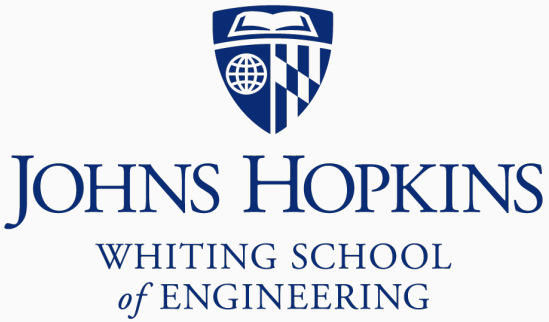
Abstract classes

Ben Langmead

ben.langmead@gmail.com

www.langmead-lab.org



Source markdown available at github.com/BenLangmead/c-cpp-notes

Abstract classes

What does the `= 0` mean here?

```
class Piece {
public:
    ...
    virtual bool legal_move_shape(CharPair, CharPair) const = 0;
    //                                                    ^^^^

    virtual bool legal_capture_shape(CharPair start,
                                     CharPair end) const
    {
        return legal_move_shape(start, end);
    }
    ...
};
```

Pure virtual functions

Declaring a virtual member function and adding `= 0` makes it a *pure* `virtual` function

When we declare a pure virtual function:

- We do not give it an implementation
- It makes the class it's declared in an *abstract class*
  - We cannot create a new object with the type, though we may be able create an object with a derived *type*

Pure virtual functions

```
class Shape {
public:
    virtual double size() const = 0;
    ...
};

class Shape2D : public Shape {
    ...
};

class Circle : public Shape2D {
public:
    virtual double size() const {
        return 3.14 * r * r;
    }
    ...
private:
    double r;
    ...
};
```

Pure virtual functions

```
int main() {
    Circle c(1.0 / sqrt(3.14));
    cout << c.size() << endl;
    return 0;
}
```

```
$ g++ -c shape_virt.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o shape_virt shape_virt.o
$ ./shape_virt
1
```

Pure virtual functions

```
int main() {
    Shape s;
    return 0;
}
```

```
$ g++ -c shape_virt2.cpp -std=c++11 -pedantic -Wall -Wextra
shape_virt2.cpp: In function 'int main()':
shape_virt2.cpp:25:11: error: cannot declare variable 's' to be of abstract type
'Shape'
     Shape s;
           ^
shape_virt2.cpp:7:7: note:   because the following virtual functions are pure
within 'Shape':
 class Shape {
       ^~~~~
shape_virt2.cpp:9:20: note:     virtual double Shape::size() const
     virtual double size() const = 0;
                    ^~~~
```

Pure virtual functions

```
int main() {
    Shape2D s2d;
    return 0;
}
```

```
$ g++ -c shape_virt3.cpp -std=c++11 -pedantic -Wall -Wextra
shape_virt3.cpp: In function 'int main()':
shape_virt3.cpp:25:13: error: cannot declare variable 's2d' to be of abstract
type 'Shape2D'
     Shape2D s2d;
             ^~~
shape_virt3.cpp:12:7: note:   because the following virtual functions are pure
within 'Shape2D':
 class Shape2D : public Shape { };
       ^~~~~~~
shape_virt3.cpp:9:20: note:     virtual double Shape::size() const
     virtual double size() const = 0;
                    ^~~~
```

Pure virtual functions

"Cannot declare variable s to be of abstract type Shape"

When a `class` has one or more pure virtual functions, it cannot be instantiated; it is *abstract*

- Similar to `abstract class` and `interface` in Java

The derived `Circle` class *can* be instantiated because it provides an implementation for the (only) pure virtual, `size()`

Abstract classes

Another way to make a `class` abstract is give it only non-`public` constructors

```
class Piece {
public:
    ...
protected:
    // This is the only constructor
    Piece( bool is_white ) : _is_white( is_white ){ }
    ...
};
```

Can't instantiate `Piece` because constructor can't be called from the outside

Abstract classes

Derived class can still use protected constructor in base class:

```
class Knight : public Piece {
    ...
  public:
    // Knight constructor calls Piece constructor
    // OK because it's protected
    Knight( bool is_white ) : Piece( is_white ) {}
    ...
};
```