11. Inheritance: Proper Inheritance, Multiple Inheritance

Public Inheritance should model “is-a”

B publicly inherits ("is-a") from A
This means:
• every object of type B is also an object of type A
• anything that is true of an object of A is also true of an object of B
• A represents a more general concept than B
• B represents a more specialized concept than A
• anywhere an object of A can be used, an object of B can be used
Requirements & Promises of Functions

- Advertised Requirements (Pre-Condition)
- Advertised Promise (Post Condition)

Expressing Requirement & Promise

Unfortunately in C++ : use Disciplined and Consistent Comments

Example:

```cpp
class Stack {
    int top() const;
    // PURPOSE: returns the top element
    // REQUIRE: numElems() != 0
    // PROMISE: nothing

    void push(int elem);
    // PURPOSE: pushes elem onto the top of stack
    // REQUIRE: numElems() < 10
    // PROMISE: numElems() == INITIAL(numElems()) + 1
    // PROMISE: top() == elem
}
```

Substitutability

Advertised Behavior of the Derived class is Substitutable for that of the Base class

Substitutability: Derived class Services Require no more and promise no less than the specifications of the corresponding services in the base class

Example:

```cpp
int Base::fn(int x);
// REQUIRE: x is odd
// PROMISE: Returns an even int

int Derived::fn(int x);
// REQUIRE: x is int
// PROMISE: Returns 8
```
Liskov Substitutability Principle (LSP)

“Any Derived class object must be substitutable wherever a Base class object is used, without the need for the user to know the difference”

Inheritance *appears* Simple

```cpp
class Bird {
    ... // has beak, wings, etc..
public:
    virtual void fly(); // Bird can fly
};
class Parrot : public Bird { // Parrot is a bird
    ...
public:
    virtual void Mimic(); // Can Repeat words, ...
};

Parrot mypet;
mypet.Mimic(); // my pet being a parrot can Mimic()
mypet.fly(); // my pet “is-a” bird, can fly
```
Yes, it just appears simple

class Penguin : public Bird {... // Penguin is Bird }

This inheritance says that Penguins can fly!!!

Result of Incorrect understanding from an imprecise (language) statement

Birds can fly does not mean *all* birds can fly.
In general, birds that have the ability to fly, can fly.

“Penguins may try to fly, but will fail” - Design

class Bird {
    ... // has beak, wings, etc..
    public:
        virtual void fly(); // Bird can fly
    }

class Penguin : public Bird { // Penguin is a Bird
    ...
    public:
        virtual void fly() { error (“Penguins don’t fly!”); }
    }

    
    • Does not model “Penguins can’t fly”
    • Models “Penguins may fly, but if they try it is an error”
    • Run-time error if an attempt is made to fly - not desirable
    • Think about Substitutability - Fails LSP

    void PlayWithBird (Bird& abird)
    {
        ...
        abird.fly(); // OK if bird
        // happens to be Parrot.
        // OOPS if bird
        // happens to be Penguin...
    }
“Not all birds fly, Penguins can’t” - Design

class Bird {
... // has beak, wings, etc..
... // No fly function - Birds have beak, wings, ... don’t want to say anything about
// flying here
};

class FlyingBird : public Bird { ... // A Bird that can fly
public:
    virtual void fly();
};

class NonFlyingBird : public Bird { ... // A Bird that can’t fly
};
class Parrot : public FlyingBird {...};
class Penguin : public NonFlyingBird {...};

More examples on Inheritance?!

• Does class Square inherit from class Rectangle?

• Does class “Basket of Bananas” inherit from “Basket of Fruits”? 
Design should be based on Requirements and Requirements

“The best design depends on what the system is expected to do, both now and in the future.”

Multiple Inheritance

An object is a “kind-of” more than one type

- **AquaticBeing**
  - swim()

- **TerrestrialBeing**
  - walk()

- **Amphibian**
  - adopt()
Ambiguous Functions in Multiple Inheritance

TA sam;
sam.GoToClass(); ???

TA has to override the GoToClass() function

Duplication of Base Class Data in Multiple Inheritance

class Window { ... } // Represents a Window on the Computer
class WindowWMenu : public Window {...}
class WindowWScrollBar : public Window {...}
class WindowWMenuAndScrollBar : public WindowWMenu, public WindowWScrollBar {...}

WindowWMenuAndScrollBar mywindow;

Window Data
WindowWMenu Data
Window Data
WindowWScrollBar Data
WindowWMenuAndScrollBar Data
Virtual Base Class

class Window { ... } // Represents a Window on the Computer
class WindowWMenu : public virtual Window {...}
class WindowWScrollBar : public virtual Window {...}
class WindowWMenuAndScrollBar : public virtual Window, public virtual WindowWMenu, public virtual WindowWScrollBar {...}

WindowWMenuAndScrollBar mywindow;

Problems With Repetitive Calls to Functions in Multiple Inheritance

WindowWMenu::draw()
{Window::draw(); ... }

WindowWScrollBar::draw()
{Window::draw(); ... }

WindowWMenuAndScrollBar::draw()
{
    WindowWMenu::draw();
    WindowWScrollBar::draw();
    ...
}

mywindow.draw();
// Window::draw will be called // twice.
Lab Work: Details provided on-line.