Queens College, CUNY,   Department of Computer Science
Object Oriented Programming in C++
CSCI 211 / 611
Summer 2018
Instructor: Dr. Sateesh Mane

September 7, 2018

Inheritance: Part III

- This lecture contains advanced topics about **inheritance**.

- The material in this lecture is **not for examination**.

1 Non-public inheritance

- There are actually three types of inheritance: **public**, **private** and **protected**.

  ```
  class B1 : public A;
  class B2 : private A;
  class B3 : protected A;
  ```
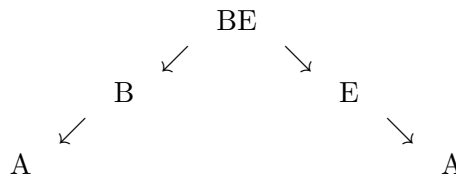
- The textbooks state that private or protected inheritance are rarely used.

- We shall employ public inheritance exclusively. ***It is by far the most common.***

2   Multiple inheritance

- A C++ class can inherit from more than one base class.

- Suppose there are two base classes `A1` and `A2`.

- A class `B12` can derive from both `A1` and `A2`.

```
class A1;
class A2;
class B12 : public A1, A2;
```

- This is called **multiple inheritance.**

- What we have studied up to now is called **single inheritance.**

- Multiple inheritance can lead to terrible complications.

  1. Suppose both `A1` and `A2` have a method "`f1()`" in their declarations.
  2. Which of the two "`f1()`" methods does an object of type `B12` invoke?
  3. To avoid ambiguity, the class `B12` must override `f1()`, even though the class `B12` may have nothing of its own to add to the method.
  4. This is the most obvious example of many complications with multiple inheritance.

- Recall the base class `A` and the derived classes `B` and `E` which both inherit from `A`.

- Now declare a class `BE` which inherits from **both `B` and `E`**.

- The structure of the inheritance tree (or chain) now looks like this.

```
                         BE
                   ↙           ↘
              B                     E
          ↙                             ↘
      A                                     A
```

- The above tree causes terrible problems.

  1. `B` "IS-A" `A` hence it contains an `A` object.
  2. `E` "IS-A" `A` hence it contains an `A` object.
  3. `BE` "IS-A" `B` and also `E`.
  4. Hence `BE` contains both a `B` object and an `E` object.
  5. However, *both of those* `B` *and* `E` *objects contain their own* `A` *objects*, **and the `A` objects in `B` and `E` are** <u>**not the same.**</u>
  6. Hence `BE` it contains **two copies of `A` objects.**
  7. It is undefined which of the `A` objects should be used, for example to access data (accessors) or to set data values (mutators).
  8. The compiler generates a warning, although it does not prevent compilation of the code.

- There is a mechanism to resolve the ambiguity so that the implicit `A` objects in the `B` and `E` components are the same object.

$$
\begin{array}{ccc}
 & \text{BE} & \\
\swarrow & & \searrow \\
\text{B} & & \text{E} \\
\searrow & & \swarrow \\
 & \text{A} & \\
\end{array}
$$
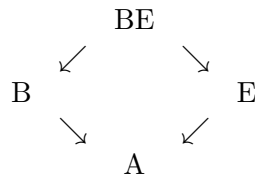
- However, in general multiple inheritance leads to a mess.

- Multiple inheritance is such a bad feature that it is **not supported in the Java language.**

- That is to say, Java supports only single inheritance.

- This was a deliberate design decision in Java because of the problems caused by multiple inheritance in C++.