Types in languages: Java

Amer Diwan

Java

• Similar to Modula-3 but notably
  – No UNSAFE/Untraced
  – ...
  – Separate interface and implementation inheritance!
  – Much bigger language definition (about 400 pages excluding libraries).
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
- Interfaces may use multiple inheritance
  - How are conflicts handled?
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 Mitchell’s slides)
  
  ```java
  interface Shape {
    public float center()
    public void rotate(float degrees) }
  ```

  ```java
  interface Drawable {
    public void setColor(Color c)
    public void draw() }
  ```

  ```java
  class Circle implements Shape, Drawable {
    ...
  }
  ```
Pros and cons of interfaces?

Conversions

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

- Can convert from S to T if \( S <: T \), i.e.,
  - 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’
- More cases than M-3 due to separation of interface and implementation

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

Why are narrowing conversions not just the opposite of widening conversions?
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
  – Mitchell’s chapter, History of objects: Simula and Smalltalk
  – Both readings should be picked up from Pat’s office (ECOT 717)