# Java Classloading

Summer School on Generative and Transformational Techniques in Software Engineering, 2005

# Outline

1. Overview

2. Motivation

# An Overview of the SSP and Class Loading

- The Sandia Secure Processor (SSP) is a hardware implementation of a significant subset of the Java Virtual Machine
- Class loading for the SSP is performed statically (prior to runtime)
  - **Class File:** a representation of a Java class (including all fields and methods declared, the class name, the parent class, and a constant pool)
  - **Class Loading:** act of resolving symbolic references within a Class File while preserving the Class File structure

# An Overview of the SSP and Class Loading

- The Sandia Secure Processor (SSP) is a hardware implementation of a significant subset of the Java Virtual Machine
- Class loading for the SSP is performed statically (prior to runtime)
  - **Class File:** a representation of a Java class (including all fields and methods declared, the class name, the parent class, and a constant pool)
  - **Class Loading:** act of resolving symbolic references within a Class File while preserving the Class File structure

# An Overview of the SSP and Class Loading

- The Sandia Secure Processor (SSP) is a hardware implementation of a significant subset of the Java Virtual Machine
- Class loading for the SSP is performed statically (prior to runtime)
  - **Class File:** a representation of a Java class (including all fields and methods declared, the class name, the parent class, and a constant pool)
  - **Class Loading:** act of resolving symbolic references within a Class File while preserving the Class File structure

# An Overview of the SSP and Class Loading

- The Sandia Secure Processor (SSP) is a hardware implementation of a significant subset of the Java Virtual Machine
- Class loading for the SSP is performed statically (prior to runtime)
  - **Class File:** a representation of a Java class (including all fields and methods declared, the class name, the parent class, and a constant pool)
  - **Class Loading:** act of resolving symbolic references within a Class File while preserving the Class File structure

# An Overview of the Constant Pool

- The Constant Pool (CP) is an implicitly indexed list of constants that correspond to symbolic references in a Class File, these constant include two types:
    - Simple constant values (ie. constant numeric values and utf8 strings)
    - Complex constant values consisting of symbolic references (indexes) into the constant pool (ie. field, method, and class references)
- Symbolic resolution of a Class File will replace symbolic references (indexes) with the data they reference.

# An Overview of the Constant Pool

- The Constant Pool (CP) is an implicitly indexed list of constants that correspond to symbolic references in a Class File, these constant include two types:
  - Simple constant values (ie. constant numeric values and utf8 strings)
  - Complex constant values consisting of symbolic references (indexes) into the constant pool (ie. field, method, and class references)
- Symbolic resolution of a Class File will replace symbolic references (indexes) with the data they reference.

# An Overview of the Constant Pool

- The Constant Pool (CP) is an implicitly indexed list of constants that correspond to symbolic references in a Class File, these constant include two types:
    - Simple constant values (ie. constant numeric values and utf8 strings)
    - Complex constant values consisting of symbolic references (indexes) into the constant pool (ie. field, method, and class references)
- Symbolic resolution of a Class File will replace symbolic references (indexes) with the data they reference.

# An Overview of the Constant Pool

- The Constant Pool (CP) is an implicitly indexed list of constants that correspond to symbolic references in a Class File, these constant include two types:
  - Simple constant values (ie. constant numeric values and utf8 strings)
  - Complex constant values consisting of symbolic references (indexes) into the constant pool (ie. field, method, and class references)
- Symbolic resolution of a Class File will replace symbolic references (indexes) with the data they reference.

# Fragment Goal

## Goal

*To further resolve all symbolic references to fields so that they satisfy the **Static Binding Property***

## Definition

*The **Static Binding Property** states that the class component of a symbolic field reference corresponds to the class that declared the field*

# Fragment Goal

### Goal

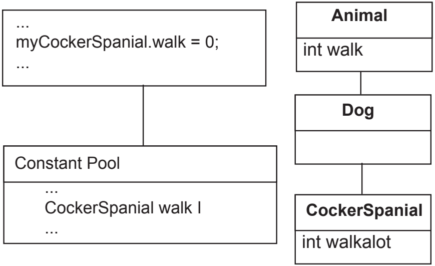
*To further resolve all symbolic references to fields so that they satisfy the* ***Static Binding Property***

### Definition

*The* ***Static Binding Property*** *states that the class component of a symbolic field reference corresponds to the class that declared the field*

# Static Binding Problem

- The **Static Binding Property** is not universally true for all symbolic field references because of inheritance



# Source

```
...
1  java/lang/Object <init> ()V
2  CockerSpaniel
3  CockerSpaniel <init> ()V
4  bark
5  one step
6  CockerSpaniel walk I
7  two steps
8  CockerSpaniel walkalot I
...
```

