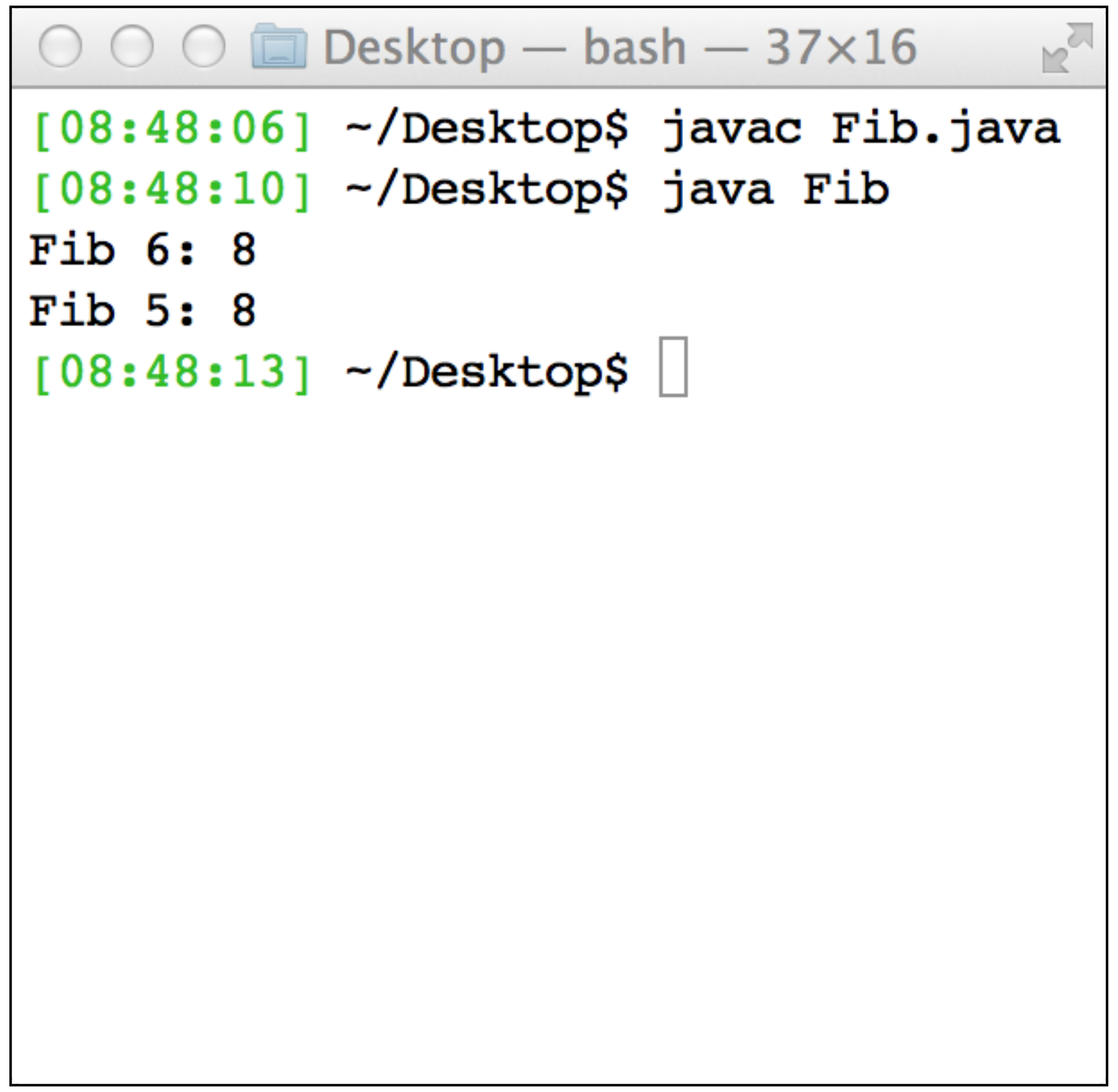

Desktop — bash — 37×16
[08:48:06] ~/Desktop$ javac Fib.java
[08:48:10] ~/Desktop$ java Fib
Fib 6: 8
Fib 5: 8
[08:48:13] ~/Desktop$

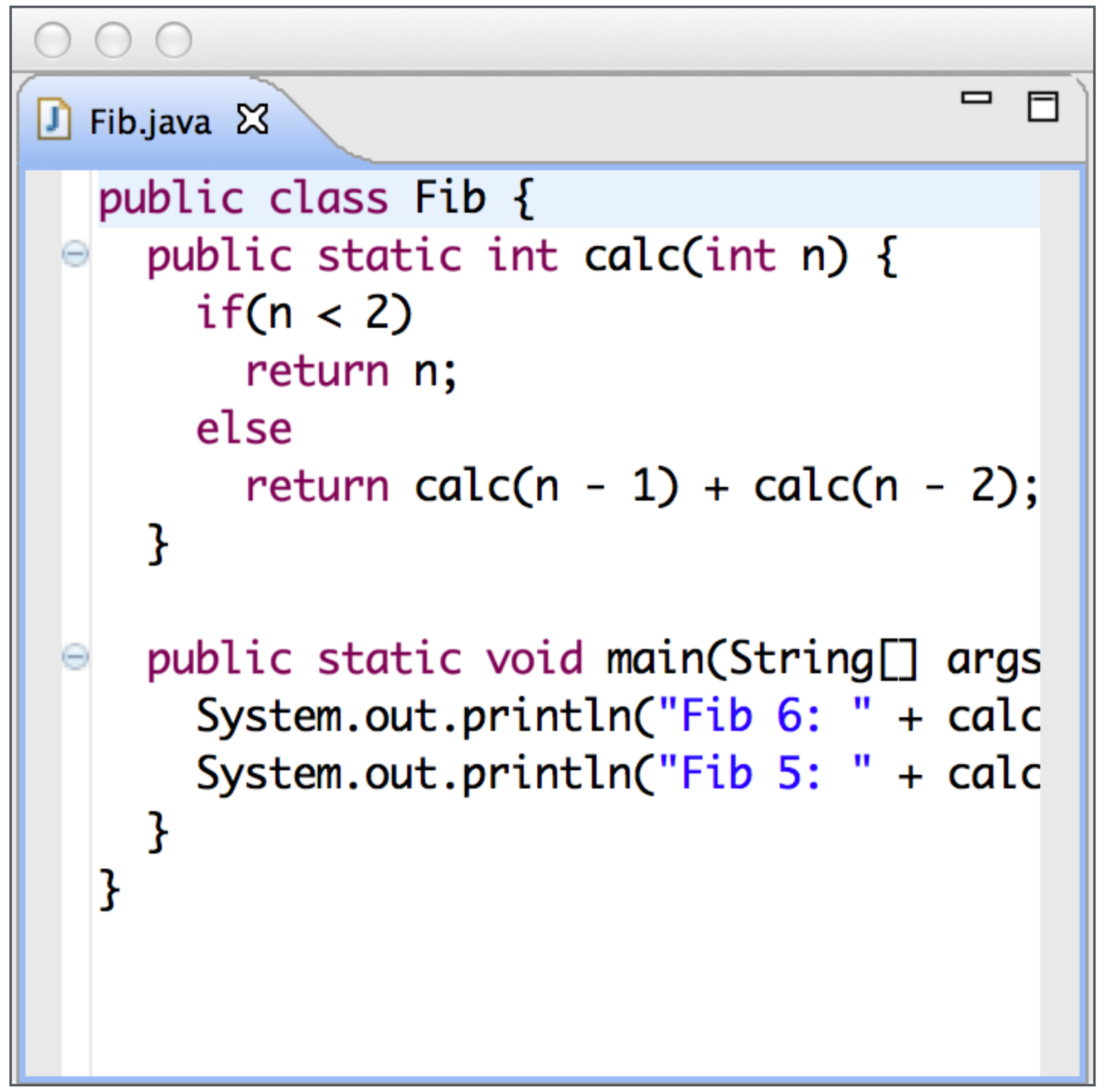

Fib.java
public class Fib {
public static int calc(int n) {
if(n < 2)
return n;
else
return calc(n - 1) + calc(n - 2);
}
public static void main(String[] args
System.out.println("Fib 6: " + calc
System.out.println("Fib 5: " + calc
}
}

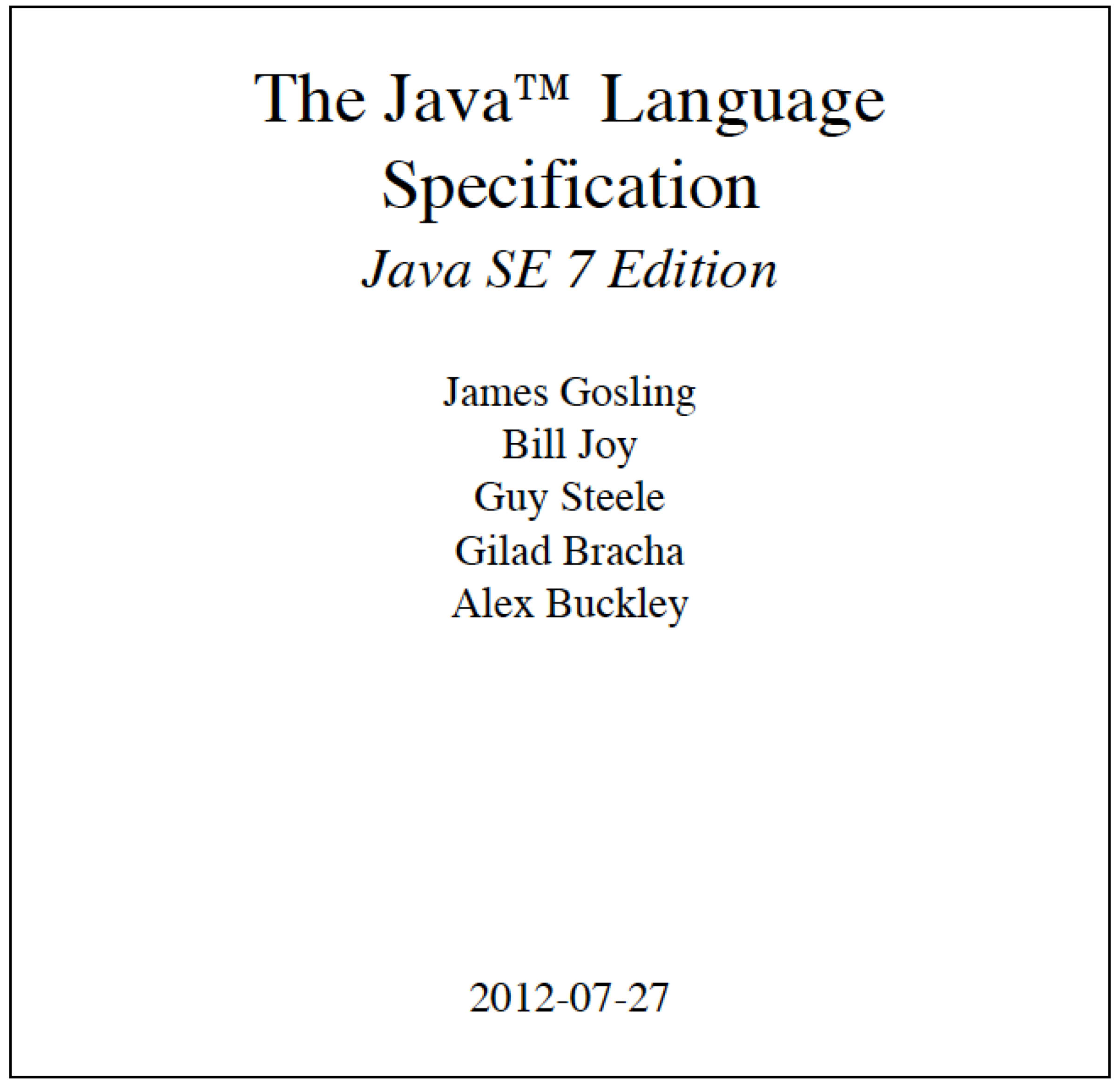

The Java™ Language
Specification
Java SE 7 Edition
James Gosling
Bill Joy
Guy Steele
Gilad Bracha
Alex Buckley
2012-07-27


Describing the Semantics of Java
and Proving Type Soundness
Sophia Drossopoulou and Susan Eisenbach
Department of Computing
Imperial College of Science, Technology and Medicine
1 Introduction
Java combines the experience from the development of several object oriented languages, such as C++, Smalltalk and CLOS. The philosophy of the language designers was to include only features with already known semantics, and to provide a small and simple language.
Nevertheless, we feel that the introduction of some new features in Java, as well as the specific combination of features, justifies a study of the Java formal semantics. The use of interfaces, reminiscent of [11,6] is a simplification of the signatures extension for C++ [4] and is – to the best of our knowledge – novel. The mechanism for dynamic method binding is that of C++, but we know of no formal definition. Java adopts the Smalltalk [15] approach whereby all object variables are implicitly pointers.
Furthermore, although there are a large number of studies of the semantics of isolated programming language features or of minimal programming languages [1], [31],[34], there have not been many studies of the formal semantics of *actual* programming languages. In addition, the interplay of features which are very well understood in isolation, might introduce unexpected effects.

parser





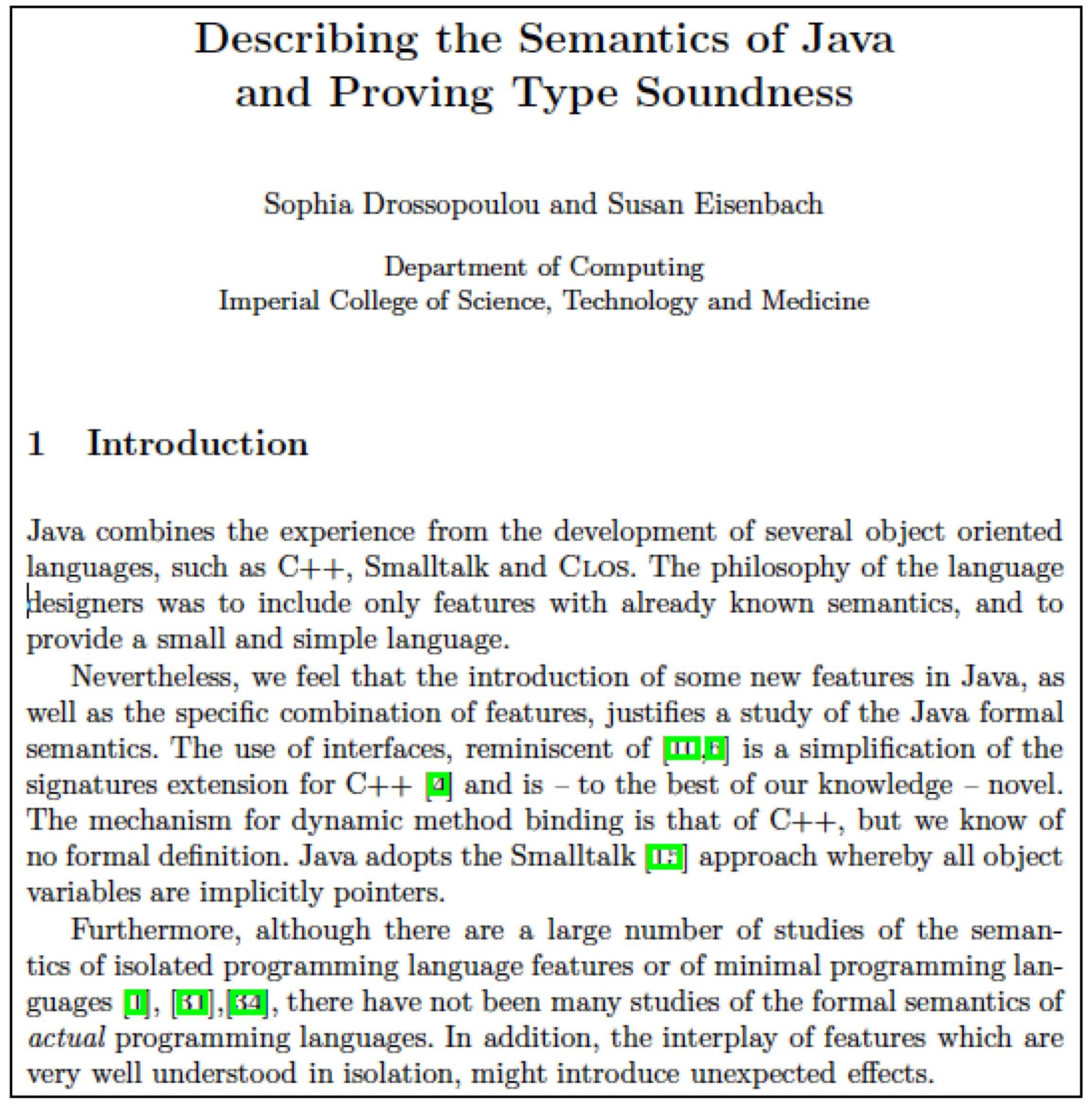


The Java™ Language Specification

Java SE 7 Edition

James Gosling
Bill Joy
Guy Steele
Gilad Bracha
Alex Buckley

2012-07-27



Describing the Semantics of Java and Proving Type Soundness

Sophia Drossopoulou and Susan Eisenbach

Department of Computing
Imperial College of Science, Technology and Medicine

1 Introduction

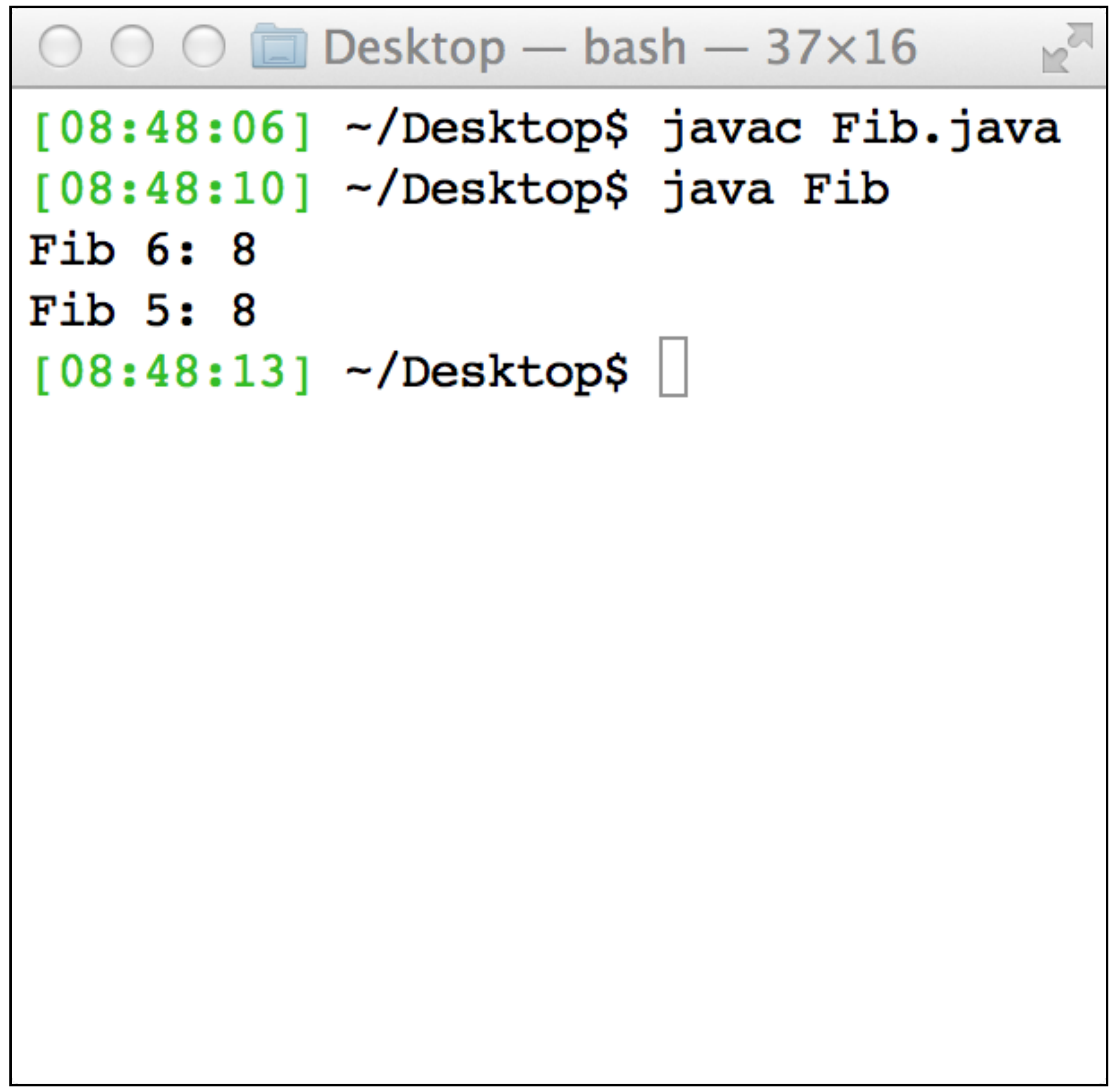
Java combines the experience from the development of several object oriented languages, such as C++, Smalltalk and CLOS. The philosophy of the language designers was to include only features with already known semantics, and to provide a small and simple language.

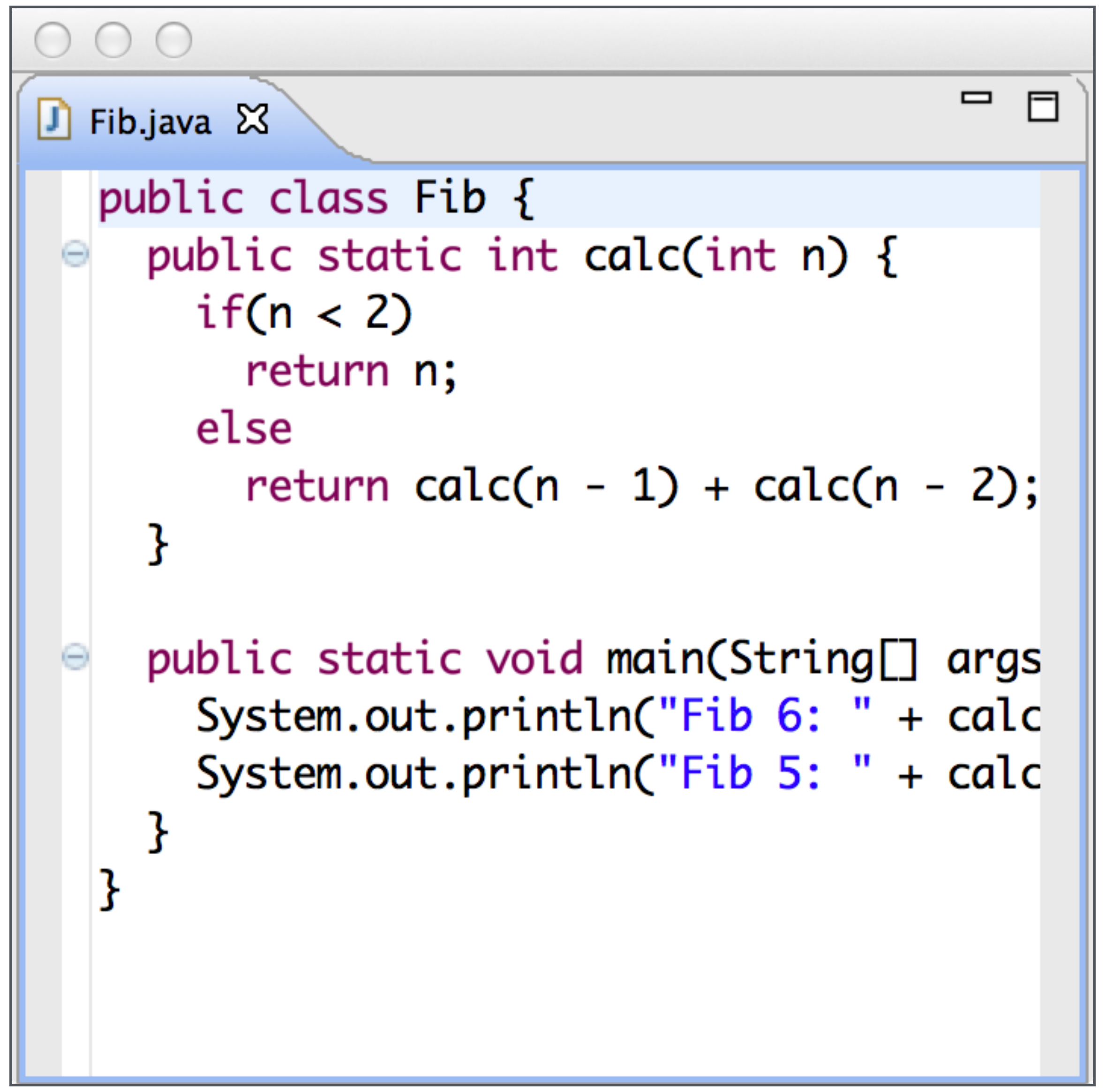
Nevertheless, we feel that the introduction of some new features in Java, as well as the specific combination of features, justifies a study of the Java formal semantics. The use of interfaces, reminiscent of [11,6] is a simplification of the signatures extension for C++ [4] and is – to the best of our knowledge – novel. The mechanism for dynamic method binding is that of C++, but we know of no formal definition. Java adopts the Smalltalk [15] approach whereby all object variables are implicitly pointers.

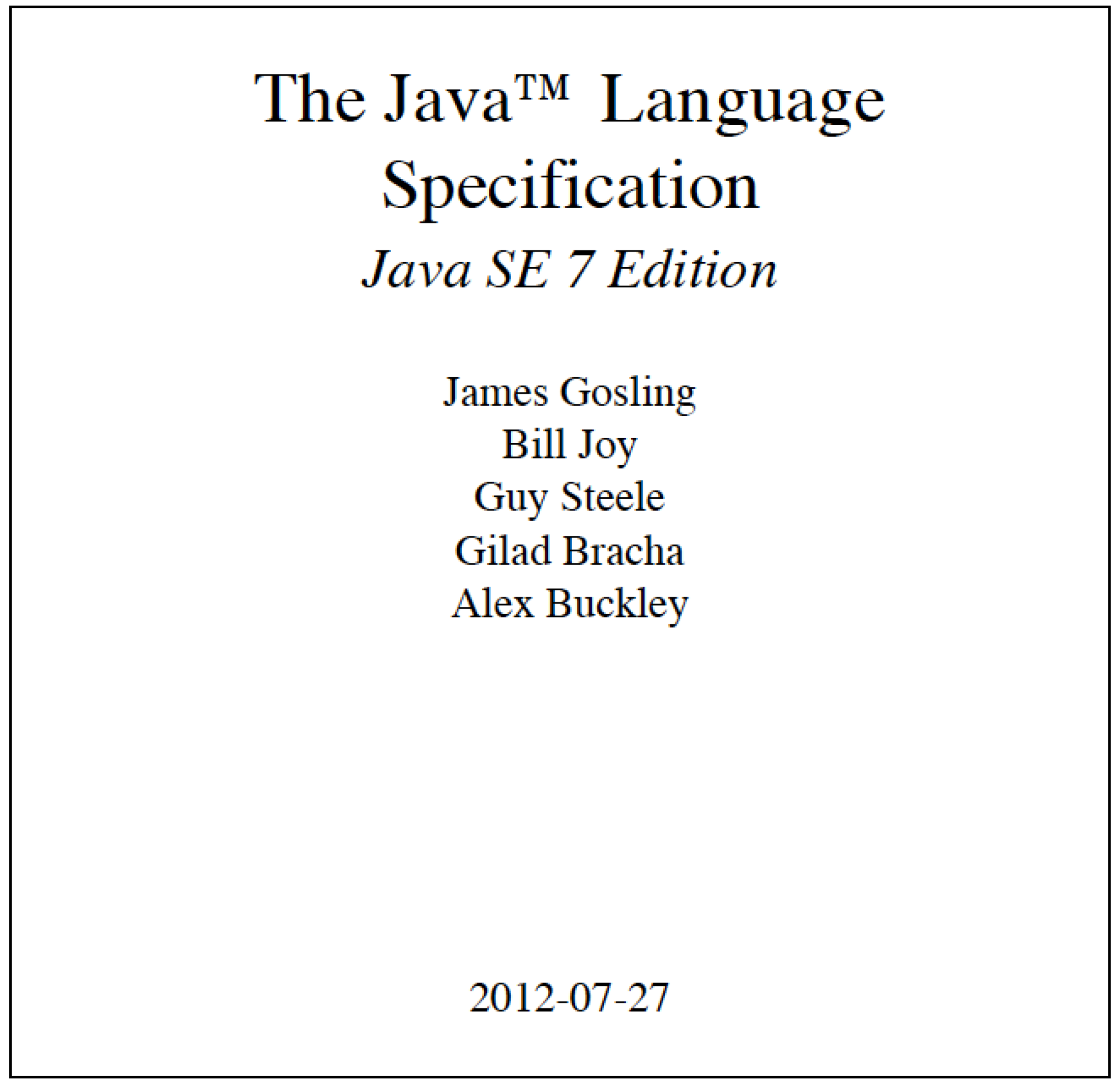
Furthermore, although there are a large number of studies of the semantics of isolated programming language features or of minimal programming languages [1], [31],[34], there have not been many studies of the formal semantics of *actual* programming languages. In addition, the interplay of features which are very well understood in isolation, might introduce unexpected effects.

parser

type checker





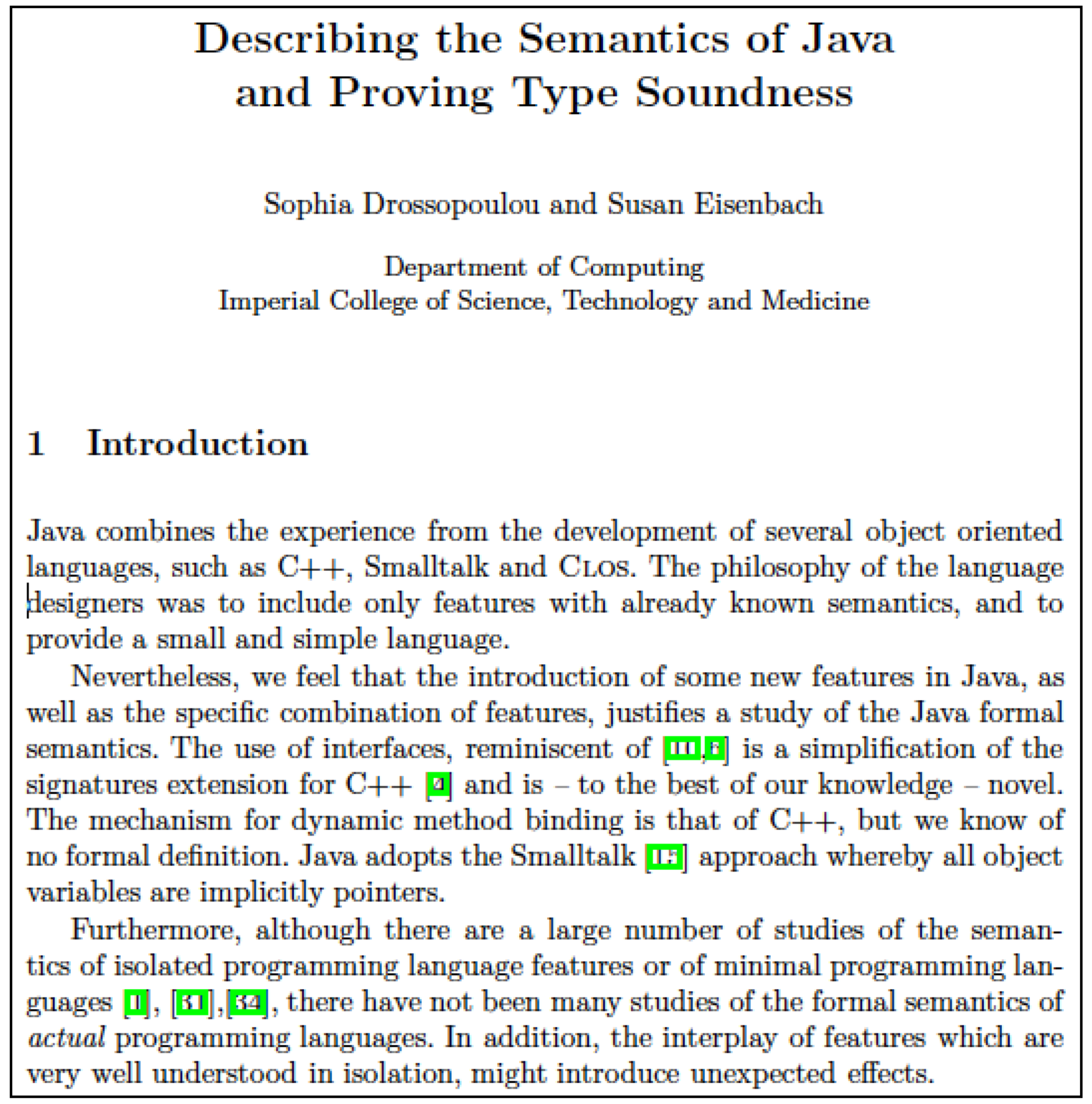


The Java™ Language Specification

Java SE 7 Edition

James Gosling
Bill Joy
Guy Steele
Gilad Bracha
Alex Buckley

2012-07-27



Describing the Semantics of Java and Proving Type Soundness

Sophia Drossopoulou and Susan Eisenbach

Department of Computing
Imperial College of Science, Technology and Medicine

1 Introduction

Java combines the experience from the development of several object oriented languages, such as C++, Smalltalk and CLOS. The philosophy of the language designers was to include only features with already known semantics, and to provide a small and simple language.

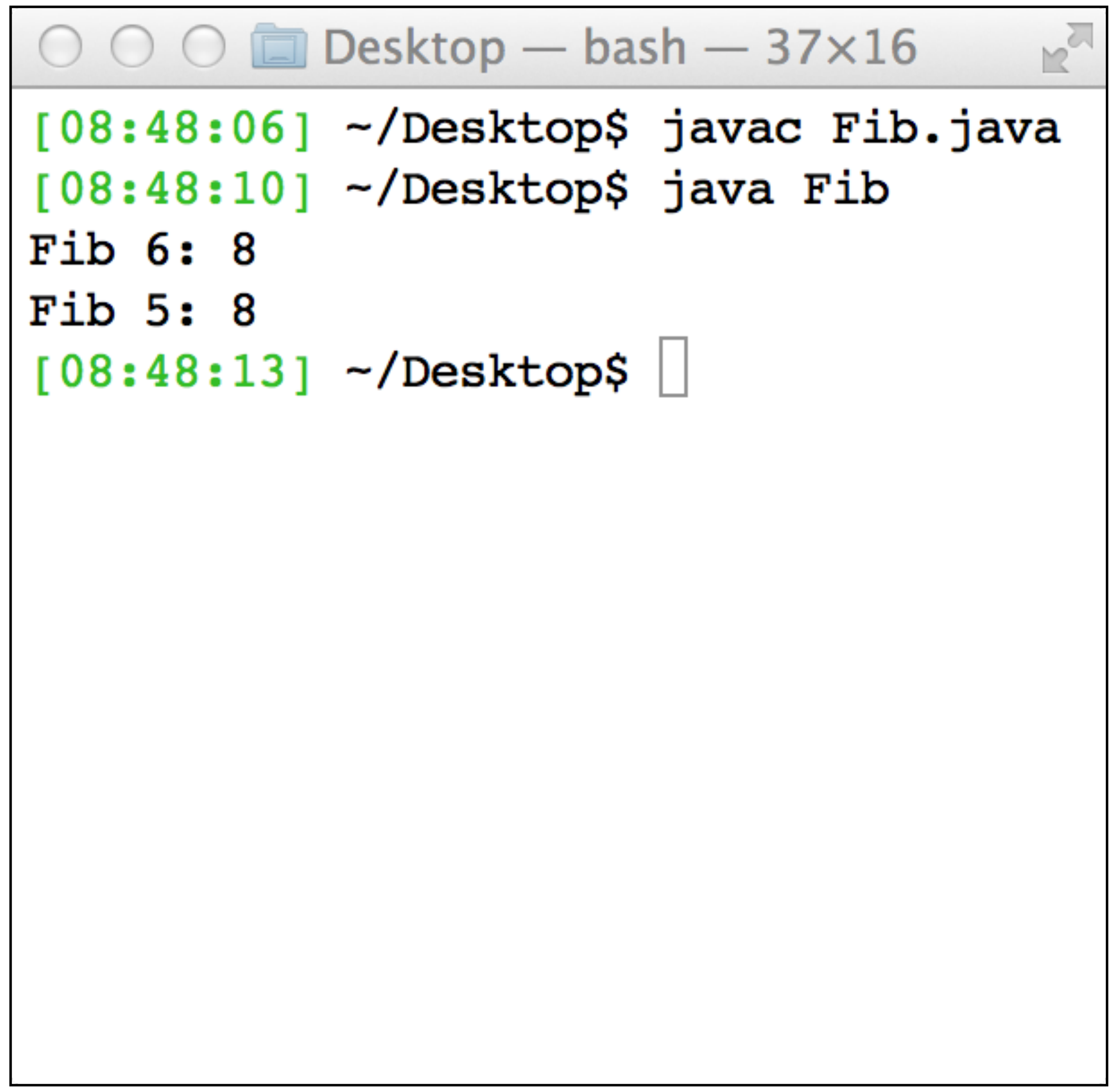
Nevertheless, we feel that the introduction of some new features in Java, as well as the specific combination of features, justifies a study of the Java formal semantics. The use of interfaces, reminiscent of [11,8] is a simplification of the signatures extension for C++ [4] and is – to the best of our knowledge – novel. The mechanism for dynamic method binding is that of C++, but we know of no formal definition. Java adopts the Smalltalk [15] approach whereby all object variables are implicitly pointers.

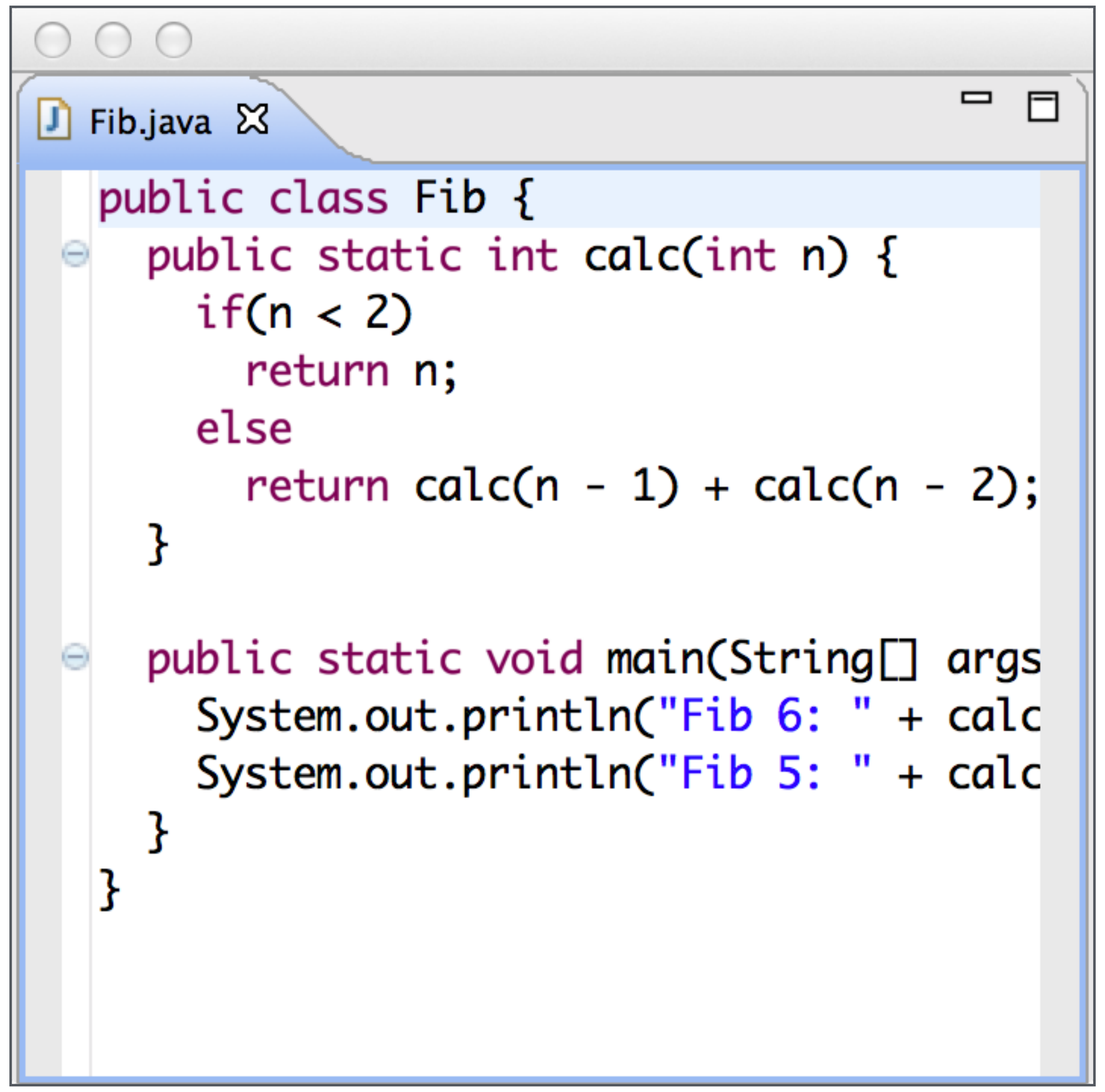
Furthermore, although there are a large number of studies of the semantics of isolated programming language features or of minimal programming languages [1], [31],[34], there have not been many studies of the formal semantics of *actual* programming languages. In addition, the interplay of features which are very well understood in isolation, might introduce unexpected effects.

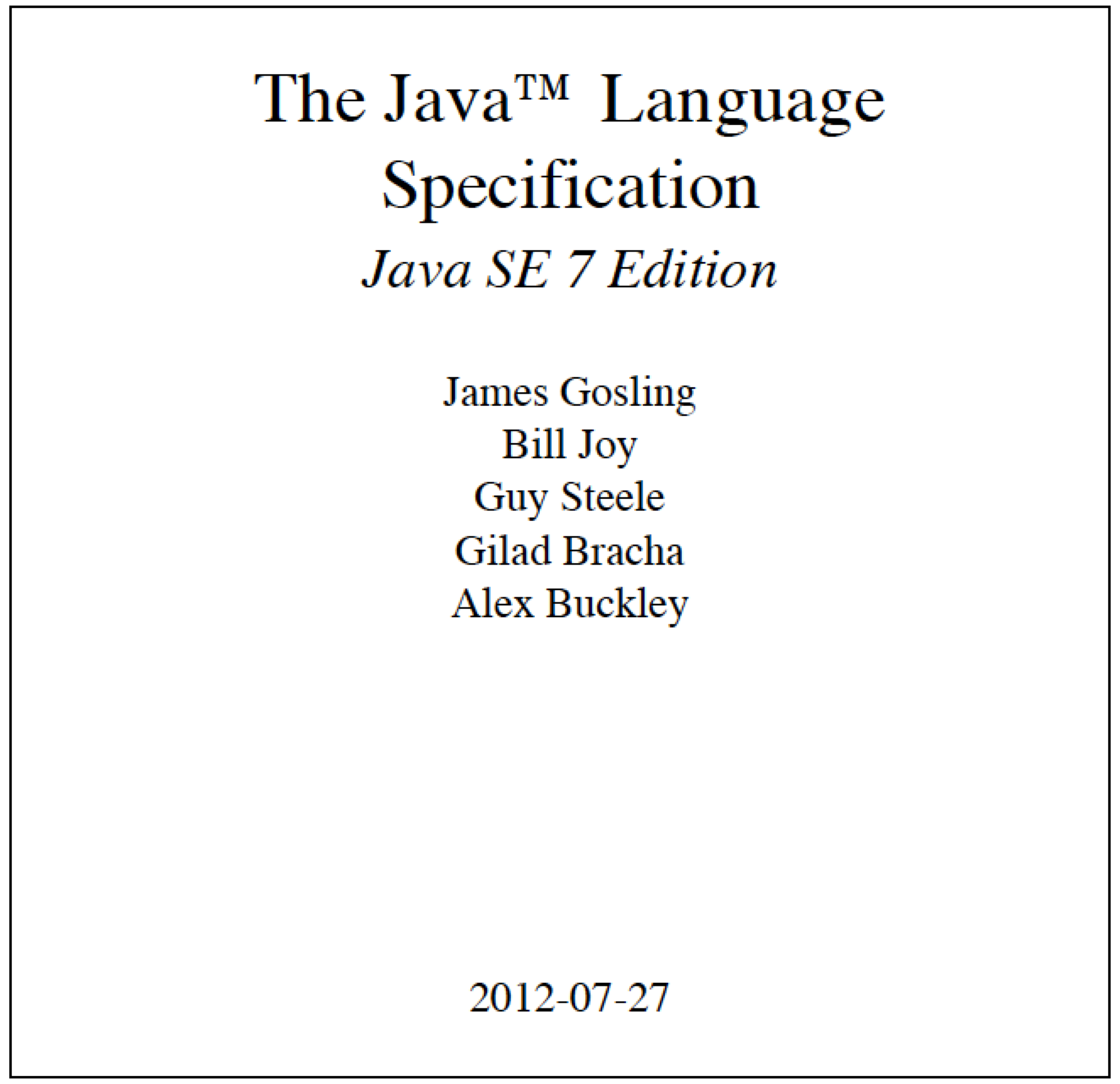
parser

type checker

code generator





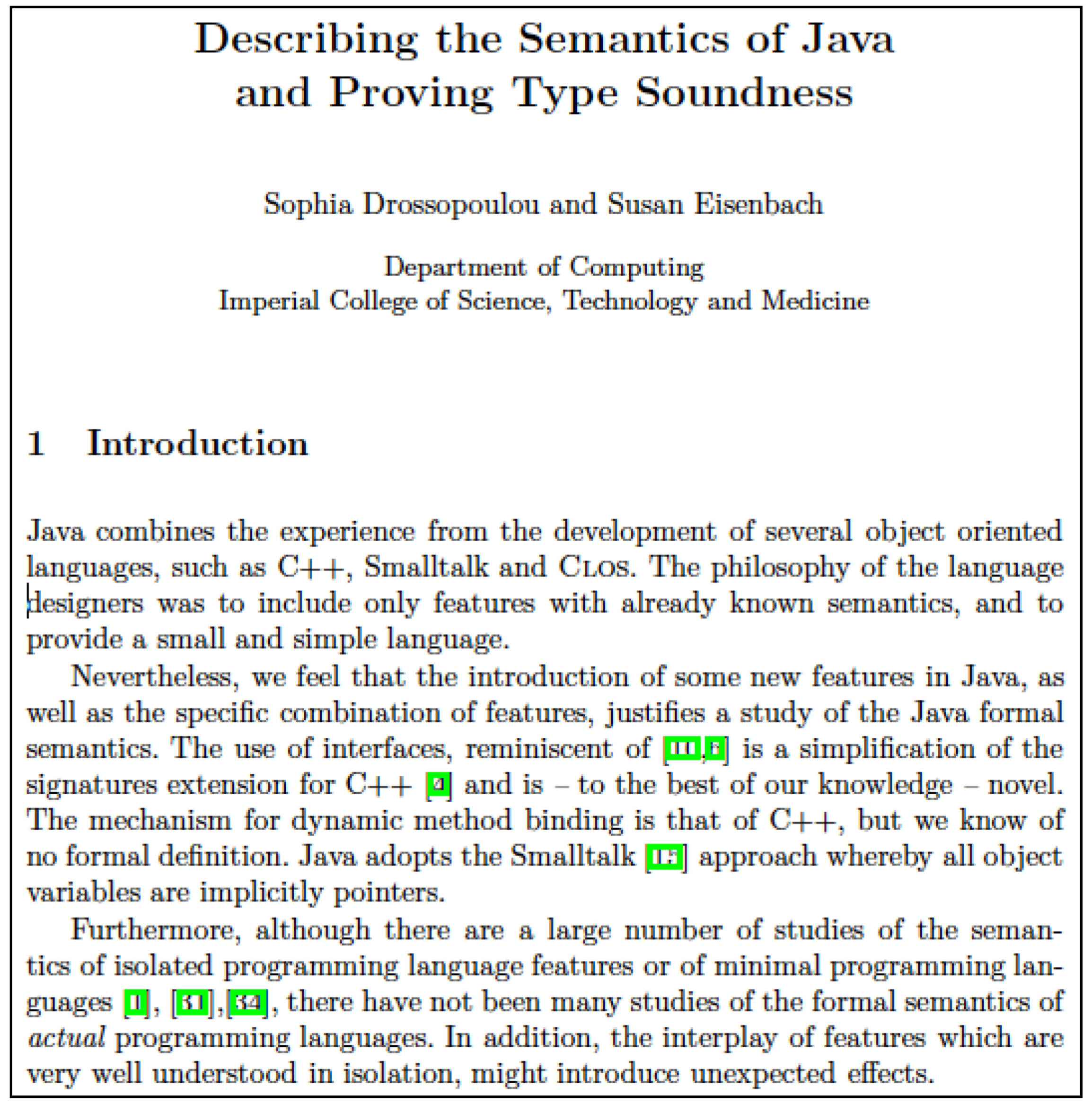


The Java™ Language Specification

Java SE 7 Edition

James Gosling
Bill Joy
Guy Steele
Gilad Bracha
Alex Buckley

2012-07-27



Describing the Semantics of Java and Proving Type Soundness

Sophia Drossopoulou and Susan Eisenbach

Department of Computing
Imperial College of Science, Technology and Medicine

1 Introduction

Java combines the experience from the development of several object oriented languages, such as C++, Smalltalk and CLOS. The philosophy of the language designers was to include only features with already known semantics, and to provide a small and simple language.

