Types in languages: Java

Amer Diwan

Java

• Some similarities to Modula-3:
  – Strongly typed
  – Single inheritance O-O model
  – Garbage collection
  – Exceptions
• Some differences from Modula-3:
  – C family syntax
  – Name type equality
  – No UNSAFE/Untraced
  – Separate interface and implementation inheritance
  – Bigger language definition (~ 400 pages excl. libs).
Java type system

- Java expresses type compatibility in terms of **narrowing** and **widening conversions**

![Diagram showing narrowing and widening conversions]

Intuition behind conversions

- **Widening**
  - Subtype to supertype. *Always legal*
- **Narrowing**
  - Supertype to subtype. *May or may not be legal*
  - Interface type to class (and vice versa). *May or may not be legal*
- I’ll focus on reference conversions in this lecture
Reference types

- **Interface**: Lists a set of methods that all implementations of the interface must have
- **Class**: Implements one or more (explicit or implicit) interfaces
- **Array**: these are objects too

Interfaces

- Interfaces are like “fully abstract” classes
  - No code: just method declarations and constants
- Interfaces **may use multiple inheritance**
Classes

- **Single inheritance** like Modula-3
- **Public, protected, private** members like C++
- Classes may be **final**: will not be subclassed
  - Has implications for implementation AND
  - Typing rules!!
- Class “Object” is the top of the inheritance hierarchy

Example: Interface and Classes

- (From John Mitchell’s slides)

  ```java
  interface Shape {
    public float center()
    public void rotate(float degrees) }
  interface Drawable {
    public void setColor(Color c)
    public void draw() }
  class Circle implements Shape, Drawable {
    ...
  }
  ```
A more complex example

interface cdDecoder {
    public void play(CD *music)}
interface dvdDecoder {
    public void play(DVD *movie) }
class dvdPlayer implements cdDecoder, dvdDecoder {
    ...
}
• dvdPlayer has two play methods. What to do?

Interfaces are a very powerful concept

• Separates implementation from specification at a fine granularity
  – A class may implement many distinct interfaces
• Different from interfaces in languages like Modula-2 and Modula-3 or header files
  – Granularity
  – Inheritance of interfaces
Conversions

- Narrowing and widening casts defined for both primitive and reference types
  - Widening casts are applied implicitly or explicitly
  - Narrowing casts must be explicitly applied

Widening conversions: subtypes

- Can convert from S to T if,
  - if S is a subclass of T
  - if S implements interface T
  - if S is NULL and T is any class type, interface type or array type
  - if S is a subinterface of T
  - if S is an interface and T is ‘Object’
  - if S is an array and T is ‘Object’
  - if S is an array and T is Clonable
Widening conversions (continued)

- If S is an array SC[], T is an array TC[], and
  - SC and TC are reference types, and
  - widening conversion from SC to TC
- What if SC and TC are not reference types?

Narrowing conversions

- S to T if
  - S is superclass of T
  - S is a class, T is an interface, S is not a final class
  - S is ‘Object’ T is any array or interface type
  - S is an interface, T is not a final class
  - S is interface, T is final class, and T implements S
Narrowing conversions (cont.)

– S and T are interfaces, but S is not a subinterface of T and they don’t declare incompatible methods
– S and T are array types and there is a narrowing conversion from their element types
  • Narrowing conversions require run-time type tests
  • What are the tests involved in the last narrowing rule?

Examples

• interface I {...}
  interface J {...}
  class S extends T implements I {...}
  • S to T?
  • T to S?
  • S[] to T[]?
  • T[] to S[]?
  • S to I?
  • S to J?
Assignment rules

• a: A = b: B is allowed when
  – B = A
  – B can be converted to A using widening conversions (i.e., B <: A)
  – For primitives, sometimes a narrowing conversion is allowed

Type equality

• Unlike Modula-3, name equality, except:
  – Array types A and B are the same if they have the same element type
  – (Array index type is not important since all arrays are dynamically allocated and have a compile-time “unknown” index type)
Some implications for implementation

• Would be similar to Modula-3, but
  – no opaque types, structural equality: need less support from linker
  – invokeinterface: in addition to v-tables also need a more complex mechanism

invokeinterface

• (example from Mitchell’s slides)
  interface Incrementable { public void inc() }
  class IntCounter implements Incrementable {
    public void add(int)
    public void inc()
    public int value()
  }
  class FloatCounter implements Incrementable {
    public void inc()
    public void add(float)
    public float value()
  }
Summary

• Type safe: unchecked type errors cannot happen at run-time
  – Particular important given Java’s target market
• Usually opts for “clean” over “fast”
• Mostly clean type system but perhaps not as elegant as that of M-3
• Separates implementation from interface

Next lecture: Smalltalk

• Smalltalk: a dynamically typed language
• Readings:
  – The Smalltalk-80 system (Byte)
  – [Sebesta (Sections 11.4, 11.5, 11.6) or Scott (Sections 10.6.1) or Ghezzi (Section 6.3.4)]