C++ Classes

CSE 1320

Alex Dillhoff

University of Texas at Arlington

Fall 2021

Classes

The **class** is the central feature of C++.

Classes are user-defined types, like `structs`, that *encapsulate* a particular idea or concept.

That is, the types defined via classes provide more than just a basic definition of an entity.

Classes

A popular analogy for understanding classes is to think of a library.

Libraries are created for a particular purpose and represent types and functions related to that purpose.

Because classes can be defined for a virtually infinite number of classes, there are many additional features in C++ that exist to support them.

Classes

The three most important types of classes to consider are:

1. Concrete classes
2. Abstract classes
3. Class hierarchies

Concrete Classes

A **concrete class** is meant to behave just like any other built-in type in the language.

Consider a numerical type such as `int`.

It represents a number and also has some set of associated operations.

Concrete Classes

When we use `int`s in our program, we don't think about the actual operations.

We understand that when we add, multiply, divide, or subtract two `int`s, it will produce some output.

Concrete Classes

Contrast this with the concept of a string in C.

The data itself is represented as a character array and the operations were explicitly defined through library functions.

Concrete Classes

In general, a **concrete class** is one in which its representation is part of its definition.

This allows the programmer to initialize them on the stack, use them with other objects, copy them, and more.

Concrete Classes

Example: The Car class

Concrete Classes

Example: The Complex Number Class

Constructor and Destructors

Constructors and destructors allow us to define exactly what happens to our objects when they are created or destroyed.

The destructor is called whenever the object is deleted.

For statically allocated objects, this happens when the object becomes out of scope.

Constructor and Destructors

If the object relies on any members that include heap memory, those members can be safely deleted.

This is another way that C++ offers safter handling of dynamically allocated memory.

Abstract Classes

In contrast to a **concrete class**, an **abstract class** does not include the concrete representation as part of its definition.

Instead, it defined an interface for which concrete classes must *inherit*.

Abstract Classes

An **abstract class** is usually identified by its use of the `virtual` keyword.

This keyword indicates that the specific item may be redefined in a class derived from the abstract class.

Abstract Classes

Example: The Shape Class

Abstract Classes

Abstract classes are typically use to define an abstract concept that will later be inherited by one more concrete types.

This is most commonly seen in class hierarchies.

Class Hierarchies

Hierarchies exist in many different contexts.

It is useful to design programs that take advantage of them.

This allows us to take advantage of ideas already created in higher-level classes without rewriting the foundational functions.

Inheritence

Classes are derived from other classes via **inheritence**.

For example, if we have created the class `Animal`, we could then create a new class that inherits from it.

```
class Dog : public Animal {
    // ...
}
```

Class Hierarchies

Example: Shapes