

# 601.220 Intermediate Programming

Function hiding and abstract classes

# Function hiding

```
// hiding.cpp
1  #include <iostream>
2
3  class Base {
4  public:
5      void fun(int i) const { std::cout << "Base " << i << std::endl; }
6  };
7
8  class Derived: public Base {
9  public:
10     void fun(char c) const { std::cout << "Derived " << c << std::endl; }
11 };
12
13 int main() {
14     Derived d;
15     d.fun(76);
16     d.fun('a');
17     return 0;
18 }
```

\$ g++ -o hiding hiding.cpp -std=c++11 -pedantic -Wall -Wextra

\$ ./hiding

Derived L  
Derived a

# Function hiding

```
// hiding2.cpp
1  #include <iostream>
2
3  class Base {
4  public:
5      virtual void fun(int i) const { std::cout << "Base " << i << std::endl; }
6  };
7
8  class Derived: public Base {
9  public:
10     void fun(char c) const { std::cout << "Derived " << c << std::endl; }
11 };
12
13 int main() {
14     Derived d;
15     d.fun(76);
16     d.fun('a');
17     return 0;
18 }
```

```
$ g++ -o hiding2 hiding2.cpp -std=c++11 -pedantic -Wall -Wextra
```

```
$ ./hiding2
```

```
Derived L
```

```
Derived a
```

# Function hiding

```
// hiding3.cpp
1  #include <iostream>
2
3  class Base {
4  public:
5      void fun(int i, int j) const { std::cout << "Base " << i << " " << j << std::endl; }
6  };
7
8  class Derived: public Base {
9  public:
10     void fun(char c) const { std::cout << "Derived " << c << std::endl; }
11 };
12
13 int main() {
14     Derived d;
15     d.fun(76, 77);
16     d.fun('a');
17     return 0;
18 }
```

```
$ g++ -o hiding3 hiding3.cpp -std=c++11 -pedantic -Wall -Wextra
```

```
hiding3.cpp: In function int main():
```

```
hiding3.cpp:15:17: error: no matching function for call to Derived::fun(int, int)
    d.fun(76, 77);
               ^
```

```
hiding3.cpp:10:10: note: candidate: void Derived::fun(char) const
```

```
    void fun(char c) const { std::cout << "Derived " << c << std::endl; }
           ^~~~
```

# Function hiding

```
// hiding4.cpp
1  #include <iostream>
2
3  class Base {
4  public:
5      virtual void fun(int i, int j) const { std::cout << "Base " << i << " " << j << std::endl; }
6  };
7
8  class Derived: public Base {
9  public:
10     void fun(char c) const { std::cout << "Derived " << c << std::endl; }
11 };
12
13 int main() {
14     Derived d;
15     d.fun(76, 77);
16     d.fun('a');
17     return 0;
18 }
```

\$ g++ -o hiding4 hiding4.cpp -std=c++11 -pedantic -Wall -Wextra

hiding4.cpp: In function int main():

hiding4.cpp:15:17: error: no matching function for call to Derived::fun(int, int)

```
    d.fun(76, 77);
        ^
```

hiding4.cpp:10:10: note: candidate: void Derived::fun(char) const

```
    void fun(char c) const { std::cout << "Derived " << c << std::endl; }
        ~~~
```

# Function hiding

```
// hiding5.cpp
1  #include <iostream>
2
3  class Base {
4  public:
5      virtual void fun(int i, int j) const { std::cout << "Base " << i << " " << j << std::endl; }
6  };
7  class Derived: public Base {
8  public:
9      void fun(char c) const { std::cout << "Derived " << c << std::endl; }
10     void fun(int i, int j) const override { std::cout << "Derived " << i << " " << j << std::endl; }
11 };
12 int main() {
13     Derived d;
14     d.fun(76, 77);
15     d.Base::fun(76, 77);
16     ((Base&) d).fun(76, 77);
17     ((Base) d).fun(76, 77);
18     return 0;
19 }

$ g++ -o hiding5 hiding5.cpp -std=c++11 -pedantic -Wall -Wextra

$ ./hiding5

Derived 76 77
Base 76 77
Derived 76 77
Base 76 77
```

## Pure `virtual` functions

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

What does this `= 0` mean here?

