Polymorphism Review

- When invoking a method, <u>it first has to exist</u>!
  - The compiler finds it based on the **declared type** of the <u>class or any of its ancestors</u>.
    - Remember, every derived class extends its parent, i.e. has everything in parent and a few more.

- It then decides if it's <u>virtual or not</u>:
  - If virtual → invoke using the vptr and vtbl of the actual object
  - If not virtual → invoke using method in declared type.

Polymorphism Review – cont.

How do we determine the correct method to be invoked?

- **Is the method defined** based on the object/pointer's **declared type** or any of its parents?
  - NO → compilation error
  - YES → proceed
- **Is there name hiding?** based on the object/pointer's **declared type**:
  (declaring a method in a derived class with same name (but different parameters) as in base class does not overload but rather **hides** the base class method, be it virtual or not)
  - Yes → Derived methods obscure their Base methods of same name .. may cause compile errors
  - NO → proceed
- **Are we using an object/value or a pointer/ref?**
  - Object → declared type is actual type → use that method
  - Pointer/ref → declared type may not be actual type → proceed
- **Is the method virtual in declared type?**
  (Remember, **once a virtual, always a virtual**)
  - NO → use method of the **declared type**
  - YES → polymorphic call → use virtual table of **actual type**

Resolved at compile-time

Resolved at run-time

Polymorphism Review - ex

```cpp
#include <iostream>
#include <vector>
using namespace std;

class SoftDrink {
public:
    virtual void display() const = 0;               // Line A
};

class Soda : public SoftDrink {
public:
    void display() const override { cout << "Soda "; }   // Line B
};

class GingerAle : public Soda {
public:
    void display() { cout << "GingerAle "; }  // Line C
};

int main() {
    GingerAle ga;   // Line D
    Soda* sd = &ga; // Line E
    sd->display();  // Line F
    ga.display();   // Line G
}
```

- O The program will output: GingerAle GingerAle
- O The program will output: Soda GingerAle
- O The program will output: Soda Soda
- O The program will output: GingerAle Soda
- O The program will compile and not output anything
- O The program will compile, but will crash when run
- O Compilation error at Line A
- O Compilation error at Line B
- O Compilation error at Line C
- O Compilation error at Line D
- O Compilation error at Line E
- O Compilation error at Line F
- O Compilation error at Line G
- O None of the above

How do we determine the correct method invoked?

Polymorphism Review – ex

```cpp
#include <iostream>
#include <vector>
using namespace std;

class SoftDrink {
public:
    virtual void display() const = 0;                // Line A
};

class Soda : public SoftDrink {
public:
    void display() const override { cout << "Soda "; }   // Line B
};

class GingerAle : public Soda {
public:
    void display() { cout << "GingerAle "; }  // Line C
};

int main() {
    GingerAle ga;   // Line D
    Soda* sd = &ga; // Line E
    sd->display();  // Line F
    ga.display();   // Line G
}
```

Soda GingerAle

How do we determine the correct method invoked?

O The program will output: GingerAle GingerAle	O Compilation error at Line B
O The program will output: Soda GingerAle	O Compilation error at Line C
O The program will output: Soda Soda	O Compilation error at Line D
O The program will output: GingerAle Soda	O Compilation error at Line E
O The program will compile and not output anything	O Compilation error at Line F
O The program will compile, but will crash when run	O Compilation error at Line G
O Compilation error at Line A	O None of the above

Polymorphism Review – ex 2

```
#include <iostream>
#include <vector>
using namespace std;

class Parent {
public:
    virtual  void display() const = 0;
};
class Child : public Parent {
public:
    void display() const { cout << "Child "; }
};
class Gc : public Child {
public:
    void display(int x) const { cout << "Grand child” << x; }
};
class GGc : public Gc {
public:
    void display() const { cout << "Grand Grand child "; }
};
int main() {
    // Grand child object
    Gc gc;
    Child* pc = &gc;
    pc->display(4);
    pc->display();
    gc.display(4);
    gc.display();


    // Grand grand child object
    GGc ggc;
    Gc* pgc = &ggc;
    pgc->display(4);
    pgc->display();
    ggc.display(4);
    ggc.display();

}
```

