(12-2) OOP: More Polymorphism in C++
D & D Chapter 12

Instructor - Andrew S. O'Fallon
CptS 122 (April 5, 2019)
Washington State University

Washington State University
World Class. Face to Face.

Key Concepts

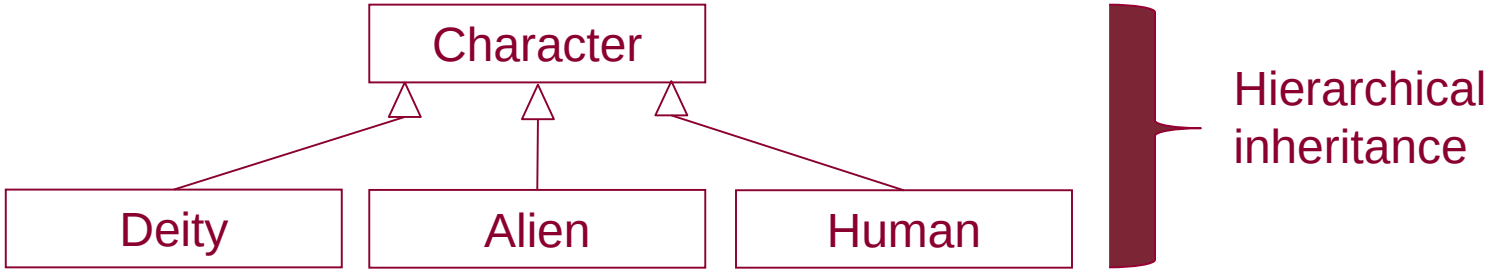
- Downcasting
- Keyword `dynamic_cast`
- Keyword `static_cast`

What is Downcasting?

- The compiler provides a means to access derived-class-only members via a base-class pointer that refers to a derived-class object
  - We can explicitly cast the base-class pointer to a derived-class pointer – this is downcasting
  - C++ provides a few different ways for downcasting – some are safer than others
- Be very careful and cautious when working with a downcast!

Hierarchical Inheritance - Inheritance Structure of Video Game Characters (I)

- Let's revisit our example - Deity, Alien, and Human classes are derived from a base class Character:



Hierarchical Inheritance - Inheritance Structure of Video Game Characters (II)

- Recall the following for the base class Character:

```
class Character
{
        public:
                // Will not show setters, getters, constructors explicitly
                virtual ~Character (); // virtual destructor
                virtual void move (int x, int y);
                virtual void render ();
        private:
                int mPosX;
                int mPosY;
                Image mSprite;
};
```


Hierarchical Inheritance - Inheritance Structure of Video Game Characters (III)

- Let's add some extra attributes to the three derived classes, which are not accessible in the base class

```
class Deity : public Character // public inheritance
{
        public:
        private:
                string mType;
};
class Alien : public Character
{
        public:
        private:
                int mPower;
};
class Human : public Character
{
        public:
        private:
                char mGender;
};
```


Explicit Downcast – C Style

- Given the following fragment:

```
Character *pBase1, *pBase2, *pBase3;
pBase1 = new Deity; // Character * - base-class pointer
pBase2 = new Alien;
pBase3 = new Human;

// We will NOT be able to access the derived-class-only members
// through the base-class pointer unless we downcast to each
// of the specific derived-class types
((Deity *) pBase1)->mType
((Alien *) pBase2)->mPower
((Human *) pBase3)->mGender
```

- Generally, this form of downcasting is not considered type-safe

Dynamic Downcast – C++ Style

- *Safely* converts pointers and references to derived-class types in an inheritance hierarchy – allows for runtime checks – only works with ***polymorphic*** types (i.e. must have at least one virtual function)

```
// Note: if the dynamic cast is successful, then dynamic_cast
// returns a value of the new type, i.e. Deity *, Alien *, or
// Human *. If the cast fails, then null pointer is returned for pointers
// or an exception is thrown for references.
(dynamic_cast <Deity *> (pBase1))→mType
(dynamic_cast <Alien *> (pBase2))→mPower
(dynamic_cast <Human *> (pBase3))→mGender
```

- If you want to check the result of `dynamic_cast`, then consider (runtime check):

```
Deity *pDeity = dynamic_cast <Deity *> (pBase1);
if (pDeity != nullptr)  // Was address to object returned?
{
        // pDeity->mType – access mType
}
```


Static Downcast – C++ Style

- Not *guaranteed* to *safely* convert pointers and references to derived-class types in an inheritance hierarchy – avoids the cost of a runtime check, but only safe if program has other logic to guarantee that a valid cast can be performed

```
(static_cast <Deity *> (pBase1))→mType
(static_cast <Alien *> (pBase2))→mPower
(static_cast <Human *> (pBase3))→mGender
```


Summary

- We can explicitly cast the base-class pointer to a derived-class pointer – this is downcasting – to access members in the derived-class that are not in the base-class
- The casts should only be performed between types that are in the same inheritance hierarchy

References

- P.J. Deitel & H.M. Deitel, *C++ How to Program (9th Ed.)*, Pearson Education , Inc., 2014.
- J.R. Hanly & E.B. Koffman, *Problem Solving and Program Design in C (7th Ed.)*, Addison-Wesley, 2013