Programming Languages

Dr. Axel Simon and Dr. Michael Petter
Winter term 2013

Outline

Inheritance Principles
- Interface Inheritance
- Implementation Inheritance
- Liskov Substitution Principle and Shapes

C++ Object Heap Layout
- Basics
- Single-Inheritance
- Virtual Methods

C++ Multiple Parents Heap Layout
- Multiple-Inheritance
- Virtual Methods
- Common Parents

Discussion & Learning Outcomes

Excursion: Linearization
- Ambiguous common parents
- Principles of Linearization
- Linearization algorithm
“Wouldn’t it be nice to inherit from several parents?”

Interface Inheritance

**Interface vs. Implementation** inheritance

The classic motivation for inheritance is implementation inheritance:

- **Code reuseage**
- Child specializes parents, replacing particular methods with custom ones
- Parent acts as library of common behaviours
- Implemented in languages like C++ or Lisp

Code sharing in interface inheritance inverts this relation:

- **Behaviour contract**
- Child provides methods with signatures predetermined by the parent
- Parent acts as generic code frame with room for customization
- Implemented in languages like Java or C#

Implementation inheritance
Excursion: LSP and Square-Rect-Problem

The Liskov Substitution Principle

Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it.

```java
class Rectangle {
    void setWidth (int w){ this.w=w; }
    void setHeight(int h){ this.h=h; }
    void getWidth () { return w; }
    void getHeight () { return h; }
}

class Square extends Rectangle {
    void setWidth (int w){ this.w=w;h=w; }
    void setHeight(int h){ this.h=h;w=h; }
}
```

Excursion: Brief introduction to LLVM IR

Low Level Virtual Machine as reference semantics:

```llvm
; (recursive) struct definitions
%struct.A = type { i32, %struct.B, i32(i32) }
%struct.B = type { i64, [10 x [20 x i32]], i32 }

; allocation of objects
%a = alloca %struct.A

; address adjustments for selection in structures:
%1 = getelementptr %struct.A*, %a, i64 2

; load from memory
%2 = load i32(i32)* %1

; indirect call
%retval = call i32 (i32)* %2(i32 42)
```

Object layout

```java
class A {
    int a; int f(int);
};

class B : public A {
    int b; int g(int);
};

class C : public B {
    int c; int h(int);
};
```

Rectangle r = new Square(2);
void setWidth (int w){ this.w=w; }
void setHeight(int h){ this.h=h; }
void getWidth () { return w; }
void getHeight () { return h; }

Rectangle r = new Square(2);
void setWidth (int w){ this.w=w;h=w; }
void setHeight(int h){ this.h=h;w=h; }

Behavioural assumptions

Multiple Inheritance | Inheritance Principles | 7 / 11
--- | --- | ---

Multiple Inheritance | Standard Object Heap Layout | Object layout & inheritance | 10 / 31
Object layout – virtual methods

class A {
    int a; virtual int f(int);
    virtual int g(int);
    virtual int h(int);
};
class B : public A {
    int b; int g(int);
};
class C : public B {
    int c; int h(int);
};
...
C c;
c.g(42);

Multiple Base Classes

class A {
    int a; int f(int);
};
class B {
    int b; int g(int);
};
class C : public A, public B {
    int c; int h(int);
};
...
C c;
c.g(42);

"So how do we include several parent objects?"

Multiple Base Classes

class A {
    int a; int f(int);
};
class B {
    int b; int g(int);
};
class C : public A, public B {
    int c; int h(int);
};
...
C c;
c.g(42);

Multiple Base Classes

class A {
    int a; int f(int);
};
class B {
    int b; int g(int);
};
class C : public A, public B {
    int c; int h(int);
};
...
C c;
c.g(42);
Ambiguities

```java
class A { void f(int); }
class B { void f(int); }
class C : public A, public B {
    C* pc;
    pc->f(42);
    // Which method is called?
}
```

Solution I: Explicit qualification
```java
pc->A::f(42);
pC->B::f(42);
```

Solution II: Automagical resolution
Idea: The Compiler introduces a linear order on the nodes of the inheritance graph

Linearization

Inheritance Relation $H$
Defined by ancestors.

Multiplicity $M$
Defined by the order of multiple ancestors.

Principles

1. An inheritance mechanism (maps Object to sequence of ancestors) must follow the inheritance partial order $H$
2. The inheritance is a uniform mechanism, and its searches (→ total order) apply identically for all object properties (→fields/methods)
3. In any case the inheritance relation $H$ overrides the multiplicity $M$
4. When there is no contradiction between multiplicity $M$ and inheritance $H$, the inheritance search must follow the partial order $H \cup M$. 

Ambiguities

```java
class A { void f(int); }
class B { void f(int); }
class C : public A, public B {
    C* pc;
    pc->f(42);
    // Which method is called?
}
```

Solution I: Explicit qualification
```java
pc->A::f(42);
pC->B::f(42);
```

Solution II: Automagical resolution
Idea: The Compiler introduces a linear order on the nodes of the inheritance graph

Linearization

Inheritance Relation $H$
Defined by ancestors.

Multiplicity $M$
Defined by the order of multiple ancestors.

Principles

1. An inheritance mechanism (maps Object to sequence of ancestors) must follow the inheritance partial order $H$
2. The inheritance is a uniform mechanism, and its searches (→ total order) apply identically for all object properties (→fields/methods)
3. In any case the inheritance relation $H$ overrides the multiplicity $M$
4. When there is no contradiction between multiplicity $M$ and inheritance $H$, the inheritance search must follow the partial order $H \cup M$. 

**Linearization algorithm candidates**

**Depth-First Search**

A B W C

⚠️ Principle 1 *inheritance* is violated

**Breadth-First Search**

A B C W D

⚠️ Principle 1 *inheritance* is violated

**Linearization algorithm candidates**

**Reverse Postorder Rightmost DFS**

A B F D C E G H W

✓ Linear extension of inheritance relation
Linearization Algorithm

Idea [Ducournau and Habib (1987)]
Successively perform Reverse Postorder Rightmost DFS and refine inheritance graph $G$ with contradiction arcs.

The reservoir set of potential contradiction arcs $CA$ is initially $M$, while the inheritance graph $G$ starts from $H$.

1. $G \leftarrow G \cup CA$
2. $CA \leftarrow (\text{contradiction arcs of upper search}) \cap M$
3. $G \leftarrow G \cup CA$;
   while $(CA \neq \emptyset) \land (\text{search violates } H \cup M)$

Idea [Ducournau and Habib (1987)]
Successively perform Reverse Postorder Rightmost DFS and refine inheritance graph $G$ with contradiction arcs.

The reservoir set of potential contradiction arcs $CA$ is initially $M$, while the inheritance graph $G$ starts from $H$.

1. $G \leftarrow G \cup CA$
2. $CA \leftarrow (\text{contradiction arcs of upper search}) \cap M$
3. $G \leftarrow G \cup CA$;
   while $(CA \neq \emptyset) \land (\text{search violates } H \cup M)$
Virtual Tables for Multiple Inheritance

class A {
    int a; virtual int f(int);
};
class B {
    int b; virtual int f(int);
    virtual int g(int);
};
class C : public A, public B {
    int c; int f(int);
};

B* pb = &c;
C c;
B* pb = &c;
pb->f(42);

\begin{tabular}{|c|c|}
\hline
Multiple Inheritance & Implementation of Multiple Inheritance \\
\hline
Virtual Table & 20/31 \\
\hline
\end{tabular}