

Bytecode Support for the Universe Type System

SCT Semester Project

Alex Suzuki
ETH Zürich

Summary

- Extensions to the MultiJava compiler to emit type information used in the Universe Type System to the classfile
- Two approaches were implemented
 - Attributes
 - Annotations (new in J2SE 5.0)

Overview

- The Problem
- The Java Classfile format
 - Attributes
 - Annotations
- Implementation
- Conclusion/Future Work

The Problem

- Until now, the compiler would parse and typecheck Java files with universe keywords (`peer`, `rep`, etc.)...
- ... but **not** emit the universe type information into the resulting classfile.
 - Universe type modifiers and method purity were lost.

The Problem (cont.)

- Result: Sourcecode needed all classes
- Until now: for binary classes...
 - ... all references were implicitly **peer**
 - ... all methods were non-pure

What we want

- Store universe type information in class file
 - Allows compiler to typecheck classes that are only available in form of a .class file
- Store other useful information:
 - availability of run-time checks
 - version of the encoding used (extensibility)

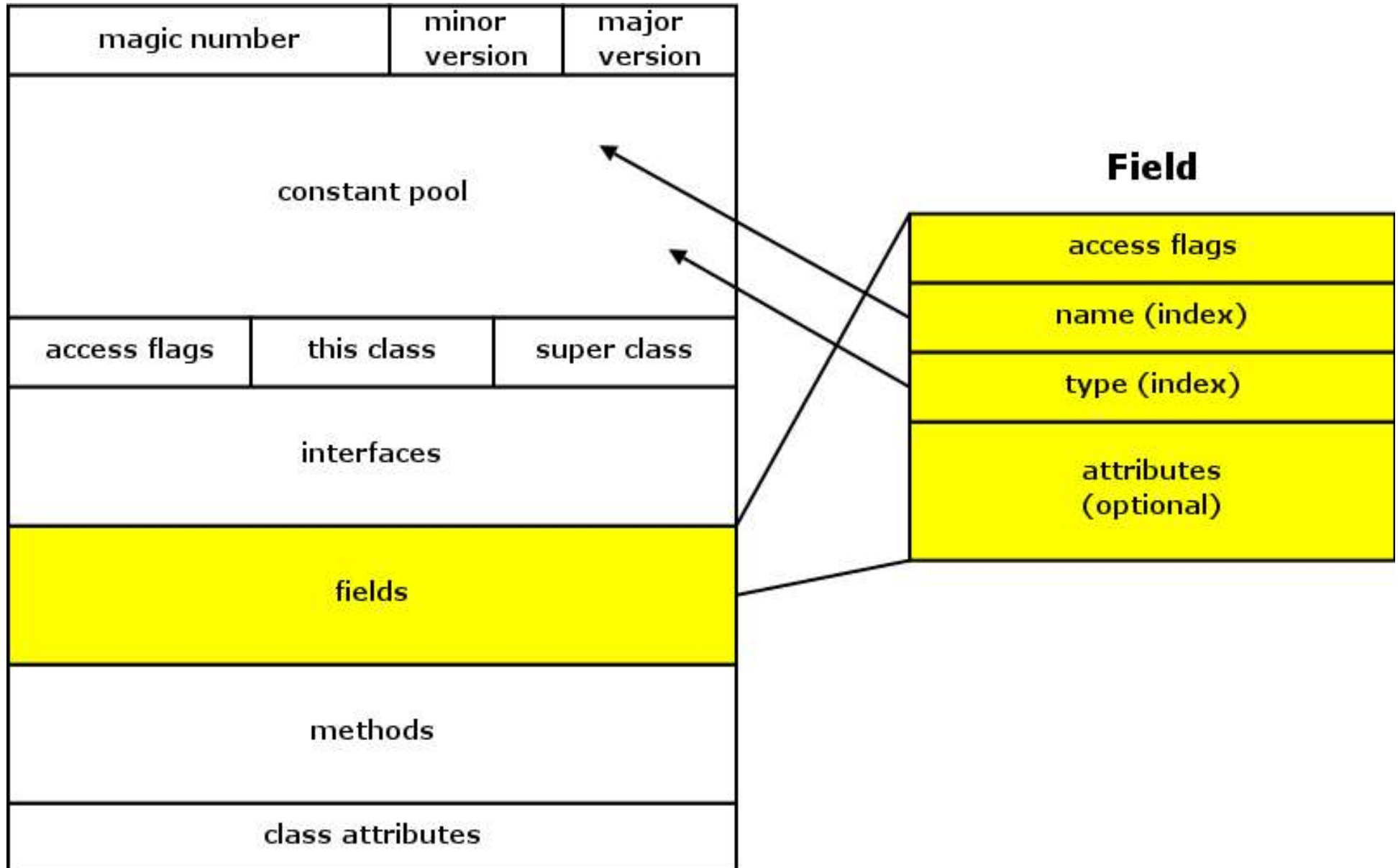
The Java Classfile Format

magic number	minor version	major version
constant pool		
access flags	this class	super class
interfaces		
fields		
methods		
class attributes		

- Binary file format
- Contains all information associated with a class
