Abstract Classes

- Sometimes implementation of a method cannot be provided in a base class because we don't know the implementation. Such methods are called *pure virtual* methods or *abstract*.
- For example, in a `Shape` base class, we cannot provide implementation of function `draw()` in `Shape`, but we know every derived class must have implementation of `draw()`.
- Similarly an `Animal` class doesn't have implementation of `move()` (assuming that all animals move), but all animals must know how to move.
- A pure virtual function in C++ is a virtual function for which there is no implementation, it's only declared. A pure virtual function is declared by assigning 0 in declaration, e.g.,

```
    public:
            // Pure Virtual Function
            virtual void show() = 0;
```

- Any class that includes a *pure virtual* function is considered *abstract* and objects may not be created directly from it.
- For example, if we take the `Shape` class provided in the Polymorphism example, and change the `getArea()` method from,

```
            // Note: getArea() needs to be virtual
            virtual float getArea()
            {
                    cout << "getArea():";
                    return 0;
            }
```

    to

```
            // Note: getArea() in Shape is pure virtual
            virtual float getArea() = 0;
```

- Now `getArea()` in `Shape` has become *pure virtual* and has no base implementation, and `Shape` is now an abstract class.
- Notice that you cannot directly create instances of abstract classes such as `Shape`. Once `getArea()` in `Shape` is pure virtual the compiler will complain if you try to create a `new Shape()`, e.g.,

```
            Shape *spot = new Shape(6, 6);
```

    Will now give an "Allocating object of abstract type" error.
- When creating a virtual class you also need to declare the abstract base class's destructor as `virtual, e.g.,`

```
    virtual ~Shape() = default;
```

- `O`therwise the compiler will use the base class destructor, which can cause problems. If your destructor is not virtual the compiler will give warnings on your `delete` calls, "Delete called on 'Shape' that is abstract but has non-virtual destructor."