Java: An Operational Semantics

Gaurav S. Kc

B. Eng. Project

Department of Computing

Semantics of Java -- why?

- Semantics:
  - *Assignment of meanings to programs*
- Java:
  - *A simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language.*

Why? ... contd.

- Better “mental model” of language
- Acceptance
  - reliability
  - expected behaviour
- Java : Security v.s. functionality
- Widespread use
- Reasoning: towards a compromise

Project Goals

- Semantics extension
- Better understanding of Java
- Research based -- no implementation

Break-down into parts

- Existing features:
  - inheritance
  - instance variables
  - overloading and overriding
- Additions:
  - access modifiers
  - final, static, abstract
  - constructors

An example in the new syntax

```
abstract class Animal extends Object {
    int age;
    Animal() {
        super();}
    int getAge() {…} }

public final class Dog extends Animal {
    final int legs = 4;
    final static boolean hasTail = yes;
    Dog() {
        this(“Laika”); }
    Dog(String s) {
        super(); ... }
    int getAge(String name) {...} }
```

Access Modifiers

- Public
- Protected
- Private
- [default]

- No packages

Semantics extension ...

- Accessibility checks

Final and Abstract modifier

- Classes
  - sub-classing not permitted
  - instantiating not permitted
- Fields
  - Constant behaviour?

Semantics extension ...

- Well-formedness
- Constructor invocation
- Assignment

Static modifier

- Fields that don’t belong to objects
- Class and interface fields
- State extension:
  - Class/interface entries
  - References

Semantics extension ...

- Runtime checks
- Class or interface v.s. object

Constructor

- A *constructor* is used in the creation of an object that is an instance of a class.
- It is the basis with which the run-time system allocates space from memory to objects during execution.
  - Instance fields & [inherited] instance fields
  - OutOfMemory exception
  - Static initialisation

Syntax of a Constructor

- Explicit constructor invocation
  - this();
    - same class
  - super();
    - parent class
    - static initialisation
- Statements

```
public class C extends B  {
    int x = 5;
    public C (int n) {
            this(true);
            print(n);
    }
    private C (boolean b)  {
            super();
            if (b) ...
    }
}
```

Constructor execution

new C(true);

[boolean] C(true), σ

[] B(); x=5; if (b)...; σ

$$FirstFitConstr(\mathtt{P},\mathtt{C},\mathtt{AT}) = \mathtt{constr}$$

$$\mathtt{constr} = \mathtt{cMod}\ \mathtt{C}(\mathtt{T}_1\ \mathtt{p}_1,\ldots,\mathtt{T}_n\ \mathtt{p}_n)\ \mathbf{throws}\ \mathtt{E}_1,\ldots,\mathtt{E}_s\ \{\mathtt{constrBody}\}$$

$$\mathtt{constrBody} = \mathtt{constrCall};\ \ \mathtt{stmts}$$

$$cc = \mathtt{constrCall}\ [\mathtt{C}/\mathbf{this},\ SuperC(\mathtt{P},\mathtt{C})/\mathbf{super}]$$

$$\mathbf{super} \in \mathtt{constrCall}\ \text{and}\ \mathtt{I}_{\mathtt{Expr}} = \mathtt{InitExpr}(\mathtt{P},\mathtt{C})$$
$$\text{or}\ \mathbf{this} \in \mathtt{constrCall}\ \text{and}\ \mathtt{I}_{\mathtt{Expr}} = \epsilon$$

$$\overline{[\mathtt{AT}]\mathtt{C}(\mathtt{val}_1,\ldots,\mathtt{val}_n),\sigma \ \underset{\mathtt{P}}{\rightsquigarrow}\ (\mathtt{cc};\ \ \mathtt{I}_{\mathtt{Expr}};\ \ \mathtt{stmts}),\sigma}$$

Other research work on Java

- Within the Department of Computing:
  - Exceptions — SLURP@DoC
  - Concurrency — SLURP@DoC
  - Binary Compatibility — SLURP@DoC

- In other research institutions:
  - Generics — PLT@Rice
  - Security Issues — SIP@Princeton

Other research, ... contd.

- A comparison perhaps?
  - Different aspects of Java
  - Post-grad & post-doc work

Conclusions

- Boring? Certainly not!
- Acquired skills
- Taste of pure research
- Lots of non-trivial work
- State of the art technology
- Continued research in Java Semantics

- Improved know-how of the Java system

Acknowledgements

- Krysia Broda
- Sophia Drossopoulou
- Susan Eisenbach
- Tanya Valkevych