Nevertheless, we feel that the introduction of some new features in Java, as well as the specific combination of features, justifies a study of the Java formal semantics. The use of interfaces, reminiscent of [11,6] is a simplification of the signatures extension for C++ [4] and is – to the best of our knowledge – novel. The mechanism for dynamic method binding is that of C++, but we know of no formal definition. Java adopts the Smalltalk [15] approach whereby all object variables are implicitly pointers.

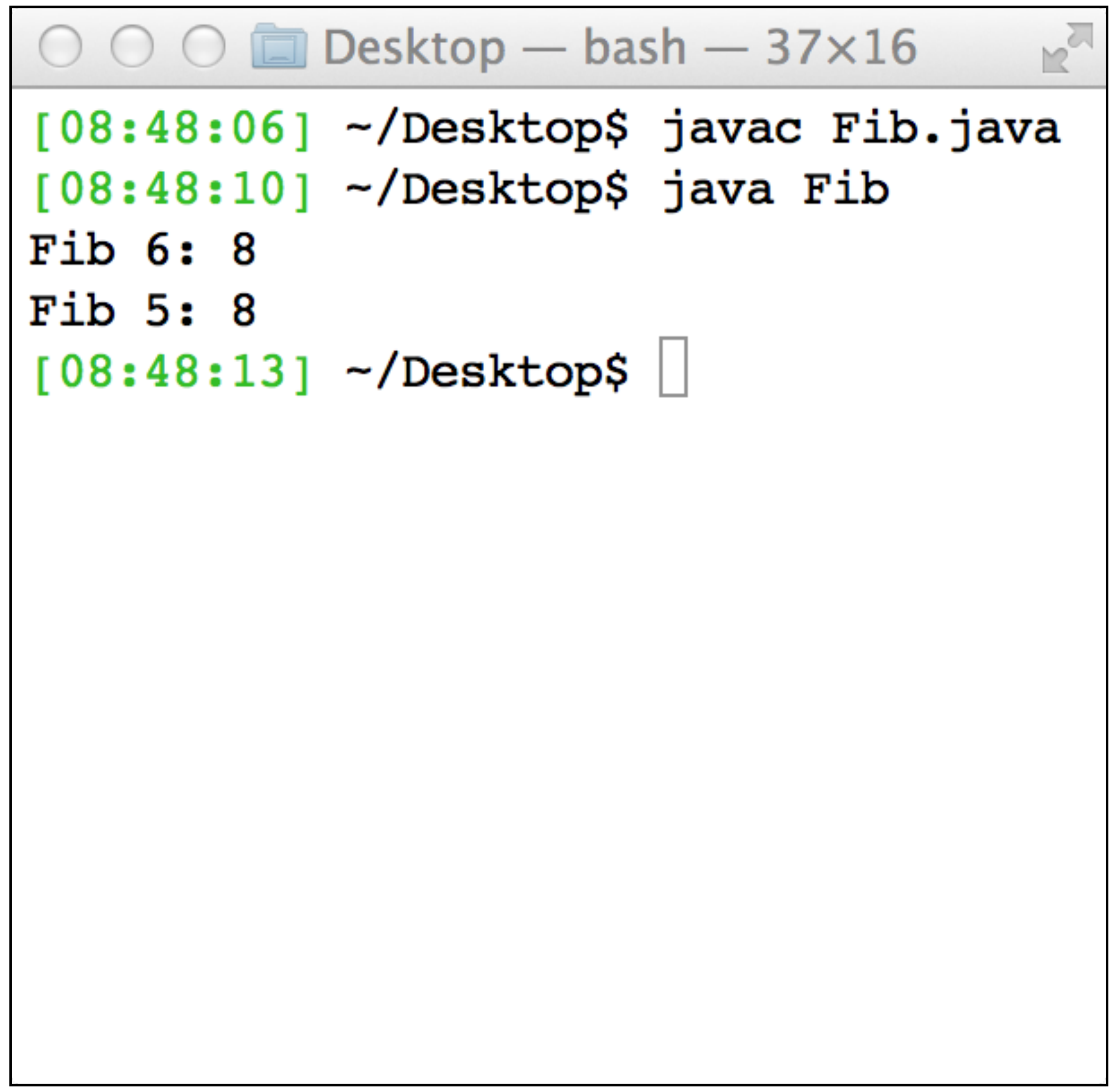
Furthermore, although there are a large number of studies of the semantics of isolated programming language features or of minimal programming languages [1], [31],[34], there have not been many studies of the formal semantics of *actual* programming languages. In addition, the interplay of features which are very well understood in isolation, might introduce unexpected effects.

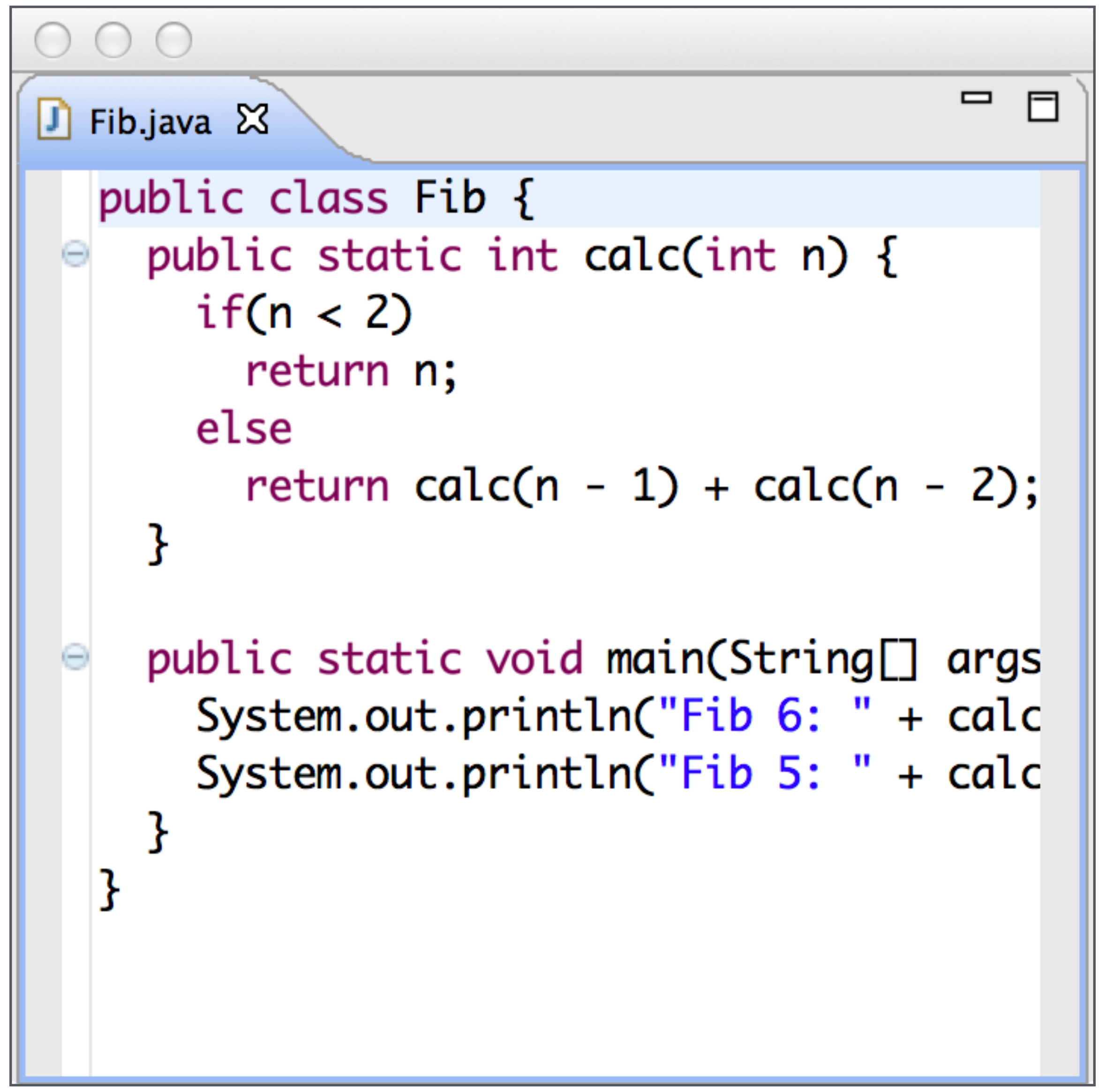
parser

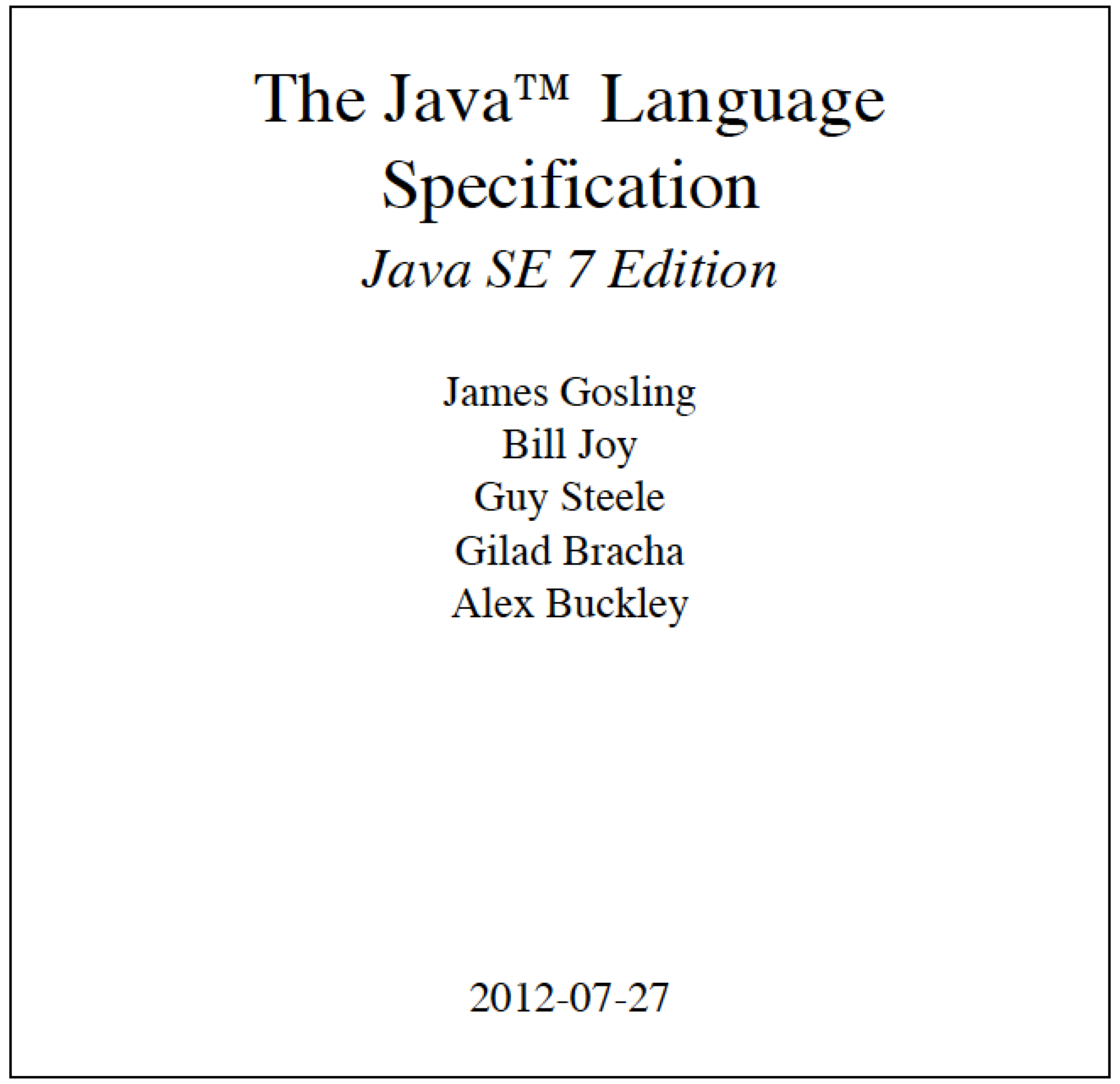
type checker

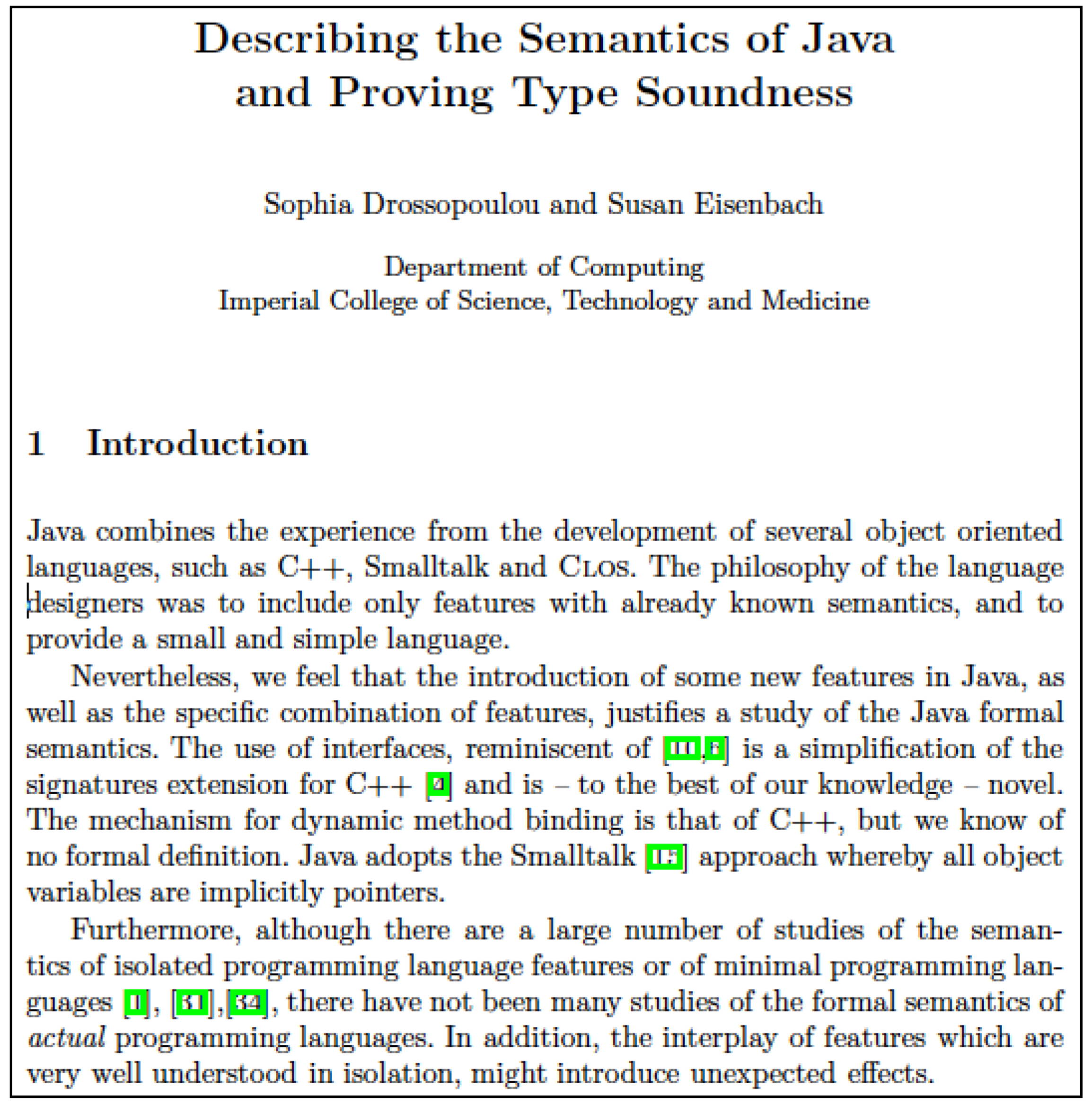
code generator

interpreter









Describing the Semantics of Java and Proving Type Soundness

Sophia Drossopoulou and Susan Eisenbach

Department of Computing
Imperial College of Science, Technology and Medicine

1 Introduction

Java combines the experience from the development of several object oriented languages, such as C++, Smalltalk and CLOS. The philosophy of the language designers was to include only features with already known semantics, and to provide a small and simple language.

Nevertheless, we feel that the introduction of some new features in Java, as well as the specific combination of features, justifies a study of the Java formal semantics. The use of interfaces, reminiscent of [11,6] is a simplification of the signatures extension for C++ [4] and is – to the best of our knowledge – novel. The mechanism for dynamic method binding is that of C++, but we know of no formal definition. Java adopts the Smalltalk [15] approach whereby all object variables are implicitly pointers.

Furthermore, although there are a large number of studies of the semantics of isolated programming language features or of minimal programming languages [1], [31],[34], there have not been many studies of the formal semantics of *actual* programming languages. In addition, the interplay of features which are very well understood in isolation, might introduce unexpected effects.

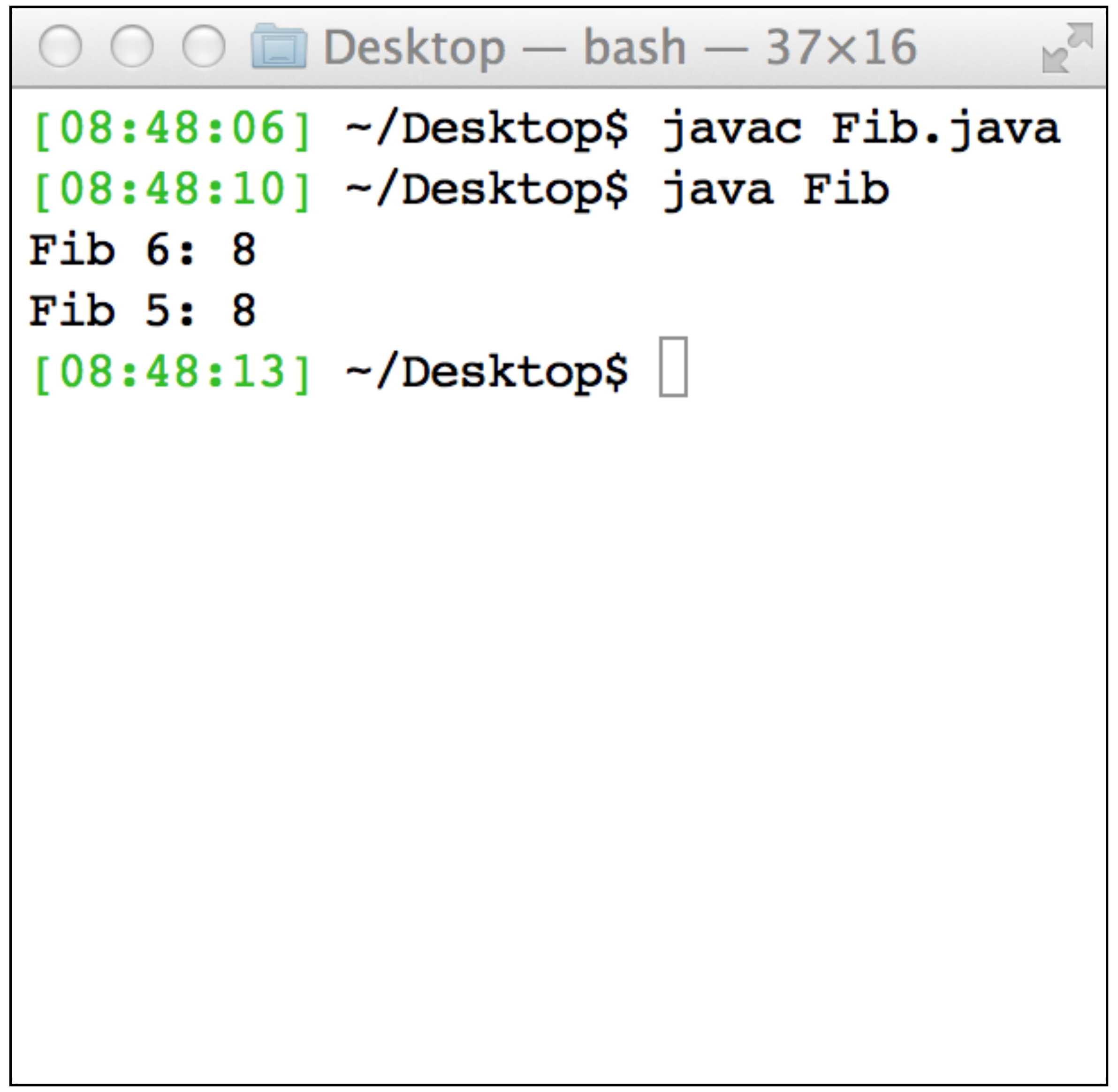
parser

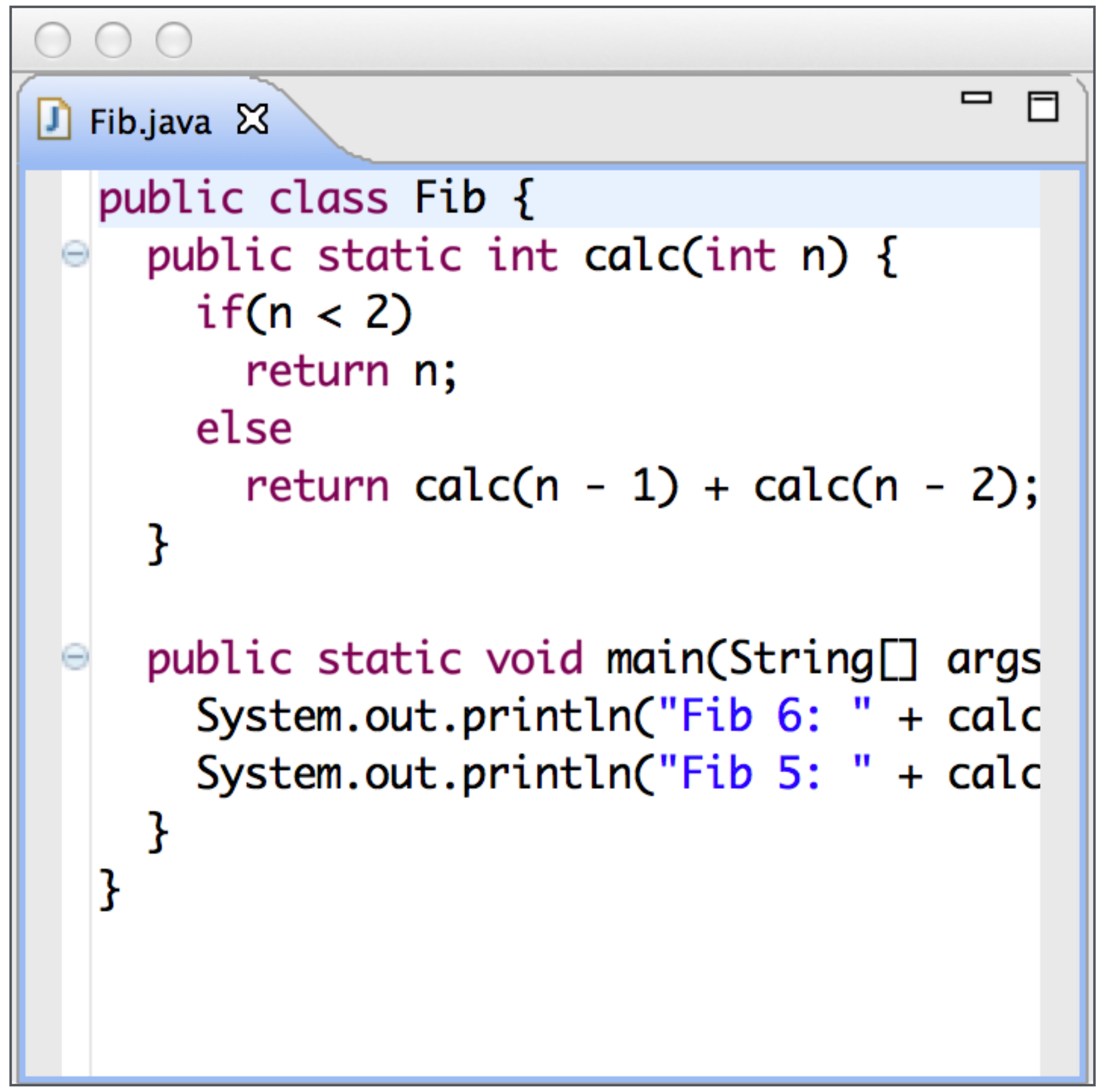
type checker

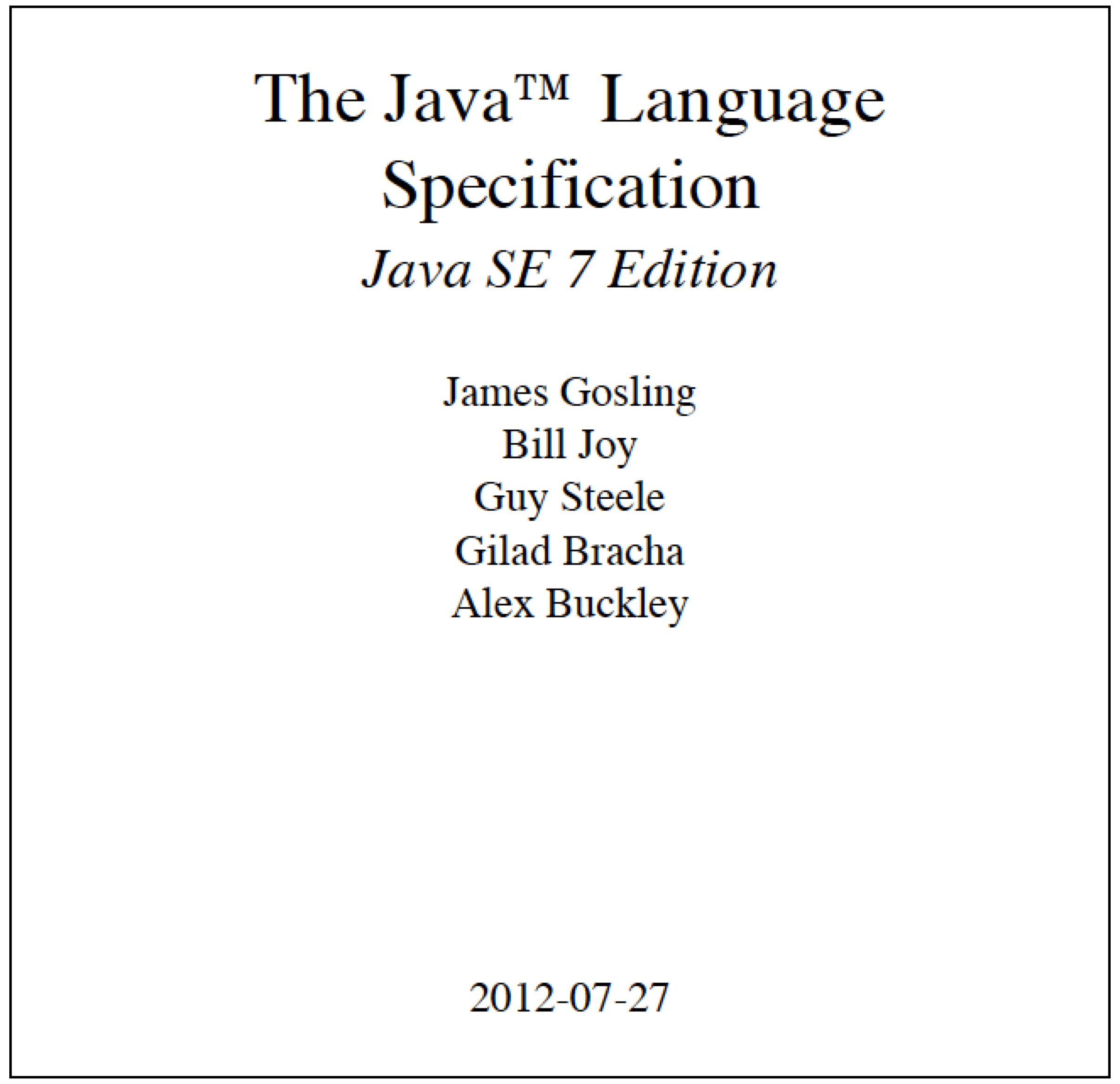
code generator

interpreter

parser





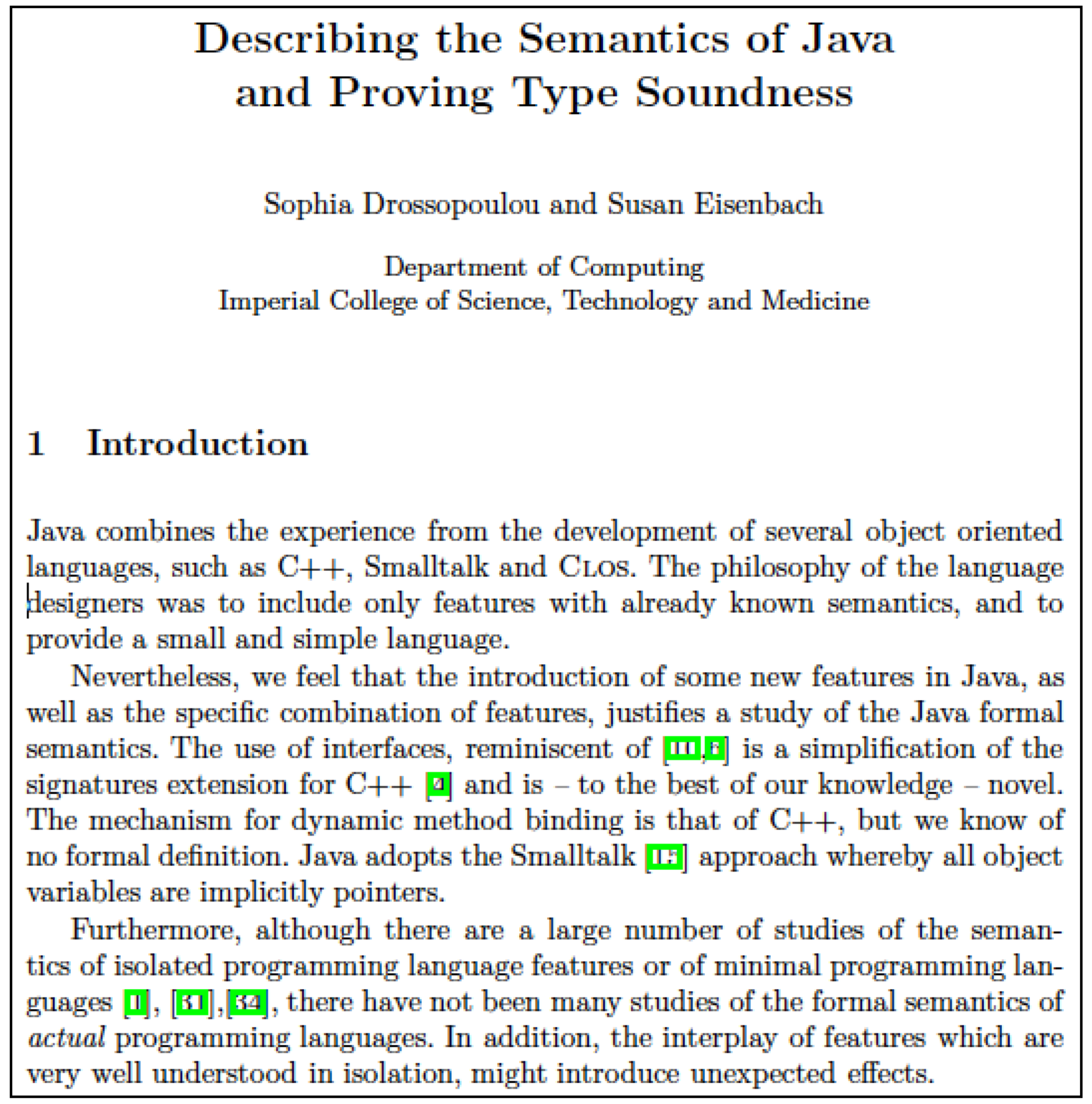


The Java™ Language Specification

Java SE 7 Edition

James Gosling
Bill Joy
Guy Steele
Gilad Bracha
Alex Buckley

2012-07-27



Describing the Semantics of Java and Proving Type Soundness

Sophia Drossopoulou and Susan Eisenbach

Department of Computing
Imperial College of Science, Technology and Medicine

1 Introduction

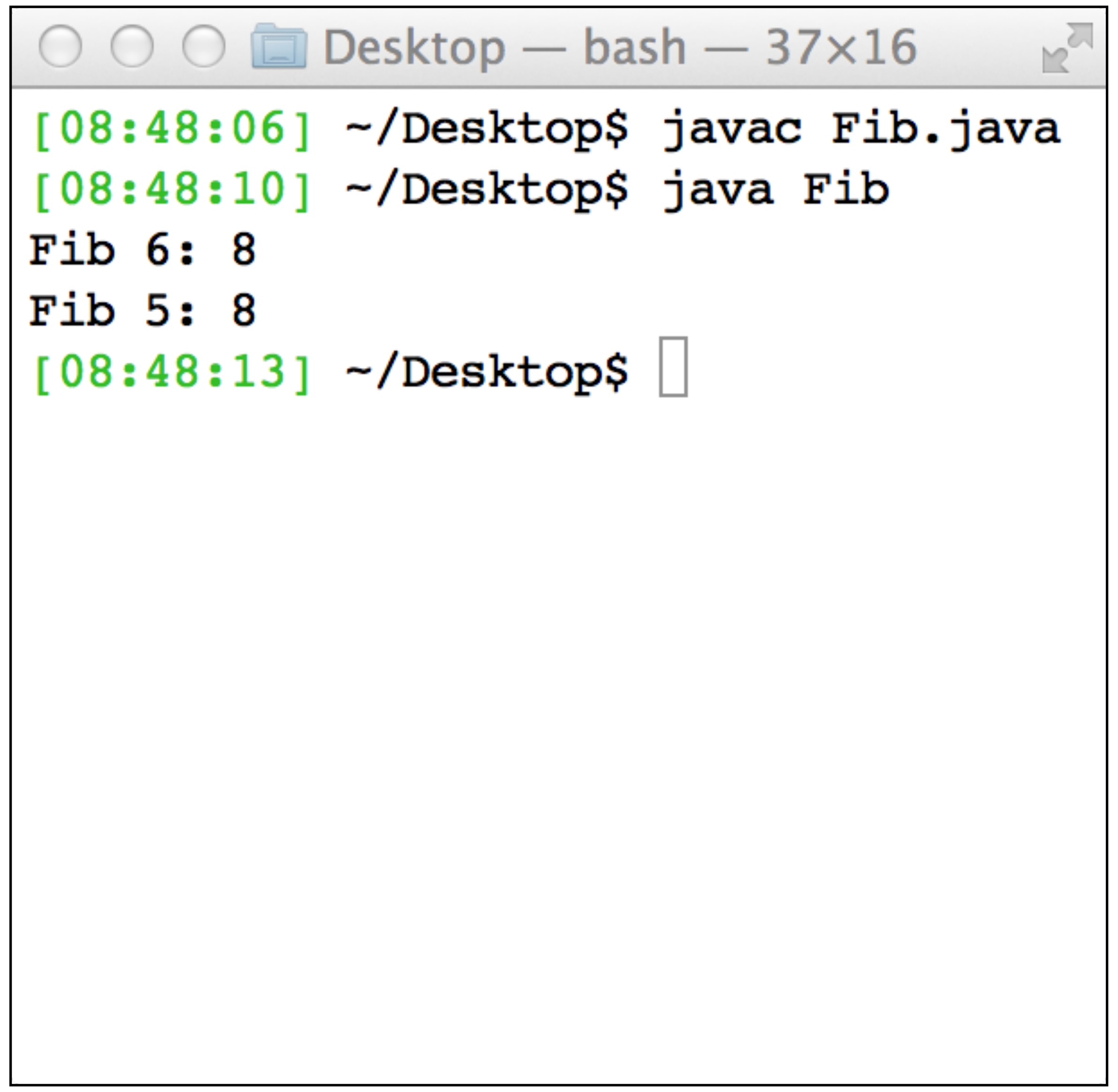
Java combines the experience from the development of several object oriented languages, such as C++, Smalltalk and CLOS. The philosophy of the language designers was to include only features with already known semantics, and to provide a small and simple language.

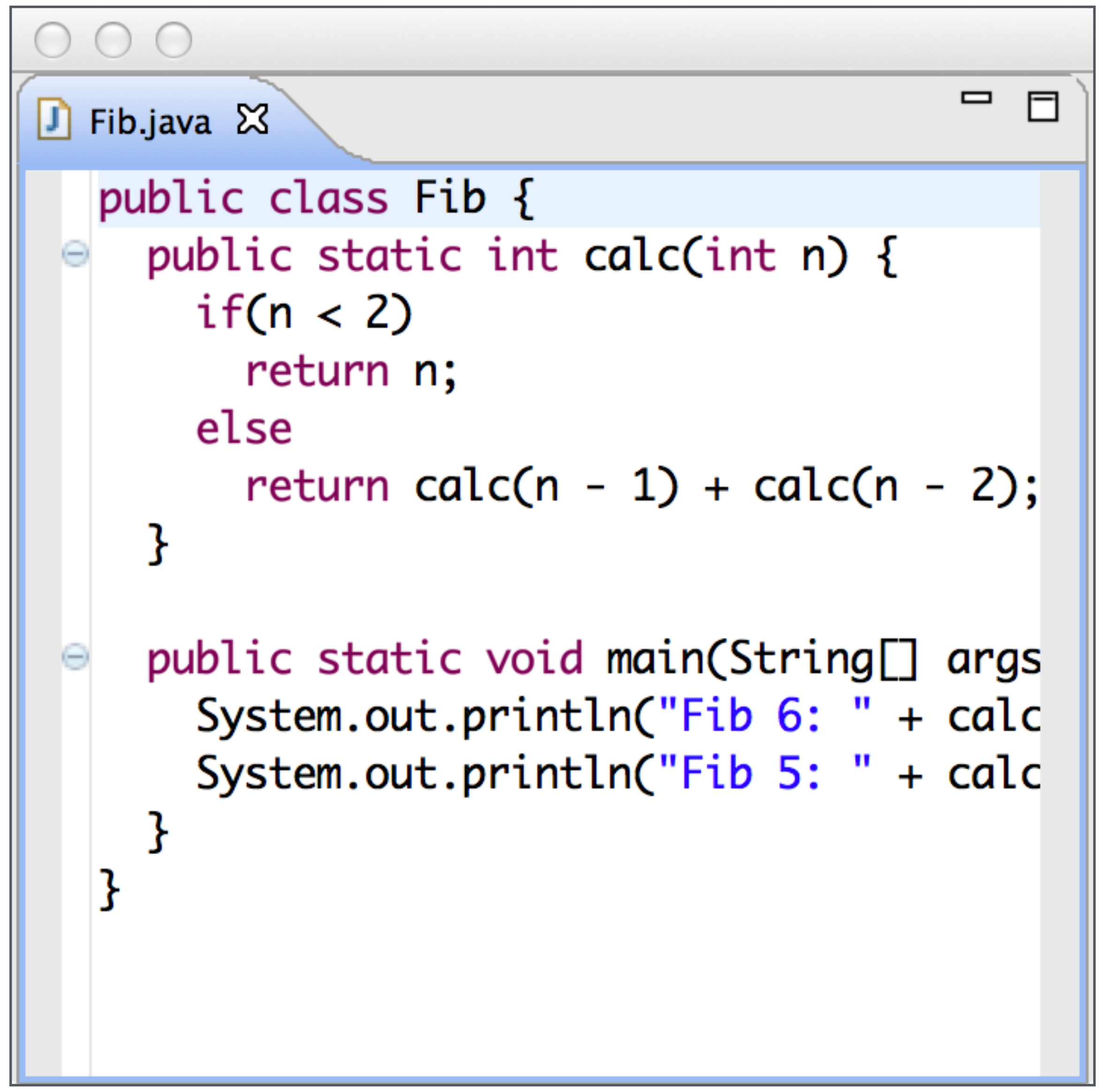
Nevertheless, we feel that the introduction of some new features in Java, as well as the specific combination of features, justifies a study of the Java formal semantics. The use of interfaces, reminiscent of [11,5] is a simplification of the signatures extension for C++ [4] and is – to the best of our knowledge – novel. The mechanism for dynamic method binding is that of C++, but we know of no formal definition. Java adopts the Smalltalk [15] approach whereby all object variables are implicitly pointers.

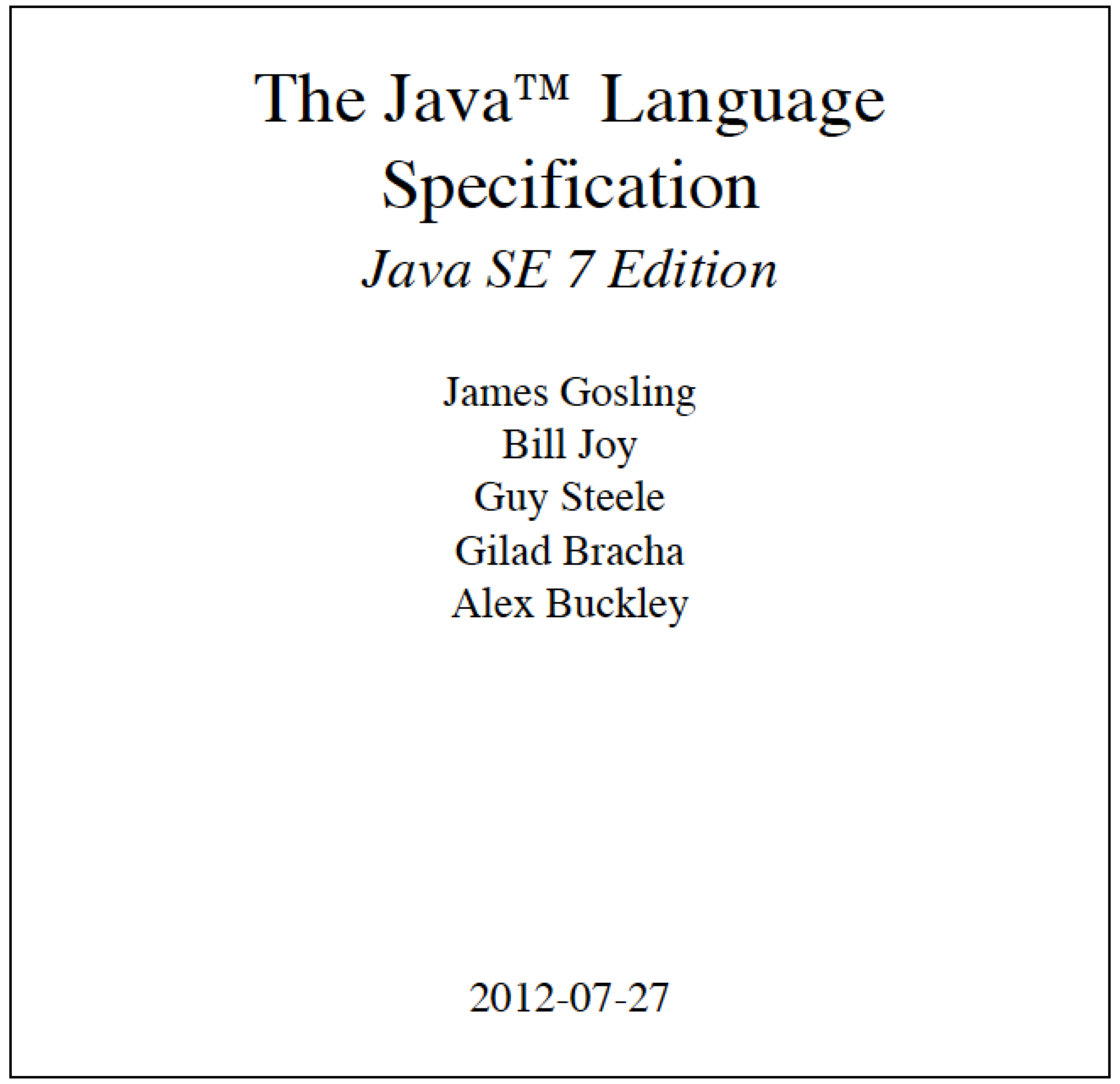
Furthermore, although there are a large number of studies of the semantics of isolated programming language features or of minimal programming languages [1], [31],[32], there have not been many studies of the formal semantics of *actual* programming languages. In addition, the interplay of features which are very well understood in isolation, might introduce unexpected effects.

parser
type checker
code generator
interpreter

parser
error recovery





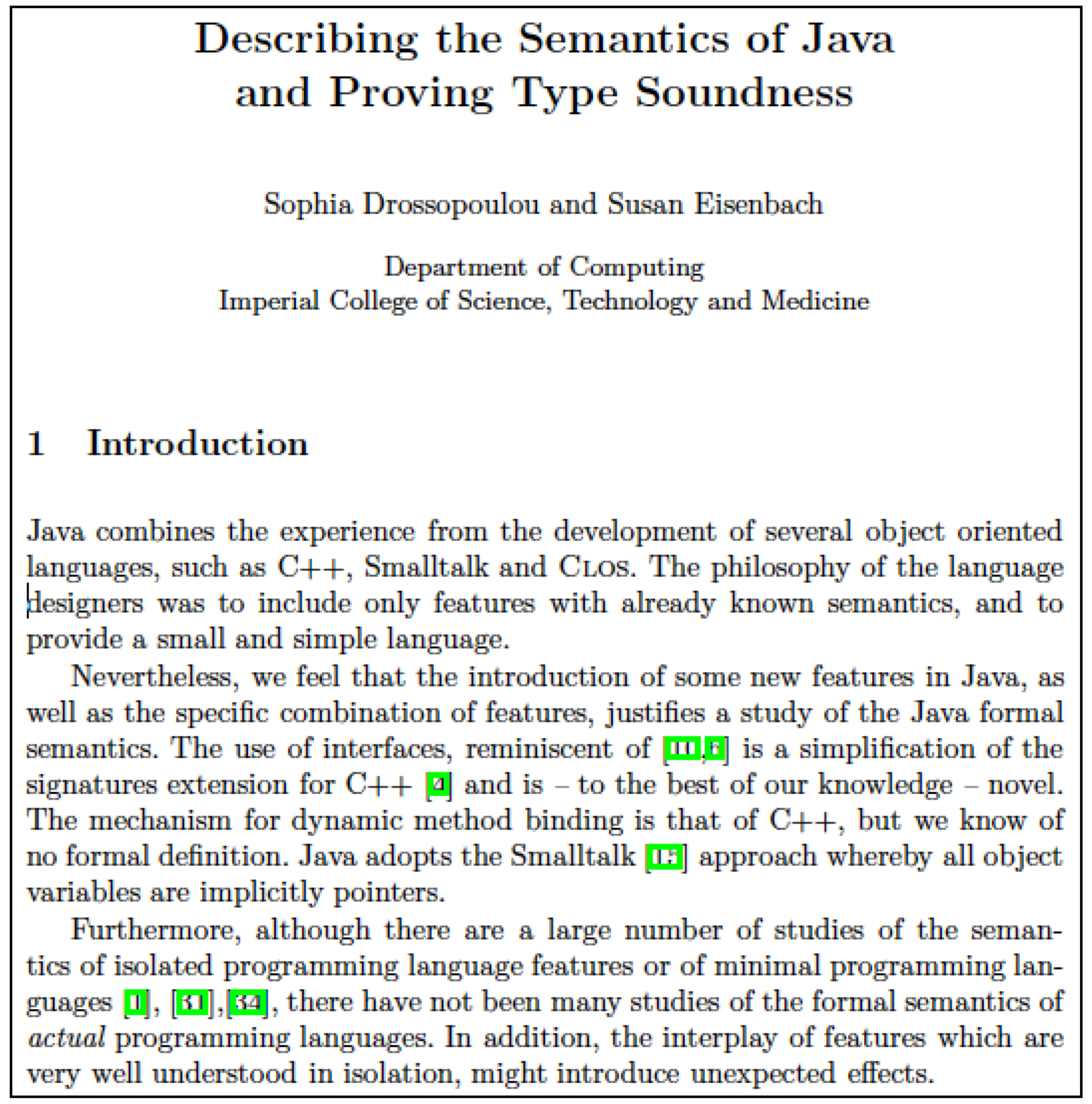


The Java™ Language Specification

Java SE 7 Edition

James Gosling
Bill Joy
Guy Steele
Gilad Bracha
Alex Buckley

2012-07-27



Describing the Semantics of Java and Proving Type Soundness

Sophia Drossopoulou and Susan Eisenbach

Department of Computing
Imperial College of Science, Technology and Medicine

1 Introduction

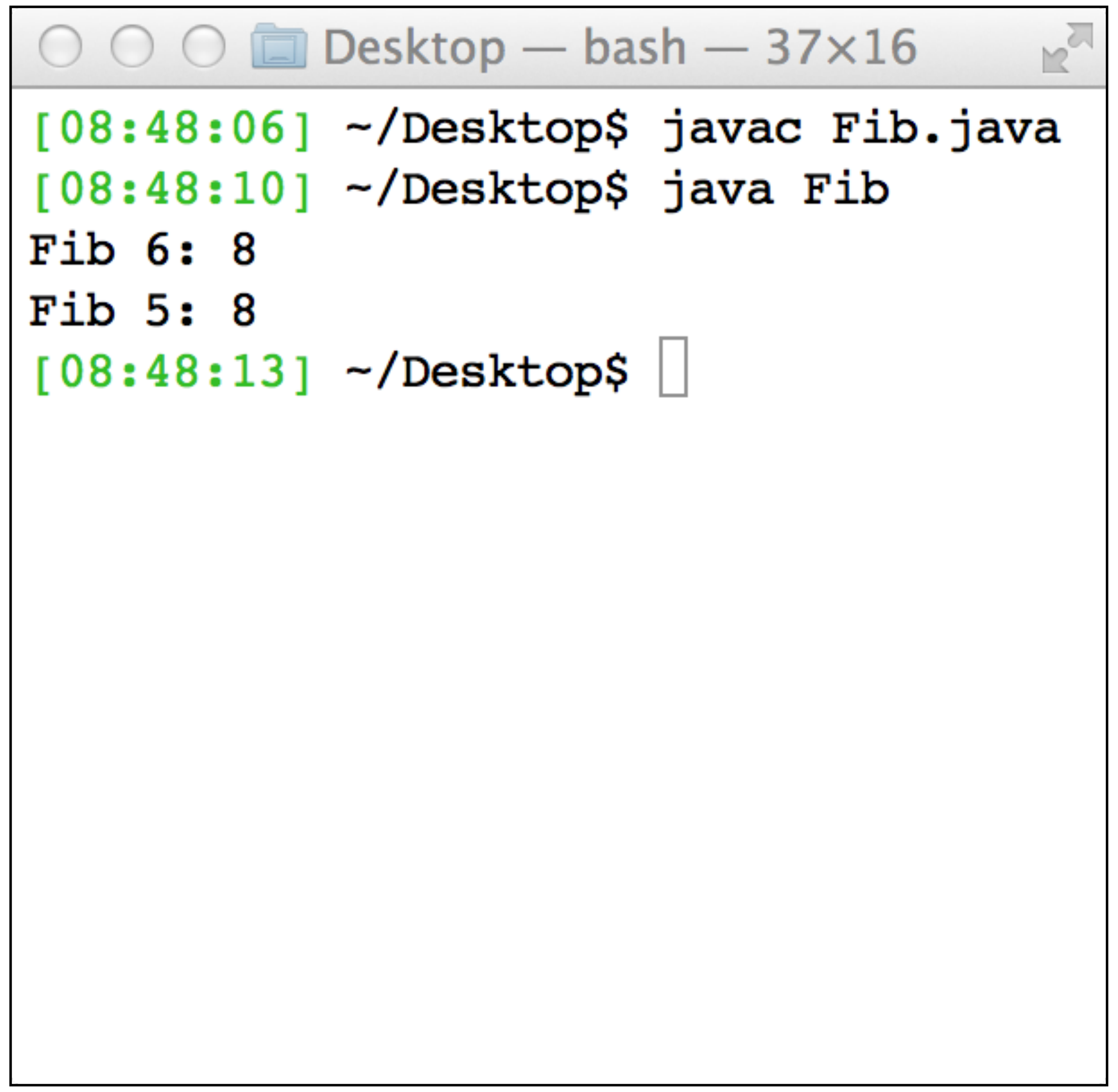
Java combines the experience from the development of several object oriented languages, such as C++, Smalltalk and CLOS. The philosophy of the language designers was to include only features with already known semantics, and to provide a small and simple language.