- Format dictated by Java VM Specification
- VM must reject non-conformant classes

0xCAFEBABE

A closer look...



Attributes

Field

access flags
name (index)
type (index)
attribute_1
...
attribute_n

- Can be used to add arbitrary data to program elements
- Named, untyped binary data

Attributes

- Basic format of all attributes:

```
attribute {  
    u2 attribute_name_index;  
    u4 attribute_length;  
    u1 info[attribute_length];  
}
```

- Constant pool entry at `attribute_name_index` is a string, the attribute's name

Example attribute: *Deprecated*

- Generated by Javadoc tag **@deprecated** on a class, interface, field or method
- Empty marker attribute, no data
- Signals tools (e.g. compiler) that the element is deprecated

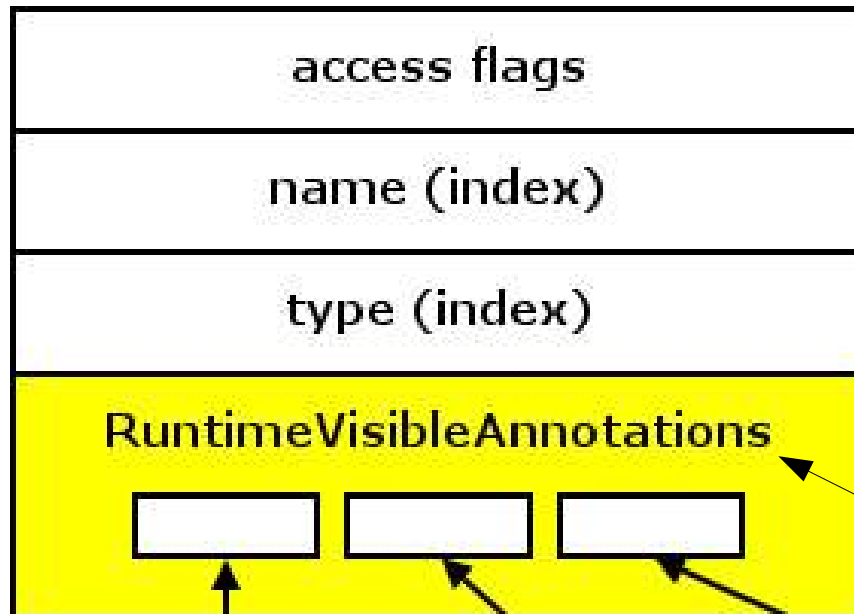
```
Deprecated_attribute {  
    u2 attribute_name_index;  
    u4 attribute_length;  
}
```

always zero

points to a constant pool entry holding the string "*Deprecated*"

Annotations

Field



- High-level metadata mechanism (J2SE 5.0)
- Typed
- Included in source file
- Built on top of attributes

annotation_1

...

annotation_n

Annotation example

Annotation



Annotation Type

```
public class C {  
    @Unsafe("Don't use this!", 42)  
    public void m();  
}
```

```
public @interface Unsafe {  
    String reason();  
    int bugsFiled();  
}
```

Universes: What do we store?

