Lecture 09a
Multiple Inheritance
Oct. 17th, 2017

Announcements
• No quiz today or next week 😊
• Recitations this week: Google Test (Part 2)
• Next midterm: two weeks from today
  – Halloween! (spooky)
• PA4 grading
  – If your code crashes on a test word…
    • We can remove that word(s) and regrade
    • 25% penalty
• PA6 due next Tuesday (more in a bit)
• PA5 due today
  – Any questions?

PA6

Professionalism
• Frustration
  – So many details
  – Compounding each other
  – Ambiguities – what does he want?
  – Pressure – big assignment, automatically graded
    • Like a demo…
• How you respond personally matters
  – Advocate for yourself
    • The ‘sentiment’ error was ours – so we fixed it
    • Crashing on test words – not our fault, but we adjusted
  – But tone matters!
    • Subject line: “I have a problem with…” – NO!
    • Avoid anger, snarkiness, ego-centrism
    • Focus on the disagreement, possible solutions
• This applies to peers as well as teachers/bosses
• This applies across all forums: in-person, email, piazza…

OO Topics Still to Discuss
• Shadowing
• Multiple Inheritance
  – New to C++ (not in Java)
• Pure Virtual Classes
  – Replaces Java interfaces
  – Benefits from multiple inheritance
• Slicing
  – A problem new to C++ (not in Java)

Inheritance Example
class Equine : public Mammal {
    protected:
        string name;
};
class Quagga : public Equine {
    public:
        Quagga() {name = “Pete”;}     
    

10/17/17
Inheritance Questions

• Is the previous code legal?
• Where does the field ‘name’ come from?
  – It is inherited from Equine
• Why can Quagga access ‘name’?
  – Because its protected
  – Child classes can access protected classes

Example II

```cpp
class Equine : public Mammal {
protected:
    string name;
};

class Quagga : public Equine {
public:
    Quagga() {name = "Pete";}
protected:
    string name;
};
```

Questions II

• Is this legal?
• How many fields called ‘name’ are there?
  – 2
  – 1 defined by Equine
  – 1 defined by Quagga
• Which field is set to ‘Pete’?
  – The one in Quagga
• What is the value of ‘name’ in Equine?
  – Uninitialized
• How do we initialize the version in Equine?

Example III

```cpp
class Equine : public Mammal {
protected:
    string name;
};

class Quagga : public Equine {
public:
    Quagga() {name = "Pete"; Equine::name = "Peter";}
protected:
    string name;
};
```

Shadowing

• Shadowing is when:
  – two fields have the same name
  – two non-virtual methods have the same signatures
• Avoid shadowing when possible
• When fields or methods are shadowed
  – The compiler will select the version defined in the current class
  – Otherwise, the one defined in the immediate parent
  – Otherwise, the parent of the parent
  – And so on up the hierarchy...

Explicit References

• Fully-qualified names over-ride this default
• Inside a Quagga method:
  – ‘name’ refers to the name in Quagga
  – ‘Equine::name’ refers to the name in Equine
• You can always use fully qualified names
  – Inside a Quagga method, you can use Quagga::name
• Generally, fully qualified names are only used when necessary
  – To resolve shadowed names
  – To resolve unknown contexts
    • E.g. in .cpp files to select the method being defined
    • Animal::Lifespan() (...)
New Topic: Multiple Inheritance

- In Java, every class has 1 parent
  - Except for 'Object', which is unique
- In C++, the number of parents varies
  - By default, classes have no parents
  - Many classes explicitly inherit 1 parent
  - A few classes have > 1 parent
    - This is called multiple inheritance
- In C++, the inheritance relation forms a DAG
  - Directed acyclic graph

Multiple Inheritance Example

class Quagga : public Equine,
  public ZooProperty {
public:
  Quagga() {name = “Pete”;}
protected:
  string name;
};

Why?

- Imagine that all known Quaggas are zoo animals
  - The Quagga "is-a" Equine
  - But a Quagga also "is-a" zoo property
    - It might have an inventory number
    - A page number in the zoo catalog
  - Quagga inherits Equine to avoid doubly-implementing methods common to all Equines
  - Quagga inherits from ZooProperty to avoid double-implementing methods common to all zoo property

Hidden Danger

- Let Animals have a field called 'lifespan'
  - Imagine that all ZooProperties are Animals
  - So ZooProperties inherit Animal
  - Mammals also inherit Animal
  - And Equine inherits Mammal
  - And Quagga inherits Equine
  - And Quagga inherits ZooProperty
  - What does a Quagga object look like?

What’s in a Quagga?

- Quagga contains Equine which contains Mammal which contains Animal...
- Quagga contains ZooProperty which contains Animal...
- Quaggas contain two Animals
  - And therefore two 'lifespan' fields
  - Which may or may not have the same value
  - Almost invariably results in bugs

Quagga Visualized

- Quagga’s contain an Equine
- Quagga’s contain a ZooProperty
- Two lifespans!
Orthagonality

- Two classes are *orthogonal* if they contain no parents in common
- Only use multiple inheritance if the classes being inherited are orthogonal

The Original Multiple Inheritance Example

- Object oriented programming was originally invented to support graphics
  - Early windowing system
- Windows had different types (old monitors)
  - Binary windows
  - Grayscale windows
  - Color windows (rare back then)
- Windows also had different types of frames
  - Frameless
  - Thin, flat boundaries
  - Beveled boundaries (a big deal at the time)

Windows (cont.)

- Problem
  - Any type of window could have any type of frame
- Solution: Multiple inheritance
  - Window class had 3 child classes
    - Binary, Gray, and Color
  - Frame class had 3 child classes
    - Frameless, ThinFrame, BeveledFrame
  - Created cross-product of classes
    - One inherited Binary & Frameless
    - One inherited Binary and ThinFrame
    - ... 
  - Cross-produce classes empty, except for parent classes
  - Sometimes called ‘mixins’