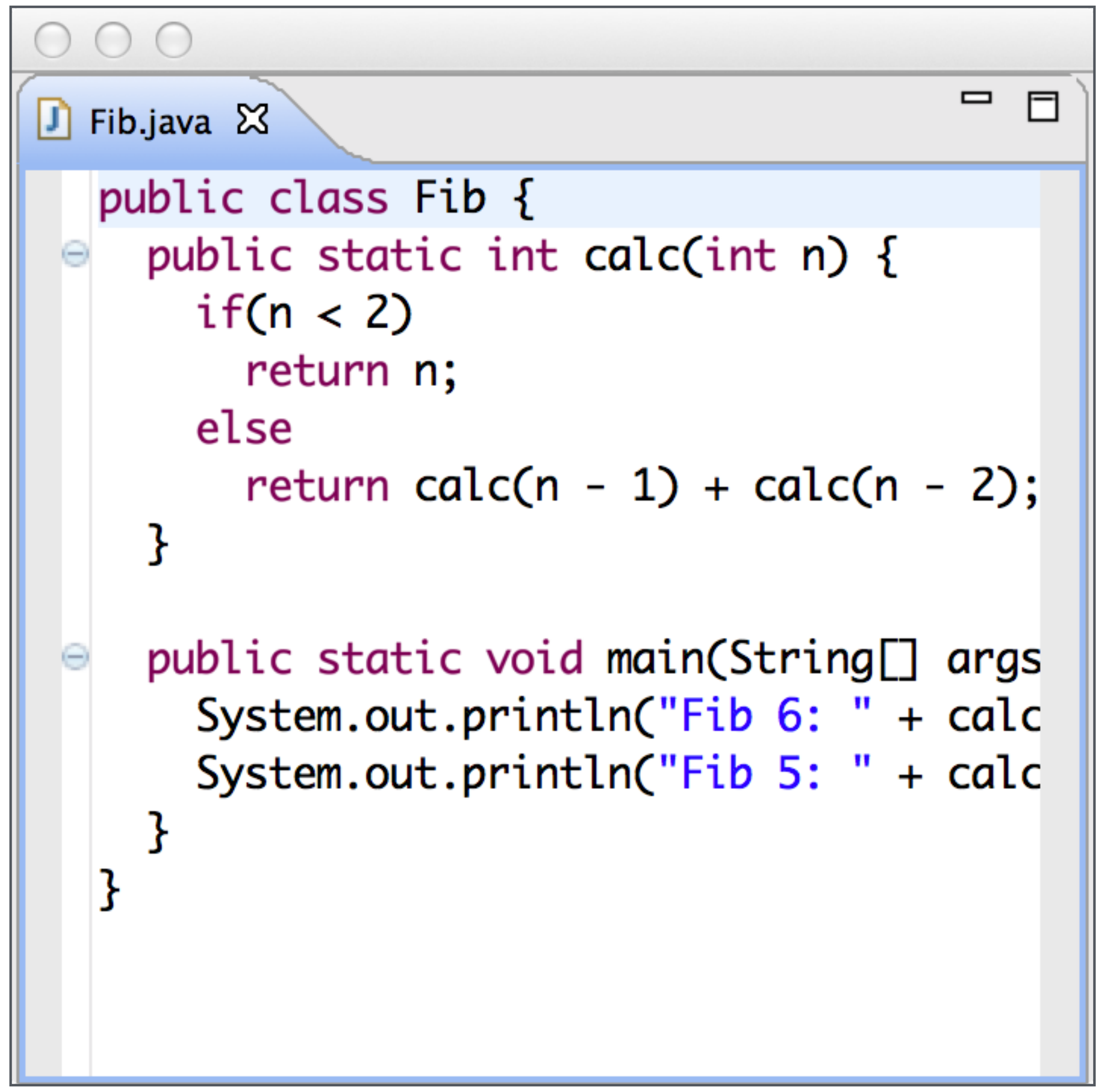
Nevertheless, we feel that the introduction of some new features in Java, as well as the specific combination of features, justifies a study of the Java formal semantics. The use of interfaces, reminiscent of [11,6] is a simplification of the signatures extension for C++ [4] and is – to the best of our knowledge – novel. The mechanism for dynamic method binding is that of C++, but we know of no formal definition. Java adopts the Smalltalk [15] approach whereby all object variables are implicitly pointers.

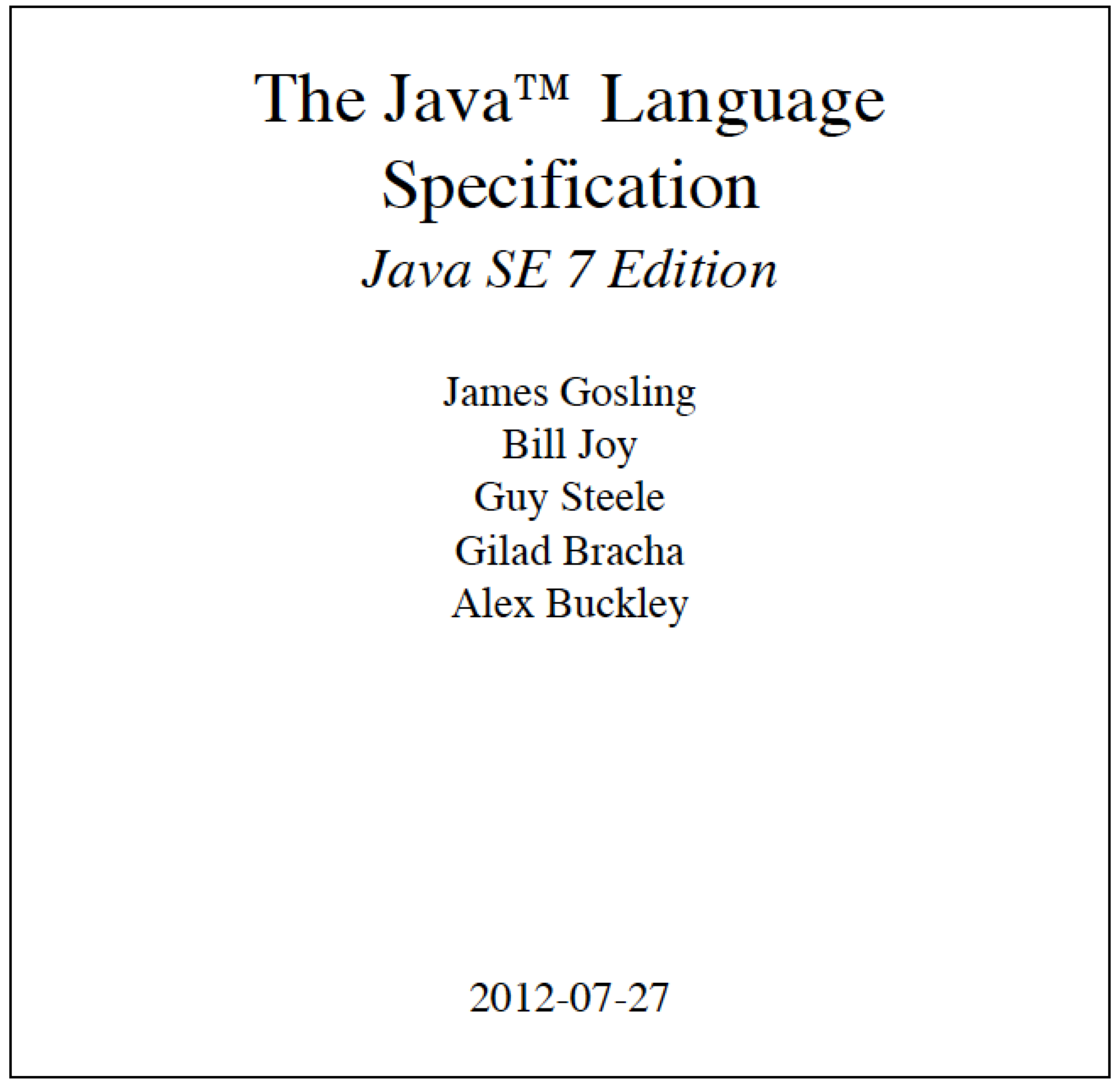
Furthermore, although there are a large number of studies of the semantics of isolated programming language features or of minimal programming languages [1], [31],[34], there have not been many studies of the formal semantics of *actual* programming languages. In addition, the interplay of features which are very well understood in isolation, might introduce unexpected effects.

parser
type checker
code generator
interpreter

parser
error recovery
syntax highlighting





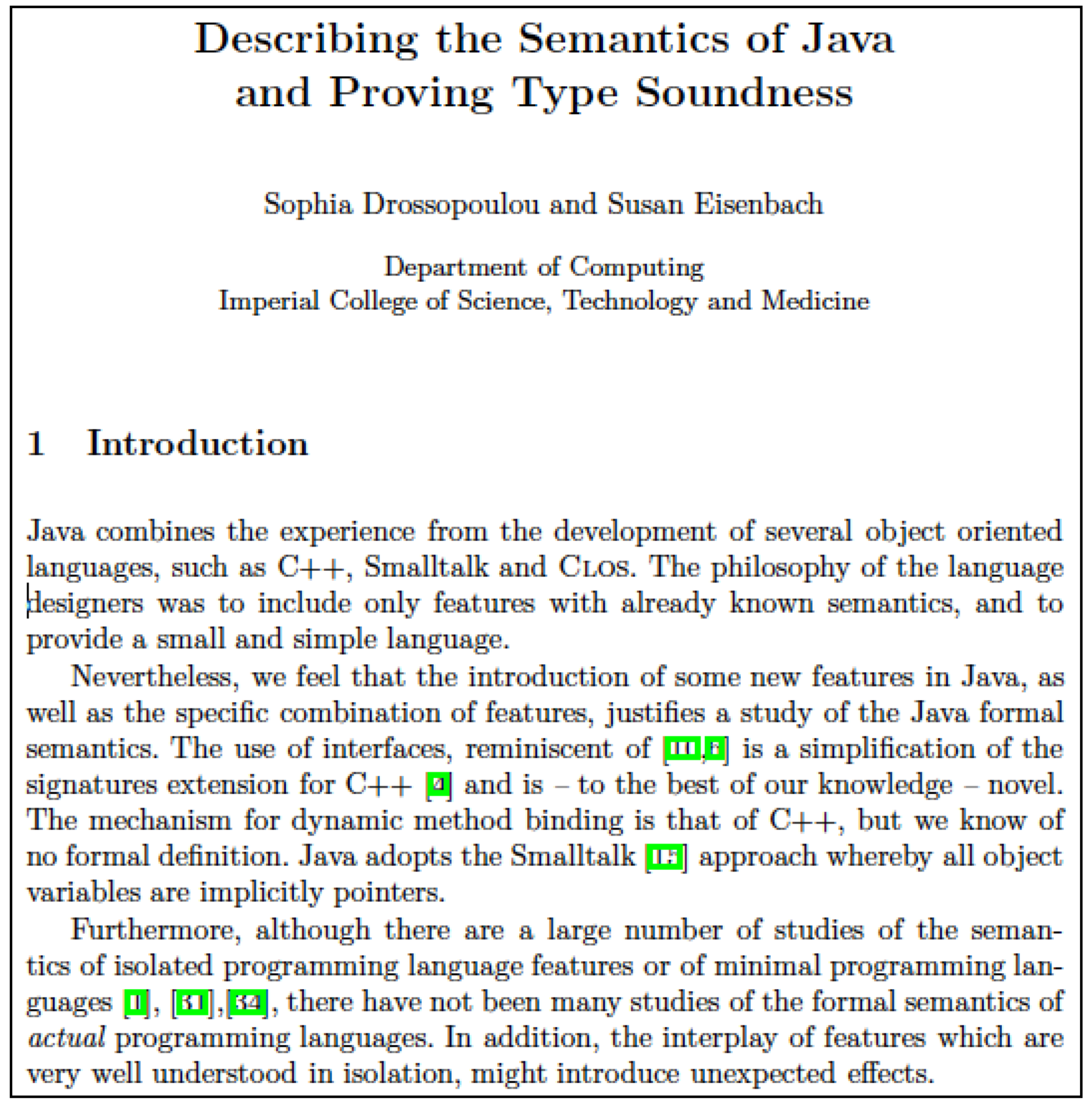


The Java™ Language Specification

Java SE 7 Edition

James Gosling
Bill Joy
Guy Steele
Gilad Bracha
Alex Buckley

2012-07-27



Describing the Semantics of Java and Proving Type Soundness

Sophia Drossopoulou and Susan Eisenbach

Department of Computing
Imperial College of Science, Technology and Medicine

1 Introduction

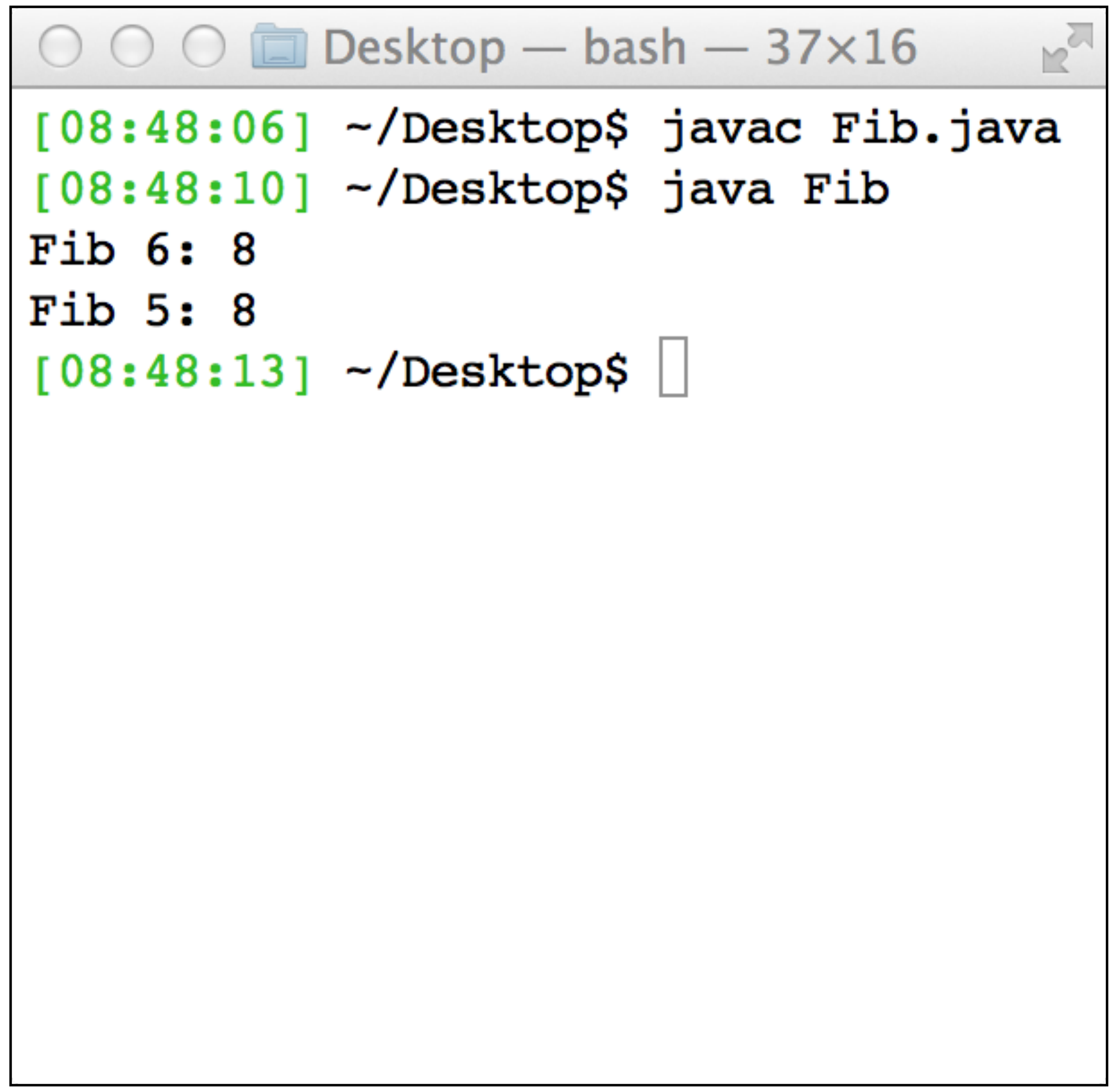
Java combines the experience from the development of several object oriented languages, such as C++, Smalltalk and CLOS. The philosophy of the language designers was to include only features with already known semantics, and to provide a small and simple language.

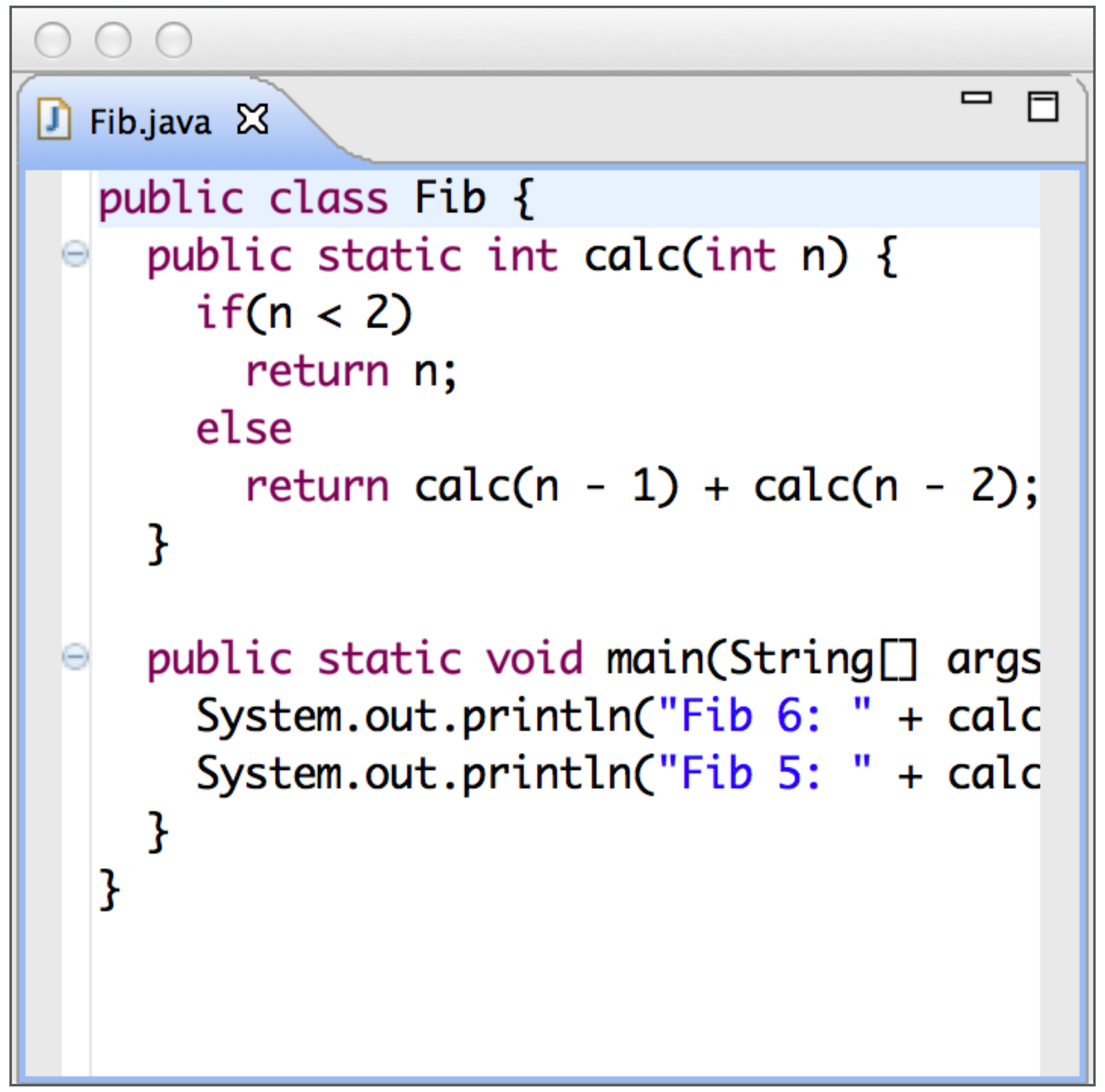
Nevertheless, we feel that the introduction of some new features in Java, as well as the specific combination of features, justifies a study of the Java formal semantics. The use of interfaces, reminiscent of [11,6] is a simplification of the signatures extension for C++ [4] and is – to the best of our knowledge – novel. The mechanism for dynamic method binding is that of C++, but we know of no formal definition. Java adopts the Smalltalk [15] approach whereby all object variables are implicitly pointers.

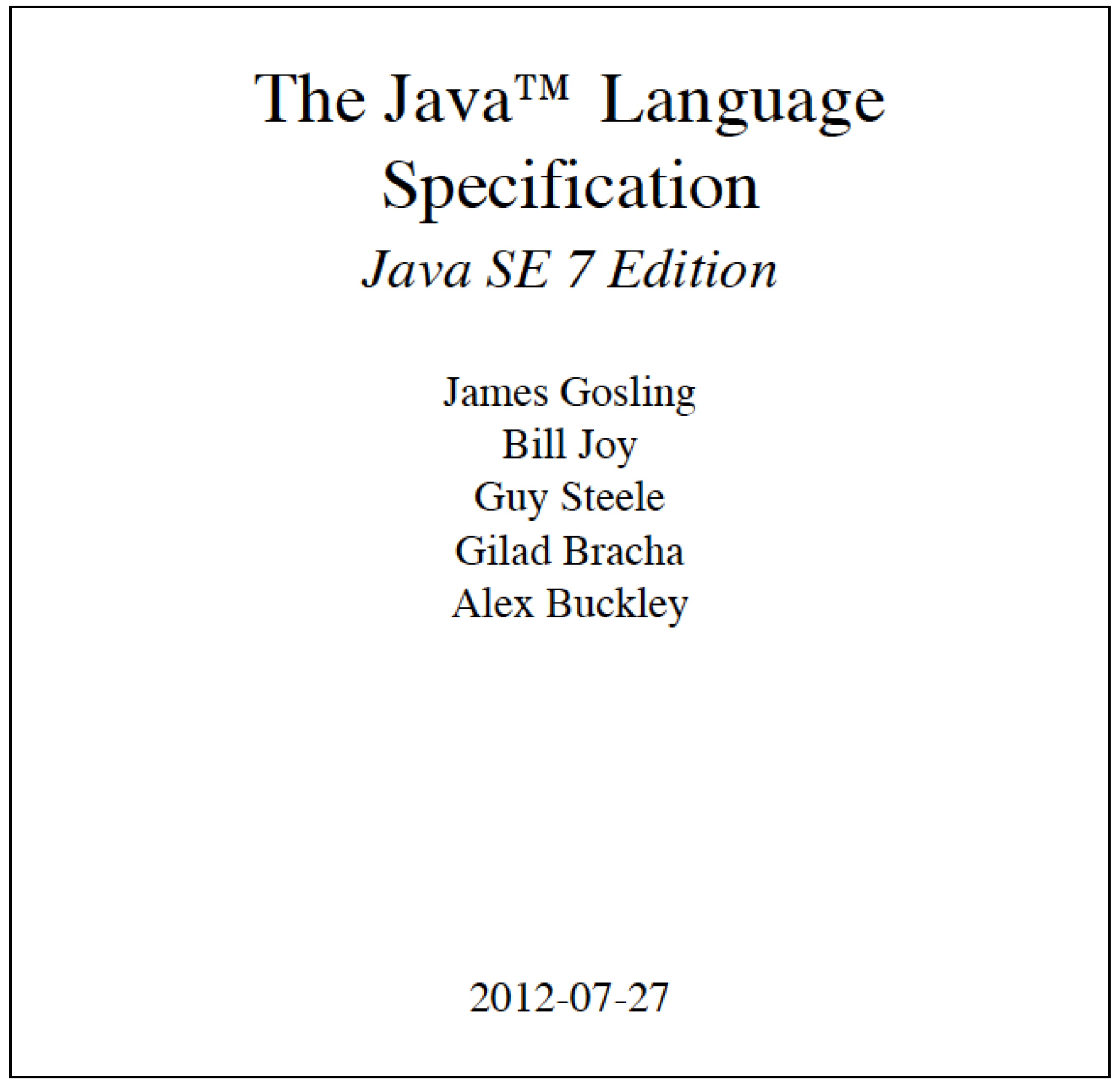
Furthermore, although there are a large number of studies of the semantics of isolated programming language features or of minimal programming languages [1], [31],[34], there have not been many studies of the formal semantics of *actual* programming languages. In addition, the interplay of features which are very well understood in isolation, might introduce unexpected effects.

parser
type checker
code generator
interpreter

parser
error recovery
syntax highlighting
outline





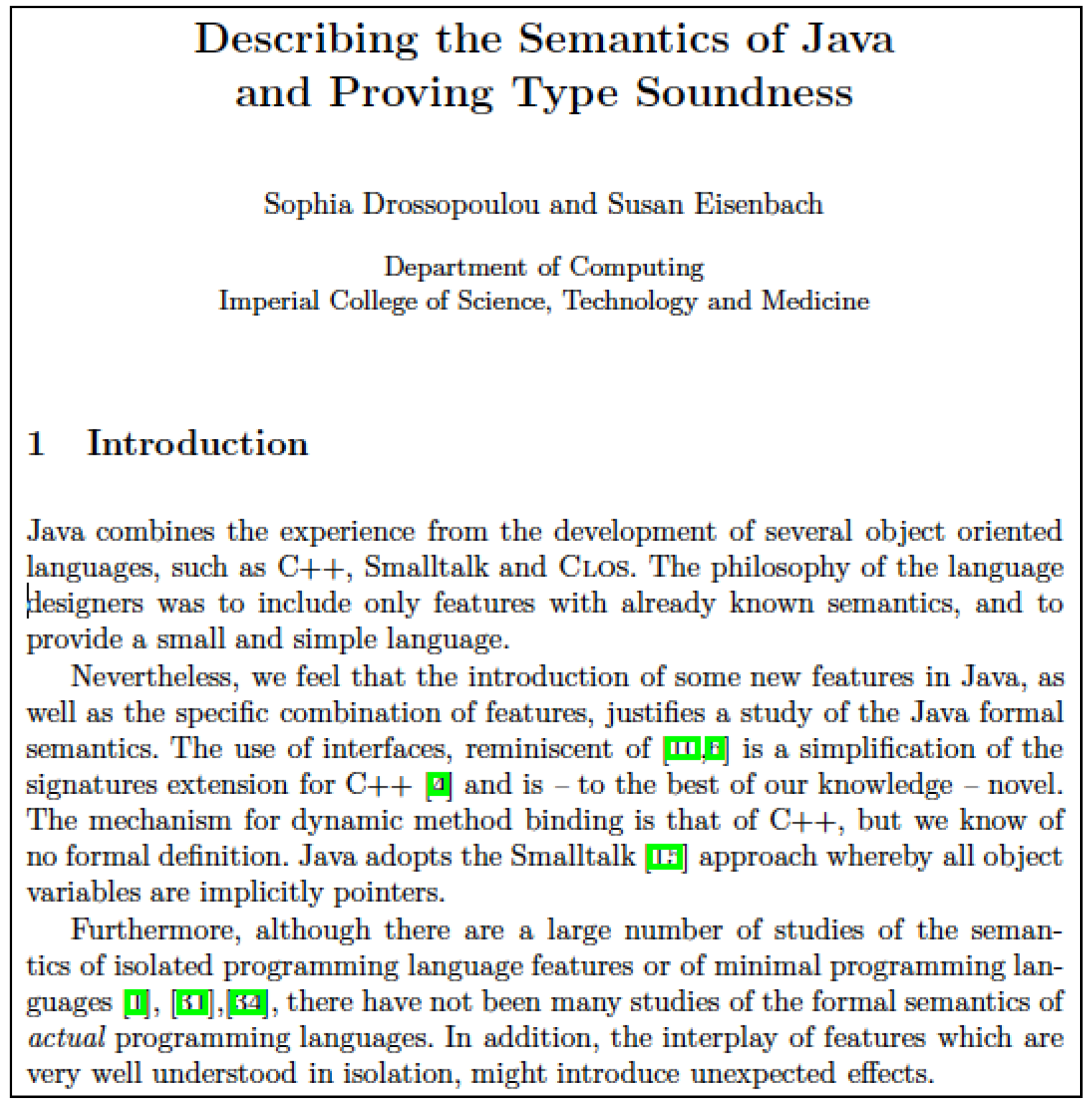


The Java™ Language Specification

Java SE 7 Edition

James Gosling
Bill Joy
Guy Steele
Gilad Bracha
Alex Buckley

2012-07-27



Describing the Semantics of Java and Proving Type Soundness

Sophia Drossopoulou and Susan Eisenbach

Department of Computing
Imperial College of Science, Technology and Medicine

1 Introduction

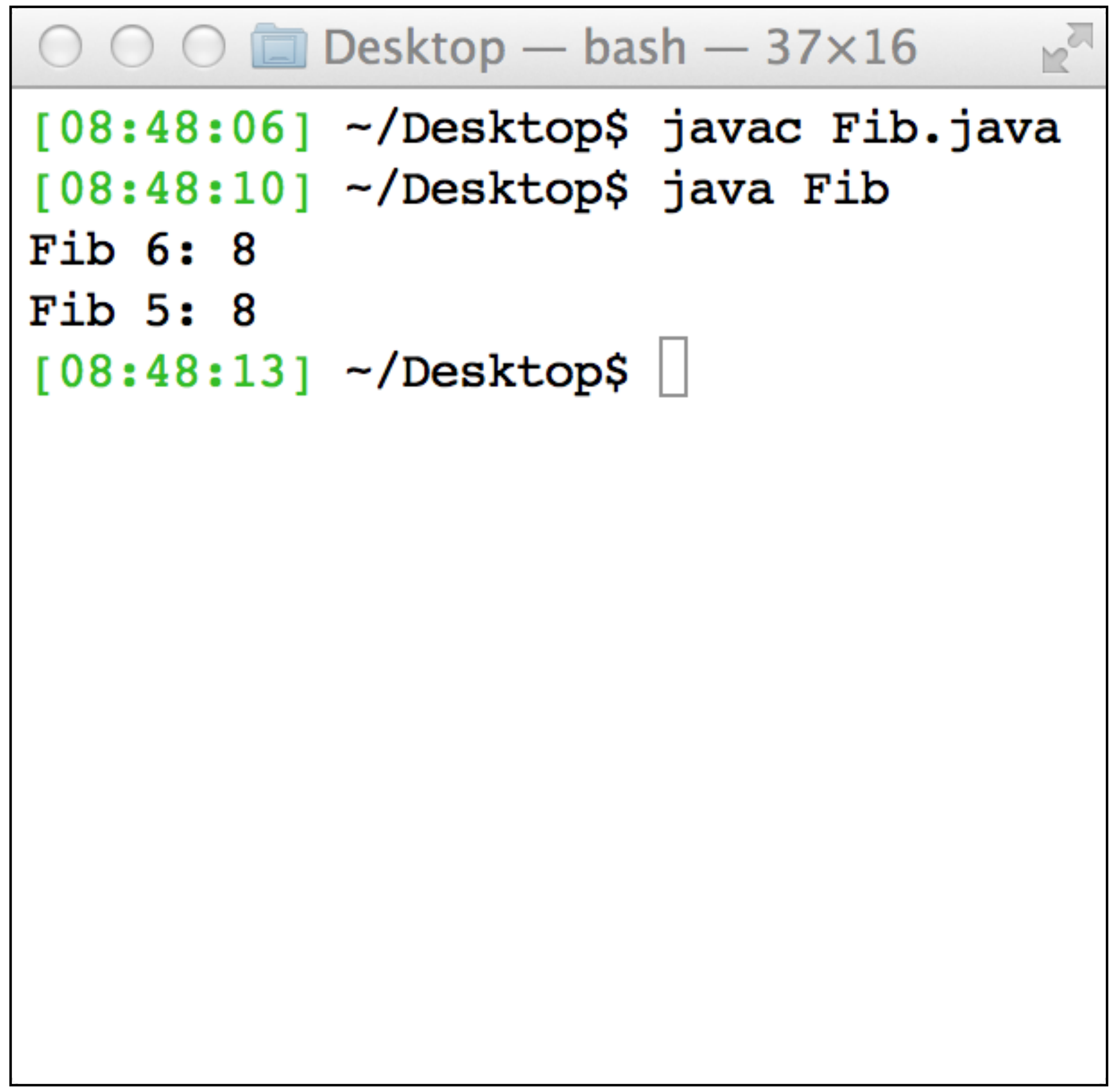
Java combines the experience from the development of several object oriented languages, such as C++, Smalltalk and CLOS. The philosophy of the language designers was to include only features with already known semantics, and to provide a small and simple language.

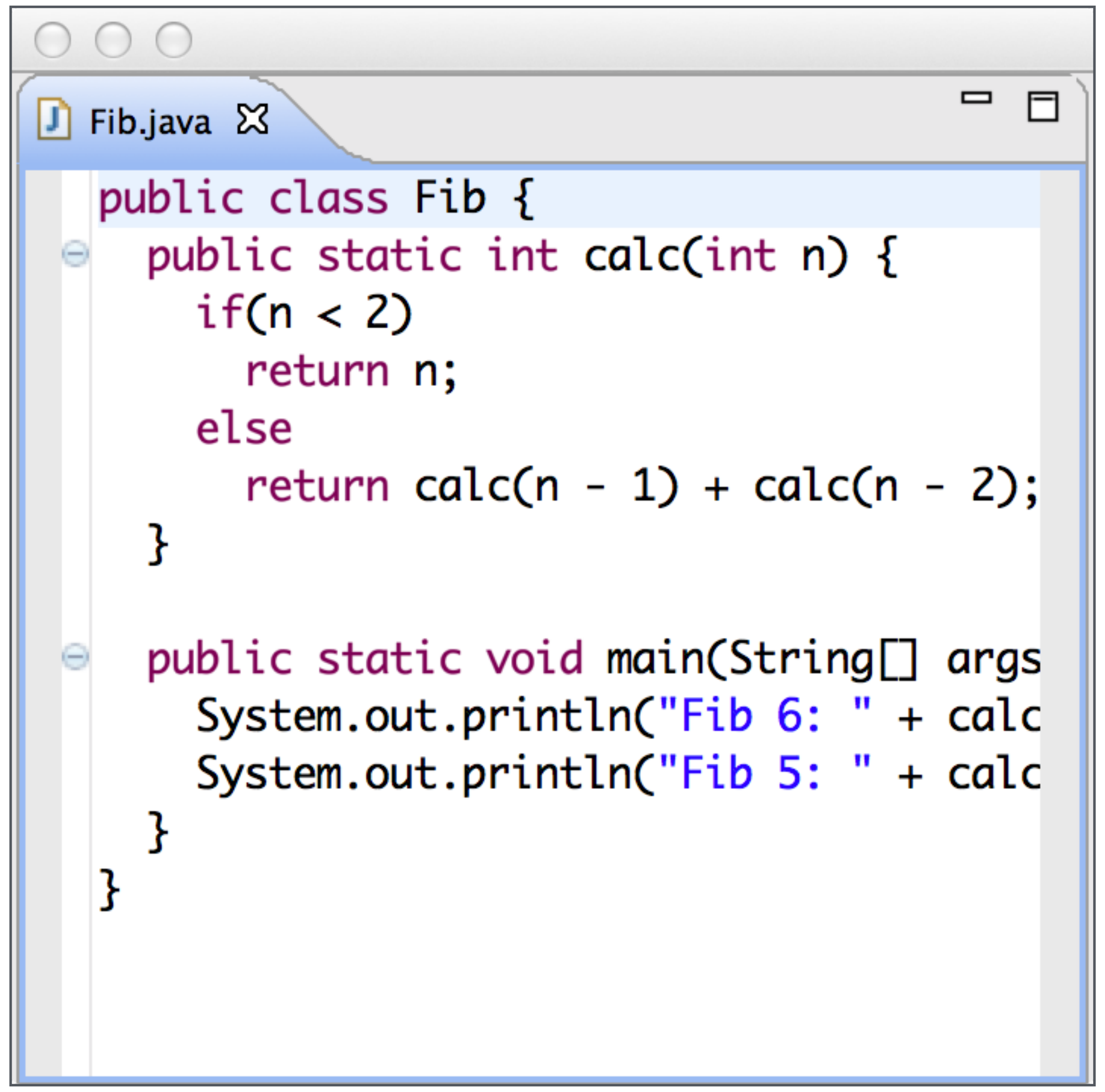
Nevertheless, we feel that the introduction of some new features in Java, as well as the specific combination of features, justifies a study of the Java formal semantics. The use of interfaces, reminiscent of [11,6] is a simplification of the signatures extension for C++ [3] and is – to the best of our knowledge – novel. The mechanism for dynamic method binding is that of C++, but we know of no formal definition. Java adopts the Smalltalk [15] approach whereby all object variables are implicitly pointers.

