Lecture 20: OO Languages

CSC 131
Fall, 2012

Kim Bruce

More Quizzes!!

Type Restrictions

Type systems limit changes in method types in subclasses.

```
class C {
      ...
   method clone() → CType {...};
   method equals(other: CType) {...};
}
```

How can we change these types in subclasses?

More flexible subclasses

Why restrict changing types in subclasses?

Methods can be mutually recursive!

```
class example {
      :
  method m(s:S,...) {... self.n(s) ...}
  method n(x: S) → T {...}
}


class subExample {
  inherits Example
  method n(x: S') → T' {...}
  method newMethod(...){...}
}
```

What must be relationship of new type of n to old to preserve type safety?

Changing Types in Subclasses

Subtype will always be fine!

I.e., S <: S' and T' <: T equivalently, S' → T' <: S → T

E.g., can change return type of clone in subclass to type of objects from subclass.

Cannot specialize parameter types in equals

Binary methods

If subclass updates method types so they are subtypes of original then type-safe.

If restricted in this way, subclasses will always generate subtypes.

What about instance variables?

Instance variables can be values and receivers

- No subtypes!

If Circle has instance variable center: Point, ColorCircle's center must have same type.

Hard to redefine getCenter in ColorCircle, even if legal!

Important problem in OO language design and type theory

OO Languages

- Simula 67
- Smalltalk-72, -74, ... -80
- C++, Object Pascal, Object Cobol, ...
- Eiffel, Sather
- Java
- Scala
- *Grace?*

Simula 67

Simula 67

- First OO language
- *You* read in text
- Also added coroutines
- Use of "inner" rather than "super" in constructors

Inner

```
Class A; Begin startA; Inner; endA End;
A Class B; Begin startB Inner; endB End;
B Class C; Begin startC Inner; endC End;
Ref(C) X;

X :- New C;
```

- Results in execution of:

  ```
  startA startB startC endC endB endA
  ```

- Beta supports similar in all methods & classes

Smalltalk

Smalltalk

- New features:
  - Everything is an object, including classes
  - No operations -- only message-sending
  - Used to build customizable environment
  - Abstraction -- private instance variables, public methods
- Dynamically typed

Dynabook

- Laptop computer -- Alan Kay 1970's
  - Turing award 2003
- Proposed in 1970's - aimed at children & adults
  - Neal Stephenson's "The Diamond Age or, a young lady's illustrated primer" is the next step
- Programmable environment
- Smalltalk as OS and programming language

Syntax

- `n <- 3+4`
  - send "+" message to 3 w/param 4 and insert in n
- `n between: 10 and: 100`
  - send "between: and:" message to n w/params 10, 100
- `[ :params | <message-expressions> ]`
  - lexical closure - equiv to lambda expression
  - positiveAmounts :=
    allAmounts select: [:amt | amt isPositive]

Smalltalk class

```
class name              Point
super class             Object
class var
instance var             x   y
class messages and methods
!...names and code for methods..."
 instance messages and methods
moveDx: dx Dy: dy ||
   x <- dx+x
   y <- dy + y
x
 ^ x
...
```