- for a class:
 - availability of run-time checks (true/false)
 - version of the encoding used
- for every field (of reference type):
 - the type modifier for the field's type
- for every method:
 - the type modifier of the return type
 - the type modifiers of the parameter types
 - the method's purity

Implementation

- Two approaches were implemented:
 1. Using class/field/method attributes
 2. Using class/field/method annotations
- Changes were made to
 - MultiJava compiler
 - MultiJava disassembler
 - jmlspec

Implementation with attributes

- Idea: use attributes to store the needed data
 - for classes: version of the encoding used, availability of run-time checks
 - for fields of reference type: encoded type modifier
 - for methods: encoded type modifiers for parameter and return types, purity

Class attribute

- *universe_class* attribute stores version of the encoding and availability of run-time checks

```
universe_class {  
    u2 attribute_name_index;  
    u4 attribute_length;  
    u2 version_index;  
    u1 runtime_support;  
}
```

index into
constant pool



boolean flag



Field attribute

- *universe_field* attribute stores encoded type modifier for fields of reference type

encoded type
modifier

```
universe_field {  
    u2 attribute_name_index;  
    u4 attribute_length;  
    ▶ u1 universe_modifier;  
}
```

Method attribute

- *universe_method* attribute stores modifiers for return and parameter types and purity

boolean flag

return type modifier

param type modifiers

```
universe_method {  
    u2 attribute_name_index;  
    u4 attribute_length;  
    u1 purity;  
    u1 return_modifier;  
    u1 param_modifiers[no_of_params];  
}
```

The diagram illustrates the structure of the `universe_method` attribute. It is a struct containing five fields. The first field is a `u2` integer representing the `attribute_name_index`. The second field is a `u4` integer representing the `attribute_length`. The third field is a `u1` integer representing the `purity`. The fourth field is a `u1` integer representing the `return_modifier`. The fifth field is a `u1` integer representing the `param_modifiers`, which is an array of size `no_of_params`. Labels with arrows point to these fields: 'boolean flag' points to the opening curly brace, 'return type modifier' points to the `return_modifier` field, and 'param type modifiers' points to the `param_modifiers` field.

Problems with attributes

- Implementation with attributes works fine, and is space-efficient, **but...**
 - Reflection doesn't work (attributes not accessible through `java.lang.reflect`)
 - Third-party bytecode processing library (e.g. BCEL) necessary
- leads to second approach...

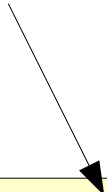
Implementation with annotations

- Use annotations to store the needed data
 - *UniverseClass* annotation (identical to *universe_class* attribute)
 - *UniverseType* annotation (for fields, methods, and parameters)
 - *UniversePure* annotation (for methods)
- Annotations are fully supported by J2SE 5.0 Reflection API

UniverseClass annotation

- Classes are annotated with the *UniverseClass* annotation type

annotation visible at run-time



```
@Retention(RetentionPolicy.RUNTIME)
public @interface UniverseClass {
    public String version();
    public boolean hasRuntimeSupport();
}
```

UniverseType annotation

- *UniverseType* annotation stores an encoded universe type modifier
- Used for fields, method return type and method parameter types
 - possible to annotate individual parameters (attributes have method-granularity)

```
@Retention(RetentionPolicy.RUNTIME)
public @interface UniverseType {
    public byte value(); // encoded modifier
}
```

UniversePure annotation

- *UniversePure* annotation marks a method as pure. Absence is interpreted as non-pure.
- Empty "marker" annotation

```
@Retention(RetentionPolicy.RUNTIME)
public @interface UniversePure {
    /* empty marker annotation */
}
```

Problems with annotations

- Not backward-compatible:
 - Class file version needs to be incremented (otherwise annotations are not recognized by the JVM)
 - Older JVMs will refuse to load the class
- Higher space requirements:
 - simple Stack example needs **988** bytes, only **847** bytes with attributes
- For reflection to work, annotation types must be distributed

