# Abstract Classes
# and
# Pure Virtual Functions

Spring 2018

# “virtual” function/method

1. Allow it to be redefined in a derived class
   - Syntax:

     virtual  void  doSomething() ;

2. Invoking a virtual Function Through a Base-Class Pointer or Reference
   - The program will choose the correct derived-class function dynamically (i.e., at execution time) based on the object type—not the pointer or reference type.
   - Known as dynamic binding or late binding.

## Software Engineering Observation 12.5

Once a function is declared `virtual`, it remains `virtual` all the way down the inheritance hierarchy from that point, even if that function is not explicitly declared `virtual` when a derived class overrides it.

## Error-Prevention Tip 12.1

Apply C++11's `override` keyword to every overridden function in a derived-class. This forces the compiler to check whether the base class has a member function with the same name and parameter list (i.e., the same signature). If not, the compiler generates an error.

# “virtual” destructor

1. If a derived-class object with a non-virtual destructor is destroyed by applying the delete operator to a base-class pointer to the object, the C++ standard specifies that the behavior is undefined. So it is important to create a public virtual destructor in the base class
   - Syntax:

     virtual ~MyCPPClass() { } ;

2. if an object in the hierarchy is destroyed explicitly by applying the delete operator to a base-class pointer, the destructor for the appropriate class is called based on the object to which the base-class pointer points.

3. The base-class destructor automatically executes after the derived-class destructor

## Common Programming Error 12.1

Constructors cannot be `virtual`. Declaring a constructor `virtual` is a compilation error.

# “final” method

1. If a derived-class object with a non-virtual destructor is destroyed by applying the delete operator to a base-class pointer to the object, the C++ standard specifies that the behavior is undefined. So it is important to create a public virtual destructor in the base class
   – Syntax:
   virtual void doSomething() final { } ;

2. “final” indicates that it cannot be overridden in any derived class—this guarantees that the base class’s final member function definition will be used by all base-class objects and by all objects of the base class’s direct and indirect derived classes

# “final” class

“final” indicates that it cannot be overridden in any derived class—this guarantees that the base class’s final member function definition will be used by all base-class objects and by all objects of the base class’s direct and indirect derived classes

```
class MyClass final // this class cannot be a base class
{
   // class body
};
```

## Software Engineering Observation 12.7

Polymorphic programming can eliminate the need for `switch` logic. By using the polymorphism mechanism to perform the equivalent logic, you can avoid the kinds of errors typically associated with `switch` logic.

# “abstract” class

1. “incomplete” class
2. Used as a base class to derive from
3. Cannot instantiate or create instances directly from the abstract class
4. Derived classes must complete the class definition to be able to create objects

# “pure” virtual

1. A pure virtual function is specified by placing “= 0” in its declaration, as in

   ```
   virtual void draw() const = 0;
   // pure virtual function
   ```

2. The “= 0” is a pure specifier.

3. Pure virtual functions typically do not provide implementations.

4. Pure virtual functions are used when it does not make sense for the base class to have an implementation of a function, but you want all concrete derived classes to implement the function.

# Important notes about abstract class and pure virtual methods

1. Although we cannot instantiate objects of an abstract base class, we can use the abstract base class to declare pointers and references that can refer to objects of any concrete classes derived from the abstract class.

2. Programs typically use such pointers and references to manipulate derived-class objects polymorphically.

## Software Engineering Observation 12.9

An abstract class defines a common public interface for the various classes in a class hierarchy. An abstract class contains one or more pure `virtual` functions that concrete derived classes must override.

## Software Engineering Observation 12.10

An abstract class has at least one pure `virtual` function. An abstract class also can have data members and concrete functions (including constructors and destructors), which are subject to the normal rules of inheritance by derived classes.

# Exercise

1. Define an abstract “Containable” class with one method of
   - bool contains(string aWord) ;

2. Define a “Movie” class that inherits from the Containable class and demonstrate that when you are calling methods in Containable, it is actually executing methods in the Movie class

# Abstract class with pure virtual methods

- An abstract class can have abstract methods.
  - It cannot be instantiated, and
  - all of the methods listed in an interface must be overridden in the derived classes.

- The purpose of an abstract class is to specify behavior for other classes.

- It is often said that an abstract is like a “contract,” and when a class inherits the abstract class, it must adhere to the contract.

- Pure virtual method:
  - the methods that have no bodies, only headers that are terminated by “= 0;”

# How to use C++ abstract class

- All abstract methods specified by an abstract class are generally declared as “public”.
- A class can inherit and implement one or more abstract classes.
- By convention, a lot of C++ abstract classes with pure virtual methods (e.g. “pure abstract class”) starts with an “I” for “interfaces.”

- It is important to declare a “virtual destructor” in the abstract class so that the right object can be destroyed.

```
public class TriangleShape : Shape, IPrintable, IMeasurable
{
};

abstract class  IPrintable
{
    public:
       ~IPrintable() { }
       void print() ;
};
```

# Exercise

1. Define the “Item” class that contains an id, a description and a price

2. Define a “Book” class that inherits from the Item class but add one more data member for the title of the book

3. Define a “IComparable” pure abstract class to compare two IComparable objects

4. Modify the Book class to implement the IComparable interface by comparing the prices

5. Provide a method to printInOrder that accepts two IComparable objects and print out the items in order

6. Write the main program that creates two Books and print out in the order of their prices.

7. How do you also print out the same two Books in the order of their titles instead of price?