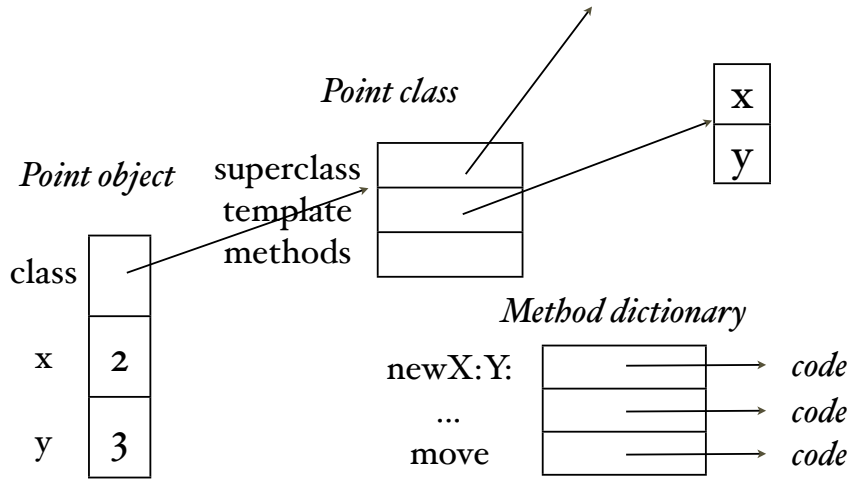
Commands

- Loops example:

```
1 to:10 do:[:i|
  Transcript show: (i asString).
].
```

- Conditional
  - `(x>0) ifTrue:[ x:=x+1. ] ifFalse:[ x:=0 ].`
  - `true and false are special values like lambda calculus encodings`

Run-time representations



Dynamic Method Invocation

- Start with object's class and search up superclasses.
- When call method inside, start search from self again.
- *Most other OO languages do not implement dmi in this way -- too inefficient!*

Key ideas of Smalltalk

- Everything is an object
- Information hiding - instance variables protected.
- Dynamic typing, so subtyping determined by whether can masquerade -- "message not understood"
- Inheritance distinct from subtyping

Smalltalk

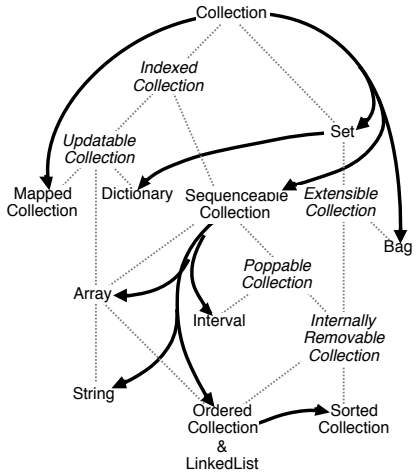


Figure 5: Interfaces versus Inheritance

C++

C++ Design Goals

- Data abstraction & OO features
- Better static type checking
- Backwards compatibility w/ C
- Efficiency: If you do not use a feature, you should not pay for it
- Explicitly hybrid language -- C w/abstraction

Additions to C

- type bool
- reference types & call by reference
- user-defined overloading
- templates
- exceptions
- public or private inheritance

Problems

- Confusing casts and conversions
- Objects allocated on stack
  - what happens w/subtyping? truncation!
- Overloading methods -- see earlier examples!
- Multiple inheritance (*later*)

Casts & Conversions

- Implicit conversions:
  - from short to int
  - class B { public: B (A a) {} }; A a; B b = a;
- Explicit conversions:
  - C c; D* d; d = (D*) &c; d -> DonlyMeth();
- Try to avoid problems by using new casts:
  - static_cast, dynamic_cast, reinterp_cast, const_cast
  - dynamic_cast checks using run-time type info (RTTI)
  - reinterp_cast trusts

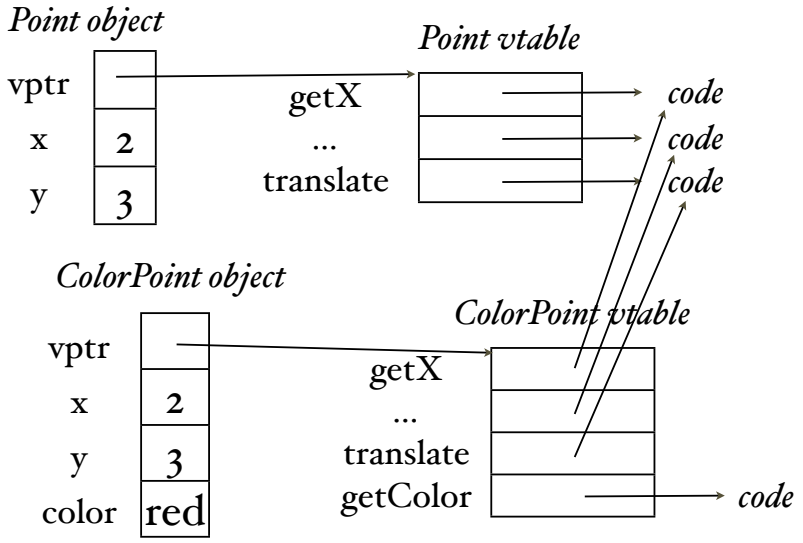
Objects on stack

- Doesn't interact well with subtyping.
- Point p; // allocates point on stack
- ColorPoint cp(3,4,BLUE);
- p = cp; // slices and converts to Point
- Call by value has similar problems
- What about reference parameters to methods?