Polymorphism Review – ex 2

```cpp
#include <iostream>
#include <vector>
using namespace std;

class Parent {
public:
    virtual  void display() const = 0;
};
class Child : public Parent {
public:
    void display() const { cout << "Child "; }
};
class Gc : public Child {
public:
    void display(int x) const { cout << "Grand child" << x; }
};
class GGc : public Gc {
public:
    void display() const { cout << "Grand Grand child "; }
};
int main() {
    // Grand child object
    Gc gc;
    Child* pc = &gc;
    pc->display(4);
    pc->display();
    gc.display(4);
    gc.display();


    // Grand grand child object
    GGc ggc;
    Gc* pgc = &ggc;
    pgc->display(4);
    pgc->display();
    ggc.display(4);
    ggc.display();

}
```

```
Build started...
1>------ Build started: Project: Lect_01_B, Configuration: Release x64 ------
1>temp.cpp
1>C:\Dropbox\CS2124_OOP_2023_Fall\temp.cpp(26,22): error C2660: 'Child::display': function does not take 1 arguments
1>C:\Dropbox\CS2124_OOP_2023_Fall\temp.cpp(11,10): message : see declaration of 'Child::display'
1>C:\Dropbox\CS2124_OOP_2023_Fall\temp.cpp(29,20): error C2660: 'Gc::display': function does not take 0 arguments
1>C:\Dropbox\CS2124_OOP_2023_Fall\temp.cpp(15,10): message : see declaration of 'Gc::display'
1>C:\Dropbox\CS2124_OOP_2023_Fall\temp.cpp(36,22): error C2660: 'Gc::display': function does not take 0 arguments
1>C:\Dropbox\CS2124_OOP_2023_Fall\temp.cpp(15,10): message : see declaration of 'Gc::display'
1>C:\Dropbox\CS2124_OOP_2023_Fall\temp.cpp(37,22): error C2660: 'GGc::display': function does not take 1 arguments
1>C:\Dropbox\CS2124_OOP_2023_Fall\temp.cpp(20,14): message : see declaration of 'GGc::display'
1>Done building project "cs2124.vcxproj" -- FAILED.
========== Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped ==========
```

Polymorphism Review – ex 3

```
#include <iostream>
using namespace std;

class Parent {
public:
    void display() const { cout << "Parent "; };
    virtual ~Parent() { cout << "Parent destroyed "; }
};

class Child : public Parent {
public:
    virtual void display() const { cout << "Child "; }
     ~Child() { cout << "child destroyed "; }
};

class Gc : public Child {
public:
    void display() const { cout << "Grand child"; }
    ~Gc() { cout << "Grand Child destroyed "; }
};

class GGc : public Gc {
public:
    void display() const { cout << "Grand Grand child "; }
    ~GGc() { cout << "Grand Grand Child destroyed "; }
};

int main() {
    Parent *pp1 = new Parent;
    Parent *pp2 = new Child;
    Parent *pp3 = new Gc;
    Parent *pp4 = new GGc;

    pp1->display();
    pp2->display();
    pp3->display();
    pp4->display();

    cout << endl;
    delete pp1; cout << endl;
    delete pp2; cout << endl;
    delete pp3; cout << endl;
    delete pp4; cout << endl;

}
```

Polymorphism Review - ex 3

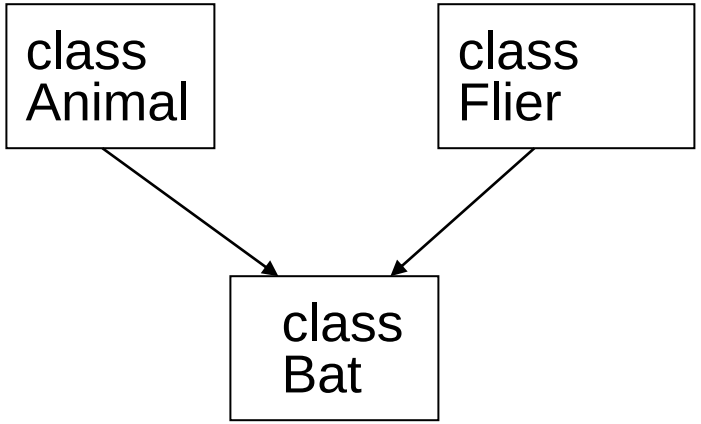
```cpp
#include <iostream>
using namespace std;

class Parent {
public:
    void display() const { cout << "Parent "; };
    virtual ~Parent() { cout << "Parent destroyed "; }
};

class Child : public Parent {
public:
    virtual void display() const { cout << "Child "; }
     ~Child() { cout << "child destroyed "; }
};

class Gc : public Child {
public:
    void display() const { cout << "Grand child"; }
    ~Gc() { cout << "Grand Child destroyed "; }
};

class GGc : public Gc {
public:
    void display() const { cout << "Grand Grand child "; }
    ~GGc() { cout << "Grand Grand Child destroyed "; }
};

int main() {
    Parent *pp1 = new Parent;
    Parent *pp2 = new Child;
    Parent *pp3 = new Gc;
    Parent *pp4 = new GGc;

    pp1->display();
    pp2->display();
    pp3->display();
    pp4->display();

    cout << endl;
    delete pp1; cout << endl;
    delete pp2; cout << endl;
    delete pp3; cout << endl;
    delete pp4; cout << endl;

}
```

```
Parent Parent Parent Parent
Parent destroyed
child destroyed Parent destroyed
Grand Child destroyed child destroyed Parent destroyed
Grand Grand Child destroyed Grand Child destroyed child
destroyed Parent destroyed
```