# Target

```
...
1  java/lang/Object <init> ()V
2  CockerSpaniel
3  CockerSpaniel <init> ()V
4  bark
5  one step
6  Animal walk I
7  two steps
8  CockerSpaniel walkalot I
...
```

### Assumption

*At this point in the execution, basic symbolic resolution has been completed.*

### Assumption

*Partial Ordering.*
*At this point in the execution, all Class Files of the program have been ordered in a list such that for any two classes A and B where B* $\succ^*$ *A, B appears in the list after A.*

## Law

*Static Binding Property.*
*For any symbolic field reference "A x T" (where T stands for the type of x), if a variable x of type T is declared in class A then "A x T" is a static binding.*

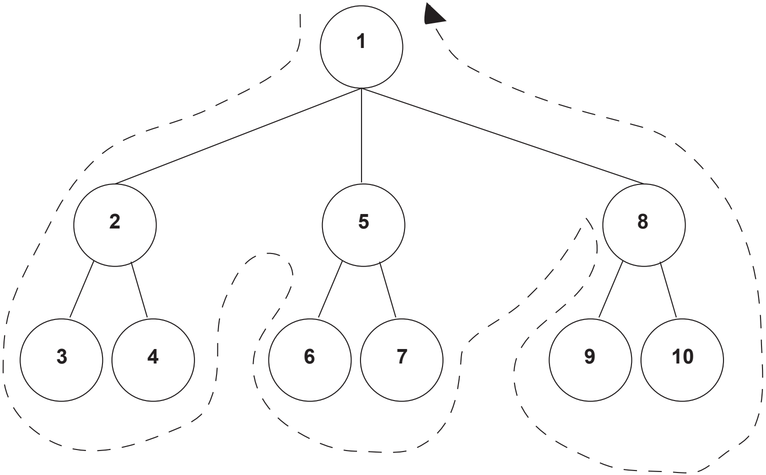
## Law

*Lifting through Hierarchy.*
*For any symbolic field reference "B x T" that is not a static binding, if B $\succ$ A then check if "A x T" is a static binding.*

# A Class File Grammar Fragment

```
...
classfile               ::=   cp this_class super_class fields methods
this_class              ::=   class
super_class             ::=   class
constant_fieldref_info  ::=   class name_and_type
field_info              ::=   access_flags name descriptor
class                   ::=   data
name_and_type           ::=   data
name                    ::=   data
descriptor              ::=   data
data                    ::=   index | utf8 | name descriptor
...
```

# TDL Traversal



# Transformations Yielding Static Binding

**def TDL s =** s $<$; all_thread_left(TDL$\{$s$\}$)

**def rcond_tdl s =** rcond( s, rcond_thread_left(rcond_tdl$\{$ s $\}$ ) )

**x_res:** app$_0$ $\longrightarrow$ TDL$\{$rcond_tdl$\{$sbind$\}$[app$_0$] $\}$(app$_0$)

**sbind:** classfile$\llbracket$ cp$_1$ class$_{this}$ class$_{super}$ fields$_1$ methods$_1$ $\rrbracket$
$\longrightarrow$
(hide(lift[class$_{this}$][class$_{super}$])
+>
rcond_tdl$\{$collect_decs[class$_{this}$]$\}$[fields$_1$]

**lift:** $\text{class}_{\mathit{this}} \longrightarrow$

$\text{class}_{\mathit{super}} \longrightarrow$

$\text{constant\_fieldref\_info}\llbracket\ \text{class}_{\mathit{this}}\ \text{name}_1\ \text{descriptor}_1\ \rrbracket$

$\longrightarrow$

$\text{constant\_fieldref\_info}\llbracket\ \text{class}_{\mathit{super}}\ \text{name}_1\ \text{descriptor}_1\ \rrbracket$

**collect_decs:** $\text{class}_{\mathit{this}} \longrightarrow$

$\text{field\_info}\llbracket\ \text{access\_flags}_1\ \text{name}_1\ \text{descriptor}_1\ \rrbracket \longrightarrow$

$\text{constant\_fieldref\_info}\llbracket\ \text{class}_{\mathit{this}}\ \text{name}_1\ \text{descriptor}_1\ \rrbracket$

$\longrightarrow$

$\text{constant\_fieldref\_info}\llbracket\ \text{class}_{\mathit{this}}\ \text{name}_1\ \text{descriptor}_1\ \rrbracket$

# For Further Reading I

- V. L. Winter and M. Subramaniam
  Dynamic Strategies, Transient Strategies, and the Distributed Data Problem.
  *Science of Computer Programming (Special Issue on Program Transformation)*, 52:165–212, Elsevier, 2004.

- V. L. Winter
  Strategy Construction in the Higher-Order Framework of TL.
  *Electronic Notes in Theoretical Computer Science (ENTCS)*, 124(1), 2004