Polymorphism

“having many forms”

Polymorphism

When a call to a member function executes different code depending on the type of object that invokes the function.

Two closely related terms…

- Polymorphism
  - When a call to a member function executes different code depending on the type of object that invokes the function.

- Virtual function
  `virtual void example();`
  - A base-class function that is declared as **virtual**, indicating to the compiler that it should wait until run-time to determine which version of the function should run.
  - A virtual function can be overridden if it is re-defined in a child class.

Some Vocabulary

- Pure virtual function (also known as abstract function)
```
virtual void example() = 0;
```
  - A virtual function that has no definition in the base class.
  - Used when you are intending for child classes to implement the function.

- Abstract class
  - Any class that has one or more pure virtual functions.
  - An abstract class cannot be instantiated (i.e. you cannot create an object out of an abstract class).

Some Vocabulary

- override specifier
  ```
  virtual void example() override;
  ```
  - Used when you want to tell the compiler that this function is intended to override some function in the base class.
  - Not required but good to use because you lower the chance of bugs

- final specifier (for a function)
  ```
  virtual void example() final;
  ```
  - Used when you want to tell the compiler that no child class is allowed to override this function.

- final specifier can also be applied to an entire class:
  ```
  class Elephant final : public Animal {}
  ```
  - In this context, **final** means that no child class can exist for Elephant. In other words, no class can inherit from Elephant.