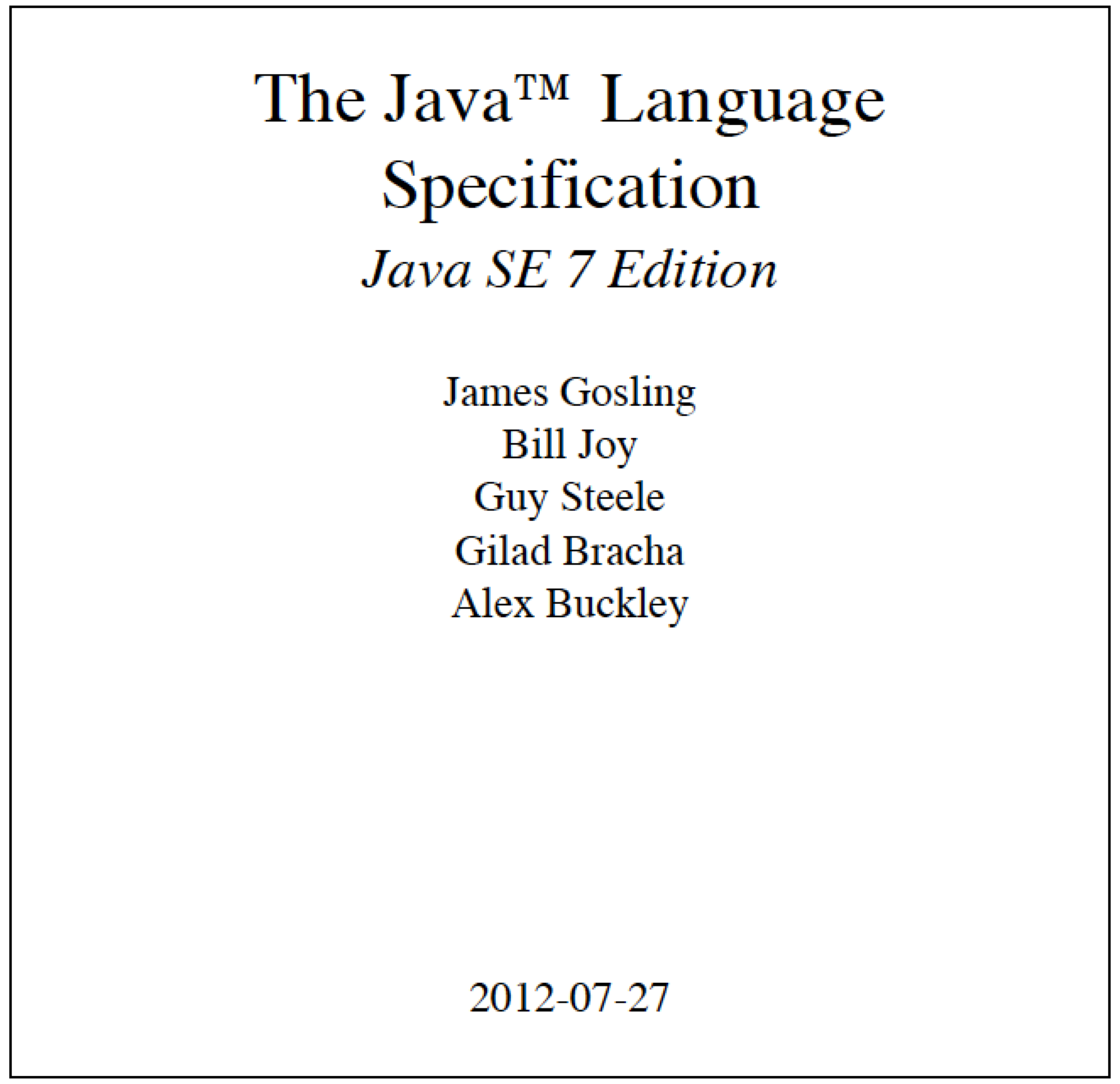
Furthermore, although there are a large number of studies of the semantics of isolated programming language features or of minimal programming languages [1], [31],[32], there have not been many studies of the formal semantics of *actual* programming languages. In addition, the interplay of features which are very well understood in isolation, might introduce unexpected effects.

parser
type checker
code generator
interpreter

parser
error recovery
syntax highlighting
outline
code completion





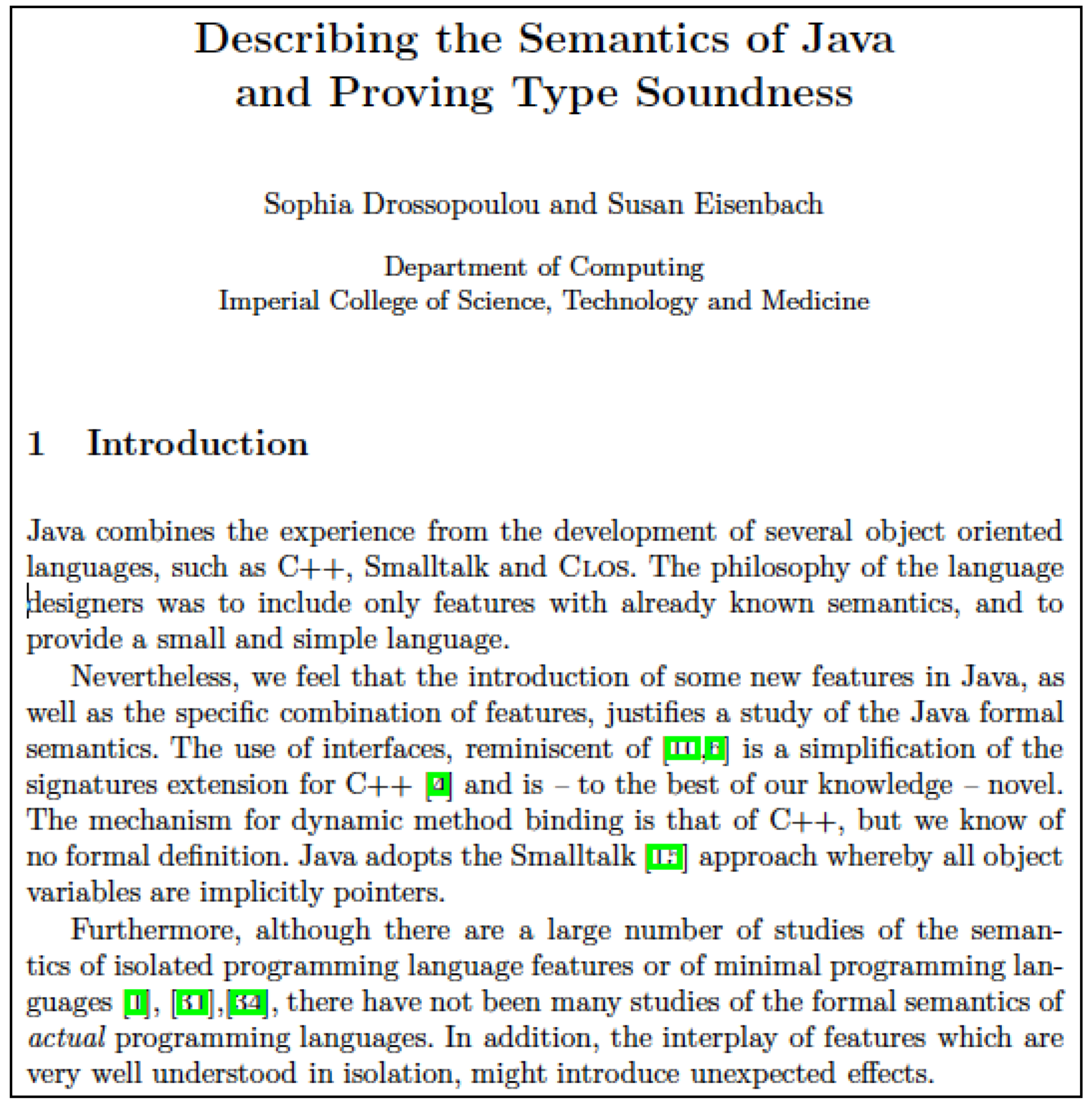


The Java™ Language Specification

Java SE 7 Edition

James Gosling
Bill Joy
Guy Steele
Gilad Bracha
Alex Buckley

2012-07-27



Describing the Semantics of Java and Proving Type Soundness

Sophia Drossopoulou and Susan Eisenbach

Department of Computing
Imperial College of Science, Technology and Medicine

1 Introduction

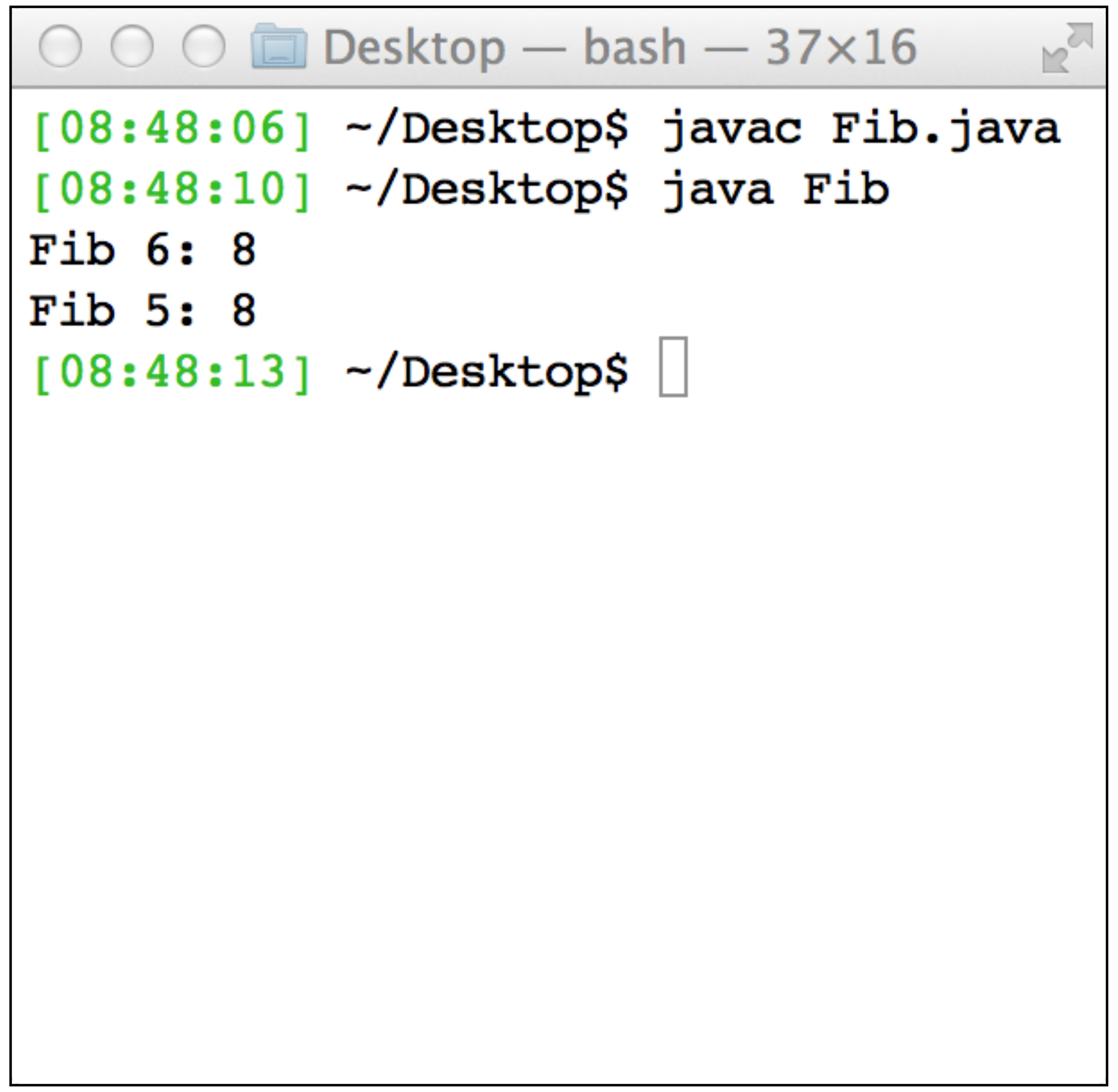
Java combines the experience from the development of several object oriented languages, such as C++, Smalltalk and CLOS. The philosophy of the language designers was to include only features with already known semantics, and to provide a small and simple language.

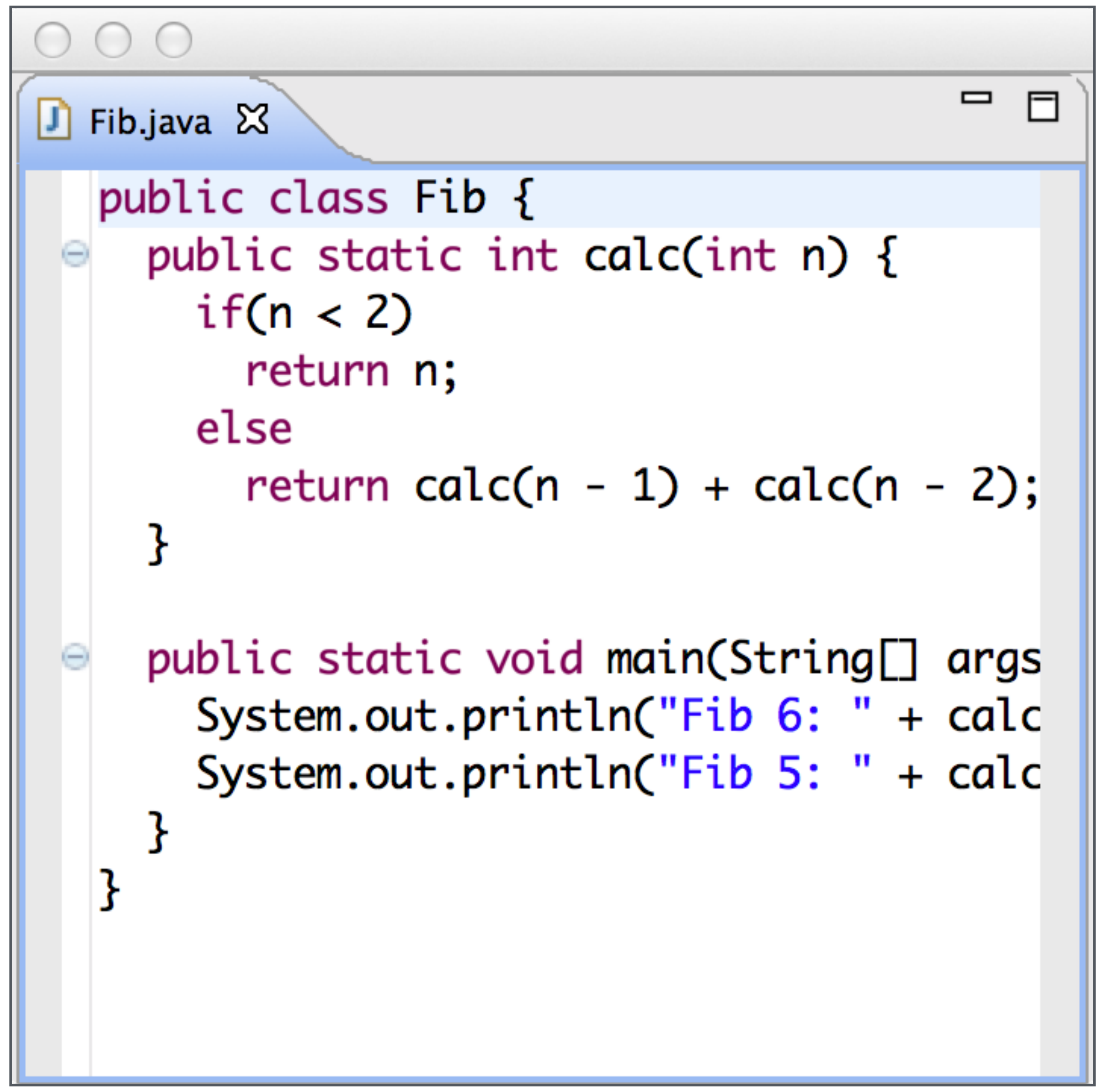
Nevertheless, we feel that the introduction of some new features in Java, as well as the specific combination of features, justifies a study of the Java formal semantics. The use of interfaces, reminiscent of [11,6] is a simplification of the signatures extension for C++ [4] and is – to the best of our knowledge – novel. The mechanism for dynamic method binding is that of C++, but we know of no formal definition. Java adopts the Smalltalk [15] approach whereby all object variables are implicitly pointers.

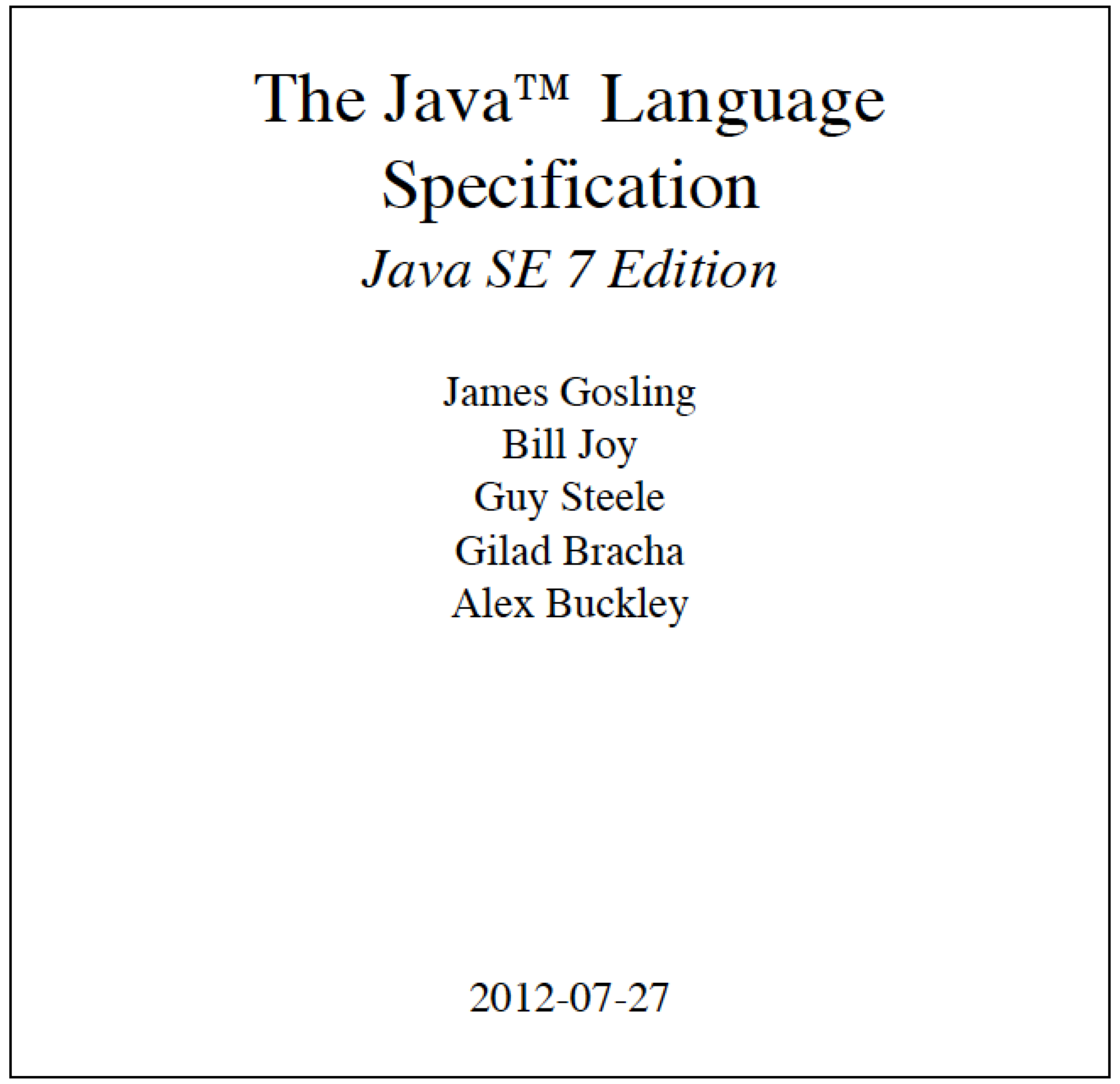
Furthermore, although there are a large number of studies of the semantics of isolated programming language features or of minimal programming languages [1], [31],[34], there have not been many studies of the formal semantics of *actual* programming languages. In addition, the interplay of features which are very well understood in isolation, might introduce unexpected effects.

parser
type checker
code generator
interpreter

parser
error recovery
syntax highlighting
outline
code completion
navigation





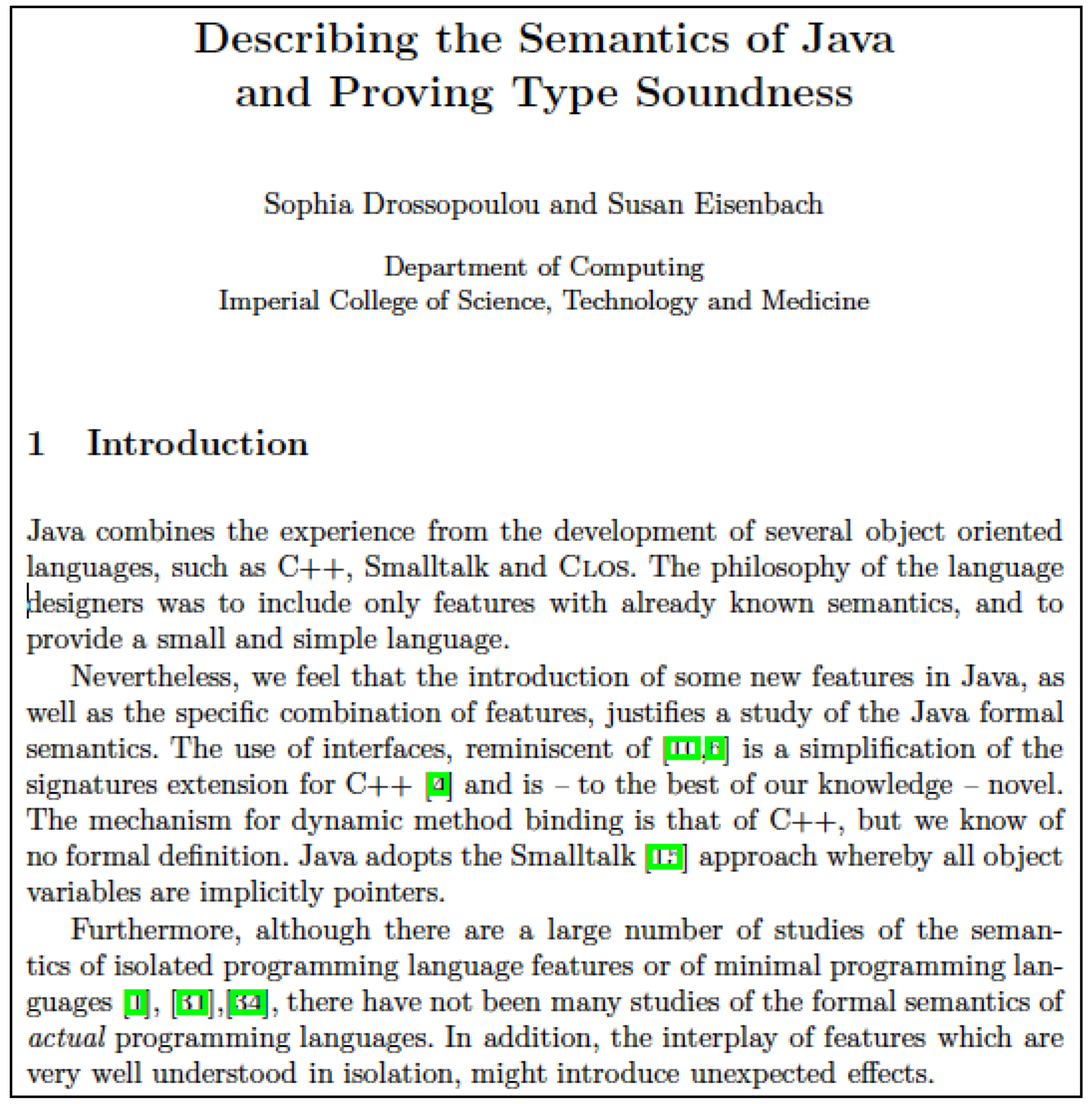


The Java™ Language Specification

Java SE 7 Edition

James Gosling
Bill Joy
Guy Steele
Gilad Bracha
Alex Buckley

2012-07-27



Describing the Semantics of Java and Proving Type Soundness

Sophia Drossopoulou and Susan Eisenbach

Department of Computing
Imperial College of Science, Technology and Medicine

1 Introduction

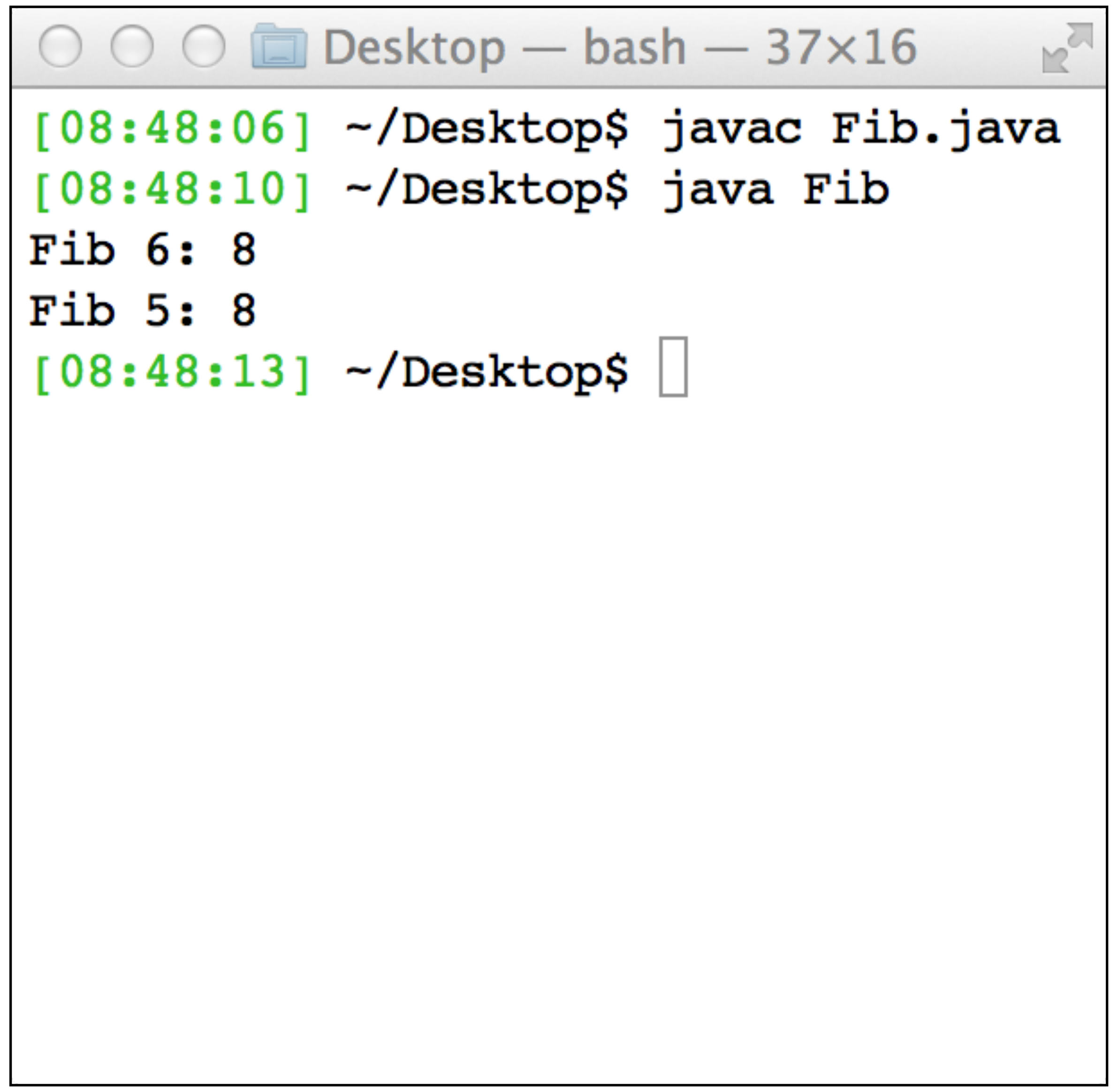
Java combines the experience from the development of several object oriented languages, such as C++, Smalltalk and CLOS. The philosophy of the language designers was to include only features with already known semantics, and to provide a small and simple language.

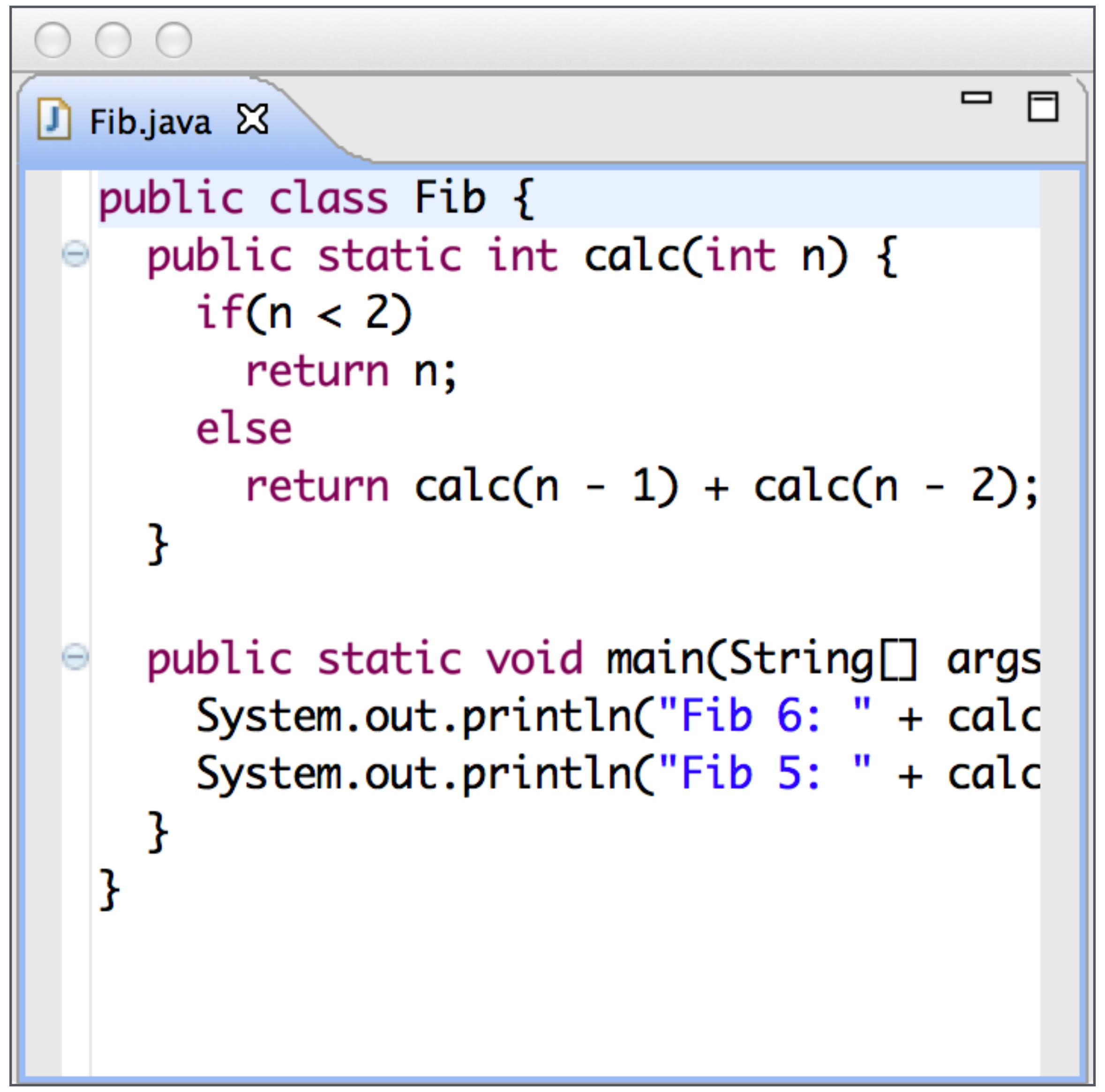
Nevertheless, we feel that the introduction of some new features in Java, as well as the specific combination of features, justifies a study of the Java formal semantics. The use of interfaces, reminiscent of [11,6] is a simplification of the signatures extension for C++ [4] and is – to the best of our knowledge – novel. The mechanism for dynamic method binding is that of C++, but we know of no formal definition. Java adopts the Smalltalk [15] approach whereby all object variables are implicitly pointers.

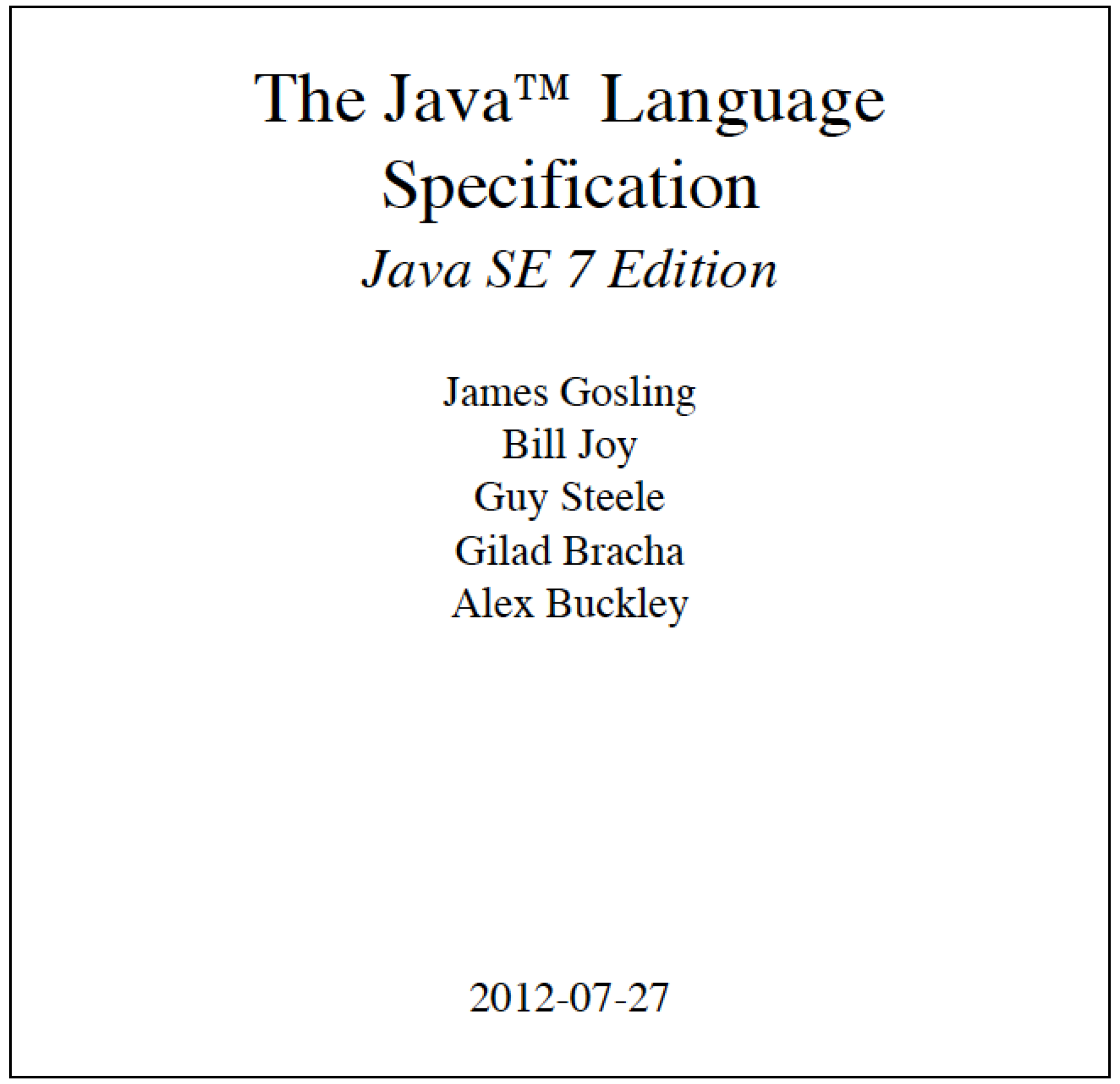
Furthermore, although there are a large number of studies of the semantics of isolated programming language features or of minimal programming languages [1], [31],[34], there have not been many studies of the formal semantics of *actual* programming languages. In addition, the interplay of features which are very well understood in isolation, might introduce unexpected effects.

parser
type checker
code generator
interpreter

parser
error recovery
syntax highlighting
outline
code completion
navigation
type checker





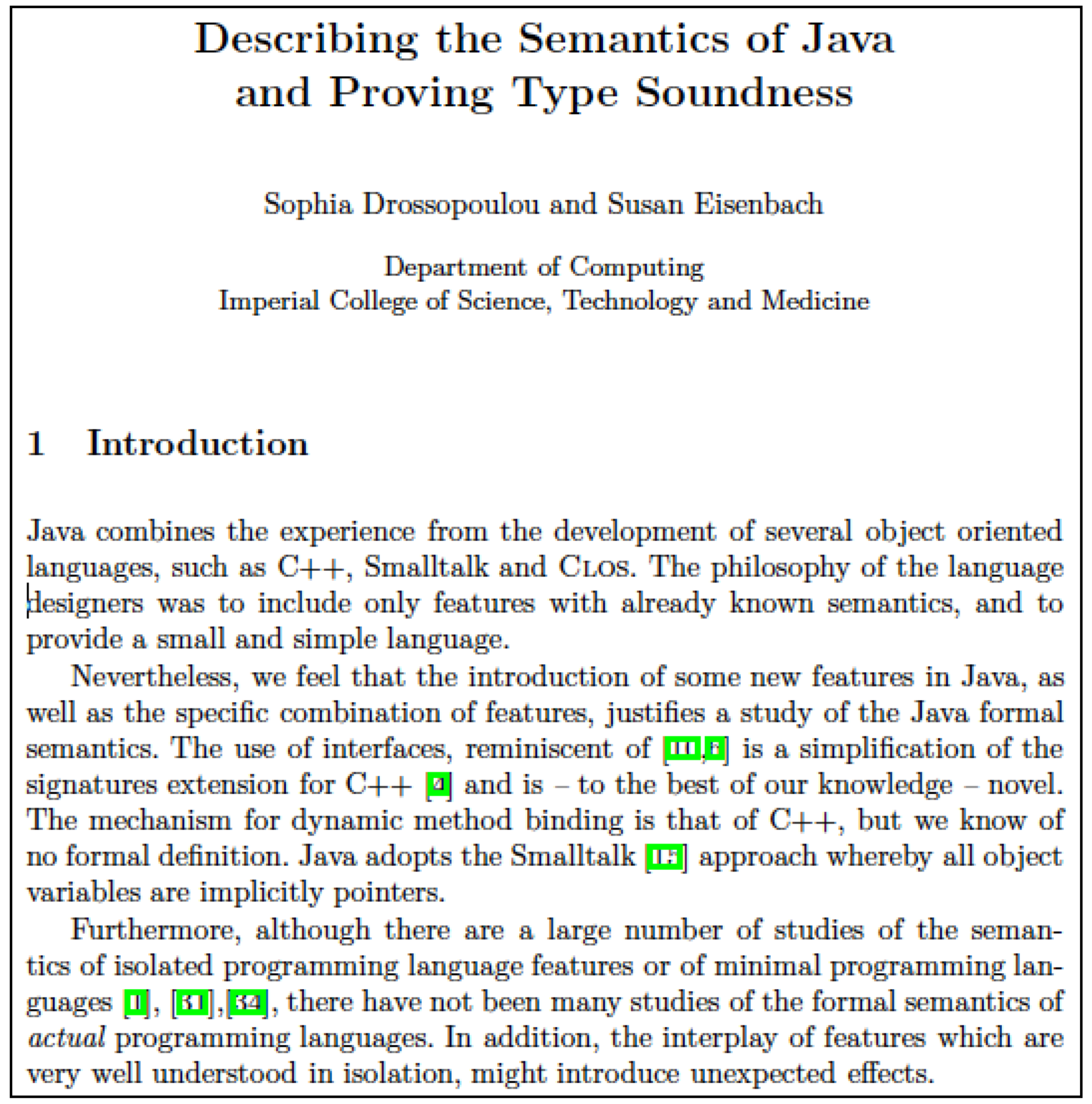


The Java™ Language Specification

Java SE 7 Edition

James Gosling
Bill Joy
Guy Steele
Gilad Bracha
Alex Buckley

2012-07-27



Describing the Semantics of Java and Proving Type Soundness

Sophia Drossopoulou and Susan Eisenbach

Department of Computing
Imperial College of Science, Technology and Medicine

1 Introduction

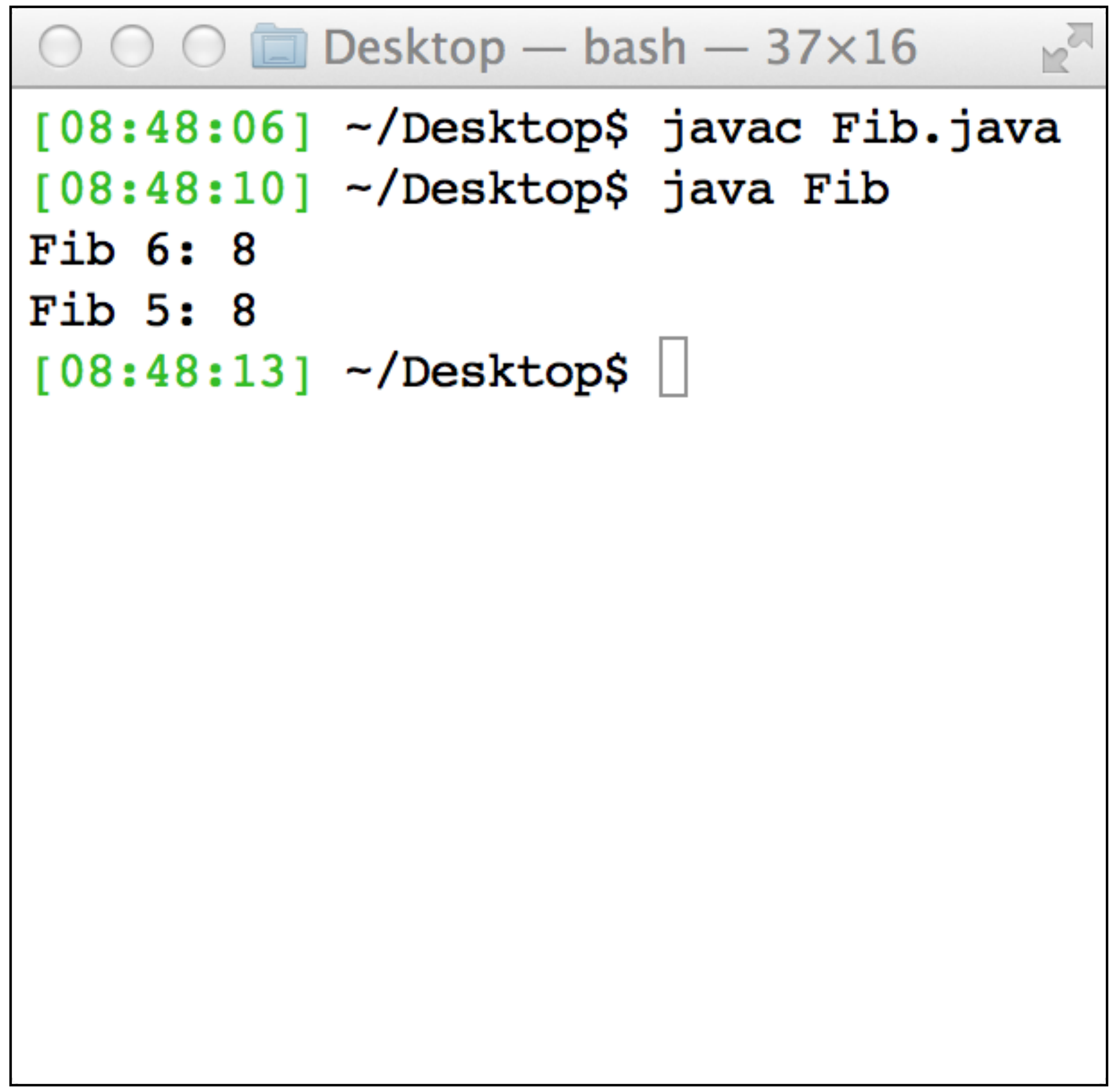
Java combines the experience from the development of several object oriented languages, such as C++, Smalltalk and CLOS. The philosophy of the language designers was to include only features with already known semantics, and to provide a small and simple language.

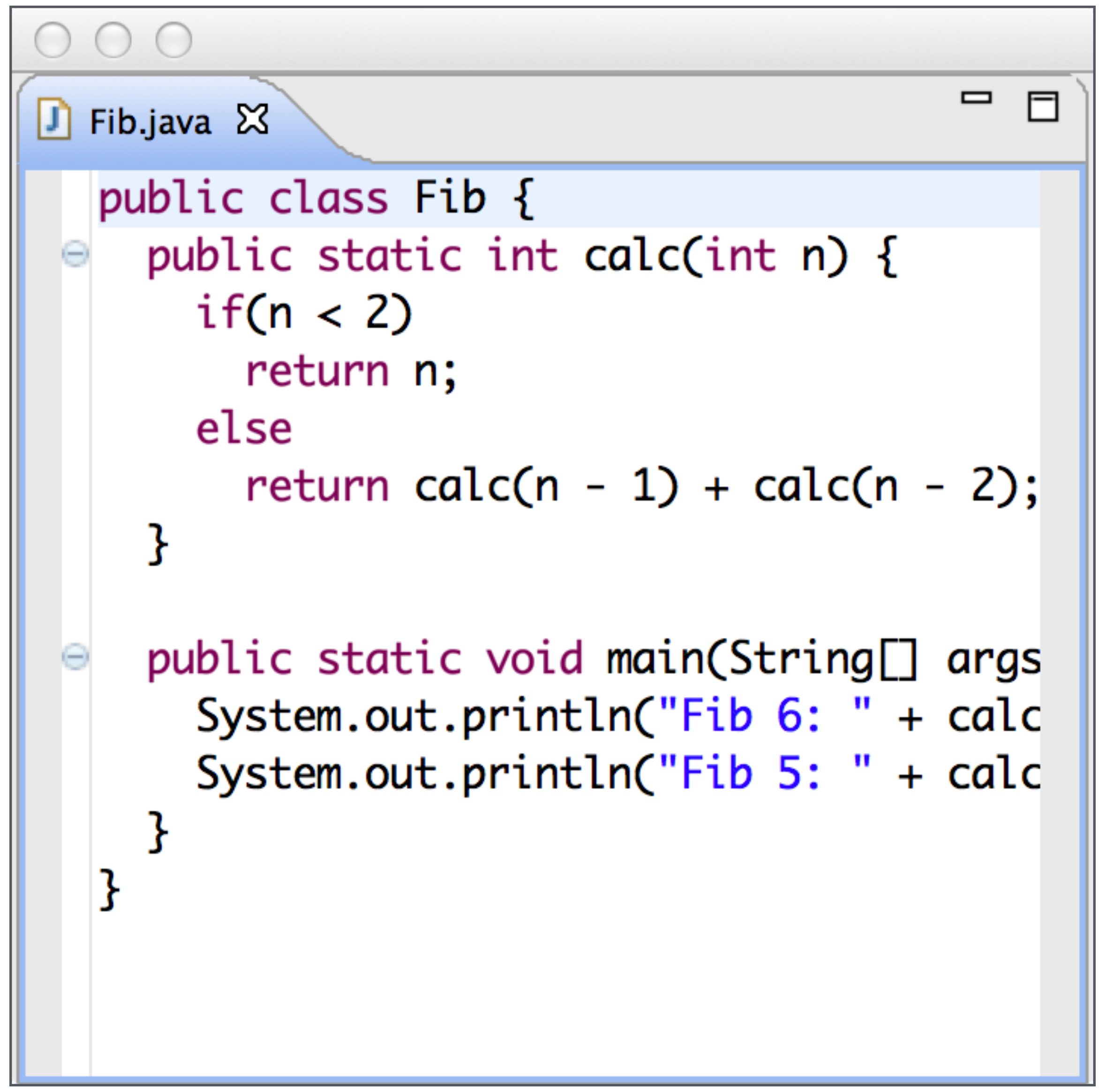
Nevertheless, we feel that the introduction of some new features in Java, as well as the specific combination of features, justifies a study of the Java formal semantics. The use of interfaces, reminiscent of [11,6] is a simplification of the signatures extension for C++ [4] and is – to the best of our knowledge – novel. The mechanism for dynamic method binding is that of C++, but we know of no formal definition. Java adopts the Smalltalk [15] approach whereby all object variables are implicitly pointers.

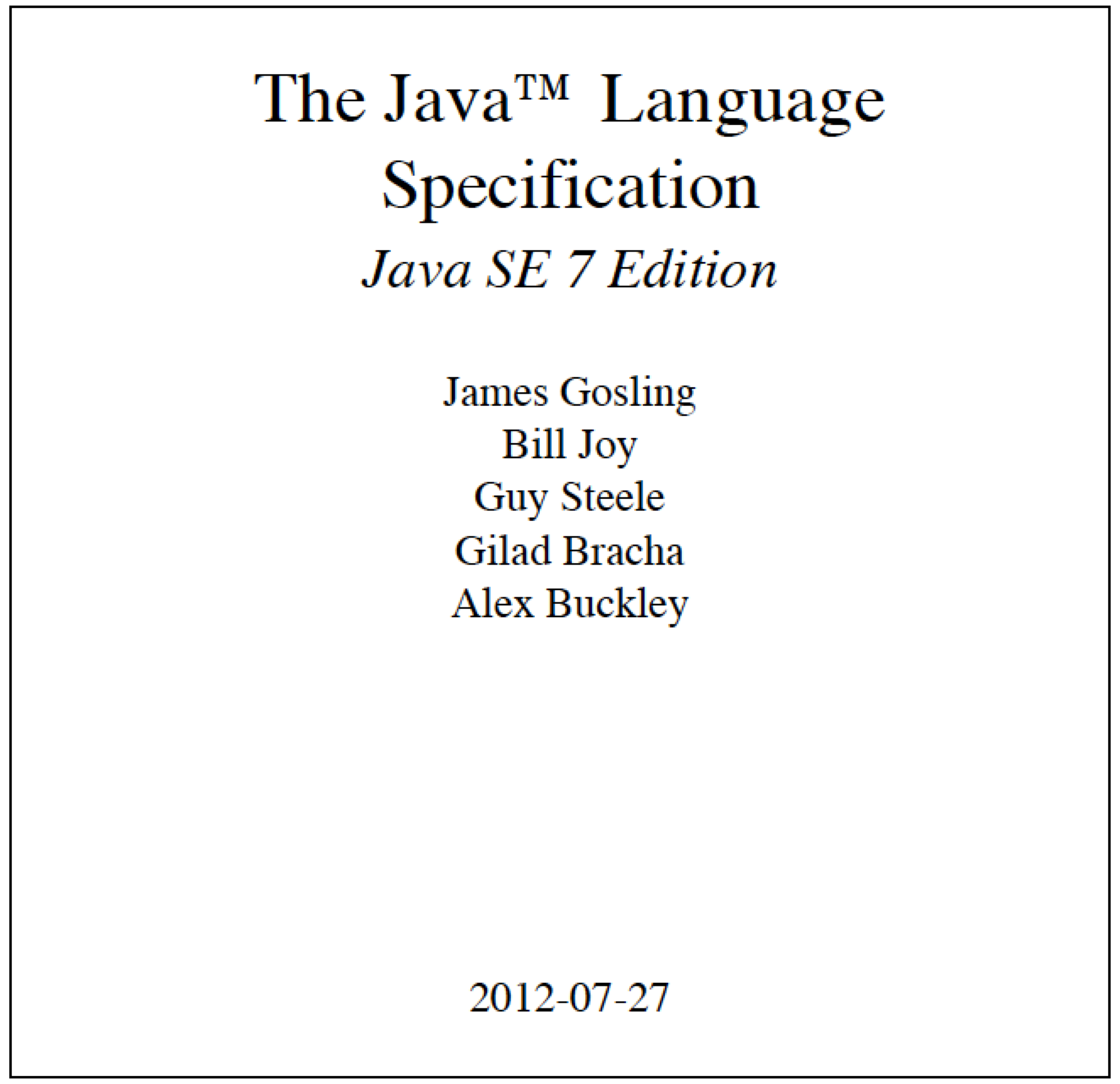
Furthermore, although there are a large number of studies of the semantics of isolated programming language features or of minimal programming languages [1], [31],[34], there have not been many studies of the formal semantics of *actual* programming languages. In addition, the interplay of features which are very well understood in isolation, might introduce unexpected effects.

parser
type checker
code generator
interpreter

parser
error recovery
syntax highlighting
outline
code completion
navigation
type checker
debugger



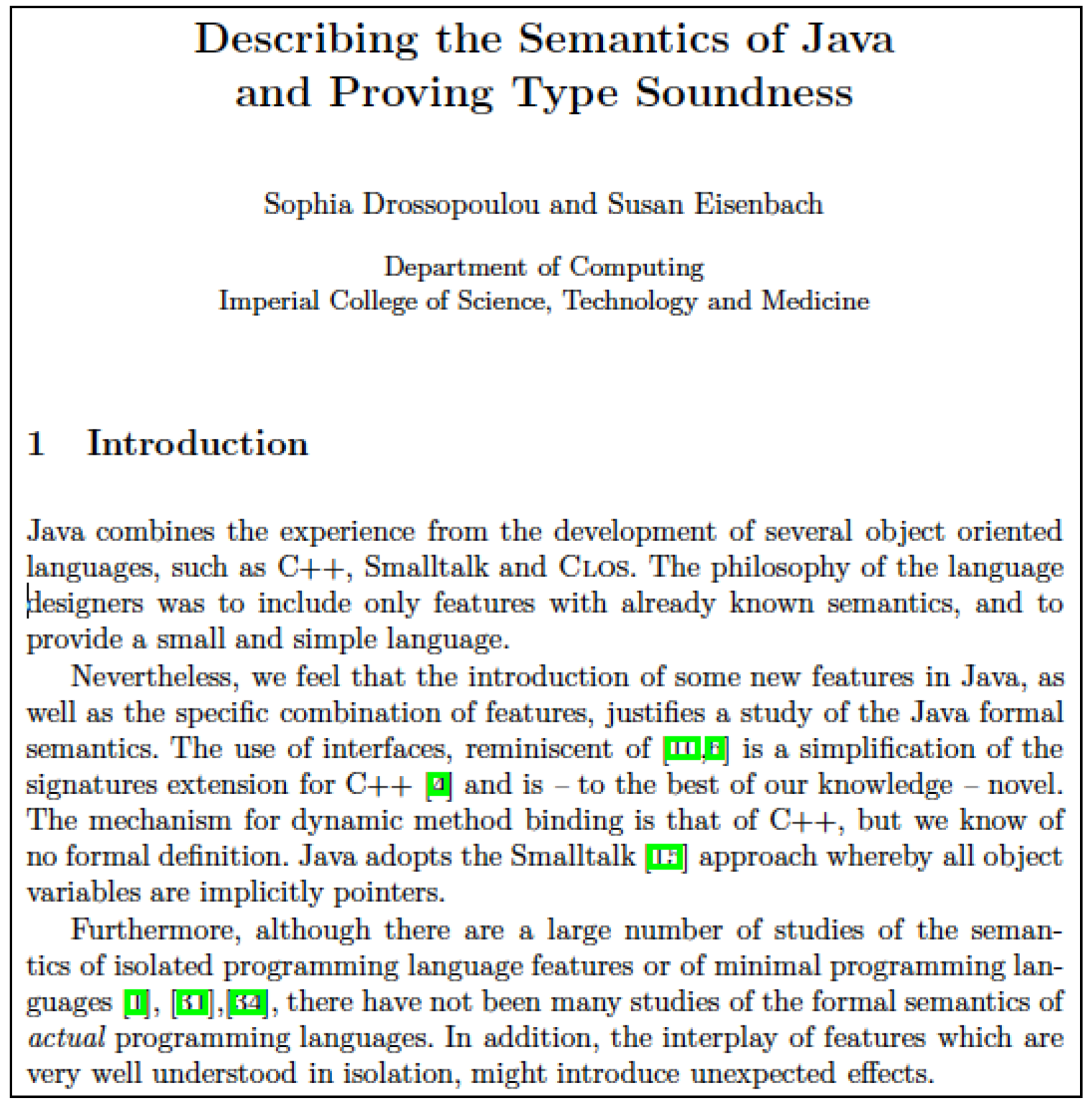



The Java™ Language Specification

Java SE 7 Edition

James Gosling
Bill Joy
Guy Steele
Gilad Bracha
Alex Buckley

2012-07-27


Describing the Semantics of Java and Proving Type Soundness

Sophia Drossopoulou and Susan Eisenbach

Department of Computing
Imperial College of Science, Technology and Medicine

1 Introduction

Java combines the experience from the development of several object oriented languages, such as C++, Smalltalk and Clos. The philosophy of the language designers was to include only features with already known semantics, and to provide a small and simple language.

