Object-oriented programming CS10003

Lecturer: Do Nguyen Kha

Contents

- Virtual Destructor
- C++11 override and final
- Multiple Inheritance
- Interface
- Template Class
- Friend Class
- Relationship
- Team project discussion

Virtual Destructor

Always define destructor as `virtual` in (base) classes

C++11 override

`override` is a C++11 identifier which means that a method is an "override" from a method from a base class

```
class Base {
public:
  virtual int foo(float x) = 0;
};
class Derived: public Base {
public:
  int foo(float x) override { ... } // OK
};
class Derived2: public Base {
public:
  int foo(int x) override { ... } // ERROR
};
```

C++11 final

To prevent prevent a class from being inherited

```
class Base1 final {};
class Derived1 : Base1 {};
```

To mark a function so as to prevent it from being overridden in the derived classes

```
class Base2 {
public:
  void f() final;
};

struct Derived2 : public Base2 {
  void f();
};
```

Multiple Inheritance

```
class MyClass {
public:
  void myFunction() { cout << "Some content in parent class."; }
};
class MyOtherClass {
public:
  void myOtherFunction() { cout << "Some content in another class."; }
};
class MyChildClass: public MyClass, public MyOtherClass { };
int main() {
  MyChildClass myObj;
  myObj.myFunction();
  myObj.myOtherFunction();
  return 0;
}
```

Multiple Inheritance

- What happened if base classes have the same property name or method name?
- Java and C# use `interface`

Interface

Interface is pure abstract class

```
class Interface1 {
public:
 virtual void method1() = 0;
};

class Interface2 {
public:
 virtual int method2(int param) = 0;
};
class ConcreteClass : public Interface1, public Interface2 {
public:
 void method1() override { }
 int method2(int param) override { }
};
```

Template Class

Templates are powerful features of C++ which allows us to write generic programs

```
template <class T>
class ClassName {
private:
 T var;
public:
 T functionName(T arg);
};
```

`class` keyword can be replaced by `typename`

Template Class

```
ClassName<int> obj1;

ClassName<float> obj2;

ClassName<string> obj3;
```

Template Class

Implement method outside of class definition

```
template <class T>
class ClassName {
 // Function prototype
 returnType functionName();
};

// Function definition
template <class T>
returnType ClassName<T>::functionName() {
 // code
}
```

Template Class

Template class with multiple parameters

```
template <class T, class U, class V = int>
class ClassName {
private:
 T member1;
 U member2;
 V member3;
};
```

Template Class

Inherit from a special class name

```
class Rectangle : public Area<int> { };
```

Inherit with template

```
template<typename T> class Rectangle : public Area<T> {};
```

Friend Class

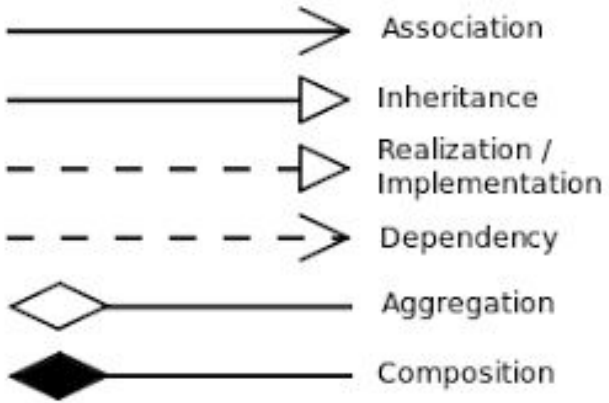
C++ provides the keyword `friend` to allow the access to private or protected fields

- Friend function
- Friend class

```
class Point{
int x, y;
friend class Rectangle;
friend void Move(Point&, int, int);
};
```

Relationship

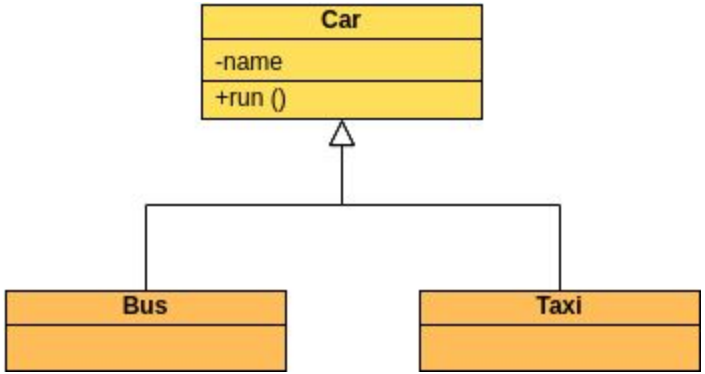
In addition to property & method, the relationship between the objects is very important



https://blog.visual-paradigm.com/what-are-the-six-types-of-relationships-in-uml-class-diagrams/

