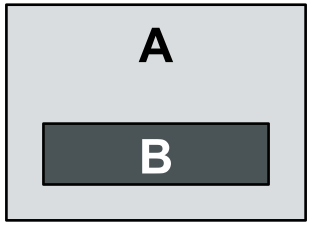
OBJECT RELATIONSHIPS

COMPOSITION

Concept	A has-a B B is part-of A The sum of the parts of A include B
Characteristics	B can only belong to one instance of A scope and lifetime of A and B are linked B is not aware of A
Code	B is stored by value within A
Example	human has-a heart



ex1_composition.cpp

AGGREGATION

Concept

A has-a B
B is part-of A
The sum of the parts of A include B

Characteristics

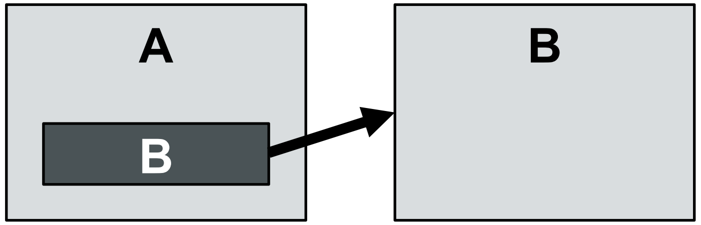
B can belong to many instances of A
scope and lifetime of A and B are independent
B is not aware of A

Code

B is linked by reference or pointer from A
A does not manage the memory of B (no big three)

Example

person has-a address



ex2_aggregation.cpp

ASSOCIATION

Concept

A uses B

Characteristics

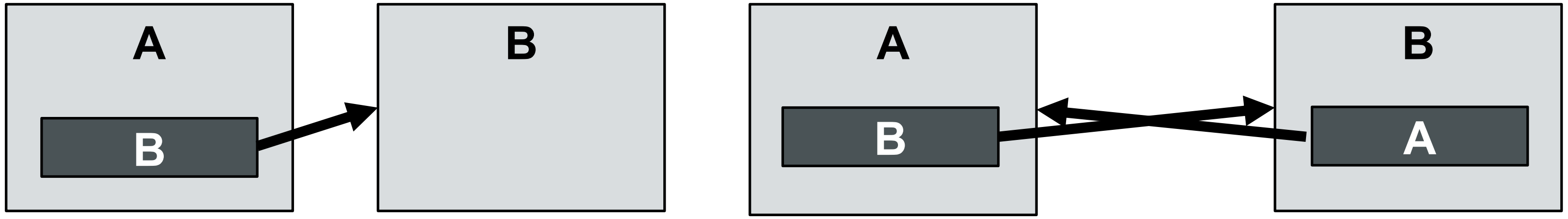
B can associate with many instances of A
scope and lifetime of A and B are independent
A and B unrelated aside from the association

Code

B is linked by pointer from A
A does not manage the lifetime memory of B (no big three)

Example

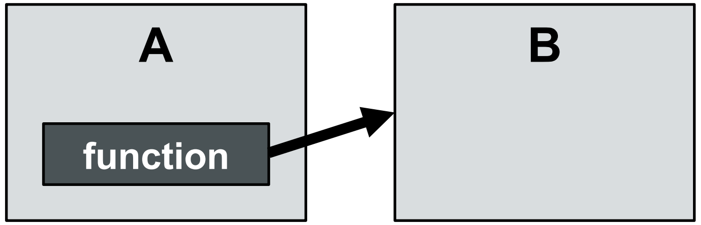
doctor uses-a patient, patient uses-a doctor



ex3_association.cpp

DEPENDENCY

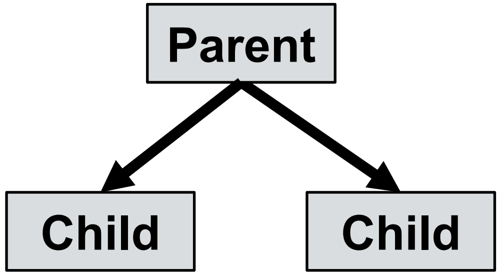
Concept	A relies upon B
Characteristics	B can associate with many instances of A scope and lifetime of A and B are independent aside from the dependency objects are unrelated
Code	B objects are not data members of A A does not manage the lifetime memory of B (no big three)
Example	person relies upon a street to drive to work



INHERITANCE

Concept	parent/child relationship
	base/derived relationship
	class B is-a class A
	class B inherits from A

Characteristics	scope and lifetime of A and B are linked inheritance is hierarchical, B inherits from A parent and child objects created for every child instance
Code	B inherits data members and functions from A
Example	student is-a person, employee is-a person