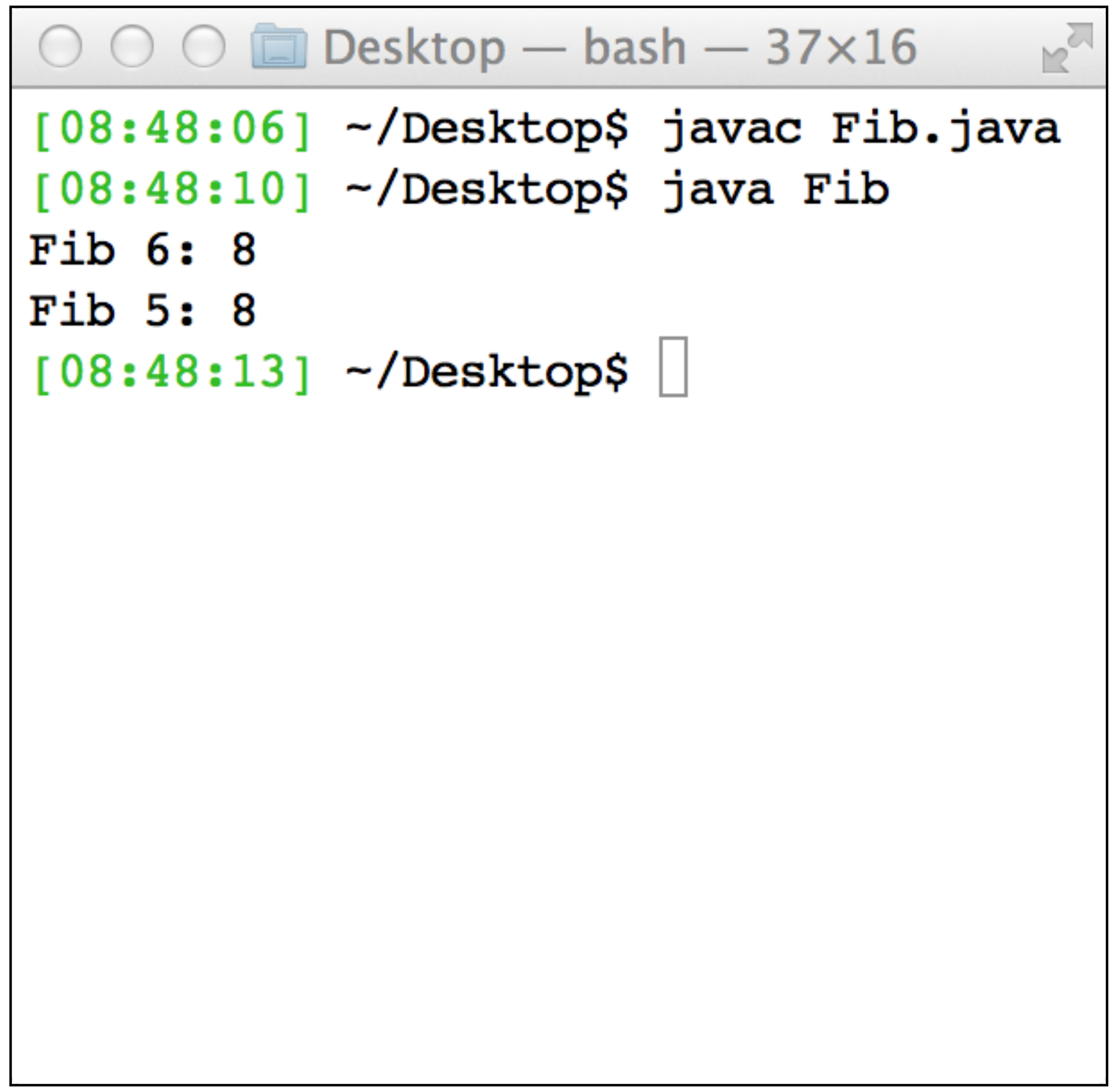
Nevertheless, we feel that the introduction of some new features in Java, as well as the specific combination of features, justifies a study of the Java formal semantics. The use of interfaces, reminiscent of [11,5] is a simplification of the signatures extension for C++ [4] and is – to the best of our knowledge – novel. The mechanism for dynamic method binding is that of C++, but we know of no formal definition. Java adopts the Smalltalk [13] approach whereby all object variables are implicitly pointers.

Furthermore, although there are a large number of studies of the semantics of isolated programming language features or of minimal programming languages [1], [31],[32], there have not been many studies of the formal semantics of *actual* programming languages. In addition, the interplay of features which are very well understood in isolation, might introduce unexpected effects.

parser
type checker
code generator
interpreter

parser
error recovery
syntax highlighting
outline
code completion
navigation
type checker
debugger

syntax definition

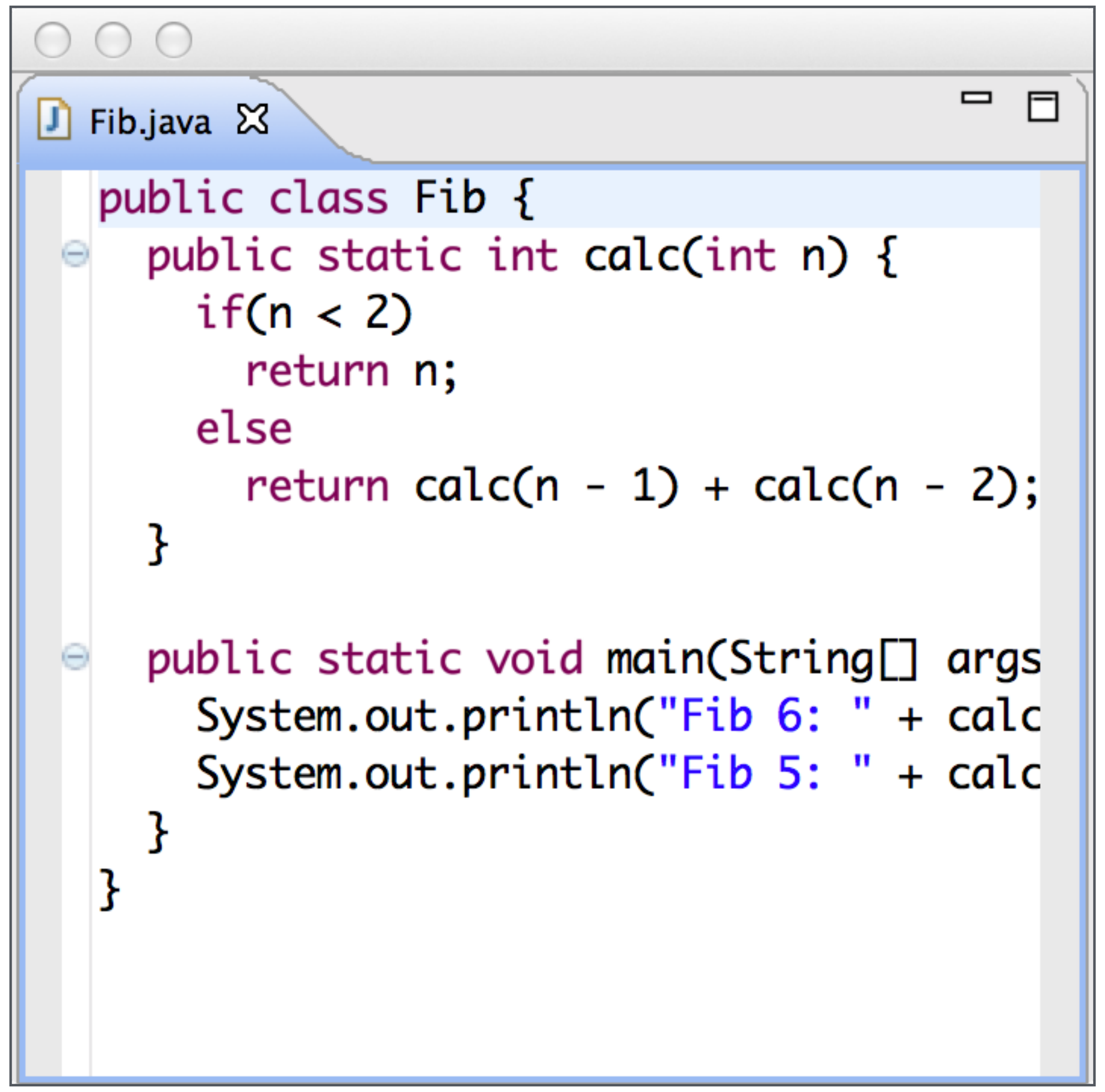


parser

type checker

code generator

interpreter



parser

error recovery

syntax highlighting

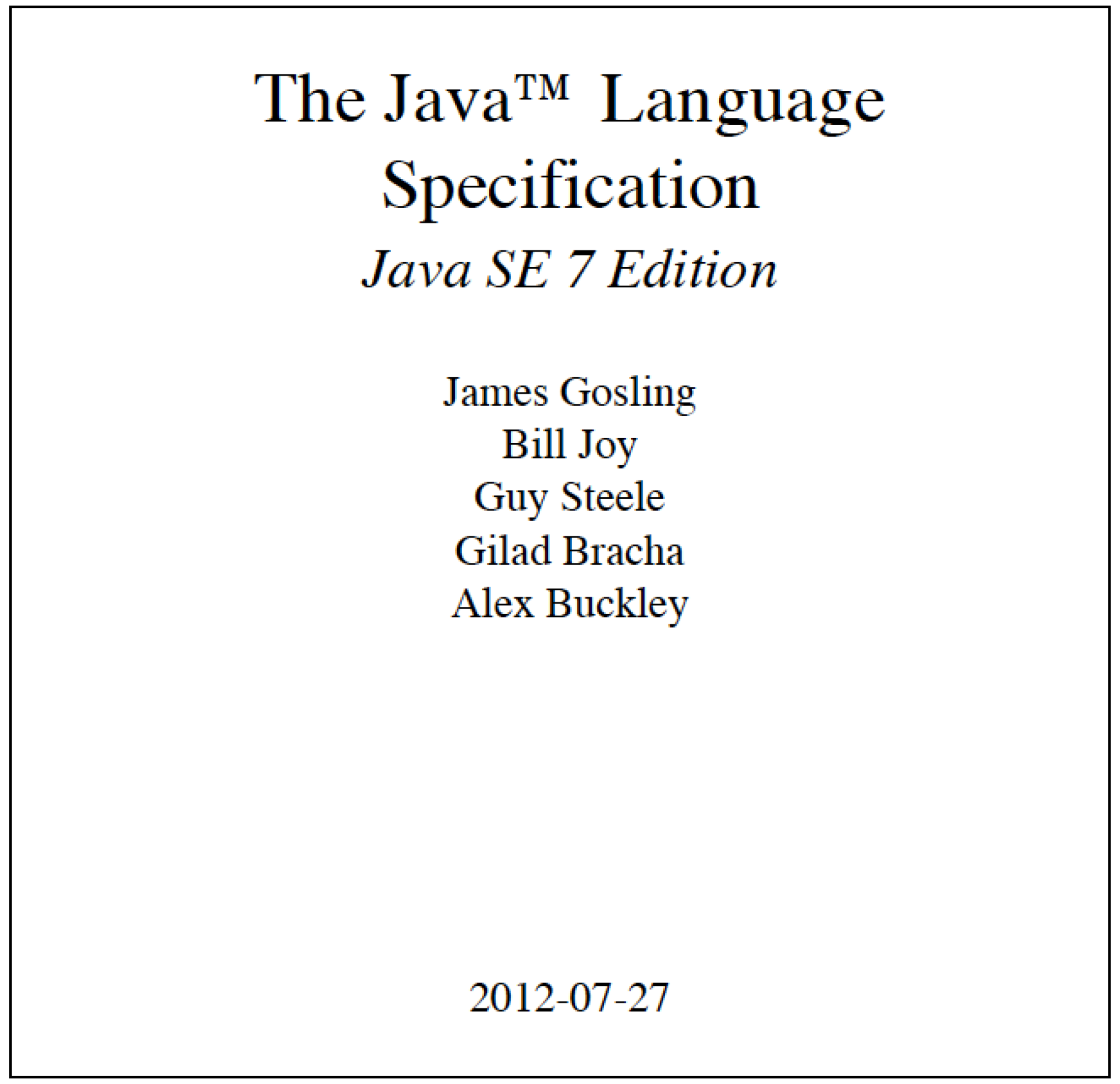
outline

code completion

navigation

type checker

debugger



The Java™ Language Specification

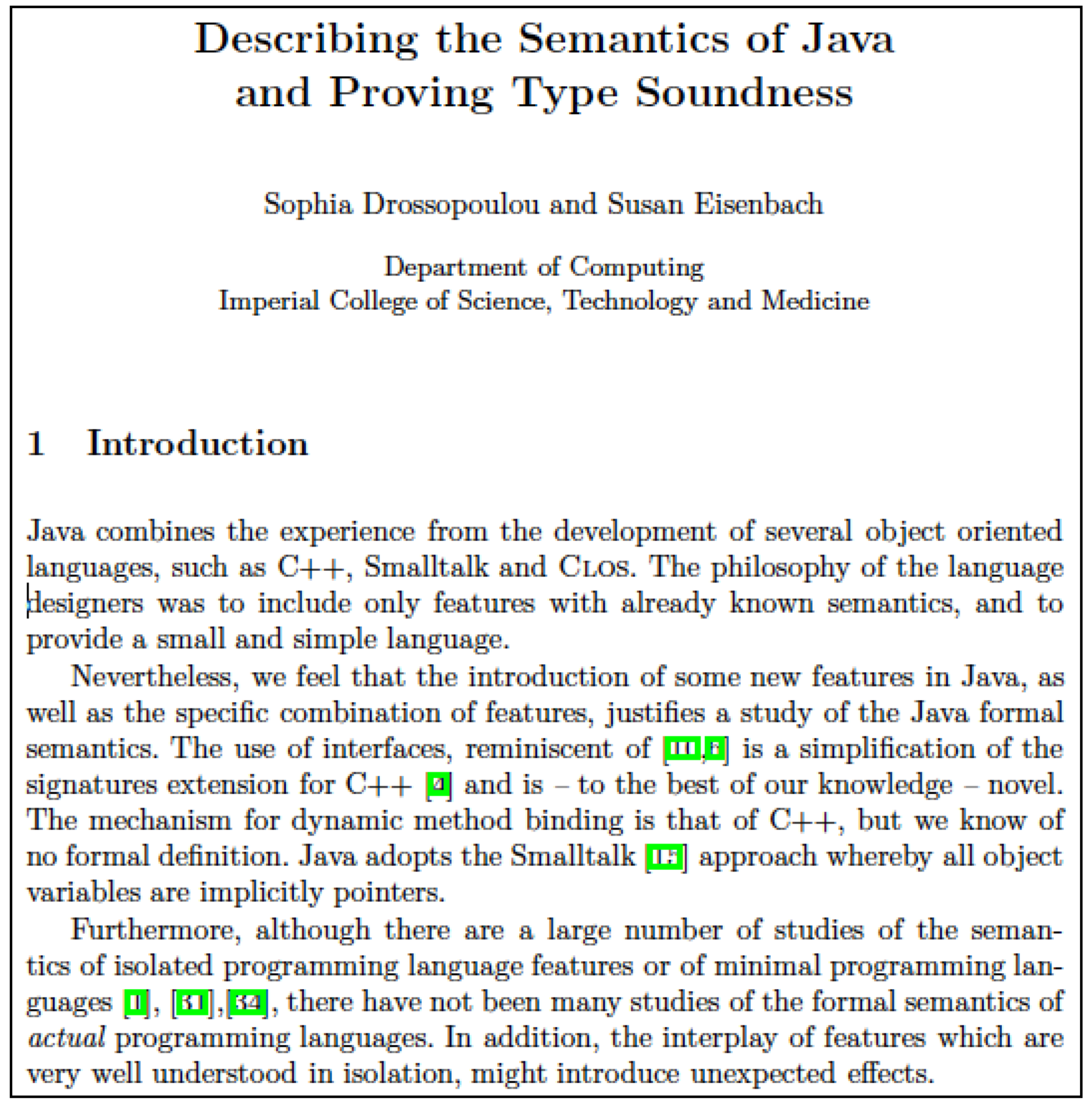
Java SE 7 Edition

James Gosling
Bill Joy
Guy Steele
Gilad Bracha
Alex Buckley

2012-07-27

syntax definition

static semantics



Describing the Semantics of Java and Proving Type Soundness

Sophia Drossopoulou and Susan Eisenbach

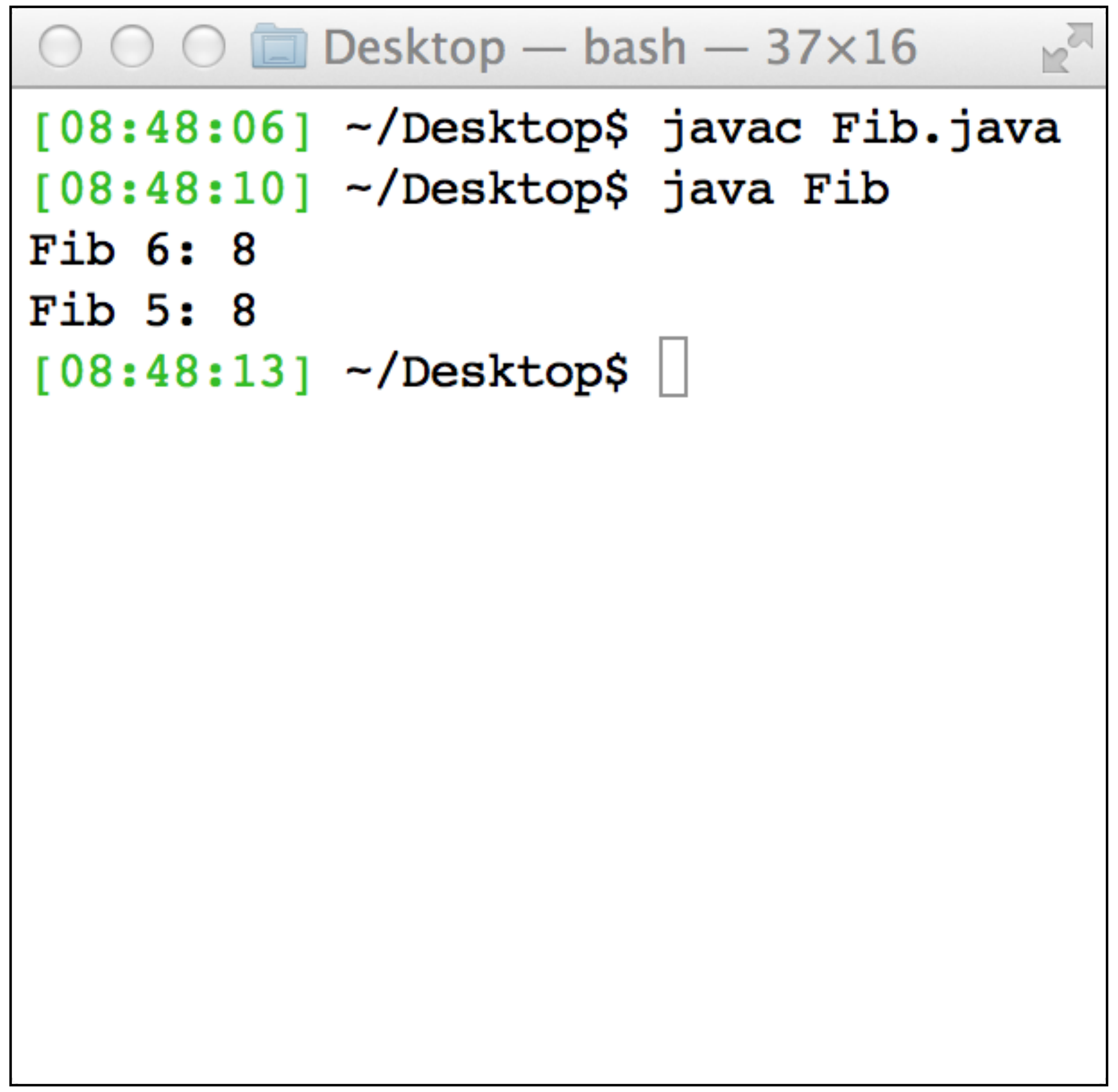
Department of Computing
Imperial College of Science, Technology and Medicine

1 Introduction

Java combines the experience from the development of several object oriented languages, such as C++, Smalltalk and CLOS. The philosophy of the language designers was to include only features with already known semantics, and to provide a small and simple language.

Nevertheless, we feel that the introduction of some new features in Java, as well as the specific combination of features, justifies a study of the Java formal semantics. The use of interfaces, reminiscent of [11,6] is a simplification of the signatures extension for C++ [4] and is – to the best of our knowledge – novel. The mechanism for dynamic method binding is that of C++, but we know of no formal definition. Java adopts the Smalltalk [15] approach whereby all object variables are implicitly pointers.

Furthermore, although there are a large number of studies of the semantics of isolated programming language features or of minimal programming languages [1], [31],[34], there have not been many studies of the formal semantics of *actual* programming languages. In addition, the interplay of features which are very well understood in isolation, might introduce unexpected effects.

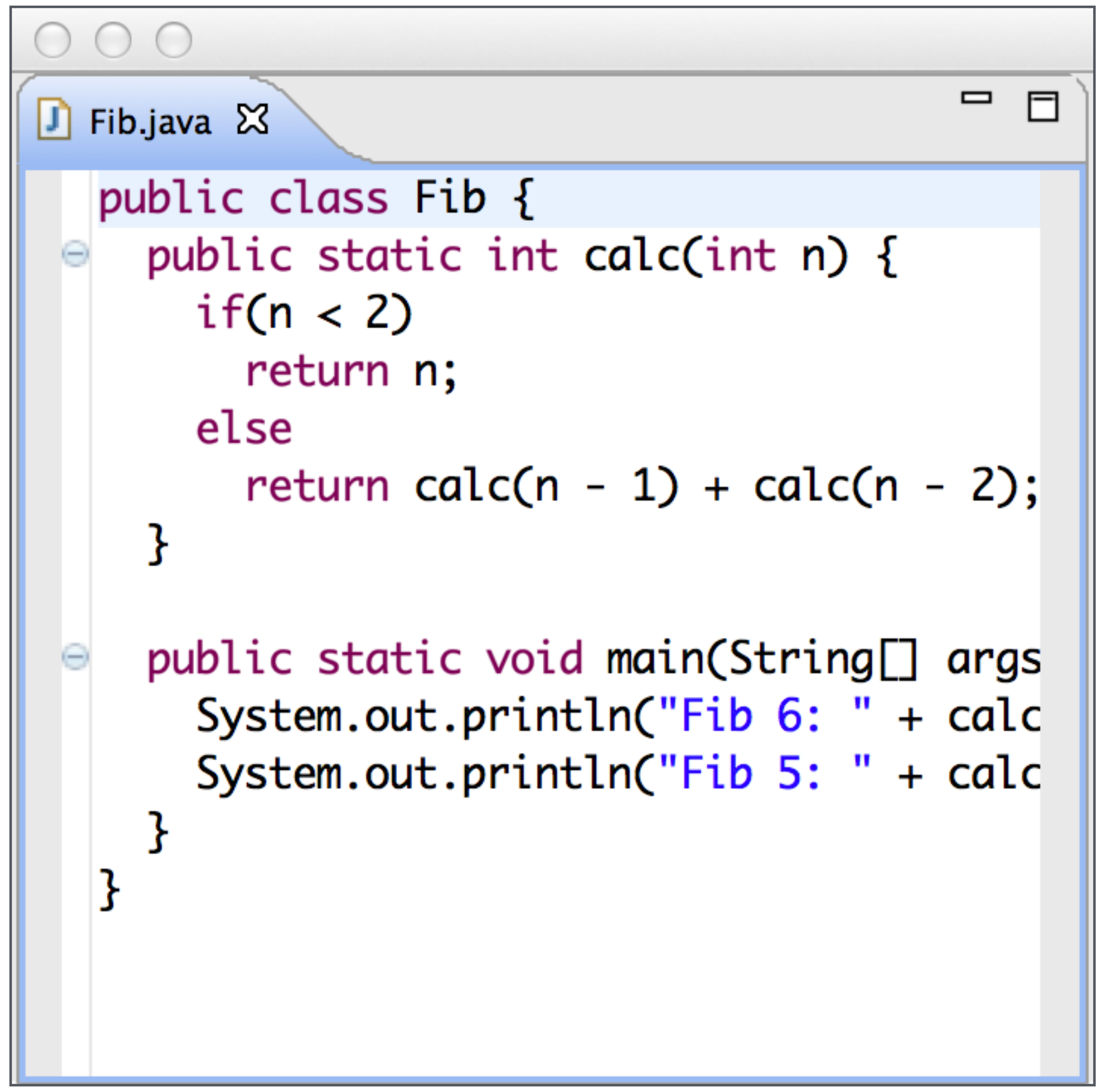


parser

type checker

code generator

interpreter



parser

error recovery

syntax highlighting

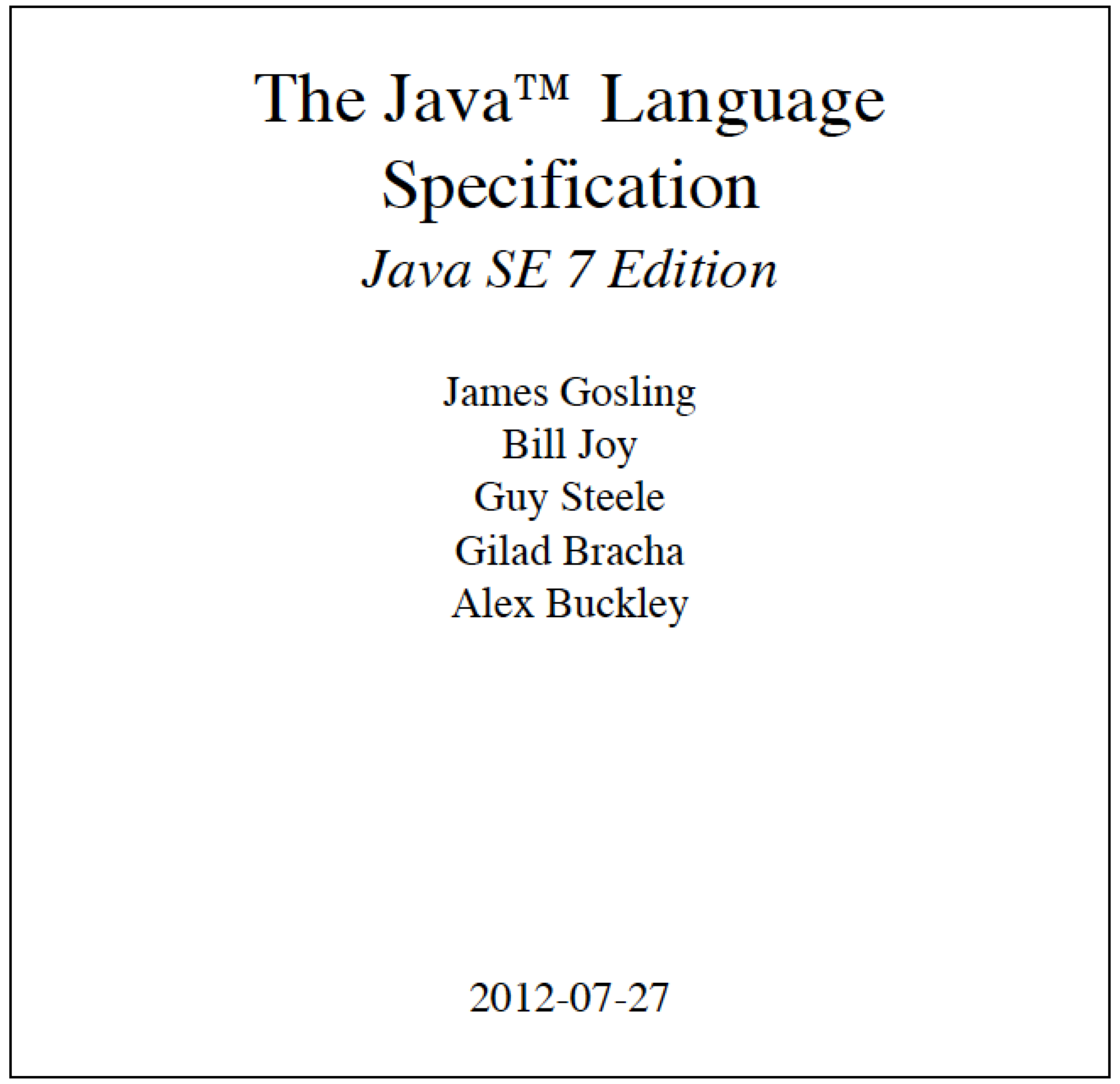
outline

code completion

navigation

type checker

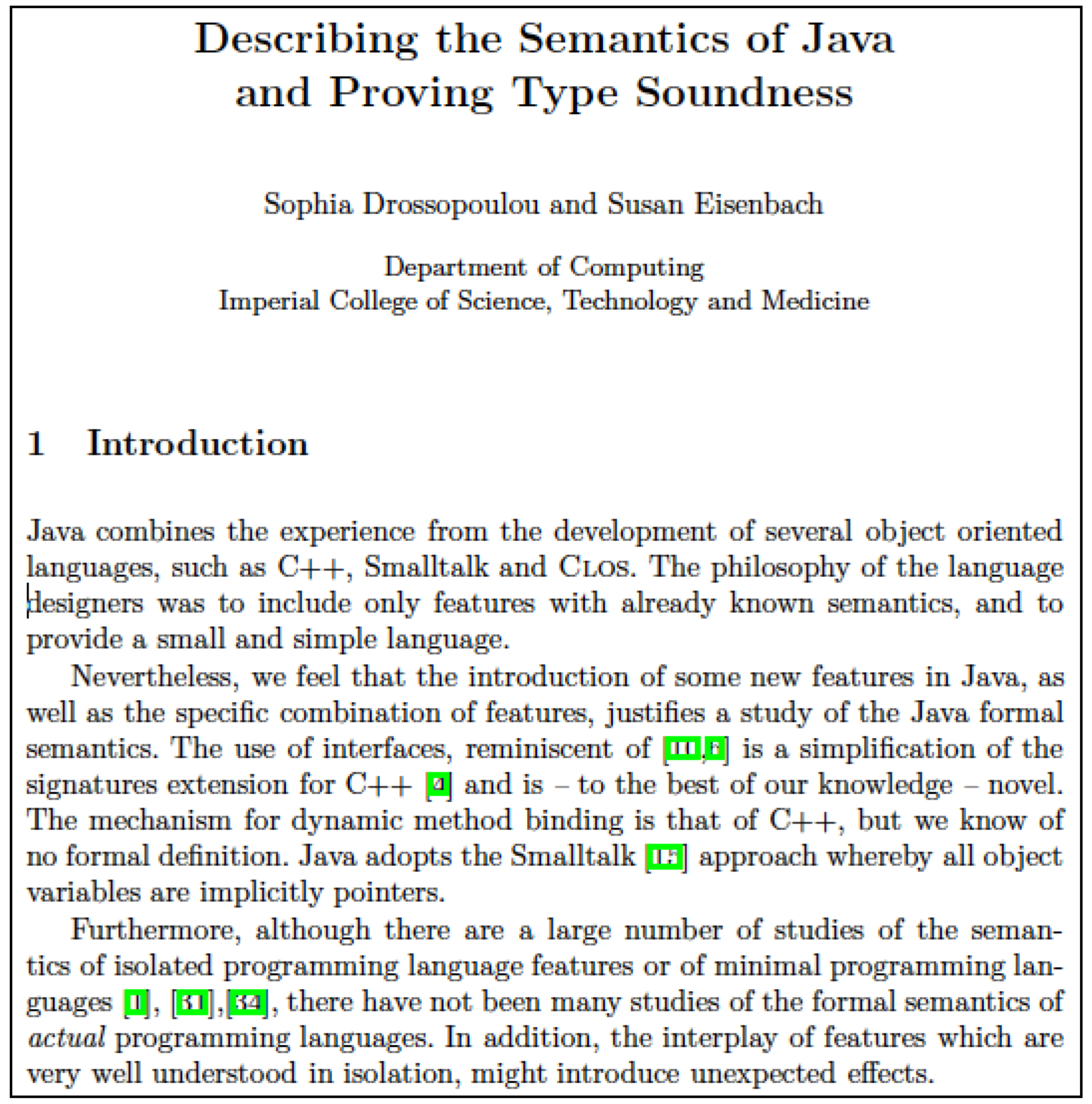
debugger



syntax definition

static semantics

dynamic semantics


Describing the Semantics of Java and Proving Type Soundness

Sophia Drossopoulou and Susan Eisenbach

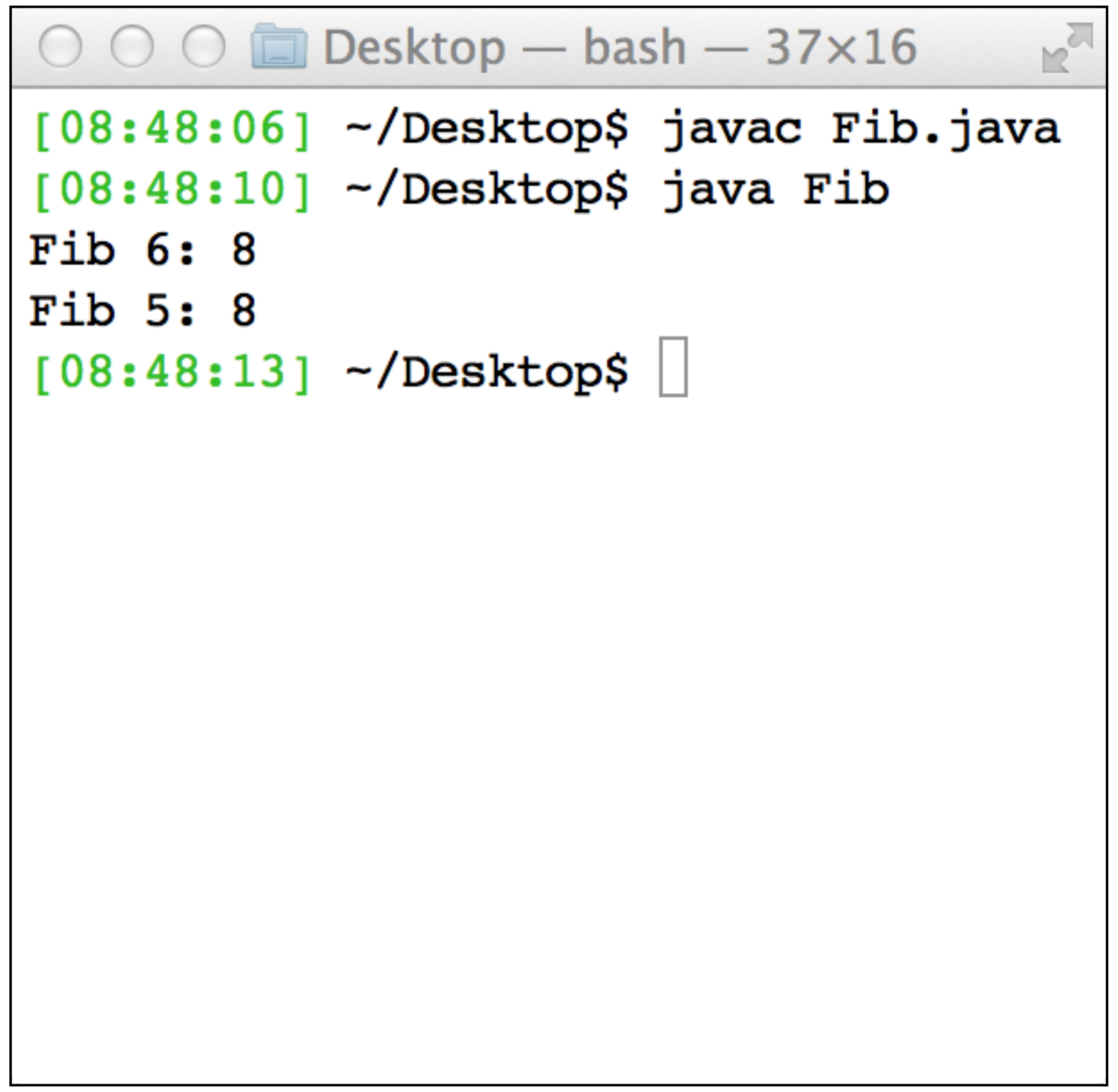
Department of Computing
Imperial College of Science, Technology and Medicine

1 Introduction

Java combines the experience from the development of several object oriented languages, such as C++, Smalltalk and Clos. The philosophy of the language designers was to include only features with already known semantics, and to provide a small and simple language.

Nevertheless, we feel that the introduction of some new features in Java, as well as the specific combination of features, justifies a study of the Java formal semantics. The use of interfaces, reminiscent of [11,6] is a simplification of the signatures extension for C++ [4] and is – to the best of our knowledge – novel. The mechanism for dynamic method binding is that of C++, but we know of no formal definition. Java adopts the Smalltalk [15] approach whereby all object variables are implicitly pointers.

Furthermore, although there are a large number of studies of the semantics of isolated programming language features or of minimal programming languages [1], [31],[34], there have not been many studies of the formal semantics of *actual* programming languages. In addition, the interplay of features which are very well understood in isolation, might introduce unexpected effects.

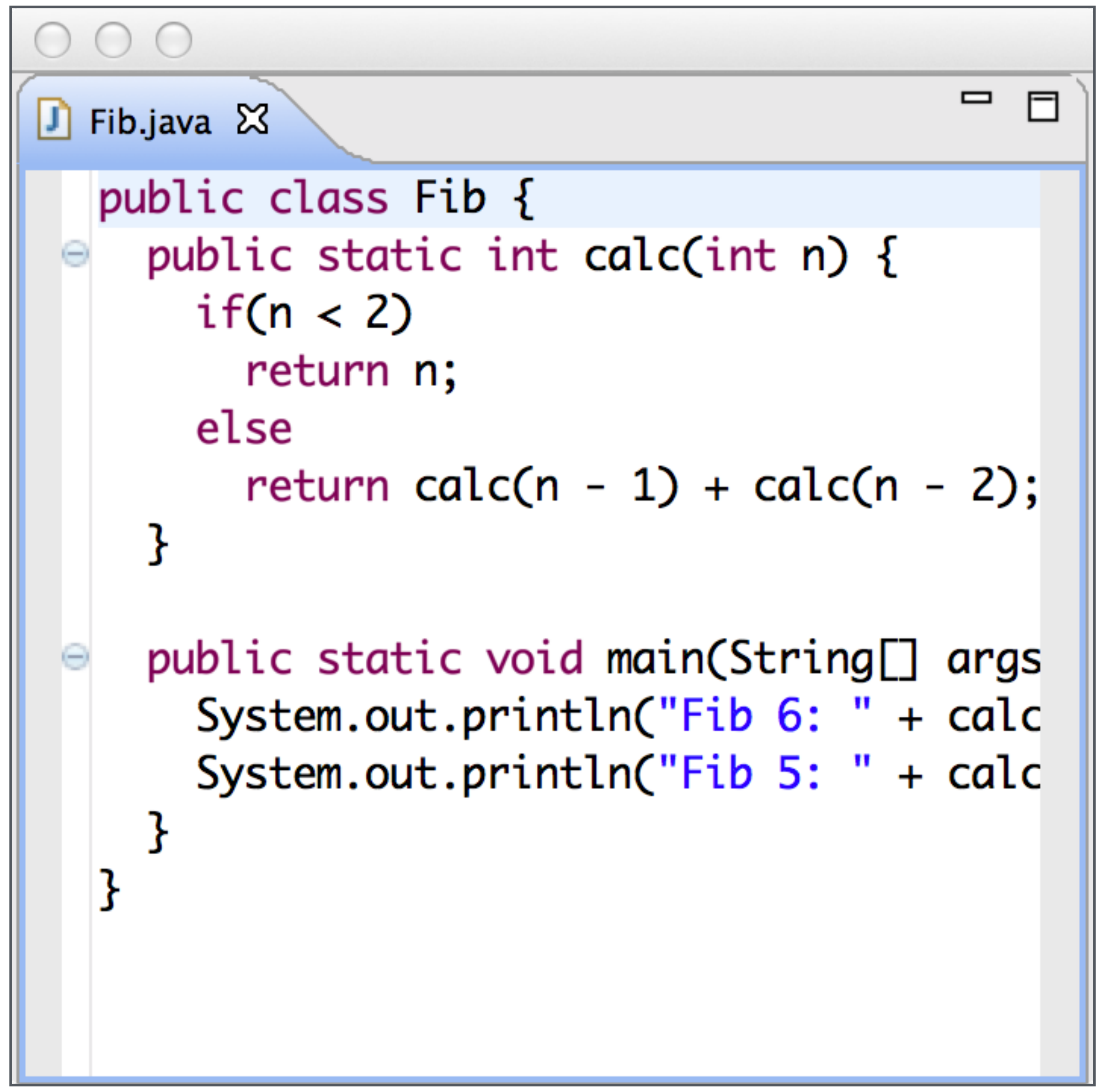


parser

type checker

code generator

interpreter



parser

error recovery

syntax highlighting

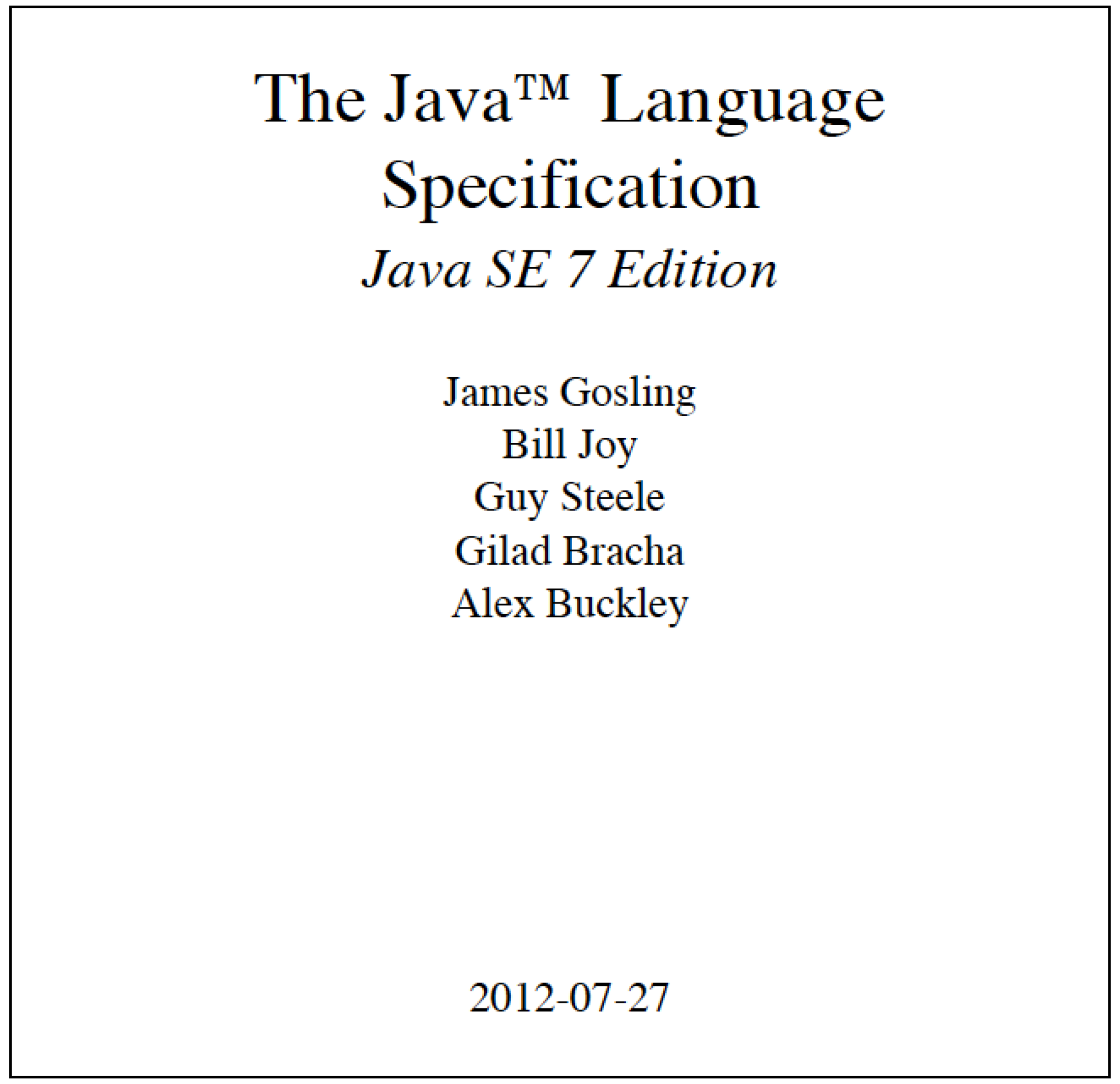
outline

code completion

navigation

type checker

debugger



The Java™ Language Specification

Java SE 7 Edition

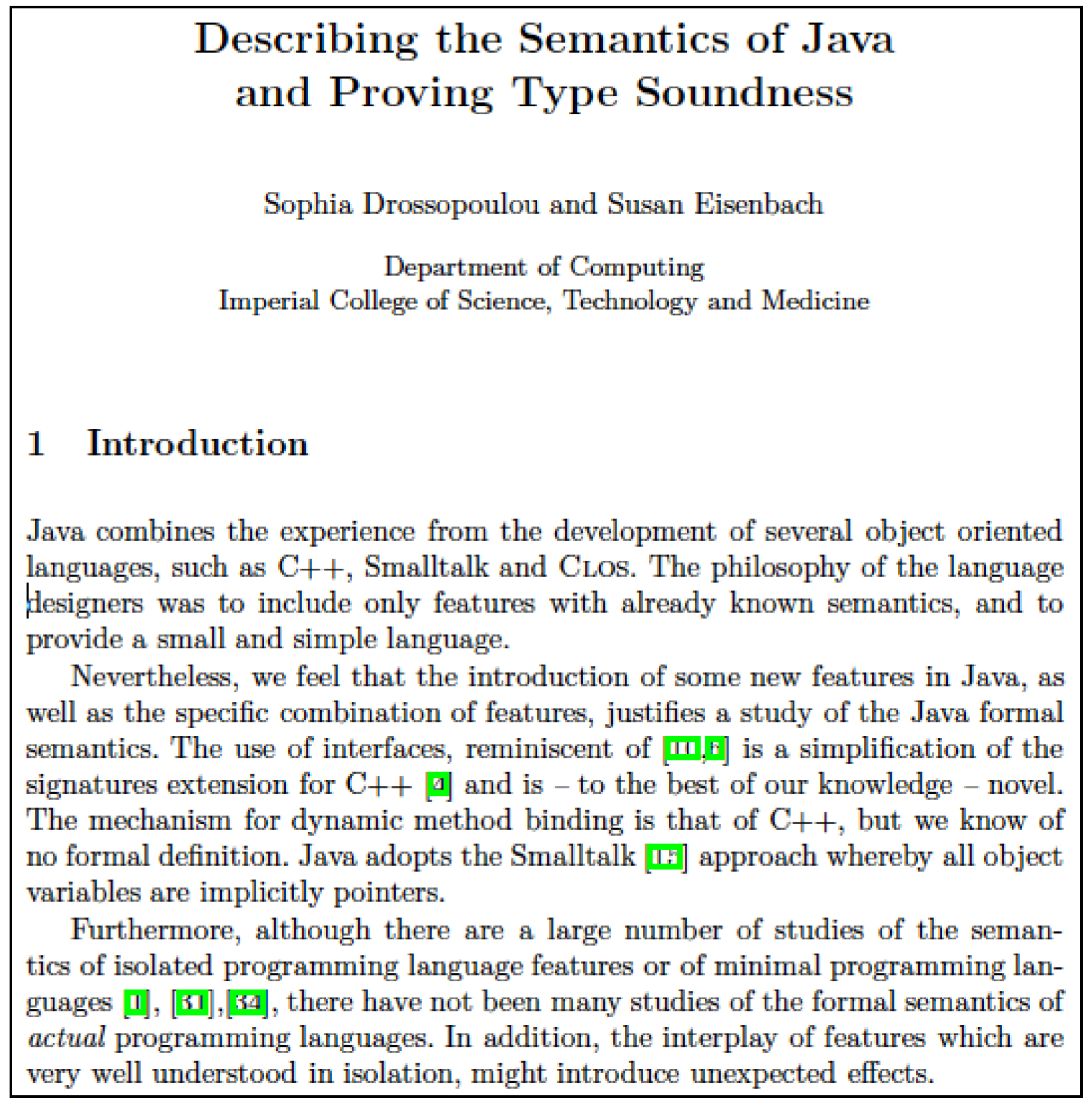
James Gosling
Bill Joy
Guy Steele
Gilad Bracha
Alex Buckley