# Pure `virtual` functions

- Declaring a `virtual` member function and adding `= 0` makes it a pure `virtual` function
- In the memory layout, it means set the address of the virtual function to `nullptr`
- When we declare a pure `virtual` function:
  - We do not give it an implementation
  - Makes the class its declared in an abstract class
  - We cannot create a new object with the type, though we might be able to create an object from a derived type



# Pure **virtual** functions

```
// shape.cpp
1  #include <iostream>
2  #include <cmath>
3
4  class Shape {
5  public:
6      virtual double size() const = 0;
7  };
8
9  class Shape2D : public Shape {};
10
11 class Circle : public Shape2D {
12 public:
13     Circle(double r) : Shape2D(), r(r) {}
14     double size() const { return 3.14 * r * r; }
15 private:
16     double r;
17 };
18
19 int main() {
20     Circle c(1.0 / sqrt(3.14));
21     std::cout << c.size() << std::endl;
22     return 0;
23 }

```

\$ g++ -o shape shape.cpp -std=c++11 -pedantic -Wall -Wextra

\$ ./shape

1

# Pure virtual functions

```
// shape2.cpp
1  #include <iostream>
2  #include <cmath>
3
4  class Shape {
5  public:
6      virtual double size() const = 0;
7  };
8
9  class Shape2D : public Shape {};
10
11 class Circle : public Shape2D {
12     public:
13         Circle(double r) : Shape2D(), r(r) {}
14         double size() const { return 3.14 * r * r; }
15     private:
16         double r;
17     };
18
19     int main() {
20         Shape s;
21         return 0;
22     }
```

```
$ g++ -o shape2 shape2.cpp -std=c++11 -pedantic -Wall -Wextra
```

```
shape2.cpp: In function int main():
```

```
shape2.cpp:20:11: error: cannot declare variable s to be of abstract type Shape
```

```
    Shape s;
```

```
    ^
```

```
shape2.cpp:4:7: note:    because the following virtual functions are pure within Shape:
```

```
    class Shape {
```

```
    ~~~~~
```

```
shape2.cpp:6:20: note:                virtual double Shape::size() const
```

```
    virtual double size() const = 0;
```

```
    ~~~~
```

# Pure virtual functions

```
// shape3.cpp
1  #include <iostream>
2  #include <cmath>
3
4  class Shape {
5  public:
6      virtual double size() const = 0;
7  };
8
9  class Shape2D : public Shape {};
10
11 class Circle : public Shape2D {
12     public:
13         Circle(double r) : Shape2D(), r(r) {}
14         double size() const { return 3.14 * r * r; }
15     private:
16         double r;
17     };
18
19     int main() {
20         Shape2D s;
21         return 0;
22     }
```

```
$ g++ -o shape3 shape3.cpp -std=c++11 -pedantic -Wall -Wextra
```

```
shape3.cpp: In function int main():
```

```
shape3.cpp:20:13: error: cannot declare variable s to be of abstract type Shape2D
    Shape2D s;
    ~
```

```
shape3.cpp:9:7: note: because the following virtual functions are pure within Shape2D:
```

```
class Shape2D : public Shape {};
```

```
shape3.cpp:6:20: note: virtual double Shape::size() const
    virtual double size() const = 0;
    ~~~~~
```

# Abstract classes

- Cannot declare variable to be of abstract type
- 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()`
- Another way to make a class abstract is give it only non-public constructors
  - Can't instantiate the abstract class because the constructor can't be called from the outside
  - Derived class can still use protected constructor in the base class

# Abstract classes using non-public constructor example

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

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

# Facts about abstract classes

- A class is abstract if it has at least one pure virtual function or has only non-public constructor
- We can have pointers and references of abstract class type but not concrete objects
- If we do not override the pure virtual function in derived class, then derived class also becomes abstract class
- An abstract class can have constructors

# Quiz - answers

Consider the following classes and partial main function:

```
1  #include <iostream>
2
3  class Op {
4  public:
5      virtual int apply(int a, int b) = 0;
6  };
7  class Add {
8  public:
9      virtual int apply(int a, int b)
10     { return a + b; }
11 };
12 class Mul {
13 public:
14     virtual int apply(int a, int b)
15     { return a * b; }
16 };
17
18 int main() {
19     Op *a, *b;
20     HERE
21     cout << a->apply(b->apply(2, 3), 4);
22 }
```

What code could be added in place of HERE (line 20) to cause the program to print the output 20?

- A. a = new Add(); b = new Add();
- B. a = new Add(); b = new Mul();
- C. a = new Mul(); b = new Add();
- D. a = new Mul(); b = new Mul();
- E. None of the above