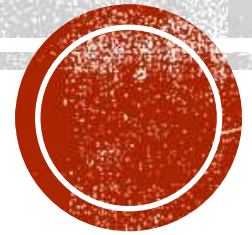


PROGRAMMING LANGUAGES 1

Inheritance & Polymorphism



INHERITANCE

- Derive a class from a base class
- “is a” relation between base class and sub class
 - Example: circle “is a” shape
- Members of base class are also members of subclass(es)

INHERITANCE - EXAMPLE

```
class shape {  
public:  
    shape();  
    ~shape();  
    double area();  
};
```

```
class square : public shape {  
private:  
    double length;  
public:  
    square();  
    ~square();  
    double getLength();  
    void setLength(double);  
};
```

INHERITANCE TYPES

Inheritance	public	protected	private
public	public	protected	✗
protected	protected	protected	✗
private	private	private	✗

ACCESS CONTROL

Access Specifier	public	protected	private
Base class	✓	✓	✓
Friend class/function	✓	✓	✓
Sub class(es)	✓	✓	✗
Other classes/functions	✓	✗	✗

FUNCTION OVERRIDING

- A subclass inherits member functions of base class
- A subclass can override a function from base class
 - Use same name and arguments
- Overridden function can have different access specifier in subclass

FUNCTION OVERRIDING - EXAMPLE

```
class square {  
protected:  
    double length;  
public:  
    square();  
    ~square();  
    double getLength() const;  
    void setWidth(double);  
    double area() const {  
        return length * length;  
    }  
};
```

```
class rectangle : public square {  
protected:  
    double width;  
public:  
    rectangle();  
    ~rectangle();  
    double getWidth() const;  
    void setWidth(double) const;  
    double area() const {  
        return length * width;  
    }  
};
```

CALLING PARENT CONSTRUCTOR

```
class square {  
protected:  
    double l;  
public:  
    square(double l) : l(l) {}  
    ...  
}  
};
```

```
class rectangle : public square {  
protected:  
    double w;  
public:  
    rectangle(double , double w) :  
        square(l), w(w) {}  
    ...  
};
```

VIRTUAL FUNCTIONS

- A pointer to base class can point to an object of the subclass
- By default, a pointer to base will call base functions
- A virtual function tells pointer to call implementation of subclass not base class
- Destructor should always be defined as virtual
 - Subclasses should call their own destructors

```
class shape {  
public:  
    shape();  
    virtual ~shape();  
    virtual double area();  
};
```

PURE VIRTUAL FUNCTION

- A virtual function that should have no definition
 - Definition does not make sense

```
class shape {  
public:  
    shape();  
    ~shape();  
    virtual double area() = 0;  
};
```

ABSTRACT CLASS

- Has at least one pure virtual function
- Cannot be instantiated
- An interface is an abstract class whose all functions are pure virtual functions

```
class shape { // abstract
    ...
    virtual double area() = 0;
};
```

```
int main() {
    shape s; // error
};
```

MULTIPLE INHERITANCE

- A subclass may have more than one base class
- Base class list is separated by commas
- Inherits members of all base classes

```
class D : public A, protected B, private C {  
...  
};
```

DIAMOND PROBLEM

- A subclass inherits from two base classes
- Both base classes have the same parent class
- Constructor/Destructor of parent class will be called twice

DIAMOND PROBLEM - EXAMPLE

```
class person {  
    string name;  
public:  
    person(string name): name(name) {}  
};  
class teacher : public person {  
...  
public:  
    teacher(string name):person(name) {}  
};
```

```
class student : public person {  
...  
public:  
    Student(int x):Person(x) {}  
};  
  
class TA : public teacher, public student  
{  
public:  
    TA(string name):student(name),  
    teacher(name) {}  
};
```

DIAMOND PROBLEM - SOLUTION

```
class person {
    string name;
public:
    person() = default;
    person(string name): name(name) {}
};

class teacher : virtual public person {
...
public:
    teacher(string name):person(name) { }
};
```

```
class student : virtual public person {
...
public:
    Student(string name):person(name) {}
};

class TA : public teacher, public
student {
public:
    TA(string name):student(name),
        teacher(name), person(name) {}
};
```

SUMMARY

- Inheritance
- Inheritance Types & Access Control
- Function Overriding
- Virtual Functions
- Pure Virtual Functions
- Abstract Class
- Multiple Inheritance
- Diamond Problem