

Virtual Functions Solutions

- Explain what is meant by static and dynamic binding when calling a member function on a class instance
 - Static and dynamic binding refer to the two different ways in which a member function can be called on an instance
 - In static binding (the norm), the static type of the instance is used
 - In dynamic binding, the dynamic type is used

- What two conditions must apply for dynamic binding to occur?
 - The member function is called on a pointer or reference to a base class
 - The member function is declared virtual

- Give an example of a situation where dynamic binding is useful
 - Container of pointers to base class, populated with pointers to derived class instances
 - When a member function is called on an element, the member function will be called on the correct derived class instance
- Describe some of the advantages of dynamic binding
 - If we add new subclasses, the correct member function will be called, without adding any extra code

- What is a virtual member function?
 - A virtual member function has the virtual keyword before it
 - When a virtual member function is called on a pointer or reference to a base class, it will be called on the dynamic type of the instance

```
Circle c;
```

```
Drawable *d = &c;
```

```
d->draw();           // Calls Circle's draw()
```

- What output would you expect from the code sample on the next page?
 - “I'm a parent”
- What output would you expect if `Parent::print()` is made virtual?
 - “I'm a child”
- Convert the code sample to a working program and compare your answers with the output

```
class Parent {  
    public:  
        void print() { cout << "I'm a parent\n"; }  
};  
class Child : public Parent {  
    public:  
        void print() { cout << "I'm a child\n"; }  
};
```

```
Child child;  
Parent& p;  
child.print();
```

- Using the Drawable and Circle classes from the code on the next page, write a program which creates a vector of pointers to Drawable instances.
- It then stores a Circle instance in this vector, iterates through the vector and calls the draw() member of the vector's elements
- Remove the virtual keyword from Drawable::draw(). Compile and run your program again. What difference does this make?

```
class Drawable {  
public:  
    virtual void draw() { cout << "I'm a Drawable shape!\n"; }  
};
```

```
class Circle : public Drawable {  
public:  
    void draw() { cout << "I'm a Circle!\n"; }  
};
```

- Add this class to your program and append an instance of it to the vector

```
class Triangle : public Drawable {  
public:  
    void draw() { cout << "I'm a Triangle!\n"; }  
};
```

- What other changes do you need to make to the program before you can compile and run it?
 - **None!**