OO Features in C++

- Visibility
  - Public, protected, private
  - Friends ...
- Virtual vs. nonvirtual functions
  - don't pay the price of dynamic method invocation
- Implemented via vtable
  - no search necessary
  - static typing makes efficient rep possible

VTable for Virtual methods



C++ vs Smalltalk implementation

- No search in C++ since offset for given method same in base and derived classes
- Smalltalk has no type declaration
  - value not known to be subtype of declared type
  - no idea where method is located

Abstract classes

- Have at least one method undefined
- "Pure" leaves all undefined
- Can't construct, but can inherit from
- Derived subclasses can be used as subtypes of abstract base class.

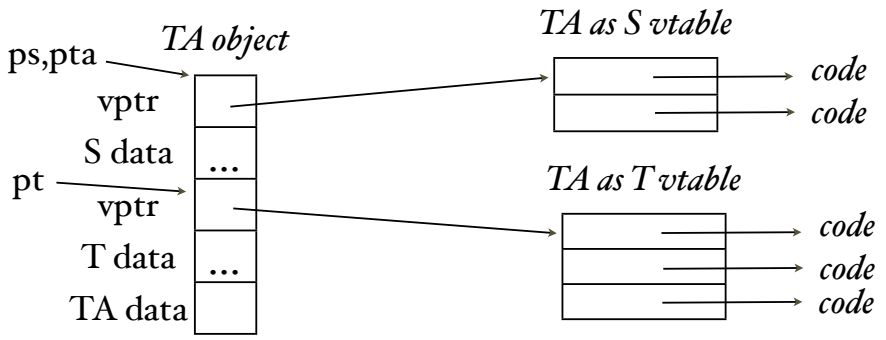
Multiple Inheritance

- Appealing: TA derived from Student and Teacher.
- Added to C++ and Smalltalk. In Eiffel from beginning.
- Problems conceptually and with implementation

MI in C++

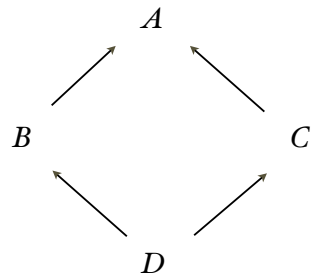
```
class S {...}
class T{...}
class TA: public S, public T
{...}

TA* pta = new TA();
S * ps = pta;
T * pt = pta;
```

Representing MI



What if T and TA both define virtual f?
T methods expect inst vbles starting at pt
How get access to instance vbles from S?

Conceptual Problems w/ MI



Diamond Inheritance: Suppose A has virtual f and B and C override it.
Which version is inherited in D?

Java Solution

- Most multiple inheritance in C++ involves pure base classes.
- Java: Single inheritance, but can implement multiple interfaces.
- Avoids problems.
- *Traits (e.g., in Scala) are modern alternative.*

C++ Summary

- One of most complicated languages ever
  - design by accretion
- Meets design goals but very hard to get right
  - "C makes it easy to shoot yourself in the foot. In C++ it's harder to shoot yourself in the foot, but when you do, you blow off your whole leg." -- Stroustrup
- Memory management is big problem
- Most programmers learn a subset.

C++ Humor

- C++: Hard to learn and built to stay that way.
- Java is, in many ways, C++--.
- How C++ is like teenage sex:
  1. It is on everyone's mind all the time.
  2. Everyone talks about it all the time.
  3. Everyone thinks everyone else is doing it.
  4. Almost no one is really doing it.
  5. The few who are doing it are:
     A. Doing it poorly.
     B. Sure it will be better next time.
     C. Not practicing it safely.

