Announcements

• Recitations:
  – Google Test (unit testing harness)
  – Mandatory
• PA5 is due today
  – \(W[i,j] = D[i,j] + M \text{ in } (W[i-1,k]) \), \(k \leq j\)
• No questions?
• Any questions?
• No programming over spring break
• Reading: Chapter 6
  – Start reading as soon as you have time
  – Quiz will be available 11:00 today
  – Quiz isn’t due until the Monday you get back
    • Start of class.

OO: C++ vs Java

• In C++, objects are optional
  – In Java, all functions are methods of a class
  – In C++, they don’t have to be

  – When should you create an object?
    • Data supports unique operations
      – Encapsulation
    • When it shares properties/code with a more abstract form
      – Polymorphism
    • Efficiency should not be a consideration

OO: C++ vs Java (cont.)

• Java has an inheritance tree
  – Every class has exactly one parent
  – Object is the root of all classes
• C++ has an inheritance forest
  – No unique root
  – Many classes have no parent
  – A few classes have multiple parents
  – Inheritance relations form a directed acyclic graph

What is inheritance?

• If class B inherits class A, what does that mean?
  – Inheritance is union
    • Every field in A is included in B
    • Every method in A is included in B
    • B then adds its own fields and methods
  – Note: you cannot inherit part of a parent
• Inheritance is mandatory in Java
  – Every class has a parent
  – Every object has extra data
• Inheritance is optional in C++
  – Most classes have no parent
  – Makes inheritance “lighter”
  – Makes run-time typing harder

Syntax of Inheritance

class Quagga : public Equine
{
  ...
};

• Choice of three inheritance levels:
  – Public: everything has the same access privileges in the child as the parent
  – Protected: public fields of parent are protected in child
  – Private: everything is private in the parent
• Always use public inheritance
  – Haven’t found a use for the rest
C++ Default: No Parent Class

- Most of you have defined objects for the programming assignments
- Most of you have not specified a parent class
- Therefore your classes do not have a parent class.
  - This is different from Java

Inheritance vs Inclusion

- Many of you have defined a Pose class
  - Should it inherit vector?
  - Or just include a field of type vector?
  - Why?
- Remember: inheritance is union
  - Can you push_back onto a Pose?
  - That would create a 26th Point3D...
  - In this case, inclusion is what you want

Method Overloading

```cpp
class Animal {
public:
    bool WarmUp() {return false;}
    void Behave() {WarmUp();}
};
class Mammal : public Animal {
public:
    bool WarmUp() {shiver(); return true;}
};
```

What Happens?

```cpp
Mammal m;
m.Behave();
```

- Does shiver() get called?
- Does Behave() return true or false?
- In Java: Shiver() gets called
- In C: Shiver() does not get called
  - OK, C doesn't have objects, but if it did...
  - In C++: your choice
  - As written, Shiver() does NOT get called

Dispatch

- How methods are selected and invoked
- Methods/functions selected by signature
  - Name must match (e.g. WarmUp)
  - Arguments must match
    - Data types must match
    - Including hidden arguments
  - Return types not used for selection
- Polymorphism creates ambiguities
  - Animal::WarmUp();
  - Mammal::WarmUp();

Dispatch (II)

- Static Dispatch
  - Functions selected at compile time
  - Based on declared data types
  - This is the C++ default
- Dynamic Dispatch
  - Functions selected at run-time
  - Based on run-time data type of object
    - Run-time type is "sticky"
    - But does not apply to copies
    - Signalled by the 'virtual' keyword in C++
Back to WarmUp()

class Animal {
  public:
    bool warmup() {return false;}
    void Behave() {warmup();}
};
class Mammal : public Animal {
  public:
    bool warmup() {shiver(); return true;}
};
Mammal m;
m.Behave();

• Note that virtual keyword never appears
• Therefore dispatch will be static

Static Dispatch: WarmUp()

• ‘m’ is created as a Mammal
• m.Behave() passes m to Animal::Behave()
  – m is the hidden argument
  – The hidden argument is of type Animal
  – Polymorphism
• In Animal::Behave(), ‘this’ is of type Animal*
• Animal::Behave() calls WarmUp()
  – ‘*this’ is the hidden argument
  – ‘*this’ is of type Animal
  – So Animal::WarmUp() is called

WarmUp() version 2

class Animal {
  public:
    virtual bool warmup() {return false;}
    void Behave() {warmup();}
};
class Mammal : public Animal {
  public:
    virtual bool warmup() {shiver(); return true;}
};
Mammal m;
m.Behave();

• ‘m’ is created as a Mammal
• m.Behave() passes m to Animal::Behave()
  – m is the hidden argument
  – The hidden argument is of type Animal
  – m’s run-time type is still Mammal
• Animal::Behave() calls WarmUp()
  – ‘*this’ is the hidden argument
  – The run-time type of ‘*this’ is still Mammal
  – So Mammal::WarmUp() is called