Relationship

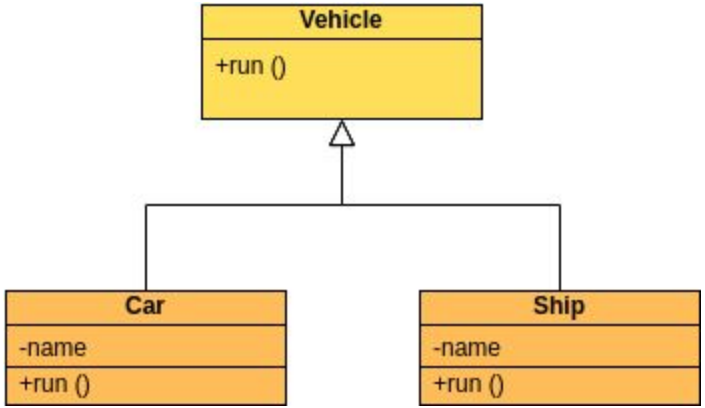
Inheritance



https://blog.visual-paradigm.com/what-are-the-six-types-of-relationships-in-uml-class-diagrams/

Relationship

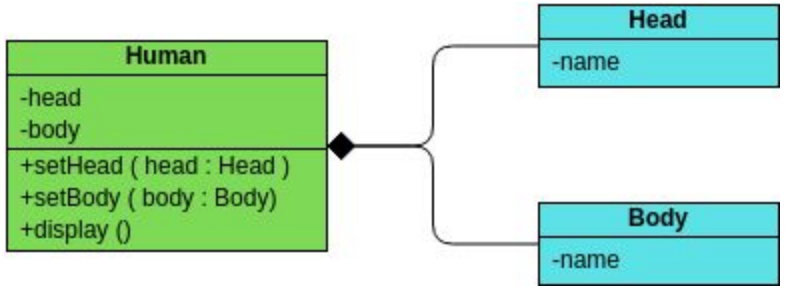
Realization / Implementation



https://blog.visual-paradigm.com/what-are-the-six-types-of-relationships-in-uml-class-diagrams/

Relationship

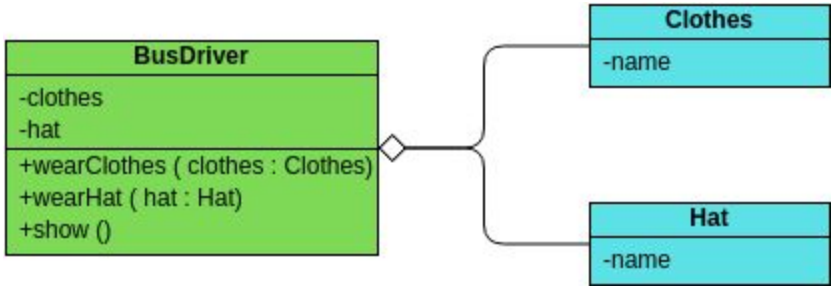
Composition



https://blog.visual-paradigm.com/what-are-the-six-types-of-relationships-in-uml-class-diagrams/

Relationship

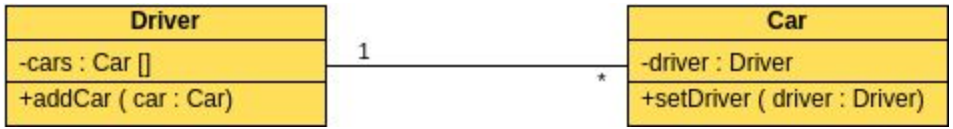
Aggregation



https://blog.visual-paradigm.com/what-are-the-six-types-of-relationships-in-uml-class-diagrams/

Relationship

Association



https://blog.visual-paradigm.com/what-are-the-six-types-of-relationships-in-uml-class-diagrams/

Relationship

Dependencies

https://blog.visual-paradigm.com/what-are-the-six-types-of-relationships-in-uml-class-diagrams/

Team project discussion