Java

Java Design Goals

- Portability across platforms
- Reliability
- Safety (no viruses!)
- Dynamic Linking
- Multithreaded execution
- Simplicity and Familiarity
- Efficiency

Java

- Original implementations slow
  - Compiled to JVML and then interpreted
  - Now JIT
  - Garbage collection
- Safety - 3 levels:
  - Strongly typed
  - JVML bytecode also checked before execution
  - Run-time checks for array bounds, etc.
- Other safety features:
  - No pointer arithmetic, unchecked type casts, etc.
  - Super constructor called at beginning of constructor

Exceptions & Subtyping

- All non-Runtime exceptions must be caught or declared in "throws" clauses
  - void method readFiles() throws IOException {...}
- Suppose m throws NewException.
- What are restrictions on throwing exceptions if m overridden in subclass? Masquerade!

Simplify from C++

- Purely OO language (except for primitives)
- All objects accessed through pointers
  - reference semantics
- No multiple inheritance -- trade for interfaces
- No operator overloading
- No manual memory management
- No automatic or unchecked conversions

Interfaces

- Originally introduced to replace multiple inheritance
- Allows pure use of subtype polymorphism w/ out confusing with implementation reuse.
- Slower access to methods as method order not guaranteed

Encapsulation

- Classes & interfaces can belong to packages:

  package MyPackage;

  public class C ...

- If no explicit package then in "default" package
- public, protected, private, "package" visibility
- Class-based privacy (not object-based):
  - If method has parameter of same type then get access to privates of parameter

Problems w/Packages

- Generally tied to directory structure.
- Anyone can add to package and get privileged access
- All classes/interfaces w/out named package in default package (so all have access to each other!)
- No explicit interface for package
- Abstraction barriers not possible for interfaces. Discourages use of interfaces for classes.

Abstraction barriers not monotonic

```
package A;
public class Fst {
   void m(int k){System.out.println("Fst m: "+k);}
   public void n(){System.out.print("Fst n: "); m(3);}
}

package B;
import A.*;
public class Snd extends Fst{
   public void m(int k){System.out.println("Snd m: "+k);}
   public void p(){System.out.print("Snd p: "); m(5);}
}

package A;
import B.*;
public class Third extends Snd{
   public void m(int k){System.out.println("Third m: "+k);}
}
```

Abstraction barriers not monotonic

```
import A.*;
import B.*;
public class Fourth{
    public static void main(String[] args){
        Fst fst = new Fst();
        fst.n();                          Fst n: Fst m: 3

        Snd snd = new Snd();
        snd.n();                          Fst n: Fst m: 3    // ????
        snd.m(5);                         Snd m: 5

        Third third = new Third();
        third.n();                        Fst n: Third m: 3
        third.m(7);                       Third m: 7
        third.p();                        Snd p: Third m: 5
    }
}
```

Warning : Method void m(int) in class B.Snd does not override the corresponding method in class A.Fst. If you are trying to override this method, you cannot do so because it is private to a different package.

Goals of Java 5

- Ease of Development
  - Increased Expressiveness
  - Increased Safety
- *Scalability and Performance*
- *Monitoring and Manageability*
- *Desktop client*
- Minimize Incompatibility
  - No changes to virtual machine
  - Only one new keyword (enum)

Java 5

- Generics
- Enhanced for loop (w/iterators)
- Auto-boxing and unboxing of primitive types
- Type-safe enumerated types
- Static Import
- Simpler I/O

Generics Finally Added

- Templates done well (unlike C++)
  - Type parameters to classes and methods.
  - Type-checked at compile time.
  - Allows clearer code and earlier detection of errors.
  - Biggest impact on Collection classes.
- *Limitations*
  - Virtual machine has not changed.
  - Translated into old code with casts
  - Casts and instanceof don't work correctly
  - Can't construct arrays involving variable type.