Polymorphism 1
CMSC 202

Warmup

What errors are present in the following hierarchy?
Assume GetCurrentTime() and RingBell() are defined elsewhere.

```cpp
class AlarmClock {
public:
  void RingAlarm();
private:
  Time alarmTime;
};

void AlarmClock::RingAlarm() {
  if (alarmTime == GetCurrentTime())
    cout << "Alarm!" << endl;
}

class ManualAlarm : public AlarmClock {
public:
  void RingAlarm();
};

void ManualAlarm::RingAlarm() {
  if (alarmTime == GetCurrentTime())
    cout << "Alarm!" << endl;
}
```

Assume methods and classes used are defined elsewhere.

Polymorphism in Inheritance

“Many-shapes”
Allows a method to take on many type-dependent forms
Ability to manipulate objects in a type-independent way
Only supported through pointers of base-type
Particular method is not decided until run-time
Pointers in Inheritance
Base pointer can point to derived object
Derived object IS a base object
Cannot call derived-class methods via base pointer
Derived pointer cannot point to base object

Binding
Determination of which method in hierarchy to call
Static Binding
Compiler determines binding
Dynamic Binding
Run-time system determines binding
Must use keyword “virtual” to indicate dynamic
A “virtual” method...

Static Binding in Action
```cpp
class Animal {
public:
    void Eat() { cout << "Food" << endl; }
};

class Lion : public Animal {
public:
    void Eat() { cout << "Meat" << endl; }
};

int main() {
    Animal animal;
    Lion lion;
    animal.Eat();
    lion.Eat();
    Animal *animalPtr = new Animal();
    animalPtr->Eat();
    Animal *animalPtr = new Lion();
    animalPtr->Eat();
    return 0;
}
```
### Dynamic Binding in Action

```cpp
class Animal {
  public:
    virtual void Eat();
};

void Animal::Eat() {
  cout << "Food" << endl;
}

class Lion : public Animal {
  public:
    virtual void Eat();
};

void Lion::Eat() {
  cout << "Meat" << endl;
}

int main() {
  Animal animal;
  Lion lion;
  animal.Eat();
  lion.Eat();
  Animal *animalPtr = new Animal();
  animalPtr->Eat();
  Animal *animalPtr = new Lion();
  animalPtr->Eat();
  return 0;
}
```

### What's so great about it?

**Polymorphism**
- Collections of base-type pointers to derived objects
- If dynamic binding is used:
  - Can access method

**Example**
```cpp
Animal *zoo[3];
for (int i = 0; i < 3; ++i)
  zoo[i] = new Animal(i); 
```

**Base-class method**
```
class Animal {
  public:
    void SetName(string name) {
      m_name = name;
    }
    virtual void Eat() = 0;
  private:
    string m_name;
  }
```

**Derived-class method**
```
class Lion : public Animal {
  public:
    virtual void Eat() {
      cout << "Meat" << endl;
    }
};
```

### Pure Virtual Methods

**Base class does not define ANY implementation for a method**
- Forces derived classes to override
- Compiler error if not

**Syntax (in class header):**
- ```virtual retType method() = 0;```
Abstract Class

Definition
Any class that has one or more pure virtual methods

Polymorphic Functions

Non-member functions can be polymorphic
Pass a pointer or reference to a base-class object
Method calls are dynamically bound
Why is this cool?
Old code calling new code when new derived classes are defined!

```c
void FeedAnimal( Animal *animal )
{
    animal->Eat();
}
```

Practice

Modify the warmup so that the AlarmClock class does not implement RingAlarm
Add an ElectricClock class that has a buzzer instead of a bell
Create a collection of AlarmClocks and use polymorphism to ring their alarms
Challenge

Define an Appliance class
Define a Microwave class that inherits from Appliance
  Microwaves have a button-based interface
  Temperature on scale 1-10
  Cooks for any number of minutes and/or seconds
Define a Stove class that inherits from Appliance
  Stoves have a knob-based interface
  Temperate on scale 100-550
  Cooks for any number of minutes
Implement a dynamically bound hierarchy of methods that perform the following:
  SetTemperature
  SetTimer