2012-07-27

syntax definition

static semantics

dynamic semantics



Describing the Semantics of Java and Proving Type Soundness

Sophia Drossopoulou and Susan Eisenbach

Department of Computing
Imperial College of Science, Technology and Medicine

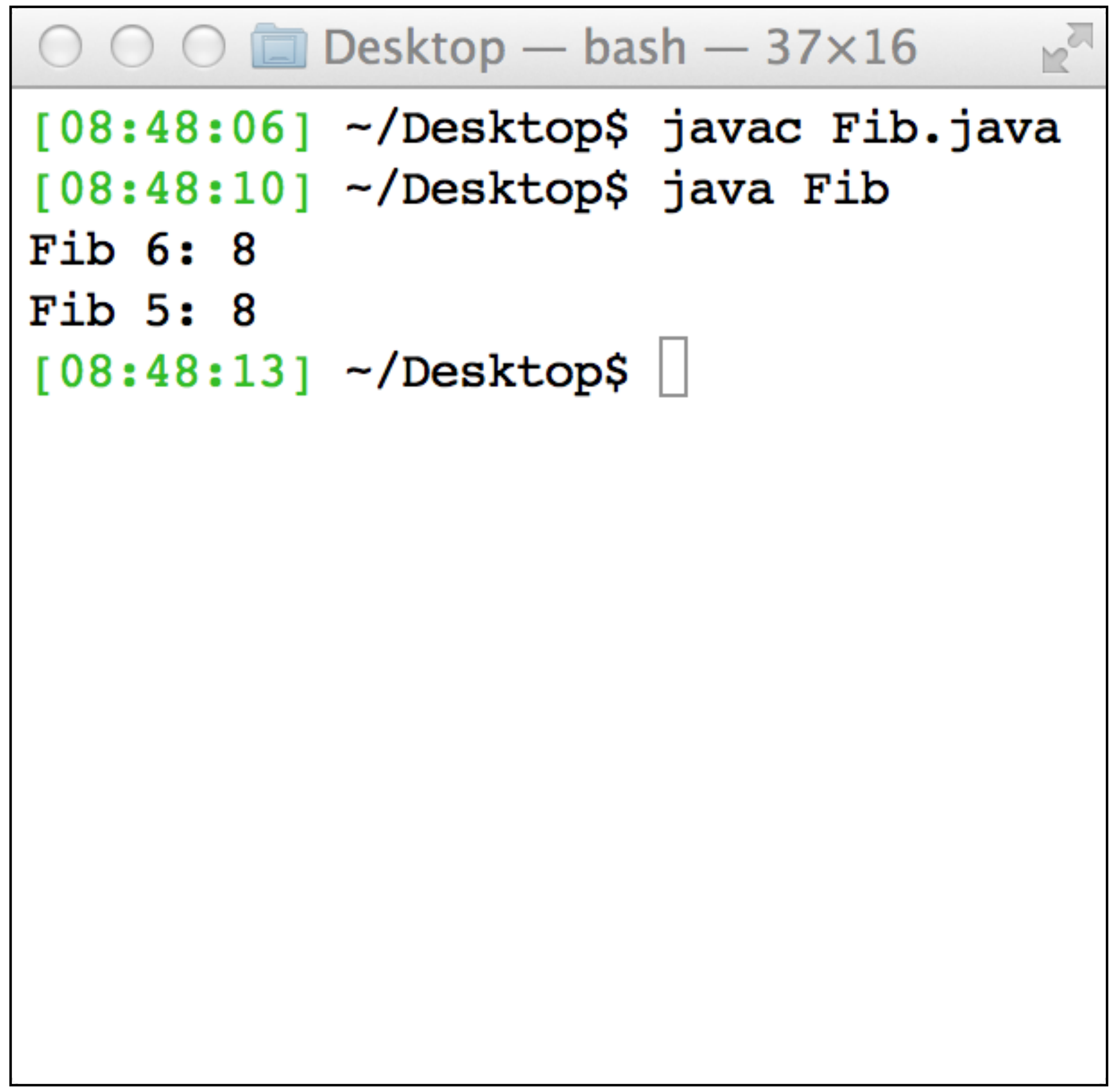
1 Introduction

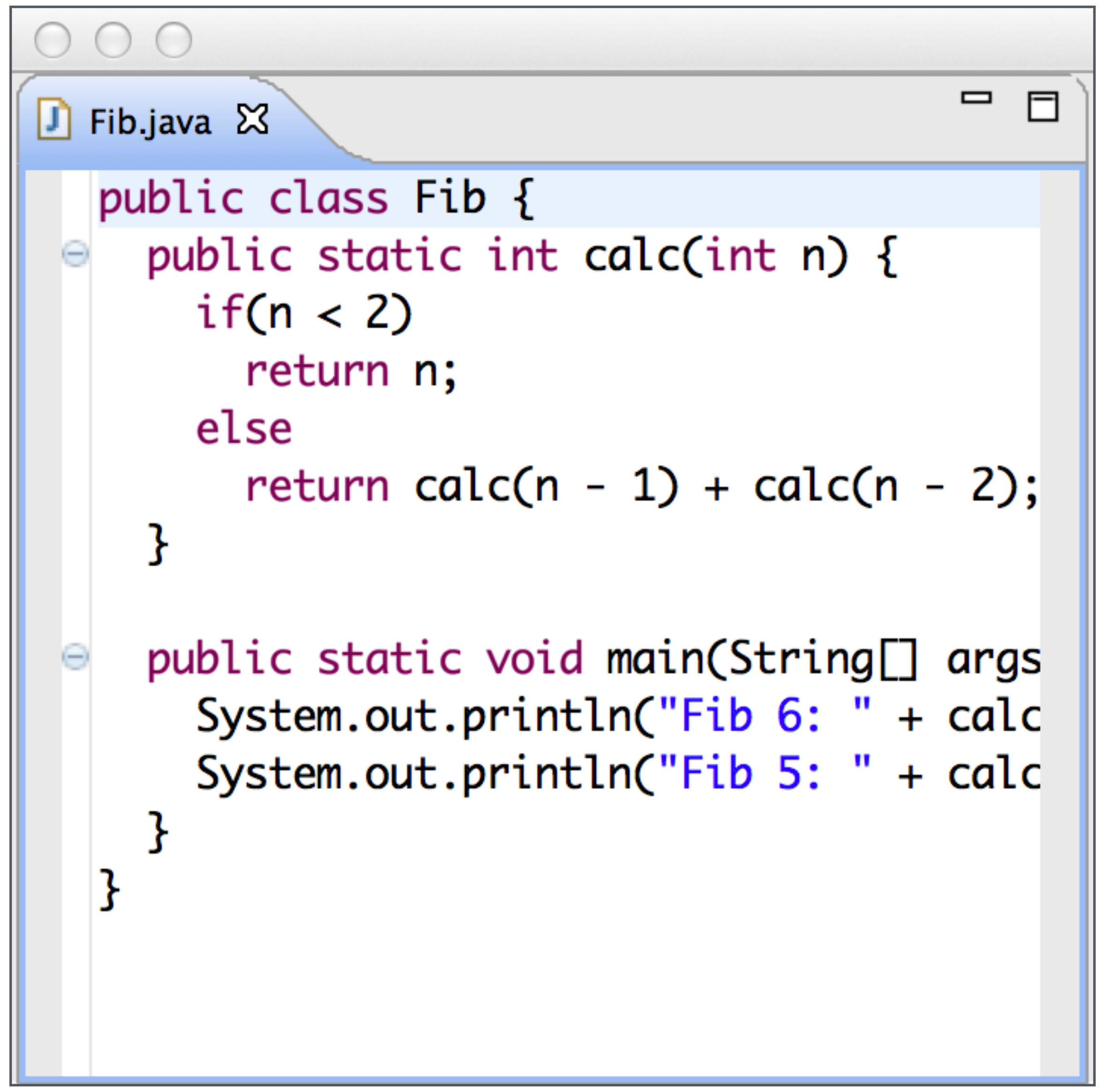
Java combines the experience from the development of several object oriented languages, such as C++, Smalltalk and CLOS. The philosophy of the language designers was to include only features with already known semantics, and to provide a small and simple language.

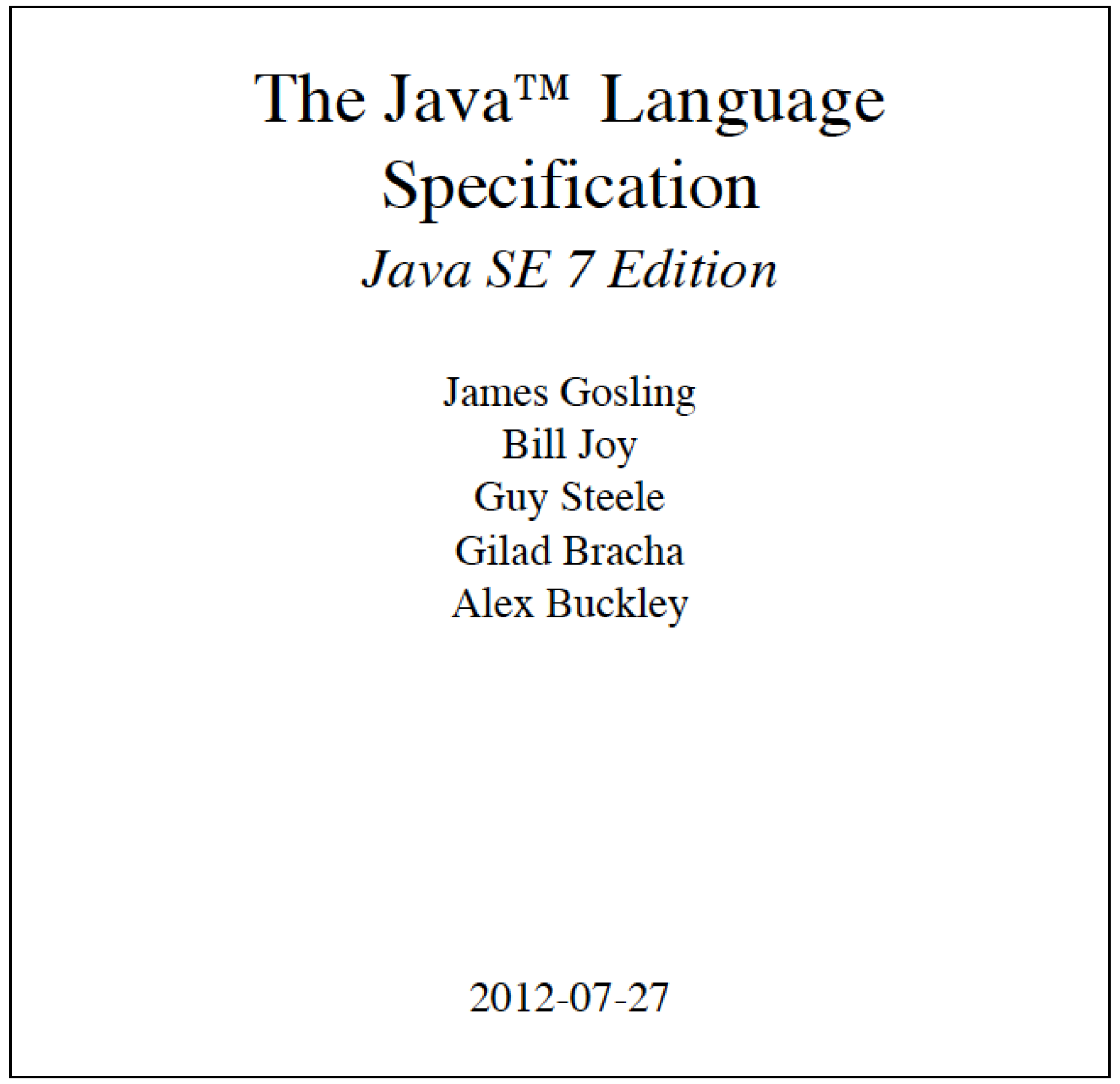
Nevertheless, we feel that the introduction of some new features in Java, as well as the specific combination of features, justifies a study of the Java formal semantics. The use of interfaces, reminiscent of [11,6] is a simplification of the signatures extension for C++ [4] and is – to the best of our knowledge – novel. The mechanism for dynamic method binding is that of C++, but we know of no formal definition. Java adopts the Smalltalk [15] approach whereby all object variables are implicitly pointers.

Furthermore, although there are a large number of studies of the semantics of isolated programming language features or of minimal programming languages [1], [31],[34], there have not been many studies of the formal semantics of *actual* programming languages. In addition, the interplay of features which are very well understood in isolation, might introduce unexpected effects.

abstract syntax





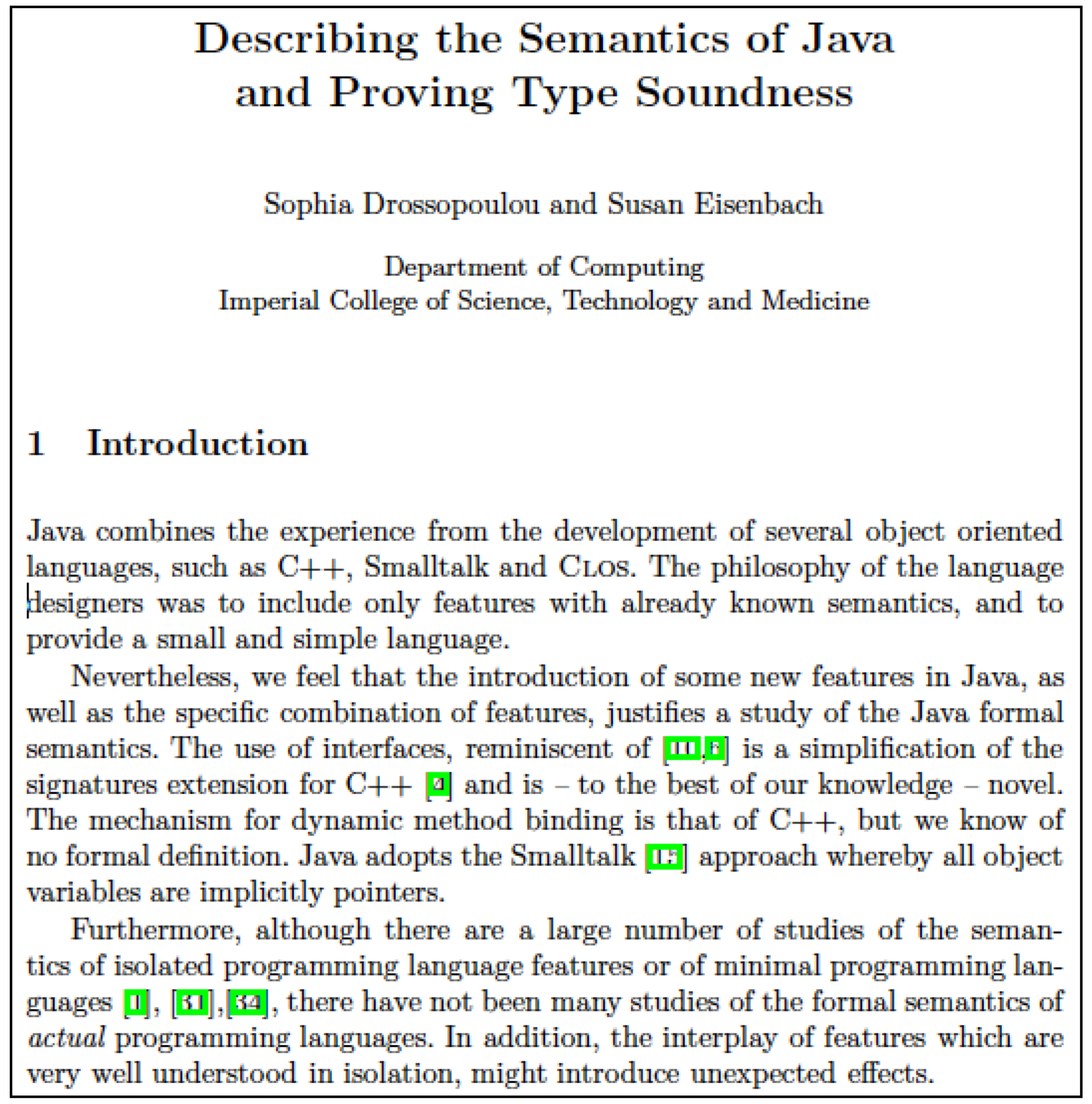


The Java™ Language Specification

Java SE 7 Edition

James Gosling
Bill Joy
Guy Steele
Gilad Bracha
Alex Buckley

2012-07-27



Describing the Semantics of Java and Proving Type Soundness

Sophia Drossopoulou and Susan Eisenbach

Department of Computing
Imperial College of Science, Technology and Medicine

1 Introduction

Java combines the experience from the development of several object oriented languages, such as C++, Smalltalk and CLOS. The philosophy of the language designers was to include only features with already known semantics, and to provide a small and simple language.

Nevertheless, we feel that the introduction of some new features in Java, as well as the specific combination of features, justifies a study of the Java formal semantics. The use of interfaces, reminiscent of [11,5] is a simplification of the signatures extension for C++ [4] and is – to the best of our knowledge – novel. The mechanism for dynamic method binding is that of C++, but we know of no formal definition. Java adopts the Smalltalk [13] approach whereby all object variables are implicitly pointers.

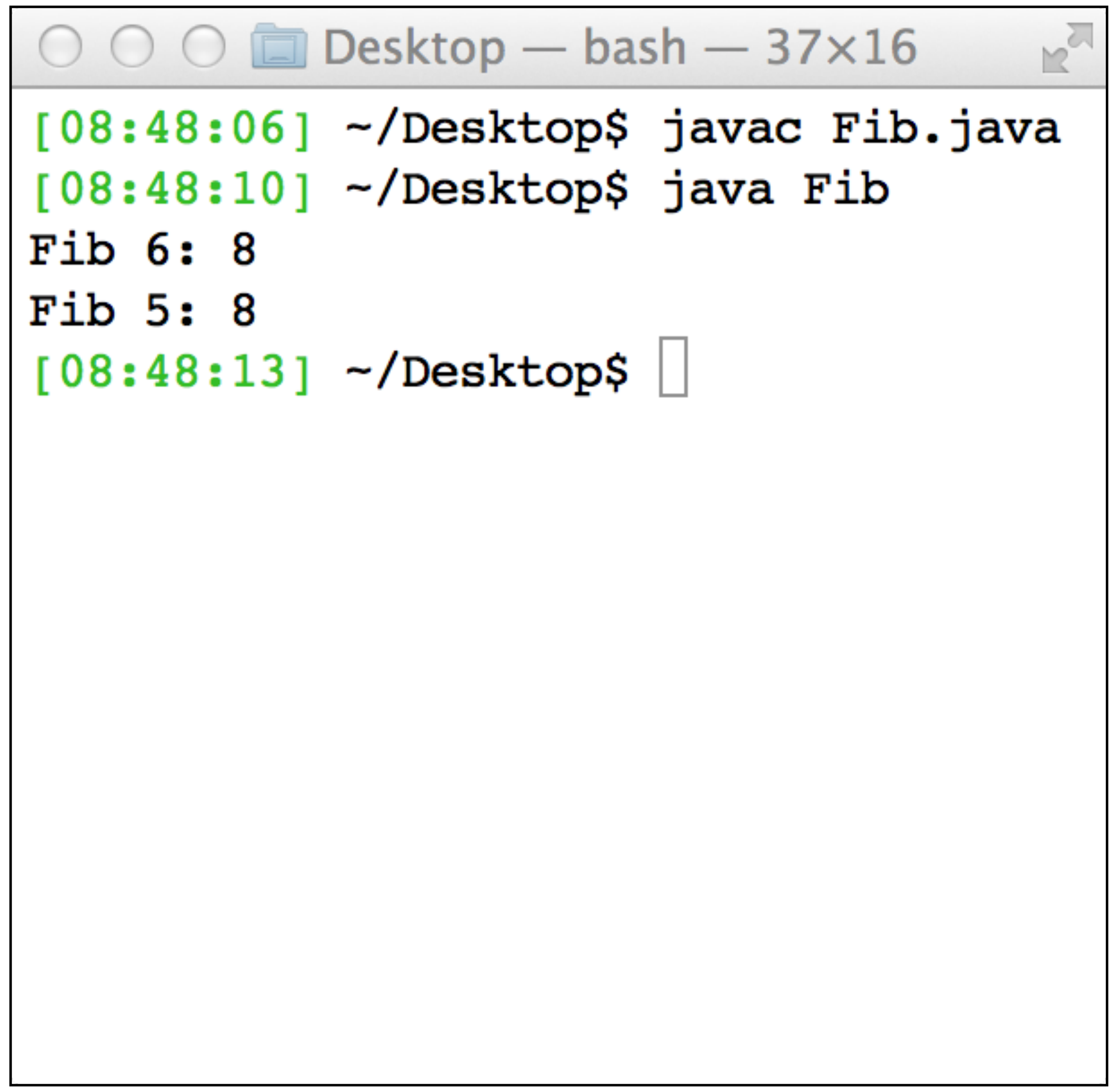
Furthermore, although there are a large number of studies of the semantics of isolated programming language features or of minimal programming languages [1], [31],[32], there have not been many studies of the formal semantics of *actual* programming languages. In addition, the interplay of features which are very well understood in isolation, might introduce unexpected effects.

parser
type checker
code generator
interpreter

parser
error recovery
syntax highlighting
outline
code completion
navigation
type checker
debugger

syntax definition
static semantics
dynamic semantics

abstract syntax
type system

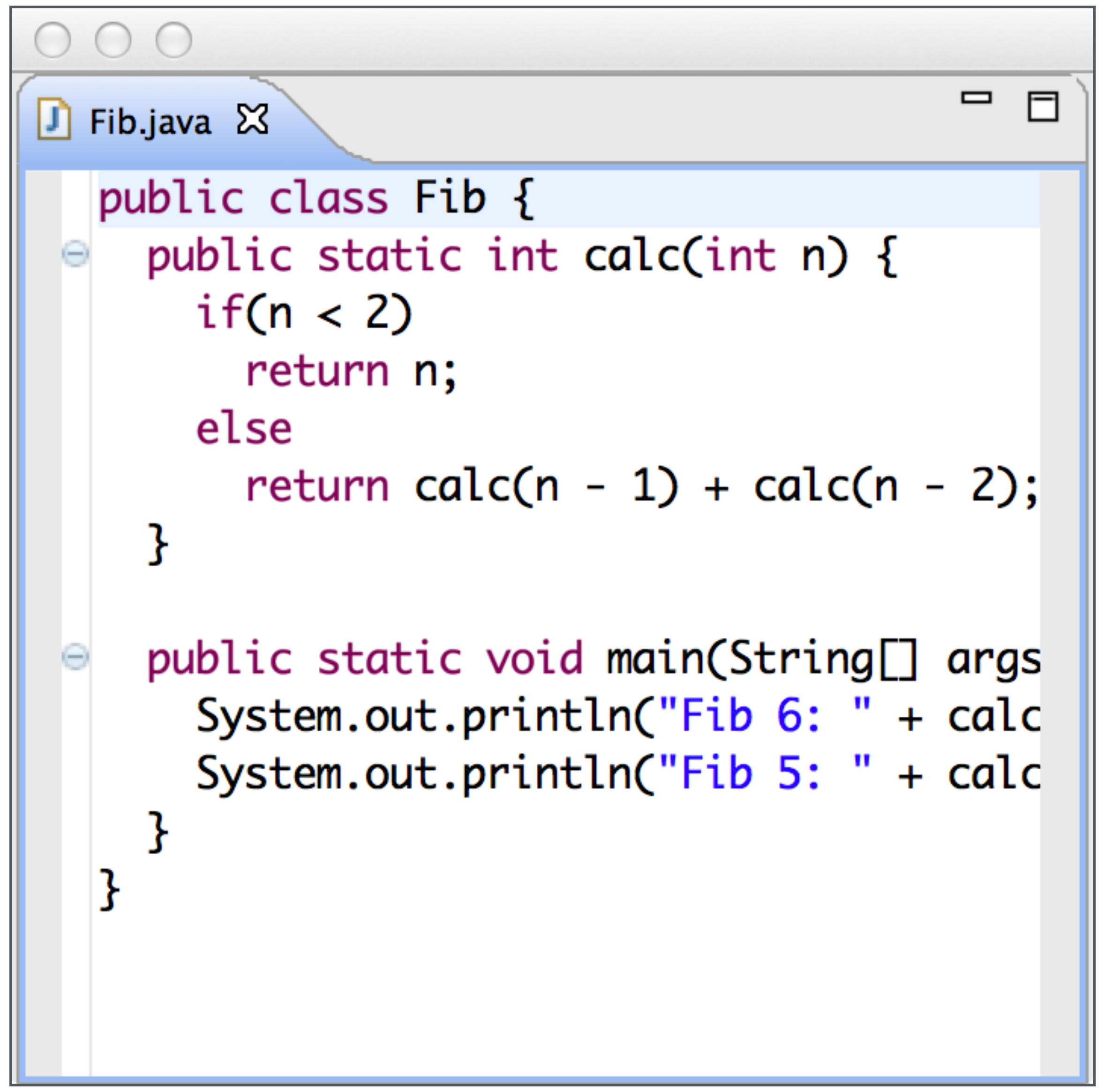


parser

type checker

code generator

interpreter



parser

error recovery

syntax highlighting

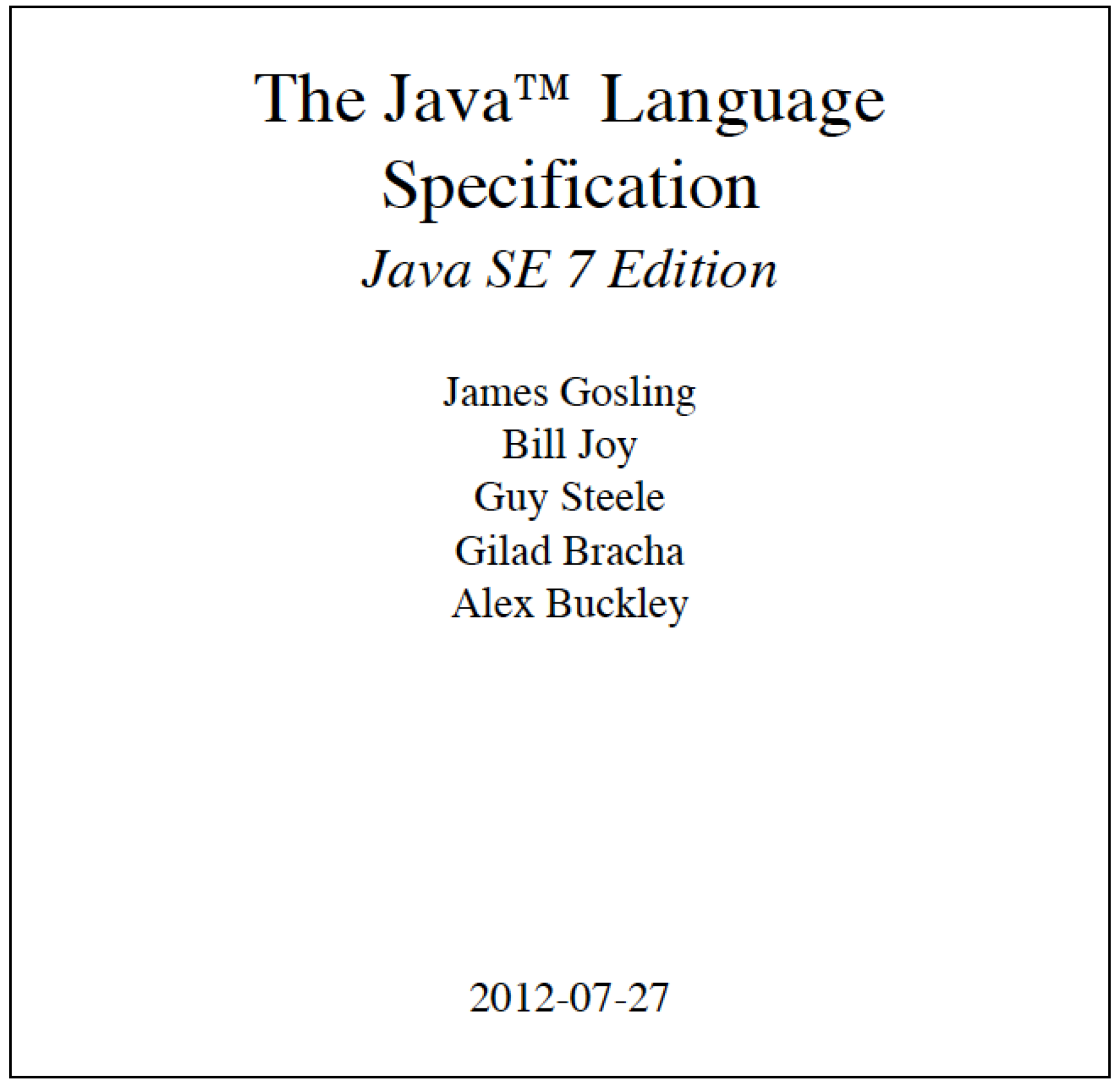
outline

code completion

navigation

type checker

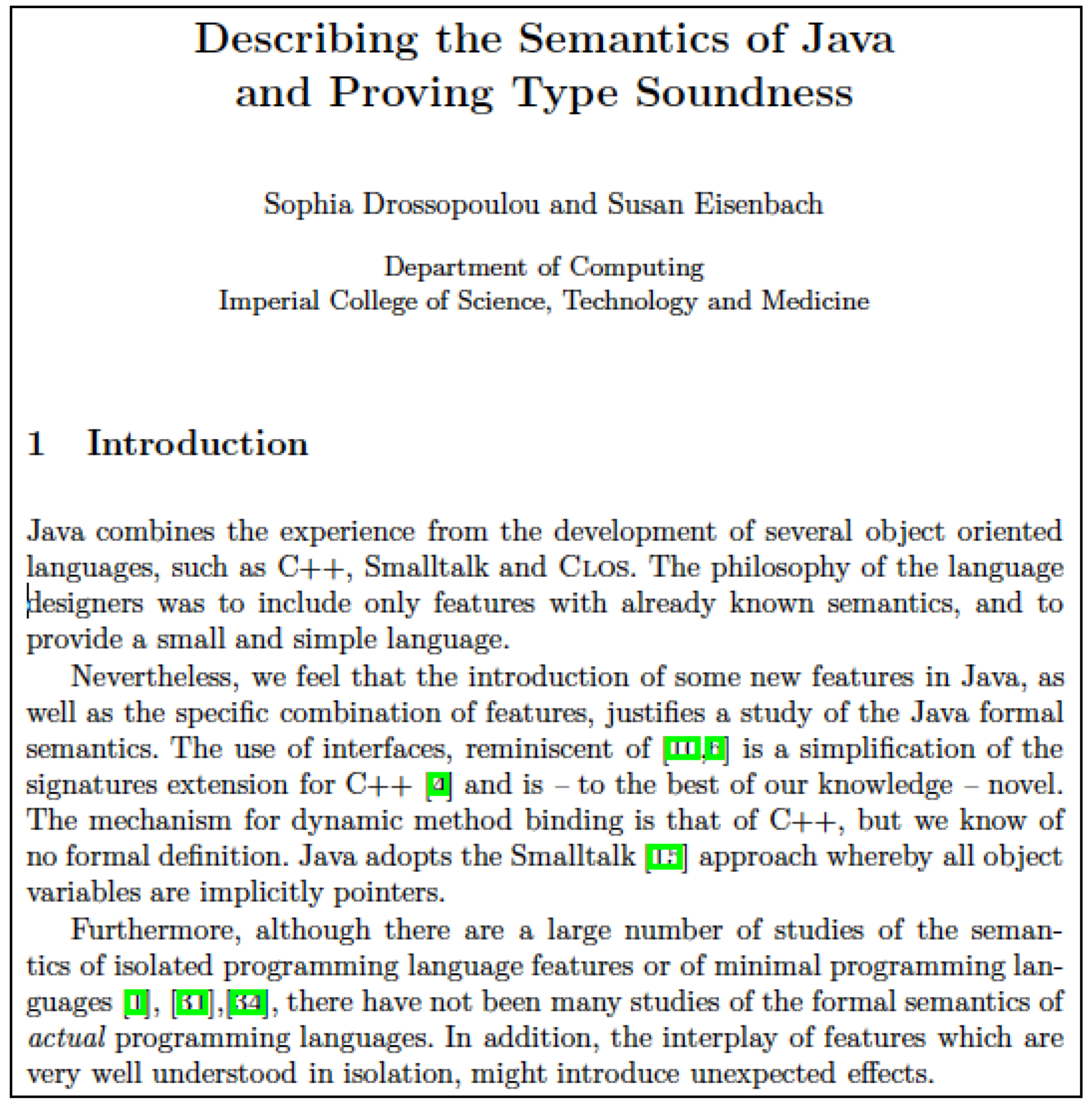
debugger



syntax definition

static semantics

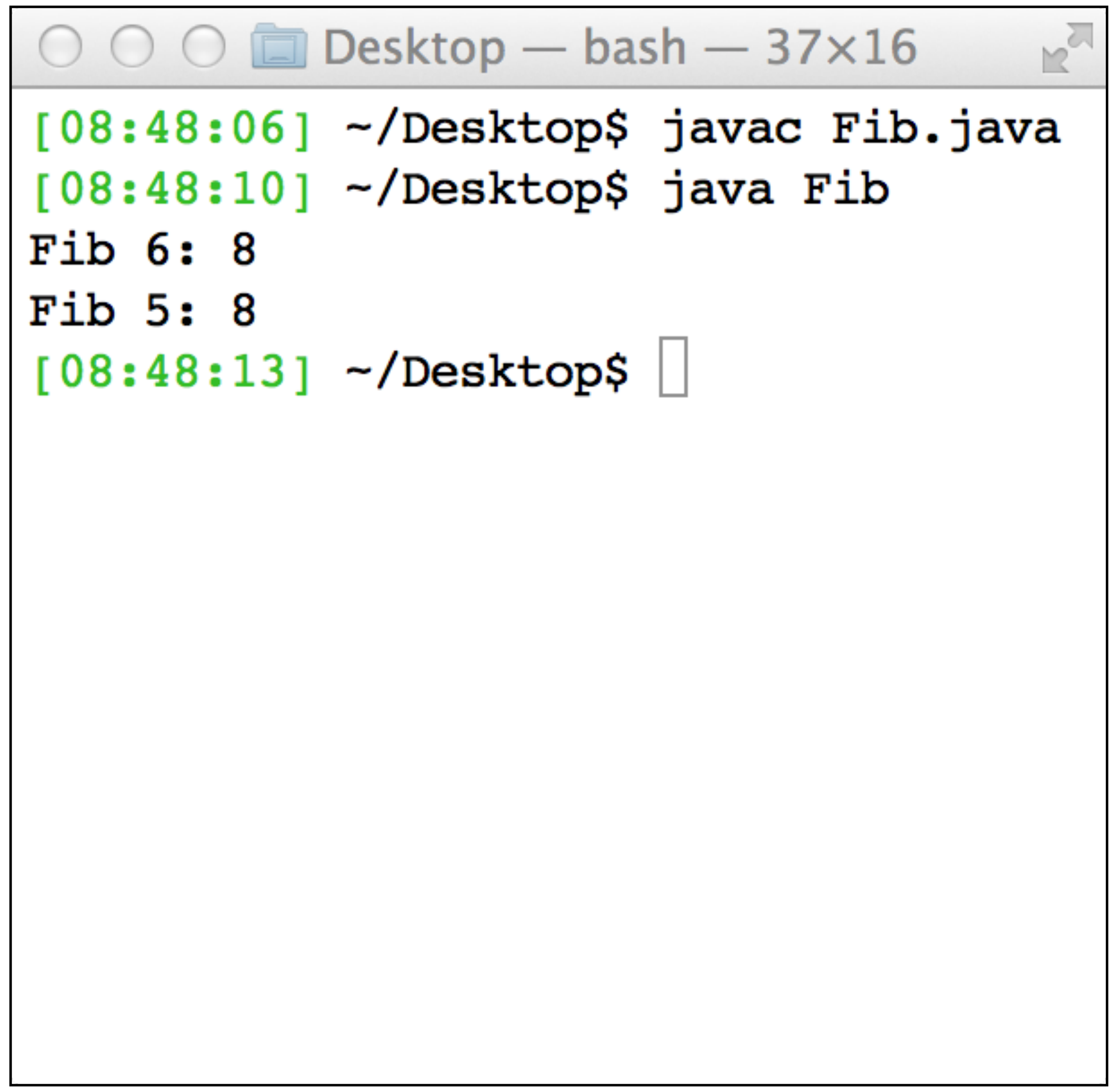
dynamic semantics



abstract syntax

type system

operational semantics

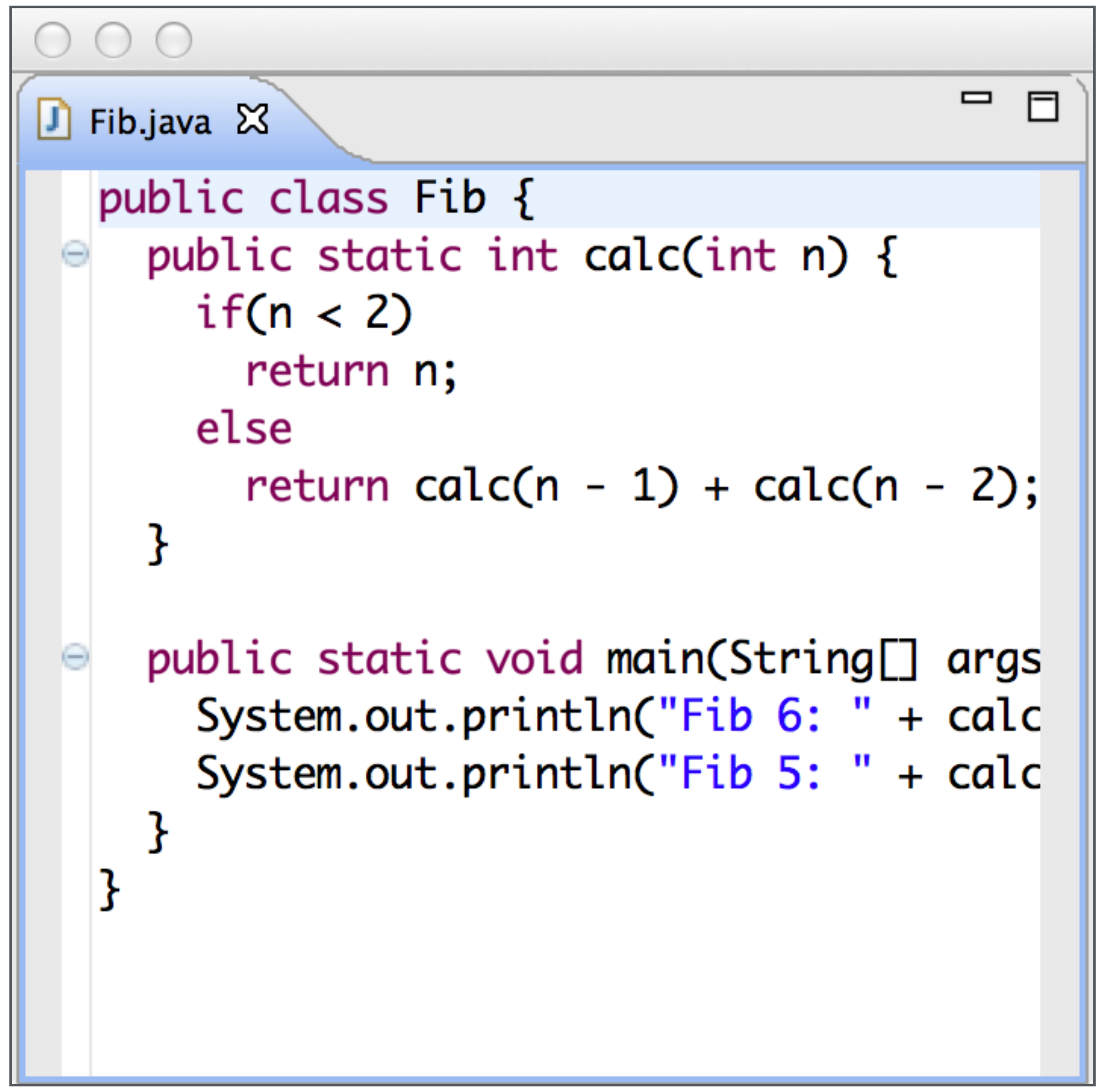


parser

type checker

code generator

interpreter



parser

error recovery

syntax highlighting

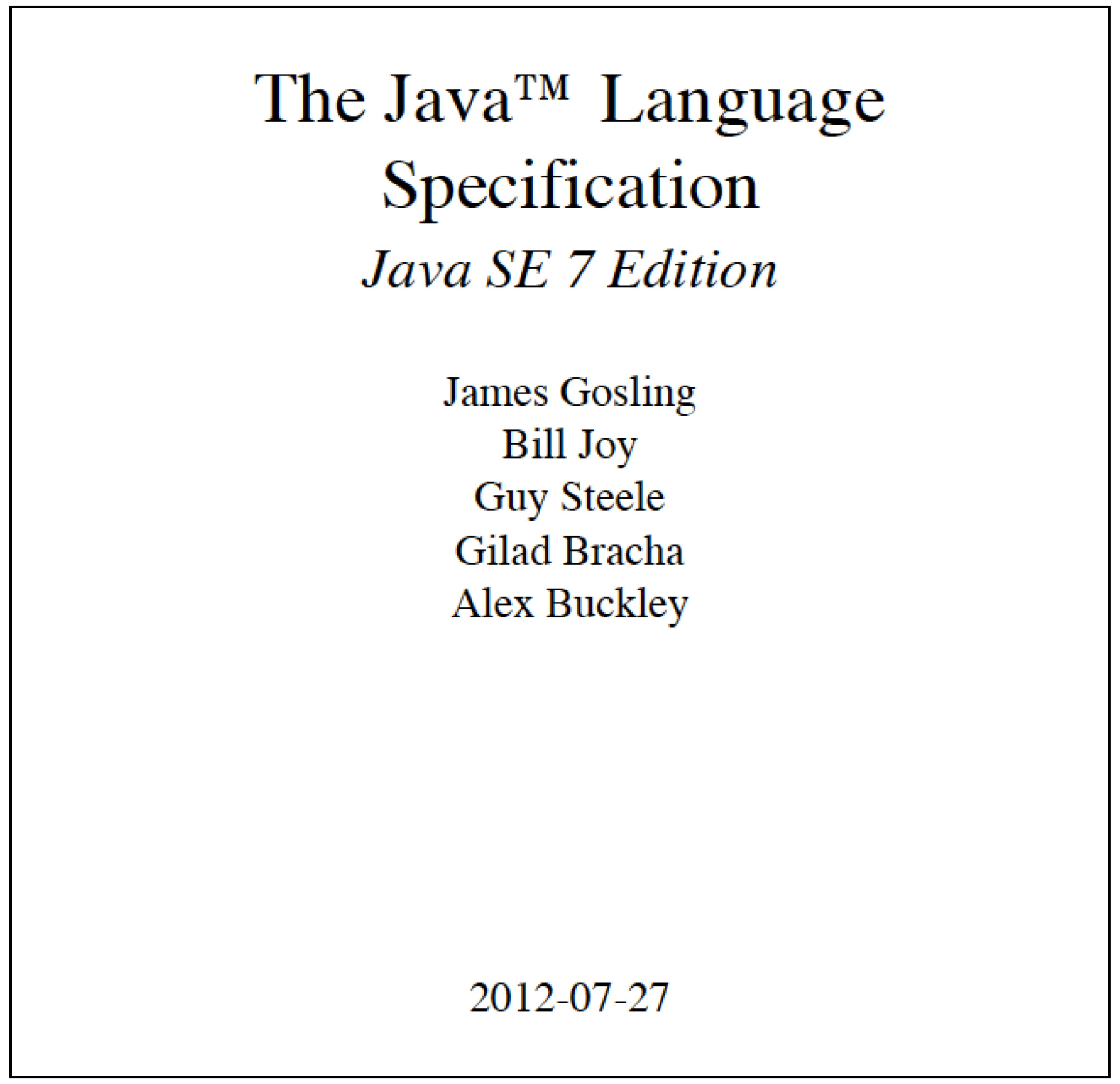
outline

code completion

navigation

type checker

debugger



The Java™ Language Specification

Java SE 7 Edition

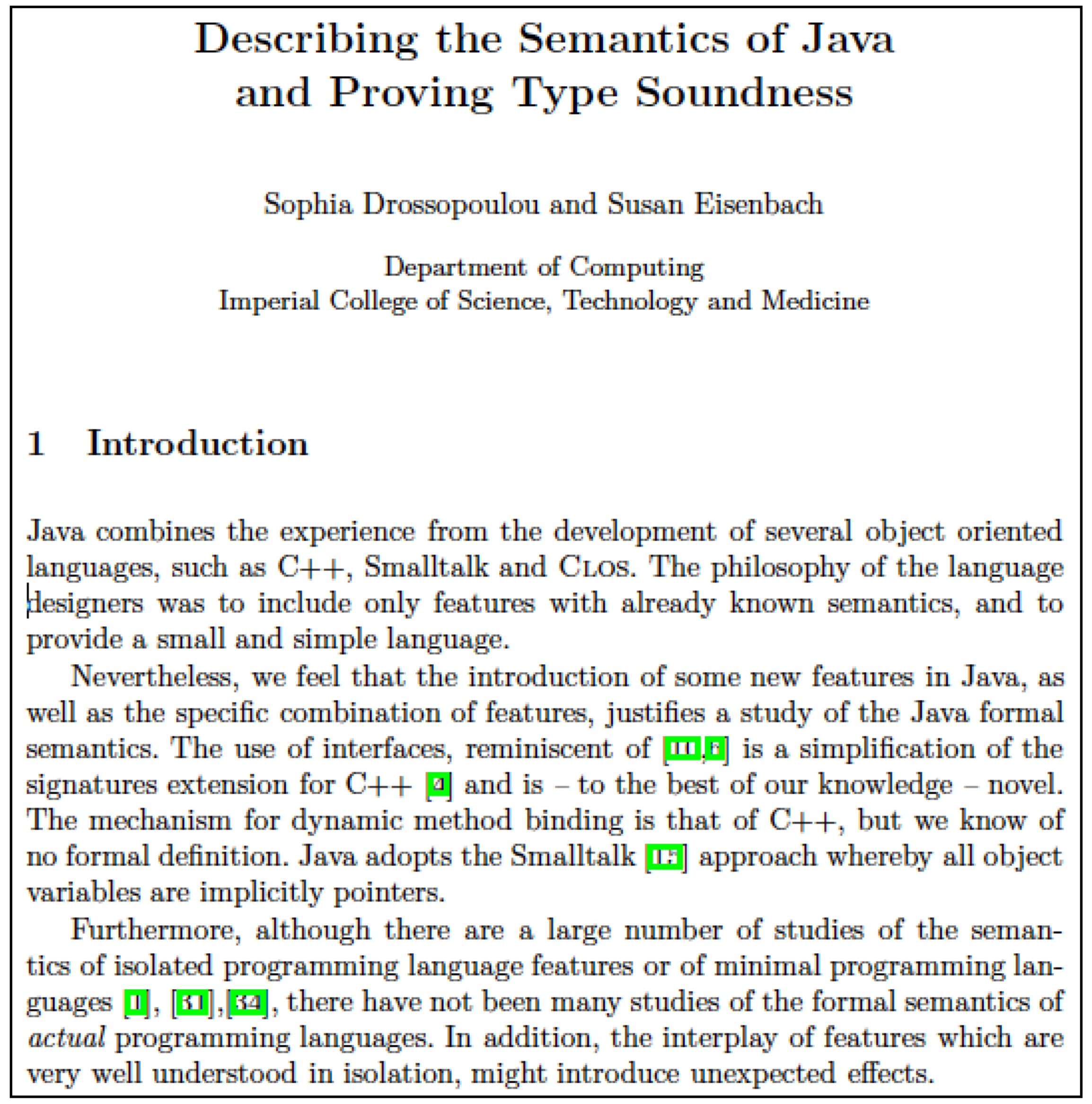
James Gosling
Bill Joy
Guy Steele
Gilad Bracha
Alex Buckley

2012-07-27

syntax definition

static semantics

dynamic semantics



Describing the Semantics of Java and Proving Type Soundness

Sophia Drossopoulou and Susan Eisenbach

Department of Computing
Imperial College of Science, Technology and Medicine

1 Introduction

Java combines the experience from the development of several object oriented languages, such as C++, Smalltalk and CLOS. The philosophy of the language designers was to include only features with already known semantics, and to provide a small and simple language.