Comparison

	Attributes	Annotations
Reflection	-	+
Space	+	-
Compatibility	+	-

- Rising adoption of J2SE 5.0 should eventually favor annotations-based approach
- Compiler switch allows to select implementation.
 - For now, default is attributes

Extensions to MultiJava Compiler

- Several extensions were made:
 - New attribute classes and attribute parser
 - Low-level support for annotations
 - Source-level support still missing
 - Class file utility classes extended
 - Type loading mechanism extended (reads universe type info from class file)
 - New testcases
- Immediate benefits for tools like jmlspec and MultiJava disassembler

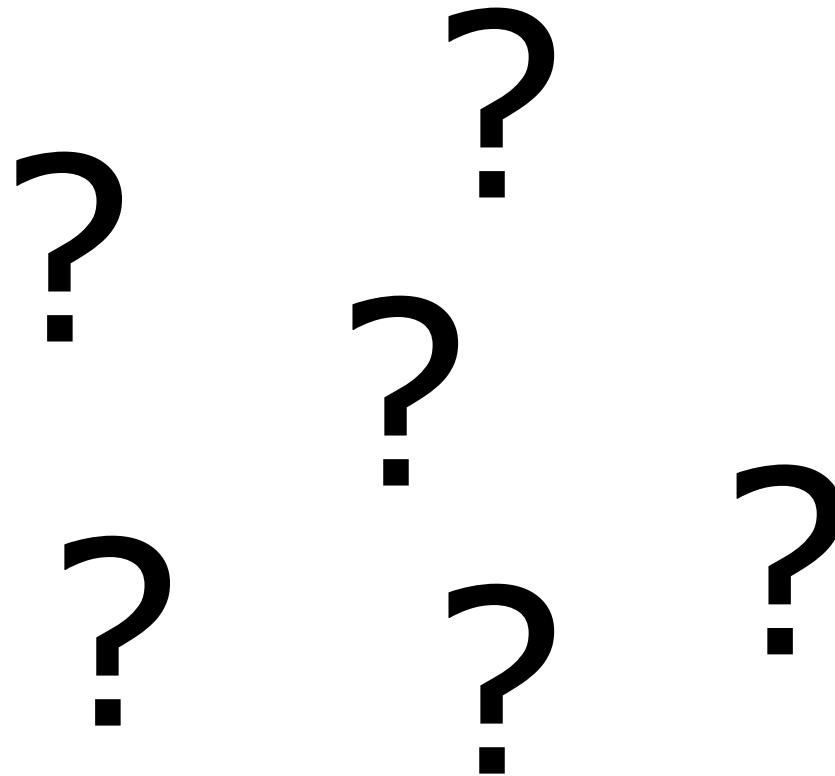
Conclusions

- Bytecode support for the Universe type system conforming to class file format
- Two implementations were shown:
 - Vary in terms of space usage, backward compatibility and reflection capability
 - Attributes preferred at the moment, later switch to annotations

Future Work

- Bytecode support enables...
 - Reflection, gaining access to full universe type information (development tools)
 - Extended bytecode verification (standard verifier does not consider universe type system)
 - Optimization of run-time checks
 - Object creation can be optimized if one knows if a class is *universe-aware* (i.e. was compiled with run-time checks enabled)

Questions



Encoding of type modifiers

- 2 bits used to encode the type modifier for simple reference types

type modifier	binary encoding
(none)	00
peer	01
rep	10
readonly	11

Encoding of array modifiers

- Array types have two modifiers, one for the array itself, and one for the elements.
- Array modifier in bits [1,0], element modifier in bits [3,2]

array modifier	binary encoding
peer peer	01 01
peer readonly	11 01
rep peer	01 10
rep readonly	11 10
readonly peer	01 11
readonly readonly	11 11

Encoding of method purity

- Method purity is encoded in a single bit

purity	binary encoding
none-pure	0
pure	1