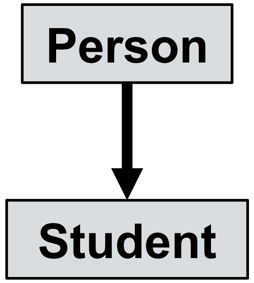
ex5_inheritance.cpp

INHERITANCE

Syntax

```
class Person {                              // base
    Person() { }
};

class Student : public Person {             // derived: student inherits from person
    Student(): Person() { }                 // create child and parent object
};
```



ex5_inheritance.cpp

MULTIPLE INHERITANCE

Concept

child has multiple parents
C is-a A and C is a B

Characteristics

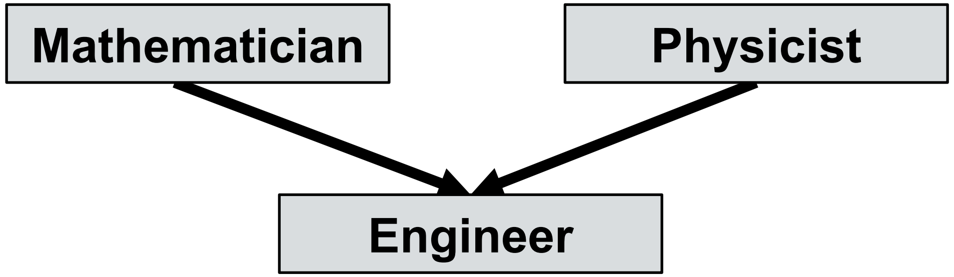
member access is ambiguous
memory usage can become significant

Code

B inherits data members and functions from A and C

Example

engineer is-a mathematician and engineer is-a physicist

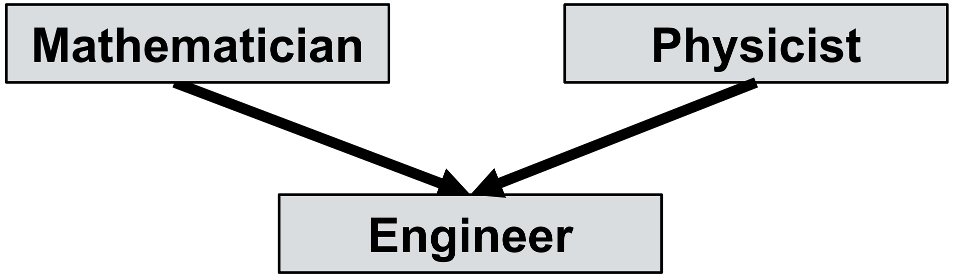


ex6_multiple_inheritance.cpp

MULTIPLE INHERITANCE

Syntax

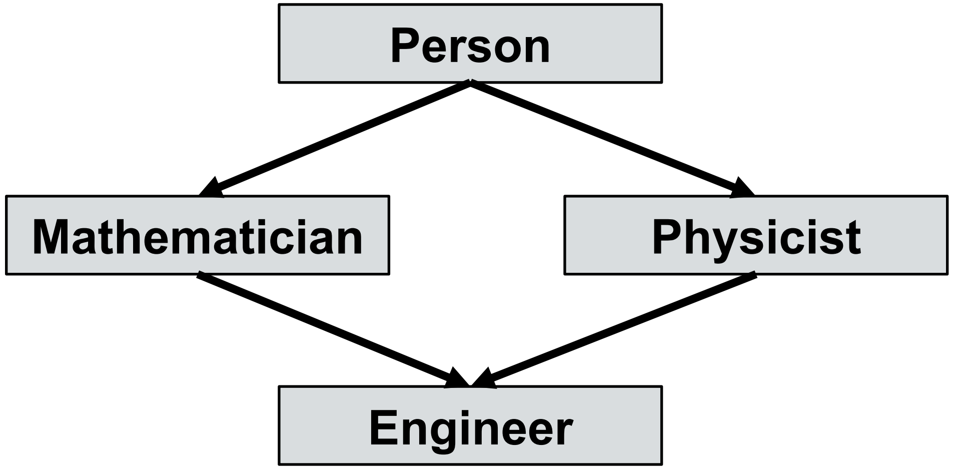
```
class Mathematician {                                              // base
        Mathematician() {}
};
class Physicist {                                                  // base
        Physicist() {}
};
class Engineer : public Mathematician, public Physicist {          // derived
        Engineer(): Mathematician(), Physicist() {}        // create child and parents
};
```



DIAMOND PROBLEM

Syntax

```
class Person {                                                  // base
};
class Mathematician : public Person {                           // derived
};
class Physicist : public Person {                               // derived
};
class Engineer : public Mathematician, public Physicist {       // derived
};
```



OVERLOAD VS REDEFINE

Overload inherited function has different parameters

Redefined inherited function has different function body

Example

```
void output() { cout << 1; }          // original function in parent class

void output(int i) { cout << i; }     // overloaded function in child class

void output() { cout << 2; }          // redefined function in child class
```

ex7_overload_vs_redefine.cpp

INHERITANCE TYPES AND ACCESS MODIFIERS

Recommendation

public inheritance
parent with protected data members

Example

```
class Student : public Person { };  // specify inheritance type
```

Base	Inheritance Type	Derived
Public	Public	Public
Protected		Protected
Private		n/a
Public	Protected	Protected
Protected		Protected
Private		n/a
Public	Private	Private
Protected		Private
Private		n/a

ex8_inheritance_types.cpp

THE BIG THREE

Requirements

child objects must deep copy parent objects
child big three functions have additional code

Example

car is-a vehicle, car inherits from vehicle

Copy Constructor

construct a child and parent from a child object

```
Car(const Car &c): Vehicle(c)  { … }     // vehicle created from car
```

Assignment Operator

copy the child and parent from a child object

```
Vehicle::operator=(c)   // call the parent assignment operator
```

Destructors

child and parent destructors independently delete memory

```
~Vehicle() { delete brand; }       // vehicle deletes its memory
~Car() { delete numDoors; }        // car deletes its memory
```

ex9_inheritance_and_the_big_three.cpp