Nevertheless, we feel that the introduction of some new features in Java, as well as the specific combination of features, justifies a study of the Java formal semantics. The use of interfaces, reminiscent of [11,6] is a simplification of the signatures extension for C++ [4] and is – to the best of our knowledge – novel. The mechanism for dynamic method binding is that of C++, but we know of no formal definition. Java adopts the Smalltalk [13] approach whereby all object variables are implicitly pointers.

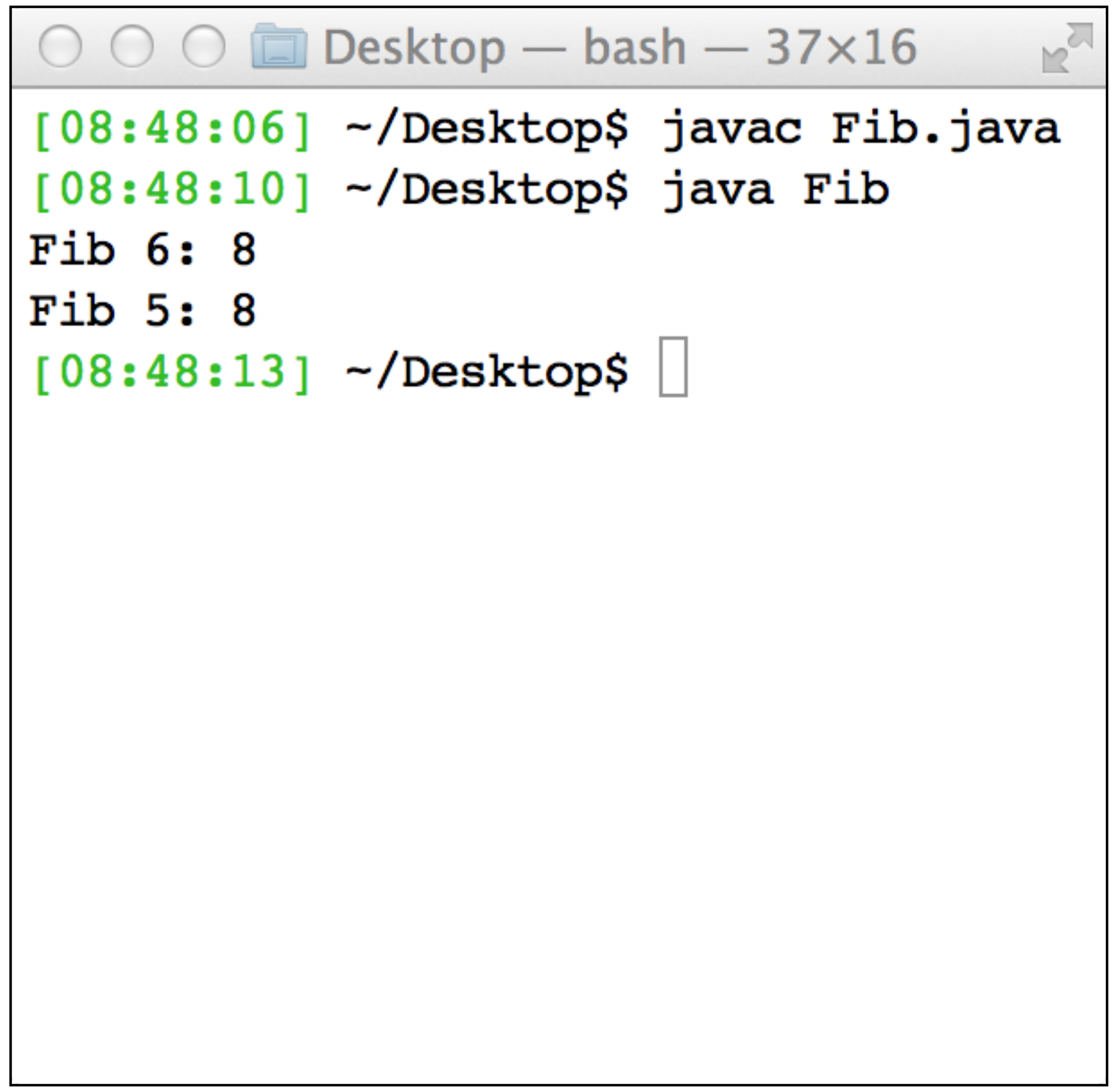
Furthermore, although there are a large number of studies of the semantics of isolated programming language features or of minimal programming languages [1], [31],[34], there have not been many studies of the formal semantics of *actual* programming languages. In addition, the interplay of features which are very well understood in isolation, might introduce unexpected effects.

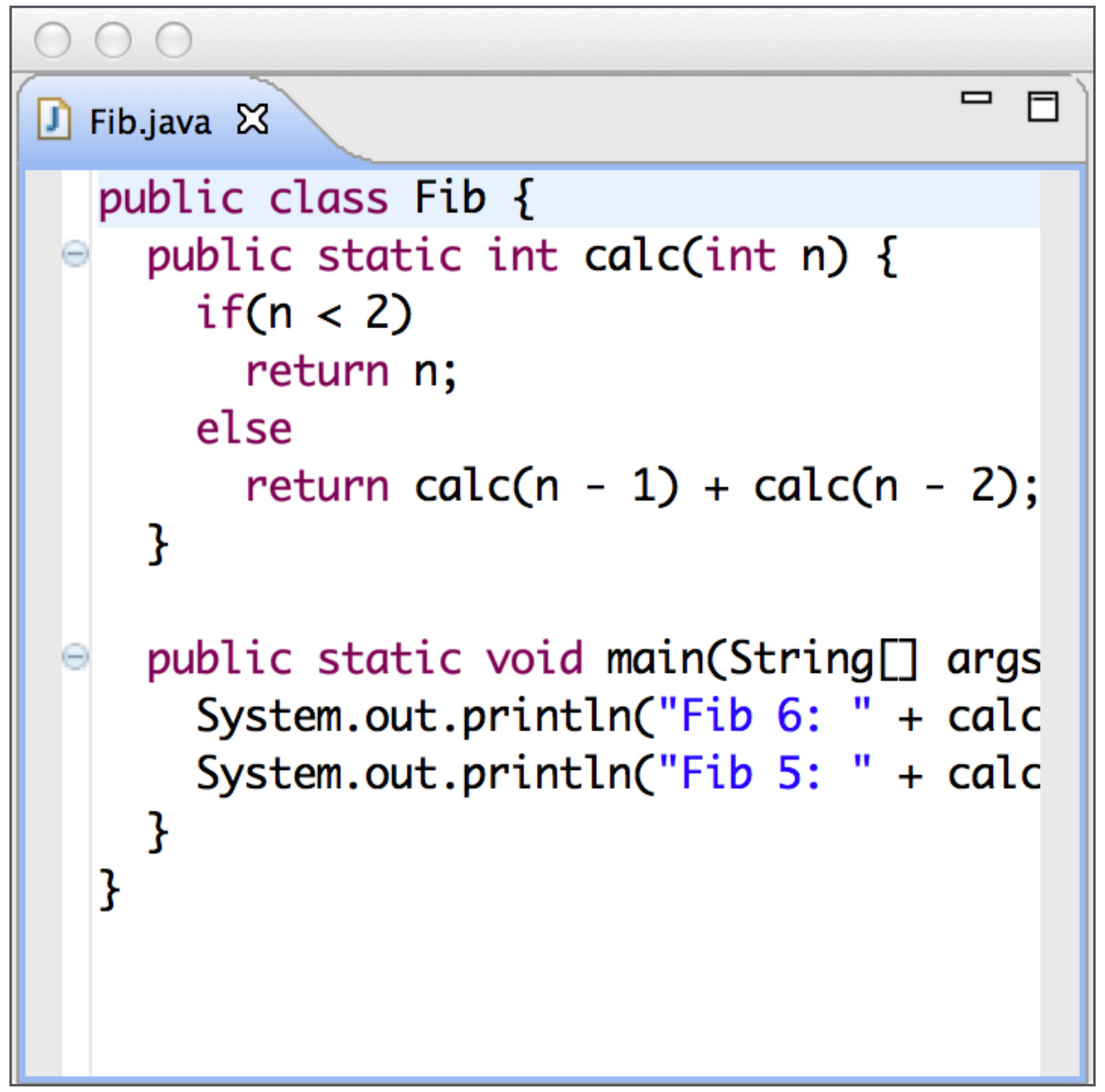
abstract syntax

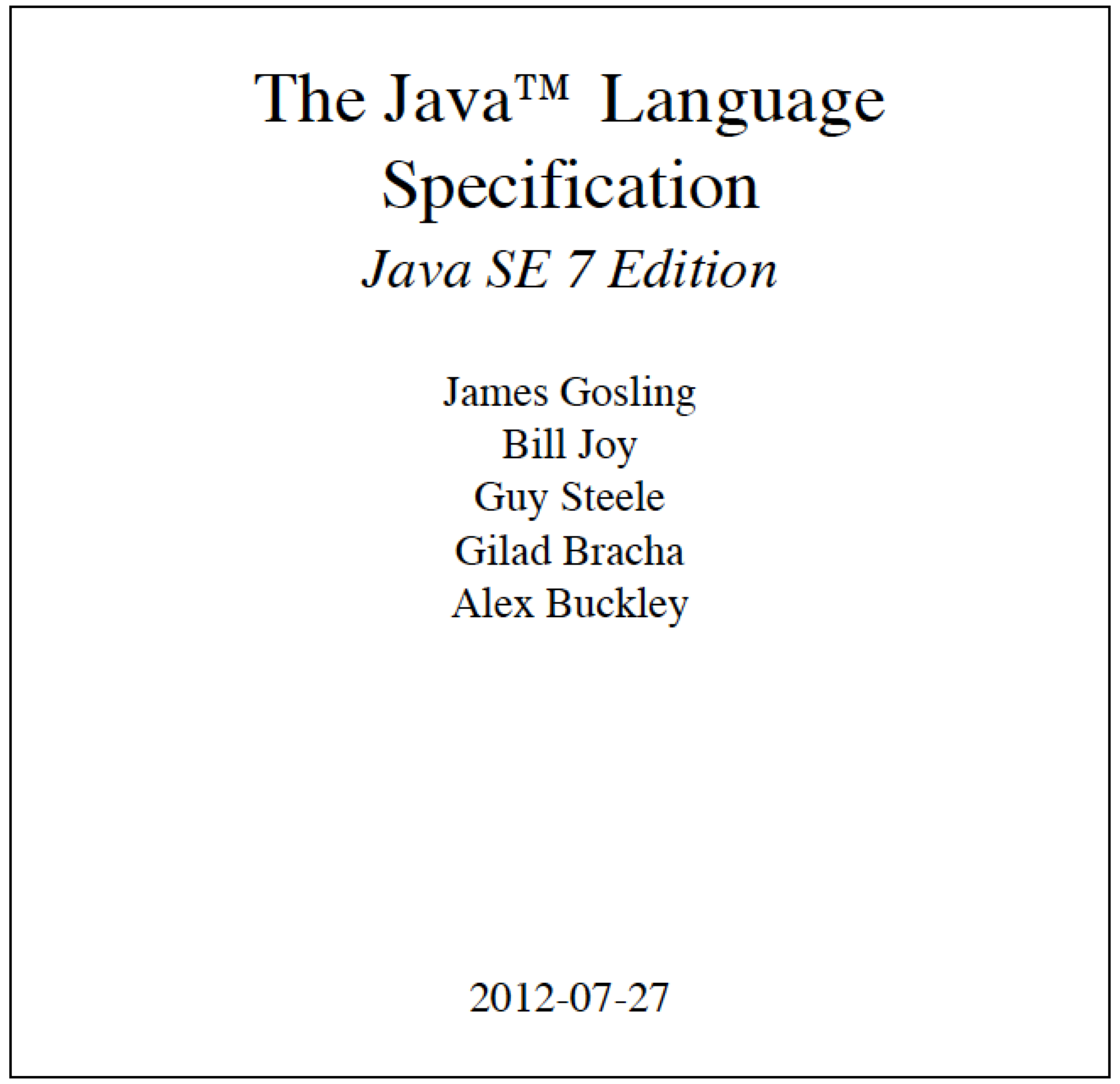
type system

operational semantics

type soundness proof





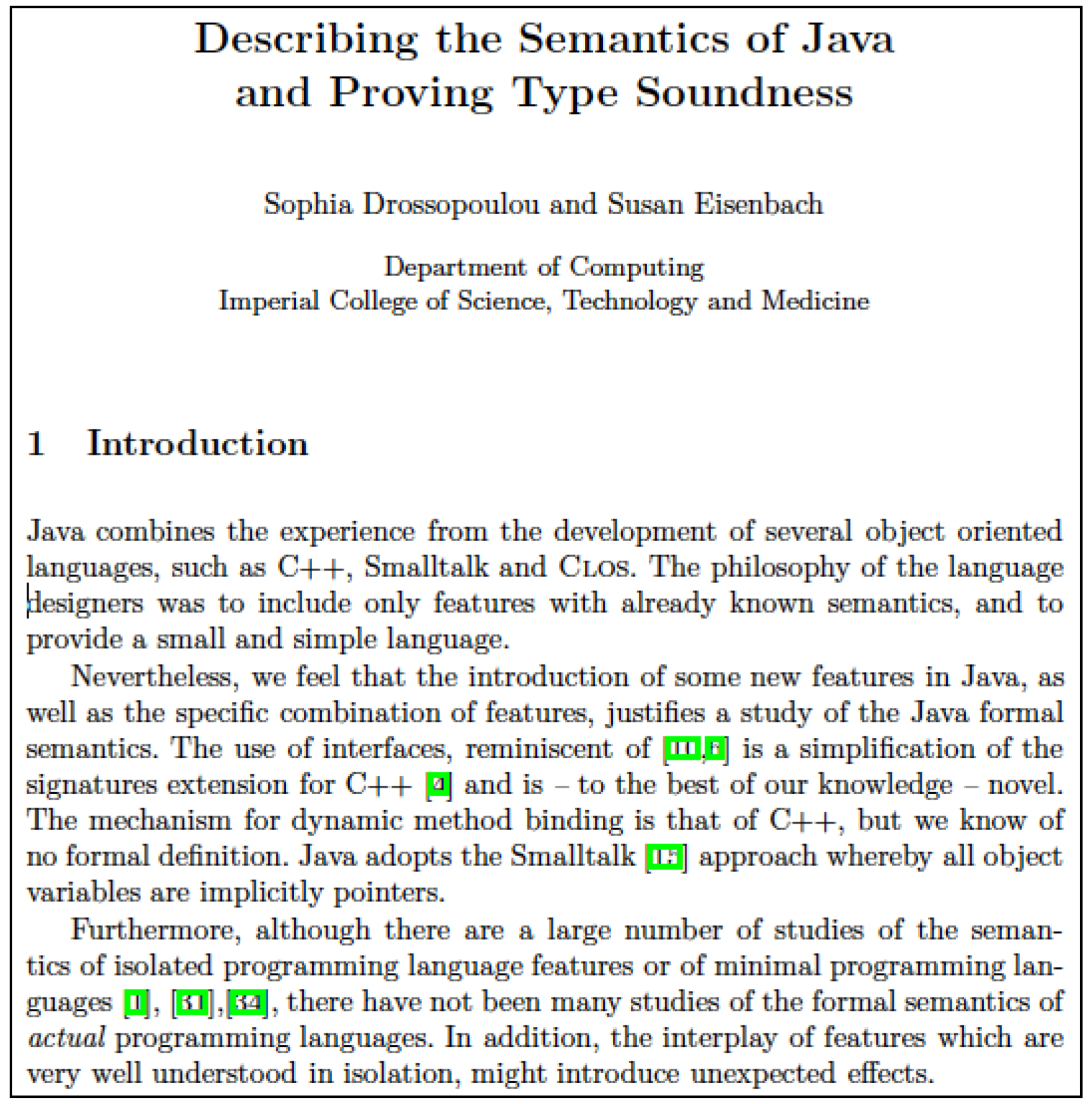


The Java™ Language Specification

Java SE 7 Edition

James Gosling
Bill Joy
Guy Steele
Gilad Bracha
Alex Buckley

2012-07-27



Describing the Semantics of Java and Proving Type Soundness

Sophia Drossopoulou and Susan Eisenbach

Department of Computing
Imperial College of Science, Technology and Medicine

1 Introduction

Java combines the experience from the development of several object oriented languages, such as C++, Smalltalk and CLOS. The philosophy of the language designers was to include only features with already known semantics, and to provide a small and simple language.

Nevertheless, we feel that the introduction of some new features in Java, as well as the specific combination of features, justifies a study of the Java formal semantics. The use of interfaces, reminiscent of [illegible] is a simplification of the signatures extension for C++ [illegible] and is – to the best of our knowledge – novel. The mechanism for dynamic method binding is that of C++, but we know of no formal definition. Java adopts the Smalltalk [illegible] approach whereby all object variables are implicitly pointers.

Furthermore, although there are a large number of studies of the semantics of isolated programming language features or of minimal programming languages [illegible], [illegible],[illegible], there have not been many studies of the formal semantics of *actual* programming languages. In addition, the interplay of features which are very well understood in isolation, might introduce unexpected effects.

parser

type checker

code generator
interpreter

parser
error recovery
syntax highlighting
outline

code completion
navigation
type checker

debugger

syntax definition

static semantics

dynamic semantics

abstract syntax

type system

operational semantics

type soundness proof


Language Design
Syntax Definition
Name Binding
Type Constraints
Dynamic Semantics
Transform
Desktop — bash — 37×16
[08:48:06] ~/Desktop$ javac Fib.java
[08:48:10] ~/Desktop$ java Fib
Fib 6: 8
Fib 5: 8
[08:48:13] ~/Desktop$
Fib.java
public class Fib {
public static int calc(int n) {
if(n < 2)
return n;
else
return calc(n - 1) + calc(n - 2);
}
public static void main(String[] args
System.out.println("Fib 6: " + calc
System.out.println("Fib 5: " + calc
}
}
The Java™ Language Specification
Java SE 7 Edition
James Gosling
Bill Joy
Guy Steele
Gilad Bracha
Alex Buckley
2012-07-27
Describing the Semantics of Java and Proving Type Soundness
Sophia Drossopoulou and Susan Eisenbach
Department of Computing
Imperial College of Science, Technology and Medicine
1 Introduction
Java combines the experience from the development of several object oriented languages, such as C++, Smalltalk and Clos. The philosophy of the language designers was to include only features with already known semantics, and to provide a small and simple language.
Nevertheless, we feel that the introduction of some new features in Java, as well as the specific combination of features, justifies a study of the Java formal semantics. The use of interfaces, reminiscent of [10,5] is a simplification of the signatures extension for C++ [4] and is – to the best of our knowledge – novel. The mechanism for dynamic method binding is that of C++, but we know of no formal definition. Java adopts the Smalltalk [15] approach whereby all object variables are implicitly pointers.
Furthermore, although there are a large number of studies of the semantics of isolated programming language features or of minimal programming languages [1], [31],[32], there have not been many studies of the formal semantics of actual programming languages. In addition, the interplay of features which are very well understood in isolation, might introduce unexpected effects.

Syntax Definition

Syntax: Phrase Structure of Programs

```
int fib(int n) {
  if(n <= 1)
    return 1;
  else
    return fib(n - 2) + fib(n - 1);
}
```

Syntax: Phrase Structure of Programs

```
int fib(int n) {
  if (n <= 1)
    return 1;
  else
    return fib(n-2) + fib(n-1);
}
```

Syntax: Phrase Structure of Programs

```
int fib(int n) {if(n <= 1) return 1; else return fib(n-2) + fib(n-1);}
```

Syntax: Phrase Structure of Programs

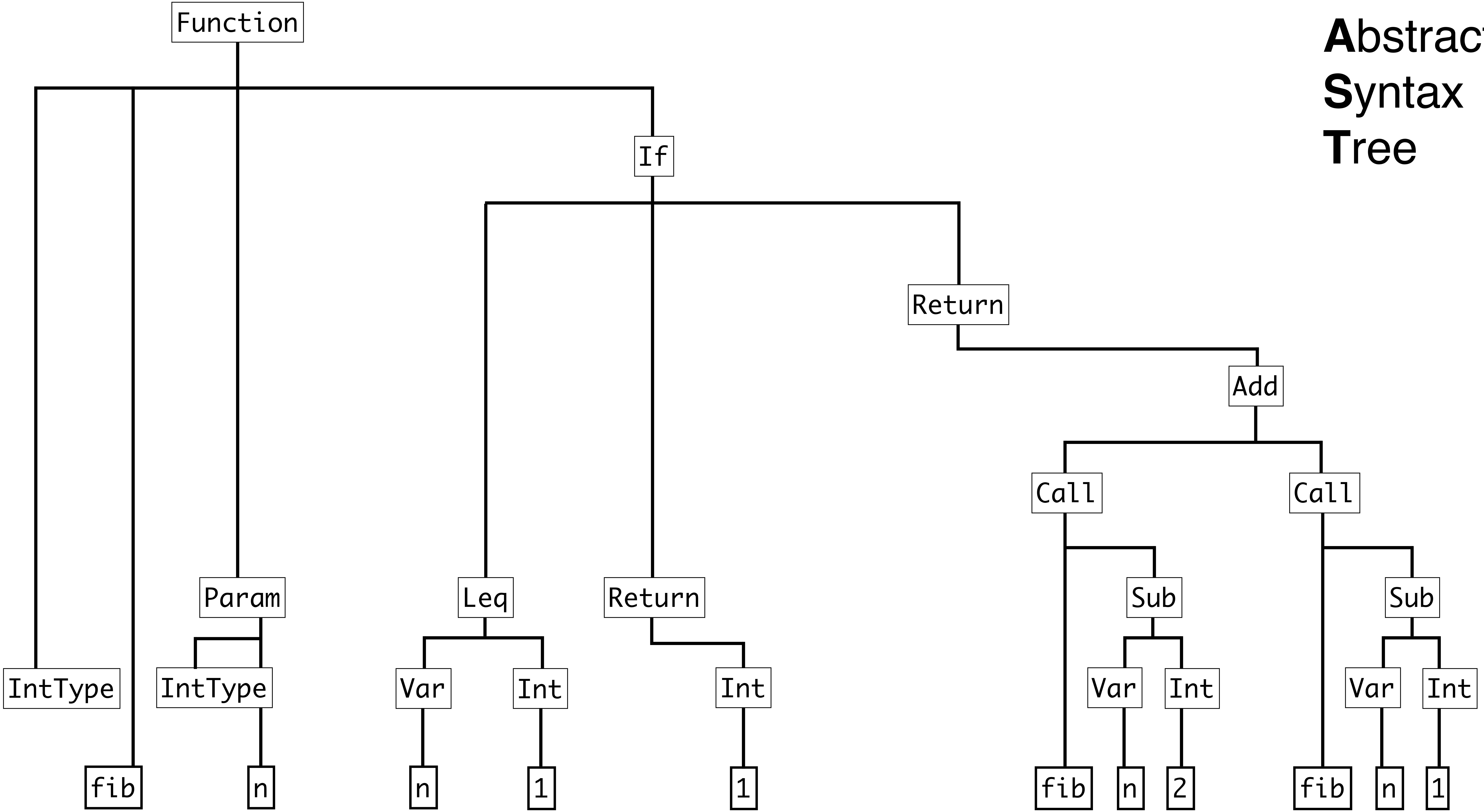
Syntax: Phrase Structure of Programs

Syntax: Phrase Structure of Programs

Syntax: Phrase Structure of Programs

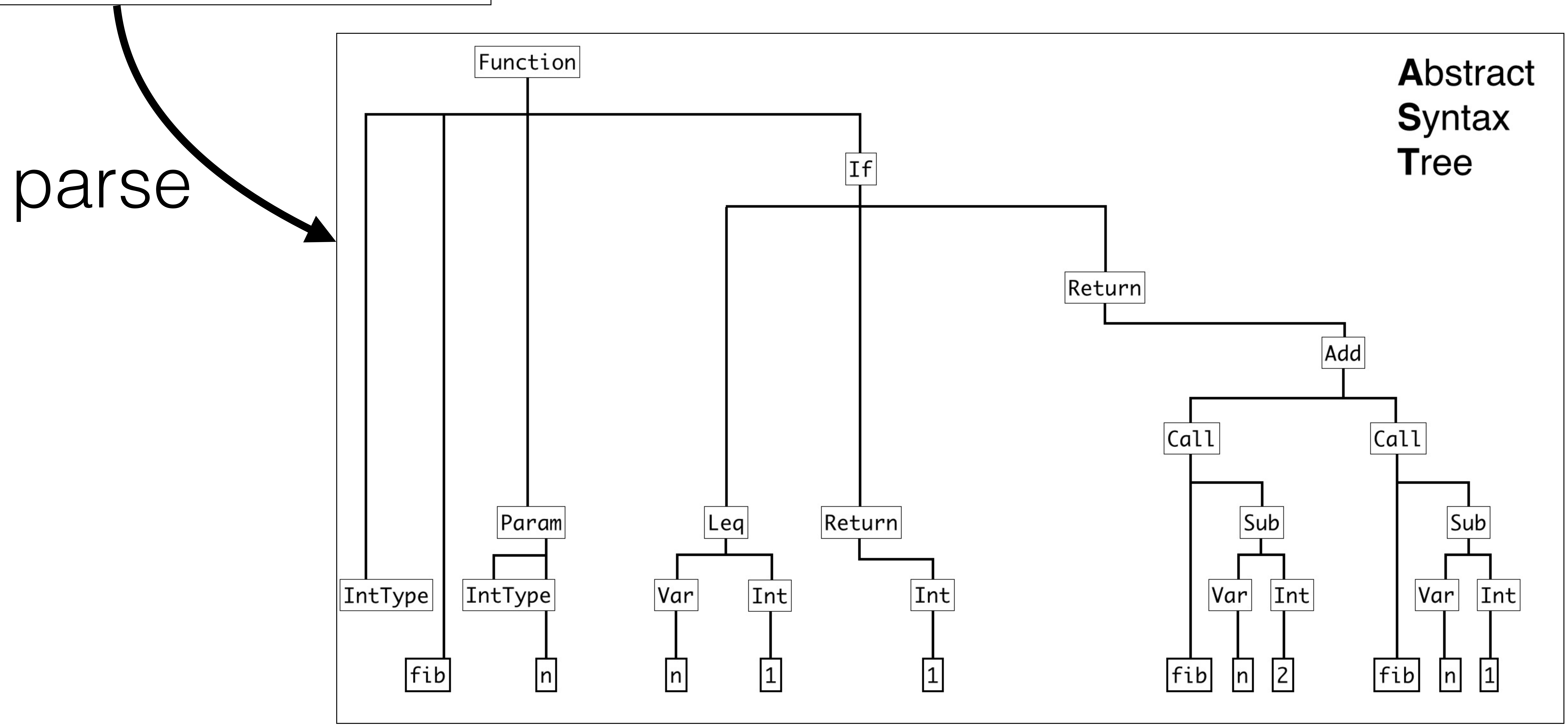
Syntax: Phrase Structure of Programs

Abstract **S**yntax **T**ree



```
int fib(int n) {
  if(n <= 1)
    return 1;
  else
    return fib(n - 2) + fib(n - 1);
}
```

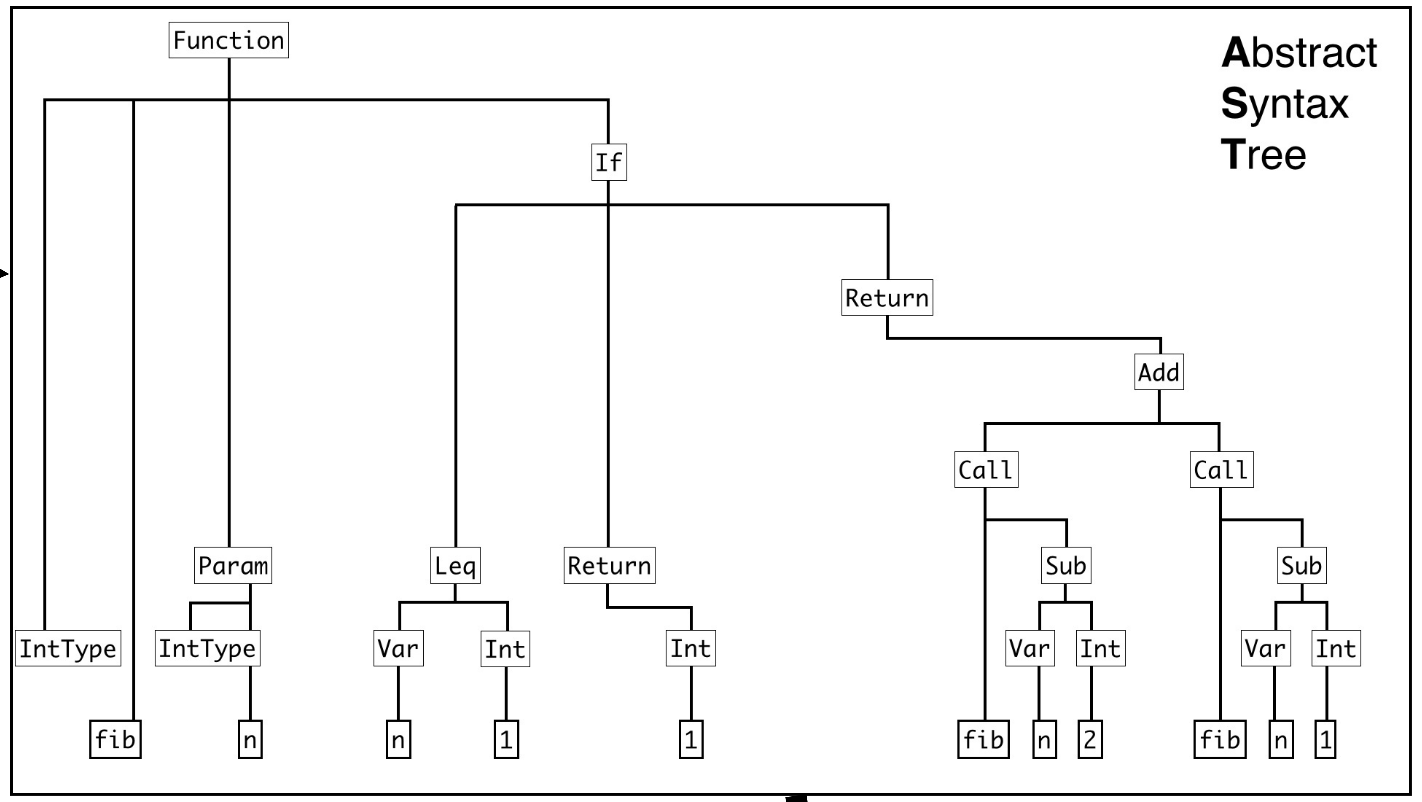
Text



```
int fib(int n) {
  if(n <= 1)
    return 1;
  else
    return fib(n - 2) + fib(n - 1);
}
```

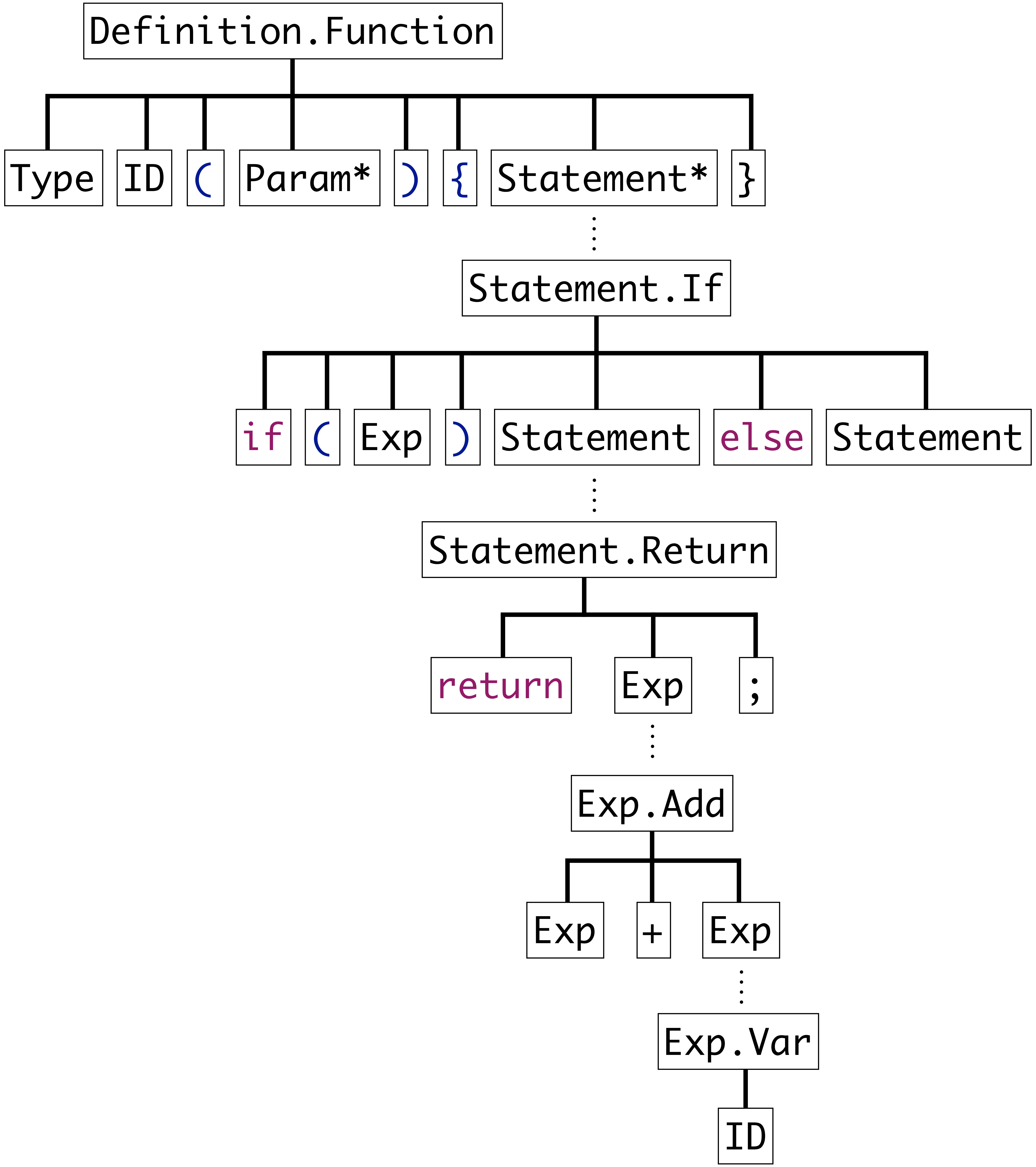
Text

parse




```
Function(
  IntType()
, "fib"
, [Param(IntType(), "n")]
, [ If(
      Leq(Var("n"), Int("1"))
    , Int("1")
    , Add(
        Call("fib", [Sub(Var("n"), Int("2"))])
      , Call("fib", [Sub(Var("n"), Int("1"))])
      )
    )
  ]
)
```

Abstract
Syntax
Term

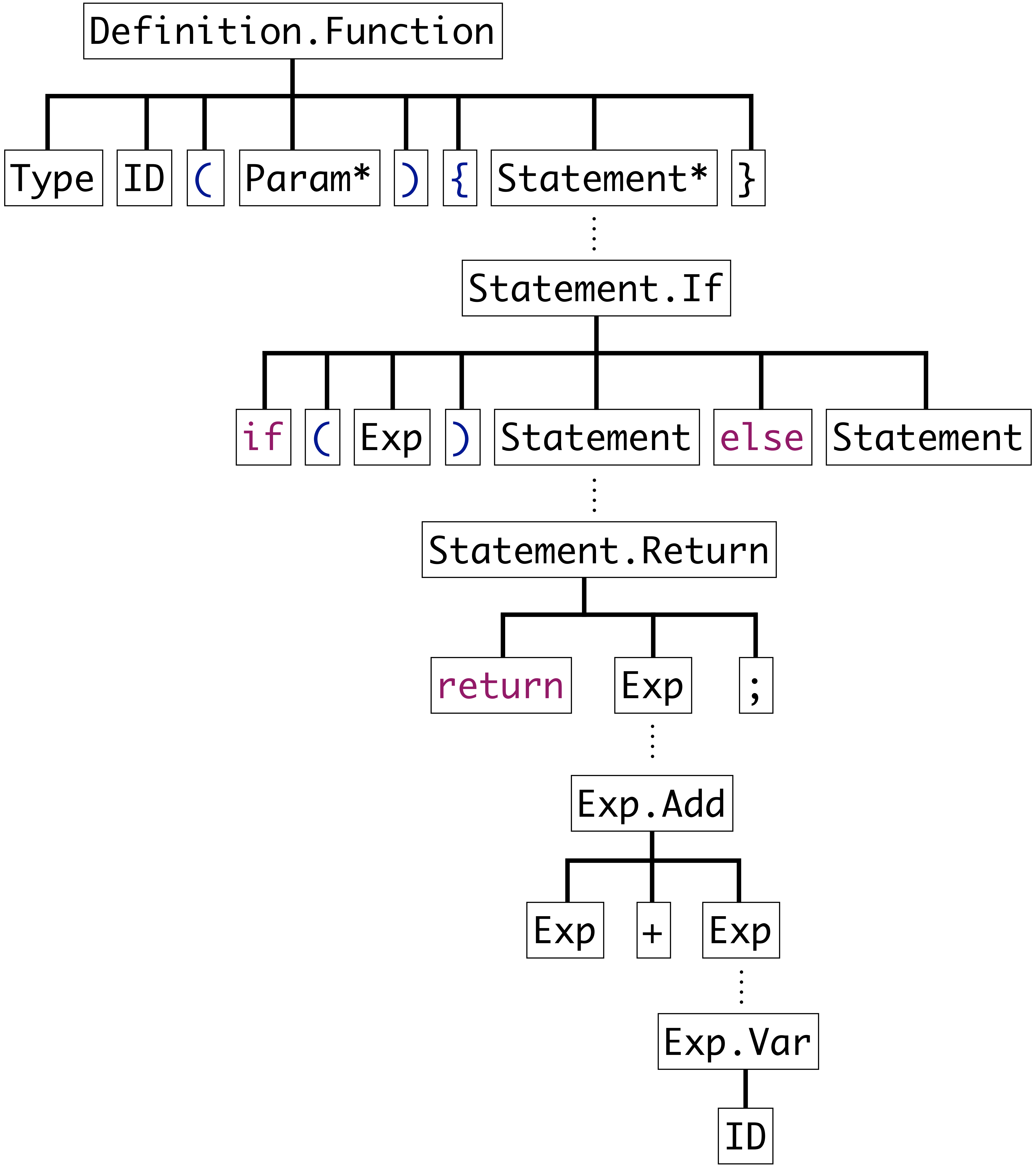


Understanding Syntax =
Understanding Tree Structure

parse(prettyprint(t)) = t

No need to understand
how parse **works!**

Demo: Syntax Definition in SDF3



The Syntax Definition Formalism SDF3

```
templates

  Definition.Function = <
    <Type> <ID>(<Param*; separator=",">) {
      <Statement*; separator="\n">
    }
  >

  Statement.If = <
    if(<Exp>)
      <Statement>
    else
      <Statement>
  >

  Statement.Return = <return <Exp>;>

  Exp.Add  = <<Exp> + <Exp>>

  Exp.Var  = <<ID>>
```

Multi-Purpose Declarative Syntax Definition

```
Statement.If = <
  if(<Exp>)
    <Statement>
  else
    <Statement>
>
```

Syntax Definition

- Parser
- Error recovery rules
- Pretty-Printer
- Abstract syntax tree schema
- Syntactic coloring
- Syntactic completion templates
- Folding rules
- Outline rules

Name and Type Analysis

Name Binding & Scope Rules

what does this variable refer to?

```
int fib(int n) {
  if(n <= 1)
    return 1;
  else
    return fib(n - 2) + fib(n - 1);
}
```

which function is being called here?

Needed for

- checking correct use of names and types
- lookup in interpretation and compilation
- navigation in IDE
- code completion

State-of-the-art

- programmatic encoding of name resolution algorithms

Our contribution

- declarative language for name binding & scope rules
- generation of incremental name resolution algorithm

- Konat, Kats, Wachsmuth, Visser (SLE 2012)
- Wachsmuth, Konat, Vergu, Groenewegen, Visser (SLE 2013)


Language Design
Syntax Definition
Name Binding
Type Constraints
Dynamic Semantics
Transform
Desktop — bash — 37×16
[08:48:06] ~/Desktop$ javac Fib.java
[08:48:10] ~/Desktop$ java Fib
Fib 6: 8
Fib.java
public class Fib {
public static int calc(int n) {
}
}
The Java™ Language Specification
Java SE 7 Edition
2012-07-27
Describing the Semantics of Java and Proving Type Soundness
Sophia Drossopoulou and Susan Eisenbach
Department of Computing
Imperial College of Science, Technology and Medicine
The mechanism for dynamic method binding is that of C++, but we know of no formal definition. Java adopts the Smalltalk [15] approach whereby all object variables are implicitly pointers.
Furthermore, although there are a large number of studies of the semantics of isolated programming language features or of minimal programming languages [1], [31], [33], there have not been many studies of the formal semantics of actual programming languages. In addition, the interplay of features which are very well understood in isolation, might introduce unexpected effects.
Demo: Name and Type Analysis in NaBL+TS

Declarative Name Binding and Scope Rules

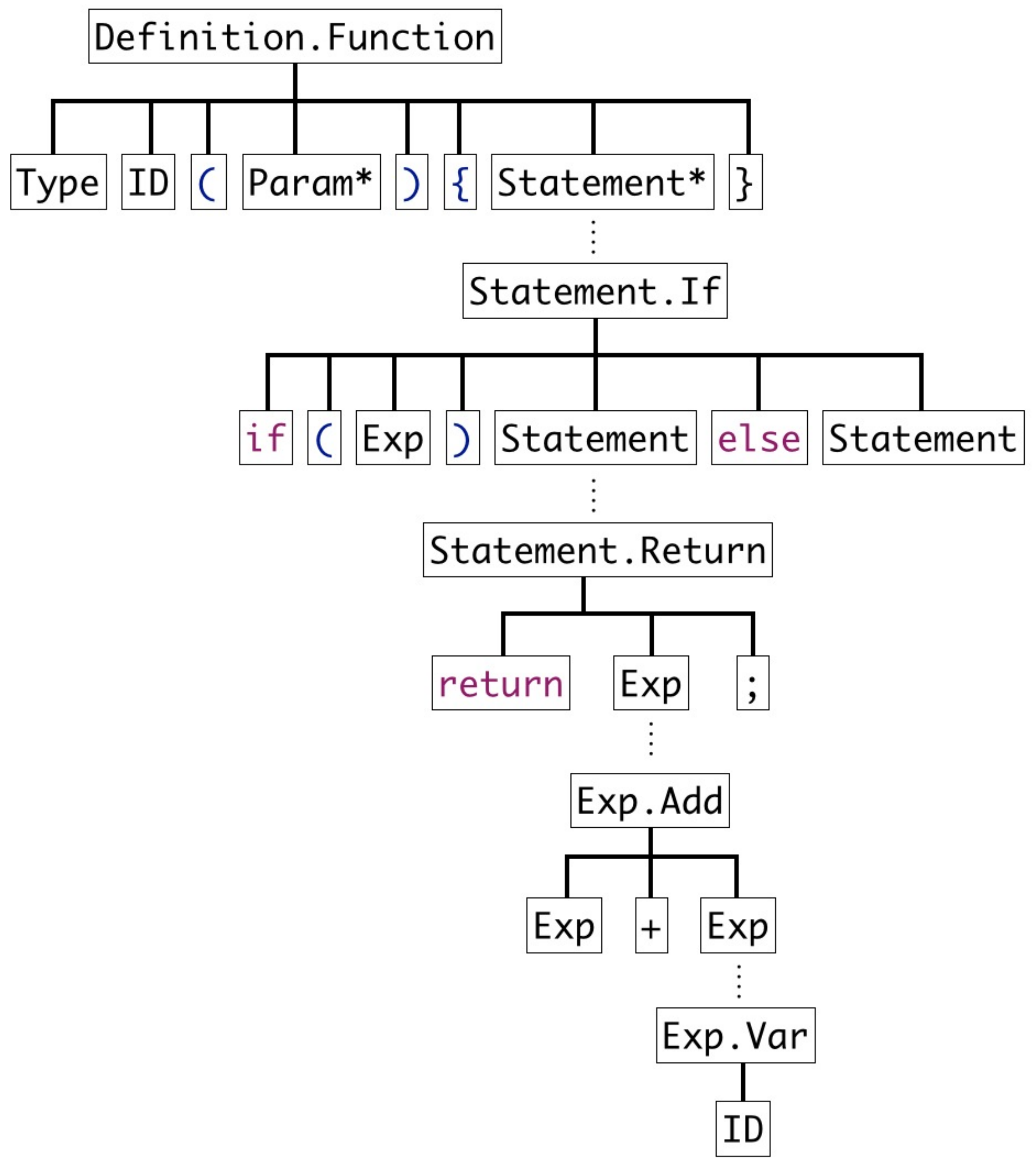
```
binding rules

  Param(t, name) :
    defines Variable name

  Var(name) :
    refers to Variable name

  Function(t, name, param*, s) :
    defines Function name
    scopes Variable, Function

  Call(name, exp*) :
    refers to Function name
```

- Incremental name resolution algorithm
- Name checks
- Reference resolution
- Semantic code completion

Semantics of Name Binding?

```
binding rules

  Param(t, name) :
    defines Variable name

  Var(name) :
    refers to Variable name

  Function(t, name, param*, s) :
    defines Function name
    scopes Variable, Function

  Call(name, exp*) :
    refers to Function name
```

Research: how to characterize correctness of the result of name resolution without appealing to the algorithm itself?



