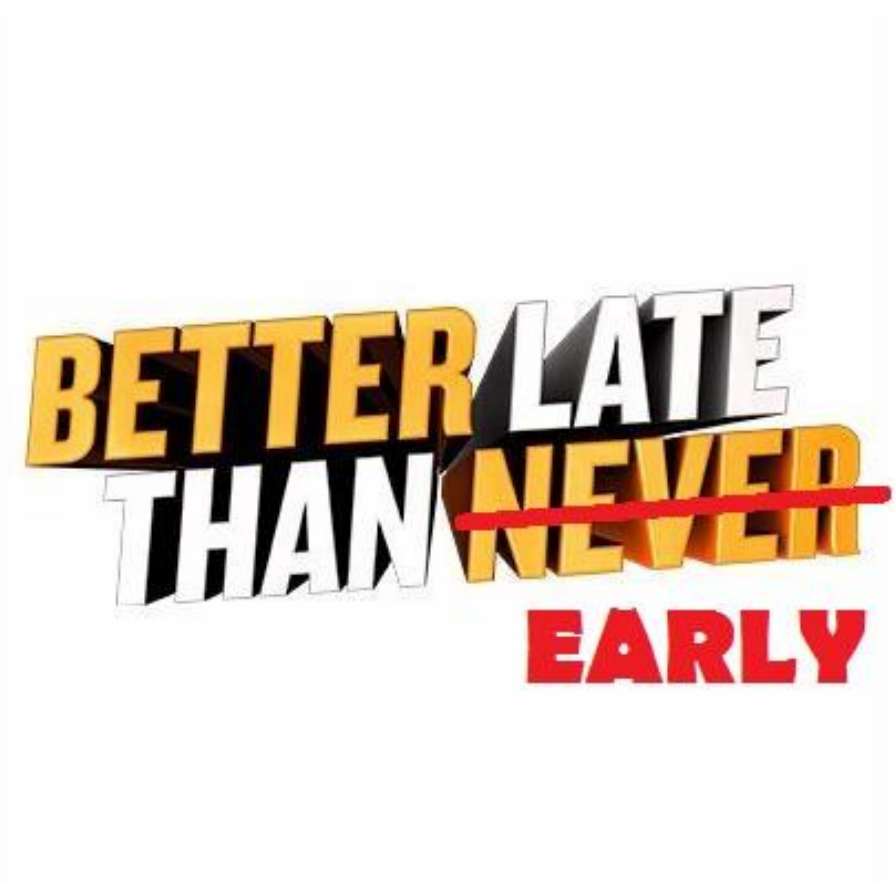
Object-oriented Programming

Pure Virtual Functions &
Abstract Classes

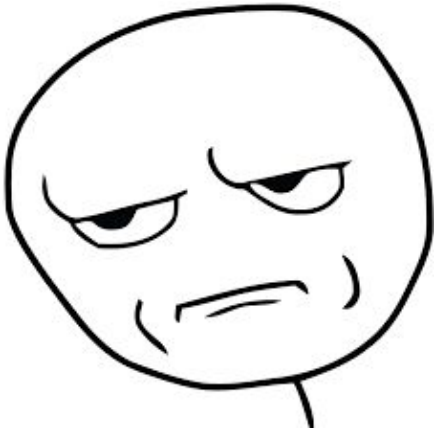
- **In object-oriented programming…**



RECAP: Virtual Functions

- Used to enforce ***Late Binding*** or ***Runtime Binding***

- The compiler dynamically chooses the correct function to call based on the type of object reference used for the call



Abstract Class

- There are cases in which it's useful to define classes that are never instantiated. Such classes are called **abstract base classes** or simply **abstract classes**

- Abstract base classes are incomplete — derived classes must define the "missing pieces"

Abstract Class

- An abstract class provides a base class from which other classes can inherit
- Classes that can be used to instantiate objects are called **concrete classes**
- An abstract can inherit another abstract class

Example

- Suppose that we have an application that works with **2D** shapes
- We have a class called **2D** with a function draw(). But… what shape would the function draw? Should draw() be implemented in base class??
- We can extend the class with derived classes such as **Square**, **Triangle** & **Circle**. And provide implementation for draw()

Abstract Class

- Abstract base classes are too generic to define real objects
- Concrete classes provide the specifics that make it reasonable to instantiate objects
- Abstract classes support the ***principle of Polymorphism***

Abstract Class

- Although we cannot create objects of an abstract class, we can use the *abstract base class* to declare pointers containing objects of its *derived class*

- Abstract class may contain concrete functions, which can be used (called) using child class instances

Pure Virtual Functions

- A class is made abstract by declaring one or more of its virtual functions to be "**pure**"

- A pure virtual function is specified by placing "**= 0**" in its declaration

- **Example:** **virtual void draw() = 0;**

Virtual vs Pure Virtual

- A virtual function has an implementation in the base class; a pure virtual function does not have an implementation in the base class

- Virtual functions ***can be*** overriden by the derived classes; pure virtual functions ***must be*** overriden by the derived classes

Example

```
 class Employee
{
        public:
 virtual void work( ) = 0;
};


 class Pilot: public Employee
{
 public:
 void work( ) { cout << “Flies a plane”; }
};


 class Technician: public Employee
 {
        public:
 void work( ) { cout << “Checks airplane for faults”;  }
}
```

Case Study

A company pays its employees weekly.

The employees are of four types: **Salaried employees** are paid a fixed weekly salary regardless of the number of hours worked, **hourly employees** are paid by the hour and receive overtime pay for all hours worked in excess of 40 hours, **commission employees** are paid a percentage of their sales and **base-salary-plus-commission employees** receive a base salary plus a % of their sales. For the current pay period, the company has decided to reward base-salary-plus-commission employees by adding 10 percent to their base salaries.

Next Lecture

- Singleton class (through coding exercise)