

Class Relationships

Category	Relationship	Inheritance (Generalization)	Association	Aggregation	Composition	Dependency (Abstraction)
Semantics (meaning)	<ul style="list-style-type: none"> is a relationship is a kind of relationship includes shared attributes operations from superclass 	<ul style="list-style-type: none"> is a relationship is a kind of relationship is a kind of relationship is a kind of relationship 	<ul style="list-style-type: none"> is a relationship is a kind of relationship is a kind of relationship is a kind of relationship 	<ul style="list-style-type: none"> is a relationship is a kind of relationship is a kind of relationship is a kind of relationship 	<ul style="list-style-type: none"> is a relationship is a kind of relationship is a kind of relationship is a kind of relationship 	<ul style="list-style-type: none"> is a relationship is a kind of relationship is a kind of relationship is a kind of relationship
Example	A student is a person.	A class has a teacher, a teacher has a class.	A car has an engine. An engine is part of a car.	A car has an engine. An engine is part of a car.	A car has an engine. An engine is part of a car.	A car has an engine. An engine is part of a car.
Class Notation	Parent (Class) Derived (Subclass)	None	None	None	None	None
UML Symbol						

Class Relationships

Unidirectional:

all but association

Bidirectional:

association

can change after instantiation

aggregation, dependency, association

cannot change after instantiation

inheritance, composition

which relationships must be created when objects instantiated: (have shared lifetime)

inheritance and composition

which relationships can be created at any convenient time (independent lifetimes)

aggregation, dependency, association

allow sharing some related objects

aggregation, dependency, association

exclusive (no sharing)

inheritance, composition

"is a" "is a kind of"

inheritance

"has a"

aggregation, composition, association

Class Relationships (cont)

whole-part

aggregation, composition

implemented with dedicated computer

syntax or keyword

inheritance

Inheritance:

add features to an existing (general) class without having to rewrite it.

```
class Foo: public Bar
```

Streams

A C++ stream is a flow of data from one place to another.

three stream classes commonly used: ifstream, ofstream, fstream

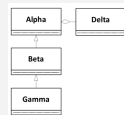
```
ofstream sales( " SALES.J UN ");
output file associated with SALES.JUN
```

```
fileOut.put(ch); OR fileOut <<<
ch put what's in variable 'ch' in fileOut (an
ofstream thing)
```

```
Cstrings char s[100]; cin.getline
(s, 100);
```

```
Strings string s; getline(cin, s);
```

Q27 ch12



delta, alpha, beta, gamma - this is the order they have to be in.

Polymorphism

Five requirements: 1. inheritance 2. function overriding 3. up casting 4. a virtual function 5. a pointer (usually) or reference variable

virtual functions allow you to use the same function call to execute member functions of objects from different classes

deciding what function call executes after a program starts is polymorphism

pure virtual function `virtual void dang() = 0;` it causes it's class to be abstract and it is in the super class

an abstract class is useful when no objects should be instantiated from it.

An abstract class can: be a base (parent or super) class have concrete features (both variables and functions) that can be inherited by derived (child or sub) classes participate in (i.e., be the target of) an upcast participate in polymorphism

Object oriented model: inheritance, encapsulation, & polymorphism

overloaded function: Are defined in the same class Must have unique argument lists May have different return types

overridden functions: Are defined in two classes that are related by inheritance Must have the same name Must have exactly the same argument list Must have the same return type

Constructors

Default: foo() no arguments `foo f1;` or `foo * f2 = new foo;`

Conversion: foo(int i) one argument to be turned into the class object `foo f1(3);` or `foo* f2 = new foo(3);`

General: foo(int x, int y) anything with more than one

Copy: foo(foo& f) pointer argument

Move: foo(foo&& f) double pointer

Chap 12 stuff

- order is not significant, follow the pattern: member-name(argument-name)
- must initialize inheritance first, this is a function call so the number, type, and order of the parameters must match the number, type, and order of the arguments in the function definition: class-name(parameters), member(argument)
- order is not significant; must use the second argument to access the members
- the general pattern is class-name::function-name()

first part

```
class Bar
{
private:
    int count;
    int balance;

public:
    Bar(int a_count, int a_balance): (a) {}
    friend ostream& operator<<(ostream& out, Bar& b)
    {
        . . . . .
        return out;
    }
    void display()
    {
        cout << count << " " << balance << endl;
    }
};
```

second part

```
class Foo : public Bar
{
private:
    string name;
public:
    Foo(string a_name, int count, int balance) : (b) {}
    friend ostream& operator<<(ostream& out, Foo& f)
    {
        out << (d) [Bar &f << " " << f.name << endl;
        return out;
    }
    void display()
    {
        (d) Bar::display();
        out << name << endl;
    }
};
```

Templates

When creating a template function, the template argument or variable is preceded by the keyword `template` or `class`.

When creating a template class, it's the same ^

The template class works with different datatypes.

Template source code is placed in a header file so that it can be included with "normal" source code where it is compiled following the type expansion or substitution. (There can be more than one template argument)

When a class is "templated" all member functions are placed in the header file: the functions can't be compiled until the templated variable is expanded. So it can't be in a regular library

```
template <class T>
class FooBar
```

correct beginning of an operation called Foo:

```
template <class T>
FooBar <T> ::Foo ()
```

```
while (true) CList< per son >pe ople;
. . . wrong because it creates a new list every time
it loops.
```



By sadieweaver

cheatography.com/sadieweaver/

Not published yet.

Last updated 14th August, 2019.

Page 2 of 2.

Sponsored by [Readable.com](https://readable.com)

Measure your website readability!

<https://readable.com>