Declarative Syntax Definition = Specifying Tree Constructors

parse(pp(t)) = t

No need to understand how parse works!

Analogy: declarative semantics of syntax definition

Interpretation & Verification

Language Design
Syntax Definition
Name Binding
Type Constraints
Dynamic Semantics
Transform
Desktop — bash — 37×16
[08:48:06] ~/Desktop$ javac Fib.java
[08:48:10] ~/Desktop$ java Fib
Fib 6: 8
Fib 5: 8
[08:48:13] ~/Desktop$
Fib.java
public class Fib {
public static int calc(int n) {
if(n < 2)
return n;
else
return calc(n - 1) + calc(n - 2);
}
public static void main(String[] args
System.out.println("Fib 6: " + calc
System.out.println("Fib 5: " + calc
}
}
The Java™ Language Specification
Java SE 7 Edition
James Gosling
Bill Joy
Guy Steele
Gilad Bracha
Alex Buckley
2012-07-27
Describing the Semantics of Java and Proving Type Soundness
Sophia Drossopoulou and Susan Eisenbach
Department of Computing
Imperial College of Science, Technology and Medicine
1 Introduction

DynSem: Dynamic Semantics Specification

```
module semantics

rules

  E env |- Var(x) --> v
  where env[x] => T(e, env'),
        E env' |- e --> v

  E env |- Fun(Param(x, t), e) --> C(x, e, env)

  E env |- App(e1, e2) --> v
  where E env |- e1 --> C(x, e, env'),
        E {x |--> T(e2, env), env'} |- e --> v

  E env |- Fix(Param(x, t), e) --> v
  where
    E {x |--> T(Fix(Param(x,t),e),env), env} |- e --> v

  E env |- Let(x, t, e1, e2) --> v
  where E {x |--> T(e1, env), env} |- e2 --> v
```

```
rules

  Num(i) --> I(i)

  Ifz(e1, e2, e3) --> v
  where e1 --> I(i), i = 0, e2 --> v

  Ifz(e1, e2, e3) --> v
  where e1 --> I(i), i != 0, e3 --> v

  Add(e1, e2) --> I(addInt(i, j))
  where e1 --> I(i), e2 --> I(j)

  Sub(e1, e2) --> I(subInt(i, j))
  where e1 --> I(i), e2 --> I(j)

  Mul(e1, e2) --> I(mulInt(i, j))
  where e1 --> I(i), e2 --> I(j)
```

Implicitly-Modular Structural Operational Semantics (I-MSOS)*

```
rules

  E env |- Var(x) --> v
  where env[x] => T(e, env'),
        E env' |- e --> v

  Add(e1, e2) --> I(addInt(i, j))
  where e1 --> I(i),
        e2 --> I(j)
```

explicate

```
rules

  E env |- Var(x) --> v
  where env[x] => T(e, env'),
        E env' |- e --> v

  E env |- Add(e1, e2) --> I(addInt(i, j))
  where E env |- e1 --> I(i),
        E env |- e2 --> I(j)
```

* P. D. Mosses. Modular structural operational semantics. JLP, 60-61:195–228, 2004.
M. Churchill, P. D. Mosses, and P. Torrini. Reusable components of semantic specifications. In MODULARITY, April 2014.

Interpreter Generation

```
rules

  Ifz(e1, e2, e3) --> v
  where e1 --> I(i), i = 0, e2 --> v

  Ifz(e1, e2, e3) --> v
  where e1 --> I(i), i != 0, e3 --> v
```

explicate
& merge

```
rules

  E env |- Ifz(e1, e2, e3) --> v
  where E env |- e1 --> I(i),
    [i = 0, E env |- e2 --> v] +
    i != 0, E env |- e3 --> v]
```

```java
package org.metaborg.lang.pcf.interpreter.nodes;

public class Ifz_3_Node extends AbstractNode
                        implements I_Exp
{
  public I_Exp _1, _2, _3;

  @Override
  public Value evaluate(I_InterpreterFrame frame){
    I_InterpreterFrame env = frame;
    I_Exp e1 = this._1;
    I_Exp e2 = this._2;
    I_Exp e3 = this._3;
    Value v1 = e1.evaluate(env);
    if (v1 instanceof I_1_Node) {
      I_1_Node c_0 = (I_1_Node) v1;
      int i = c_0._1;
      if (i != 0) {
        return e3.evaluate(env);
      } else {
        if (i == 0) {
          return e2.evaluate(env);
        } else {
          throw new
          InterpreterException("Premise failed");
        }
      }
    } else {
      throw new
      InterpreterException("Premise failed");
    }
  }
  // constructor omitted
}
```

First Little (Big) Step: From PCF in Spoofax …

```
module PCF
sorts Exp Param Type
templates
  Exp.Var = [[ID]]
  Exp.App = [[Exp] [Exp]] {left}
  Exp.Fun = [
    fun [Param] (
         [Exp]
     )
  ]
  Exp.Fix = [
    fix [Param] (
        [Exp]
     )
  ]
  Exp.Let = [
    let [ID] : [Type] =
         [Exp]
     in [Exp]
  ]
  Exp.Num = [[INT]]
  Exp.Add = [[Exp] + [Exp]] {left}
  Exp.Sub = [[Exp] - [Exp]] {left}
  Exp.Mul = [[Exp] * [Exp]] {left}
  Exp     = [([Exp])] {bracket}
  Exp.Ifz = [
    ifz [Exp] then
      [Exp]
    else
      [Exp]
  ]
  Type.IntType = [int]
  Type.FunType = [[Type] -> [Type]]
  Param.Param = [[ID] : [Type]]

context-free priorities

  Exp.App > Exp.Mul > {left: Exp.Add Exp.Sub}
  > Exp.Ifz
```

```
module names

namespaces Variable

binding rules

  Var(x) :
    refers to Variable x

  Param(x, t) :
     defines Variable x of type t

  Fun(p, e) :
    scopes Variable

  Fix(p, e) :
    scopes Variable

  Let(x, t, e1, e2) :
    defines Variable x of type t in e2
```

```
module types
type rules

  Var(x) : t
  where definition of x : t

  Param(x, t) : t

  Fun(p, e) : FunType(tp, te)
  where p : tp and e : te

  App(e1, e2) : tr
  where e1 : FunType(tf, tr) and e2 : ta
    and tf == ta
        else error "type mismatch" on e2

  Fix(p, e) : tp
  where p : tp and e : te
    and tp == te
        else error "type mismatch" on p

  Let(x, tx, e1, e2) : t2
  where e2 : t2 and e1 : t1
    and t1 == tx
        else error "type mismatch" on e1

  Num(i) : IntType()

  Ifz(e1, e2, e3) : t2
  where e1 : IntType() and e2 : t2 and e3 : t3
    and t2 == t3
        else error "types not compatible" on e3

  e@Add(e1, e2) + e@Sub(e1, e2) + e@Mul(e1, e2) : IntType()
  where e1 : IntType()
        else error "Int type expected" on e
    and e2 : IntType()
        else error "Int type expected" on e
```

```
module semantics

rules

  E env |- Var(x) --> v
  where env[x] => T(e, env'),
        E env' |- e --> v

  E env |- Fun(Param(x, t), e) --> C(x, e, env)

  E env |- App(e1, e2) --> v
  where E env |- e1 --> C(x, e, env'),
        E {x |--> T(e2, env), env'} |- e --> v

  E env |- Fix(Param(x, t), e) --> v
  where
    E {x |--> T(Fix(Param(x,t),e),env), env} |- e --> v

  E env |- Let(x, t, e1, e2) --> v
  where E {x |--> T(e1, env), env} |- e2 --> v

rules

  Num(i) --> I(i)

  Ifz(e1, e2, e3) --> v
  where e1 --> I(i), i = 0, e2 --> v

  Ifz(e1, e2, e3) --> v
  where e1 --> I(i), i != 0, e3 --> v

  Add(e1, e2) --> I(addInt(i, j))
  where e1 --> I(i), e2 --> I(j)

  Sub(e1, e2) --> I(subInt(i, j))
  where e1 --> I(i), e2 --> I(j)

  Mul(e1, e2) --> I(mulInt(i, j))
  where e1 --> I(i), e2 --> I(j)
```

```
    [Exp]
  ]
  Type.IntType = [int]
  Type.FunType = [[Type] -> [Type]]
  Param.Param = [[ID] : [Type]]

context-free priorities

  Exp.App > Exp.Mul > {left: Exp.Add Exp.Sub}
  > Exp.Ifz
```

```
  defines Variable x of type t

Fun(p, e) :
  scopes Variable

Fix(p, e) :
  scopes Variable

Let(x, t, e1, e2) :
  defines Variable x of type t in e2
```

```
where e1 : IntType() and e2 : t2 and e3 : t3
  and t2 == t3
    else error "types not compatible" on e3

e@Add(e1, e2) + e@Sub(e1, e2) + e@Mul(e1, e2) : IntType()
where e1 : IntType()
    else error "Int type expected" on e
  and e2 : IntType()
    else error "Int type expected" on e
```

```
where e1 --> I(i), i != 0, e3 --> v

Add(e1, e2) --> I(addInt(i, j))
where e1 --> I(i), e2 --> I(j)

Sub(e1, e2) --> I(subInt(i, j))
where e1 --> I(i), e2 --> I(j)

Mul(e1, e2) --> I(mulInt(i, j))
where e1 --> I(i), e2 --> I(j)
```

```
Inductive sorts  : Set :=
| Param_S
| ID_S
| INT_S
| Exp_S
| Type_S
.

Parameter Ident : Set.

Definition sort   :=
sorts
.

Definition Ident_Sort   :=
ID_S
.

Inductive Constructors   :=
| INTC (n: nat)
| VarC
| FunC
| FixC
| AppC
| LetC
| ParamC
| NumC
| AddC
| SubC
| MulC
| DivC
| IfzC
| IntTypeC
| FunTypeC
.

Definition constructors   :=
Constructors
.

Fixpoint
  get_sig (x: constructors)  : list sort * sort :=
    match x    with
      | INTC n => ([],INT_S)
      | VarC => ([ID_S],Exp_S)
      | FunC => ([Param_S;Exp_S],Exp_S)
      | FixC => ([Param_S;Exp_S],Exp_S)
      | AppC => ([Exp_S;Exp_S],Exp_S)
      | LetC => ([ID_S;Type_S;Exp_S;Exp_S],Exp_S)
      | ParamC => ([ID_S;Type_S],Param_S)
      | NumC => ([INT_S],Exp_S)
      | AddC => ([Exp_S;Exp_S],Exp_S)
      | SubC => ([Exp_S;Exp_S],Exp_S)
      | MulC => ([Exp_S;Exp_S],Exp_S)
      | DivC => ([Exp_S;Exp_S],Exp_S)
      | IfzC => ([Exp_S;Exp_S;Exp_S],Exp_S)
      | IntTypeC => ([],Type_S)
      | FunTypeC => ([Type_S;Type_S],Type_S)
      end.
```

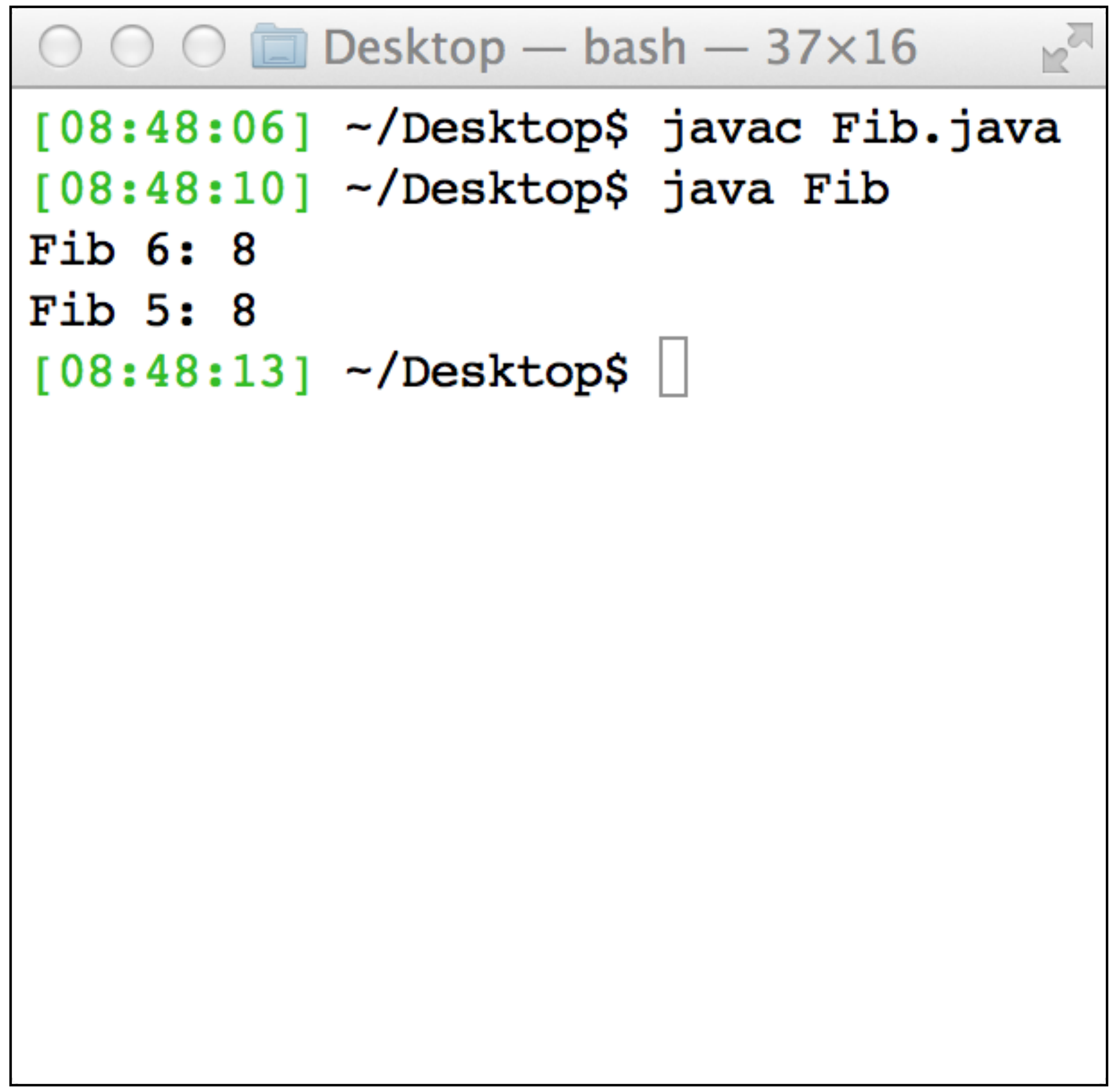
```
Inductive ID_NS  : Set :=
| VariableNS
.

Definition NS   :=
ID_NS
.

Inductive scopesR  : term -> NS -> Prop :=
| Fun_scopes_Variable p e k0 : scopesR (Co FunC [p;e] k0) VariableNS
| Fix_scopes_Variable p e k0 : scopesR (Co FixC [p;e] k0) VariableNS
.

Definition scopes_R   :=
scopesR
.

Inductive definesR  : term -> Ident -> NS -> key -> Prop :=
| Param_defines_Variable x k1 t k0 : definesR (Co ParamC [Id x k1;t] k0) x VariableNS k1
.

Definition defines_R   :=
definesR
.

Inductive refers_toR  : term -> Ident -> NS -> key -> Prop :=
| Var_refers_to_Variable x k1 k0 : refers_toR (Co VarC [Id x k1] k0) x VariableNS k1
.

Definition refers_to_R   :=
refers_toR
.

Inductive typed_definesR  : term -> Ident -> NS -> term -> key -> Prop :=
| Param_typed_defines_Variable x t k1 t k0 : typed_definesR (Co ParamC [Id x k1;t] k0) x VariableNS t k1
.

Definition typed_defines_R   :=
typed_definesR
.
```

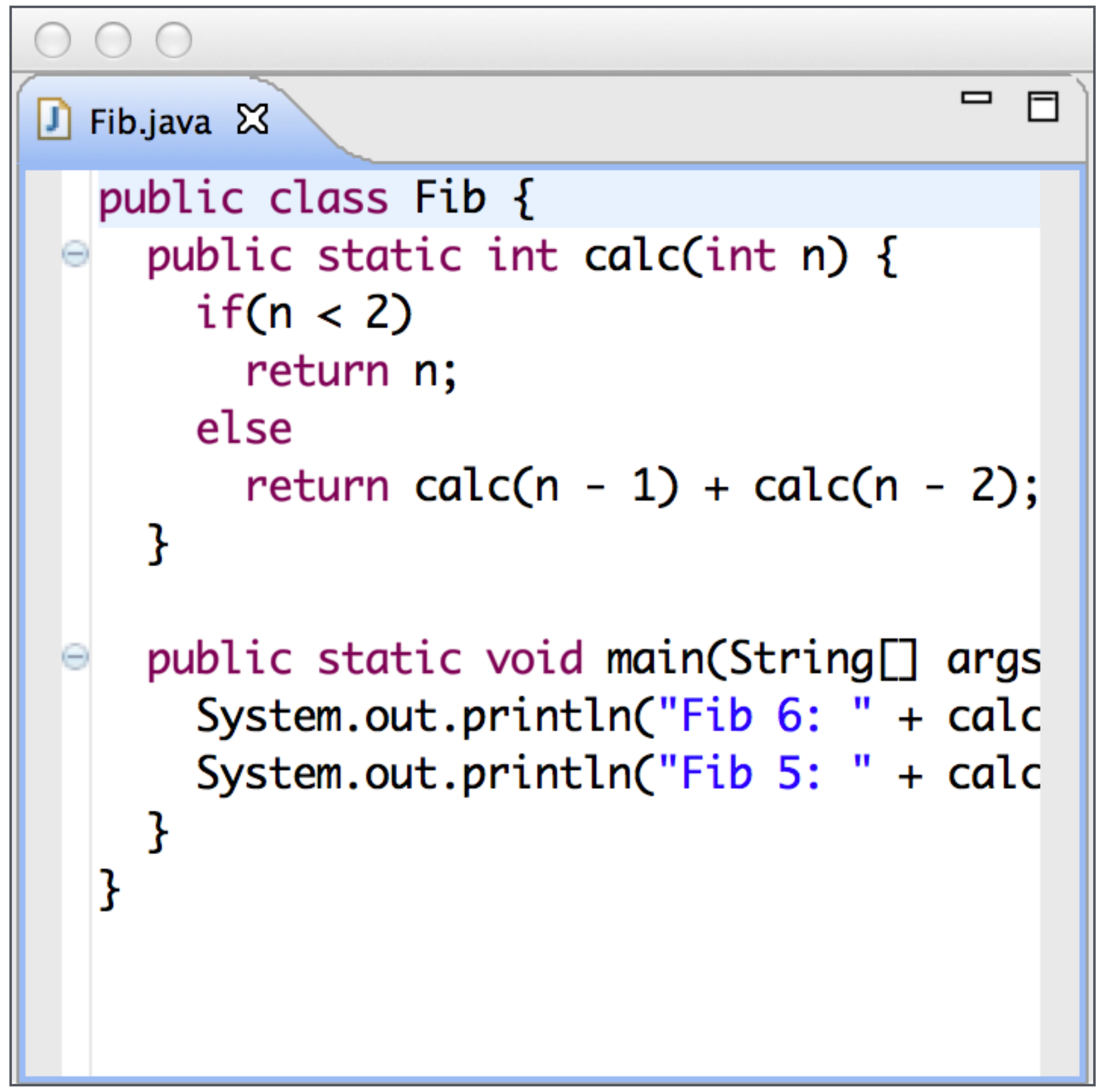
```
Inductive has_type (C: Context) : term -> term -> Prop :=
| VarC_ht ns k0 t x k1 : lookup C x ns k0 t -> has_type C (Co VarC [Id x k0] k1) t
| ParamC_ht x t k0 : has_type C (Co ParamC [x;t] k0) t
| FunC_ht k0 t_p t_e p e k1 : has_type C p t_p -> has_type C e t_e -> has_type C (Co FunC [p;e] k1) (Co FunTypeC [t_p;t_e] k0)
| FixC_ht t_p t_e p e k0 : has_type C p t_p -> has_type C e t_e -> (t_p = t_e) -> has_type C (Co FixC [p;e] k0) t_p
| AppC_ht t_r k0 t_f t_a e1 e2 k1 : has_type C e1 (Co FunTypeC [t_f;t_r] k0) -> has_type C e2 t_a -> (t_f = t_a) -> has_type C (Co AppC [e1;e2] k1) t_r
| LetC_ht t2 t1 x t_x e1 e2 k0 : has_type C e2 t2 -> has_type C e1 t1 -> (t1 = t_x) -> has_type C (Co LetC [x;t_x;e1;e2] k0) t2
| NumC_ht k0 i k1 : has_type C (Co NumC [i] k1) (Co IntTypeC [] k0)
| IfzC_ht k0 t2 t3 e1 e2 e3 k1 : has_type C e1 (Co IntTypeC [] k0) -> has_type C e2 t2 -> has_type C e3 t3 -> (t2 = t3) -> has_type C (Co IfzC [e1;e2;e3] k1) t2
| AddC_ht k2 k0 k1 e1 e2 k3 : has_type C e1 (Co IntTypeC [] k0) -> has_type C e2 (Co IntTypeC [] k1) -> has_type C (Co AddC [e1;e2] k3) (Co IntTypeC [] k2)
| SubC_ht k2 k0 k1 e1 e2 k3 : has_type C e1 (Co IntTypeC [] k0) -> has_type C e2 (Co IntTypeC [] k1) -> has_type C (Co SubC [e1;e2] k3) (Co IntTypeC [] k2)
| MulC_ht k2 k0 k1 e1 e2 k3 : has_type C e1 (Co IntTypeC [] k0) -> has_type C e2 (Co IntTypeC [] k1) -> has_type C (Co MulC [e1;e2] k3) (Co IntTypeC [] k2)
| HT_eq e ty1 ty2 (hty1: has_type C e ty1) (tyeq: term_eq ty1 ty2) : has_type C e ty2
.
```

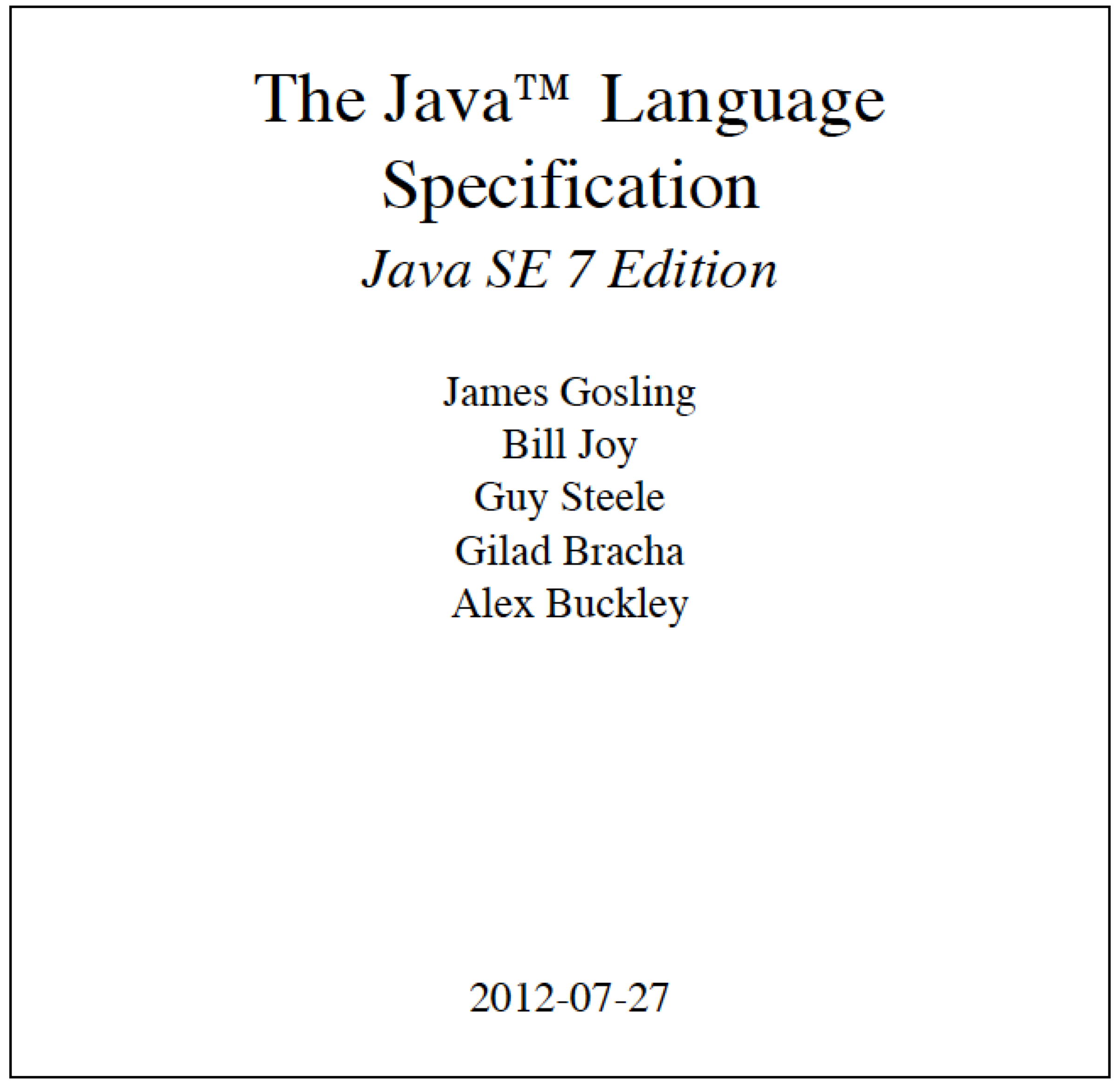
```
Inductive semantics_cbn  : Env -> term -> value -> Prop :=
| Var0C_sem env' e env x k0 v : get_env x env e env' -> semantics_cbn env' e v -> semantics_cbn env (Co VarC [x] k0) v
| Fun0C_sem t k1 k0 x e env : semantics_cbn env (Co FunC [Co ParamC [x;t] k1;e] k0) (Clos x e env)
| Fix0C_sem k1 k0 env x t k3 e k2 v : semantics_cbn { x |--> (Co FixC [Co ParamC [x;t] k1;e] k0,env), env } e v -> semantics_cbn env (Co FixC [Co ParamC [x;t] k3;e] k2) v
| App0C_sem env' x e env e1 e2 k0 v : semantics_cbn env e1 (Clos x e env') -> semantics_cbn { x |--> (e2,env), env' } e v -> semantics_cbn env (Co AppC [e1;e2] k0) v
| Let0C_sem env x t e1 e2 k0 v : semantics_cbn { x |--> (e1,env), env } e2 v -> semantics_cbn env (Co LetC [x;t;e1;e2] k0) v
| Num0C_sem env k0 i : semantics_cbn env (Co NumC [i] k0) (Natval i)
| Ifz0C_sem i env e1 e2 e3 k0 v : semantics_cbn env e1 (Natval i) -> (i = 0) -> semantics_cbn env e2 v -> semantics_cbn env (Co IfzC [e1;e2;e3] k0) v
| Ifz1C_sem i env e1 e2 e3 k0 v : semantics_cbn env e1 (Natval i) -> (i <> 0) -> semantics_cbn env e3 v -> semantics_cbn env (Co IfzC [e1;e2;e3] k0) v
| Add0C_sem env e1 e2 k0 i j : semantics_cbn env e1 (Natval i) -> semantics_cbn env e2 (Natval j) -> semantics_cbn env (Co AddC [e1;e2] k0) (plus i j)
| Sub0C_sem env e1 e2 k0 i j : semantics_cbn env e1 (Natval i) -> semantics_cbn env e2 (Natval j) -> semantics_cbn env (Co SubC [e1;e2] k0) (minus i j)
| Mul0C_sem env e1 e2 k0 i j : semantics_cbn env e1 (Natval i) -> semantics_cbn env e2 (Natval j) -> semantics_cbn env (Co MulC [e1;e2] k0) (mult i j)
.
```

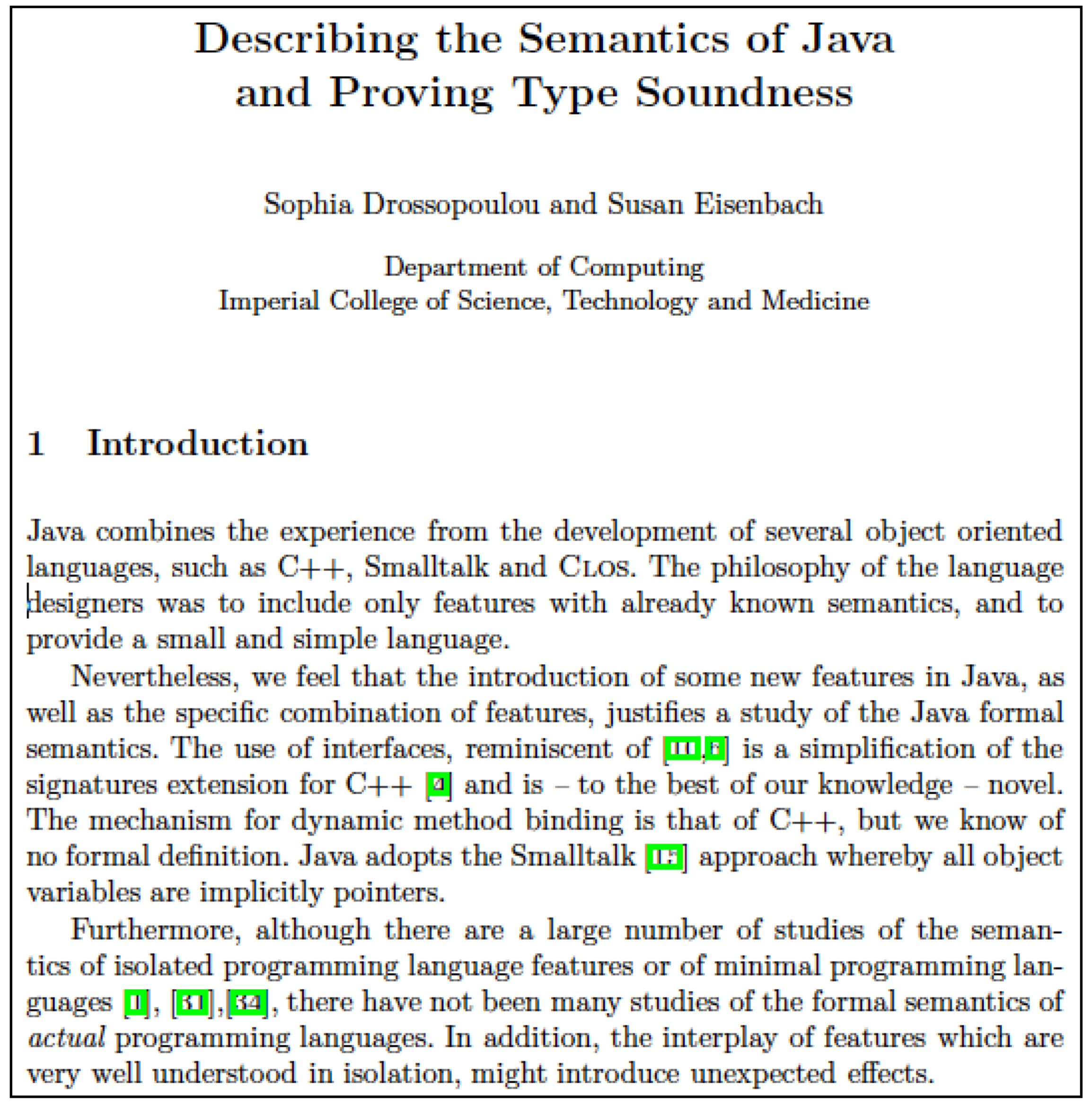
… to PCF in Coq (+ manual proof of type preservation)

Summary









parser
type checker
code generator
interpreter

parser
error recovery
syntax highlighting
outline
code completion
navigation
type checker
debugger

syntax definition
static semantics
dynamic semantics

abstract syntax
type system
operational semantics
type soundness proof


Declarative Multi-Purpose Language Definition
Syntax Definition
Name Binding
Type Constraints
Dynamic Semantics
Transform
Desktop — bash — 37×16
[08:48:06] ~/Desktop$ javac Fib.java
[08:48:10] ~/Desktop$ java Fib
Fib 6: 8
Fib 5: 8
[08:48:13] ~/Desktop$
Fib.java
public class Fib {
public static int calc(int n) {
if(n < 2)
return n;
else
return calc(n - 1) + calc(n - 2);
}
public static void main(String[] args
System.out.println("Fib 6: " + calc
System.out.println("Fib 5: " + calc
}
}
The Java™ Language Specification
Java SE 7 Edition
James Gosling
Bill Joy
Guy Steele
Gilad Bracha
Alex Buckley
2012-07-27
Describing the Semantics of Java and Proving Type Soundness
Sophia Drossopoulou and Susan Eisenbach
Department of Computing
Imperial College of Science, Technology and Medicine
1 Introduction
Java combines the experience from the development of several object oriented languages, such as C++, Smalltalk and Clos. The philosophy of the language designers was to include only features with already known semantics, and to provide a small and simple language.
Nevertheless, we feel that the introduction of some new features in Java, as well as the specific combination of features, justifies a study of the Java formal semantics. The use of interfaces, reminiscent of is a simplification of the signatures extension for C++ and is – to the best of our knowledge – novel. The mechanism for dynamic method binding is that of C++, but we know of no formal definition. Java adopts the Smalltalk approach whereby all object variables are implicitly pointers.
Furthermore, although there are a large number of studies of the semantics of isolated programming language features or of minimal programming languages, there have not been many studies of the formal semantics of actual programming languages. In addition, the interplay of features which are very well understood in isolation, might introduce unexpected effects.

Declarative Multi-Purpose Language Definition
SDF3: Syntax Definition
NaBL: Name Binding
TS: Type Constraints
DynSem: Dynamic Semantics
Stratego: Transform
Desktop — bash — 37×16
[08:48:06] ~/Desktop$ javac Fib.java
[08:48:10] ~/Desktop$ java Fib
Fib 6: 8
Fib 5: 8
[08:48:13] ~/Desktop$
Fib.java
public class Fib {
public static int calc(int n) {
if(n < 2)
return n;
else
return calc(n - 1) + calc(n - 2);
}
public static void main(String[] args
System.out.println("Fib 6: " + calc
System.out.println("Fib 5: " + calc
}
}
The Java™ Language Specification
Java SE 7 Edition
James Gosling
Bill Joy
Guy Steele
Gilad Bracha
Alex Buckley
2012-07-27
Describing the Semantics of Java and Proving Type Soundness
Sophia Drossopoulou and Susan Eisenbach
Department of Computing
Imperial College of Science, Technology and Medicine
1 Introduction

Declarative Multi-Purpose Language Definition

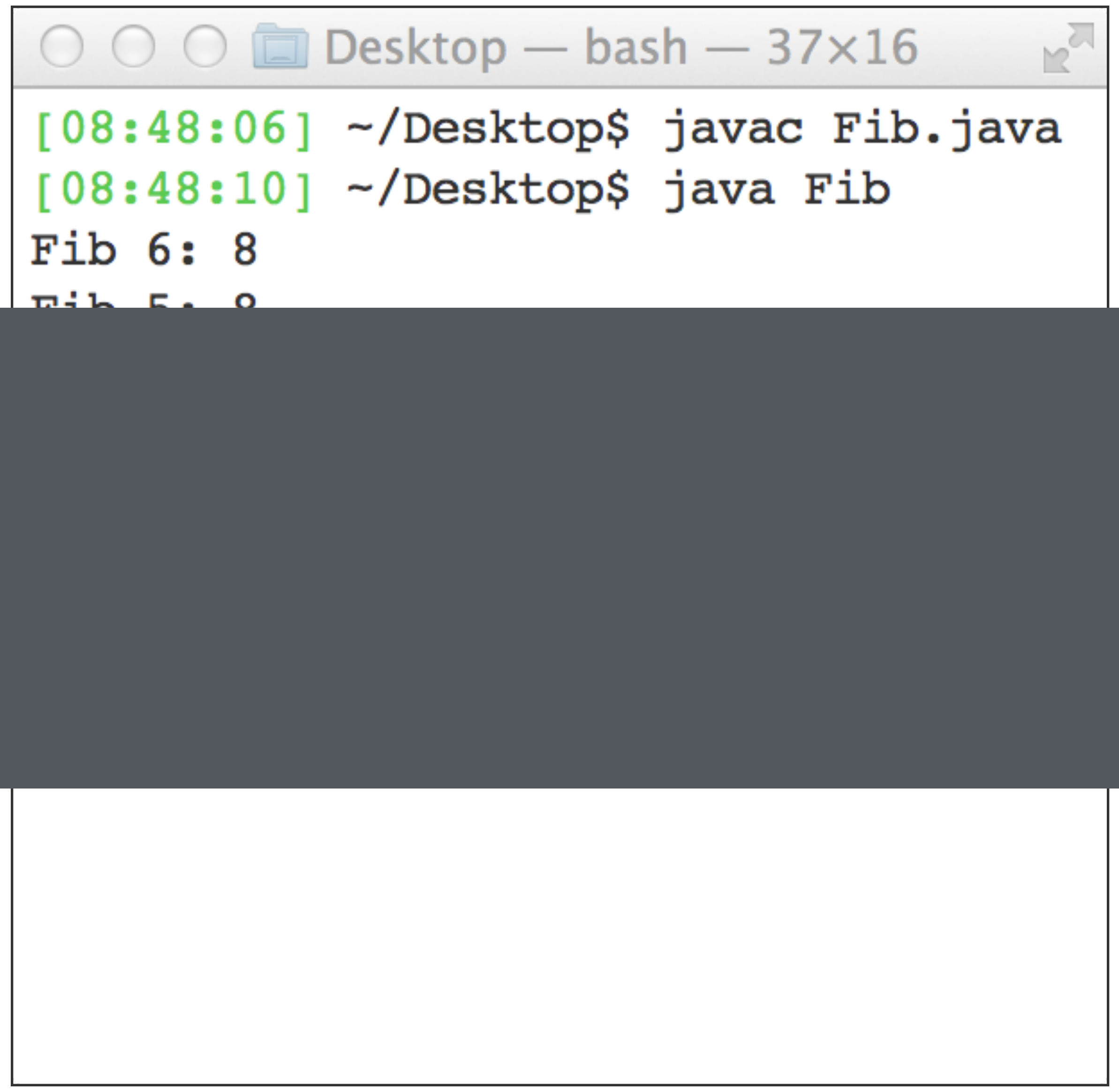
SDF3: Syntax Definition

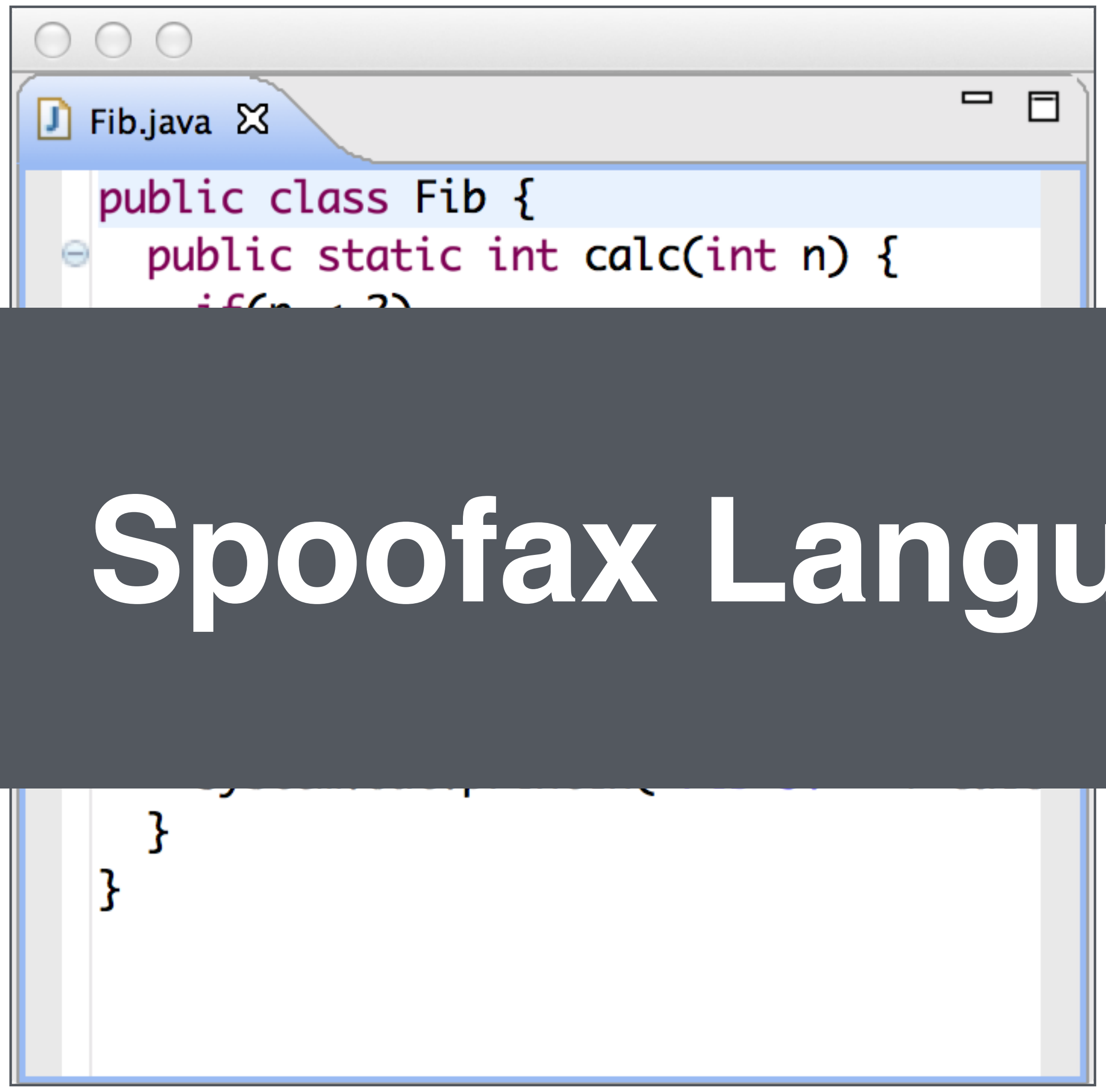
NaBL: Name Binding

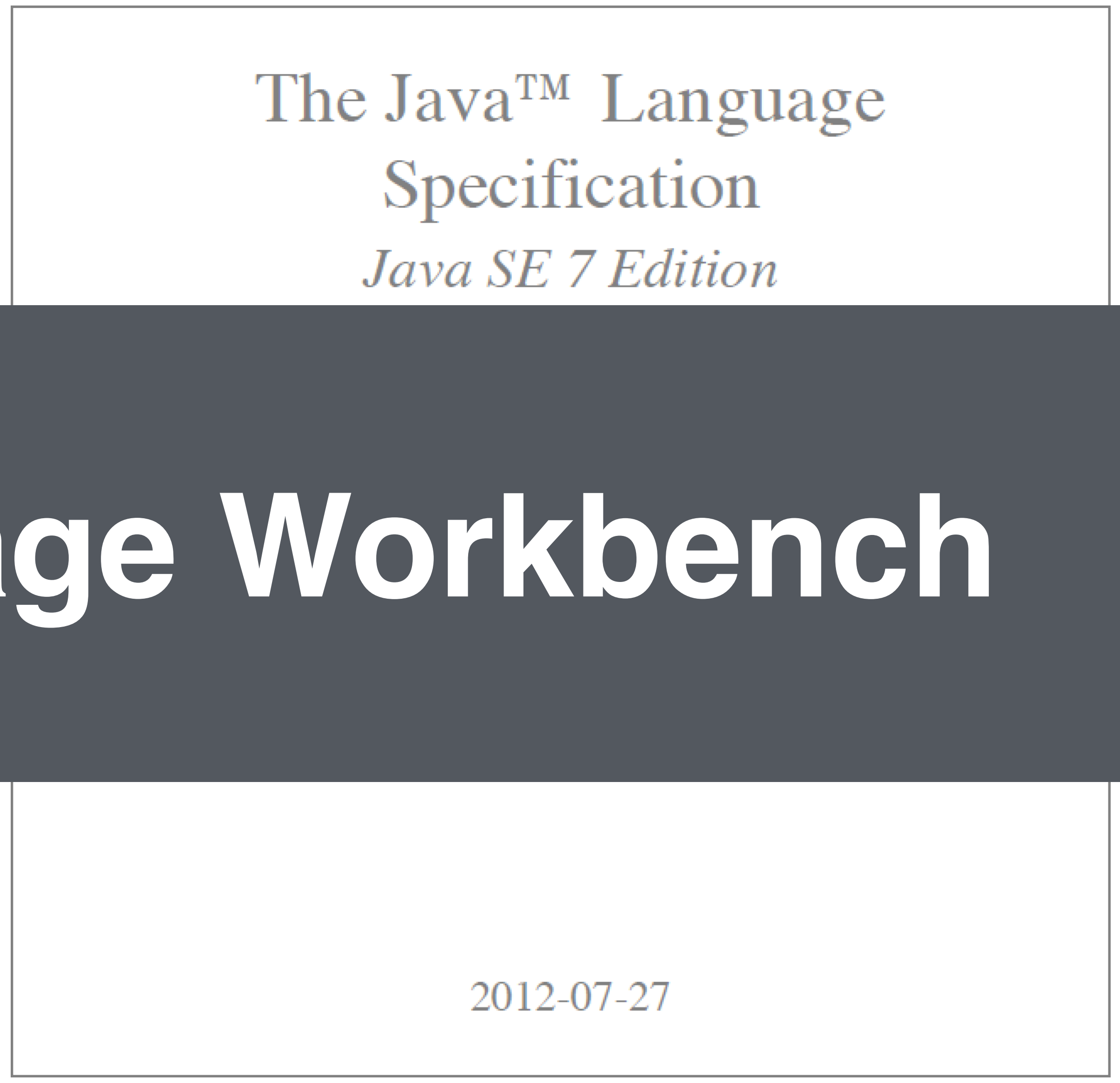
TS: Type Constraints

DynSem: Dynamic Semantics

Stratego: Transform







Spoofax Language Workbench