Multiple Inheritance

- A derived class can have more than one base class
- Different usages:
  - Inheriting multiple abstract classes (**interfaces**) → c++ answer to Java's interfaces is multiple inheritance.
  - Needing objects that are of **multiple types**
- Each base class can have its own access specification in derived class's definition:

```
class Bat : public Anmial, public Flier;
```



Multiple Inheritance

```
#include <iostream>
#include <vector>
using namespace std;

class Flier {
public:
    virtual void fly() { cout << "I can fly!!!\n"; }
};
class Animal {
public:
    virtual void display() { cout << “I am an Animal\n"; }
};

class Bat : public Animal, public Flier { };

class Insect : public Animal, public Flier {
public:
    void fly() { cout << "Bzzzz. ";  Flier::fly();  }
};

class Plane : public Flier {};

int main() {
    Bat battie;
    battie.display();
    battie.fly();
    Plane aPlane;
    Insect anInsect;

    cout << "==========\n";
    vector<Flier*> vf;
    vf.push_back(&battie);
    vf.push_back(&aPlane);
    vf.push_back(&anInsect);
    for (Flier* flier : vf) {
        flier->fly();
    }

}
```

```
I am an Animal
I can fly!!!
==========
I can fly!!!
I can fly!!!
Bzzzz. I can fly!!!
```

- What if I want to add a display() method to Flier that just says “I am a Flier”?

Multiple Inheritance

```
#include <iostream>
#include <vector>
using namespace std;

class Flier {
public:
    virtual void fly() { cout << "I can fly!!!\n"; }
    virtual void display(){ cout << "I am a Flier\n"; }
};
class Animal {
public:
    virtual void display() { cout << "I am an Animal\n"; }
};

class Bat : public Animal, public Flier {
public:

};
class Insect : public Animal, public Flier {
public:
    void fly() { cout << "Bzzzz. ";  Flier::fly();  }
};
class Plane : public Flier {};

int main() {
    Bat battie;
    Plane aPlane;
    Insect anInsect;

    battie.display();
    aPlane.display();
    anInsect.display();

}
```

```
Build started...
1>------ Build started: Project: Lect_01_B, Configuration: Release x64 ------
1>10.mi_om.cpp
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\07.Inheritance\10.mi_om
.cpp(33,19): error C2385: ambiguous access of 'display'
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\07.Inheritance\10.mi_om
.cpp(33,19): message : could be the 'display' in base 'Animal'
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\07.Inheritance\10.mi_om
.cpp(33,19): message : or could be the 'display' in base 'Flier'
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\07.Inheritance\10.mi_om
.cpp(35,21): error C2385: ambiguous access of 'display'
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\07.Inheritance\10.mi_om
.cpp(35,21): message : could be the 'display' in base 'Animal'
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\07.Inheritance\10.mi_om
.cpp(35,21): message : or could be the 'display' in base 'Flier'
1>Done building project "cs2124.vcxproj" -- FAILED.
========== Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped ==========
ailed, 0 up-to-date, 0 skipped ==========
```

- What if I want to add a display() method to Flier that just says “I am a Flier”?

Multiple Inheritance

- Problem: **Ambiguity** when member variables/functions with the same name.
- Solutions:
  - Derived class redefines the method
  - Invoke the method in a particular base class using scope resolution operator `::`
- Compiler errors occur if derived class uses base class function without one of these solutions

Multiple Inheritance

```
#include <iostream>
#include <vector>
using namespace std;

class Flier {
public:
    virtual void fly() { cout << "I can fly!!!\n"; }
    virtual void display(){ cout << "I am a Flier\n"; }
};
class Animal {
public:
    virtual void display() { cout << "I am an Animal\n"; }
};

class Bat : public Animal, public Flier {
public:
    void display() { Animal::display(); }
};
class Insect : public Animal, public Flier {
public:
    using Animal::display;
    void fly() { cout << "Bzzzz. ";  Flier::fly();  }
};
class Plane : public Flier {};

int main() {
    Bat battie;
    Plane aPlane;
    Insect anInsect;

    battie.display();
    aPlane.display();
    anInsect.display();

}
```

```
I am an Animal
I am a Flier
I am an Animal
```

- What if I want to add a display() method to Flier that just says “I am a Flier”?

Multiple Inheritance

- Arguments can be passed to both base classes' constructors:
- Base class constructors are called in order given in class declaration, not in order used in class constructor