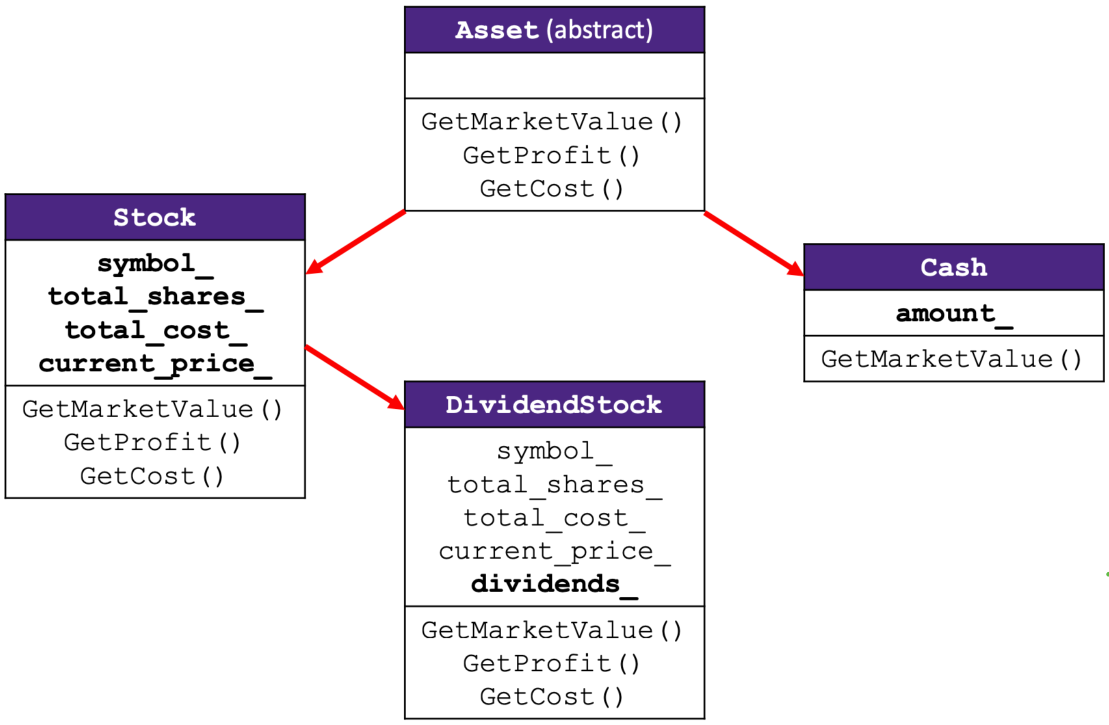
Lib : inheritance

Inheritance :

benefits :

- code reuse
- polymorphism
  - Ability to redefine existing behavior but preserve the interface
  - Children can override behaviors of parent
  - Others can make calls on objects without knowing which part of the inheritance tree it is in.
- Extensibility : children can add behavior

Design With Inheritance



modifiers :

- public :
  visible to all class
- protected:
  ~ current & derived class
  - used when class is designed to be extended ; subclass must have access but clients shouldn't
- private:
  ~ only current class

❖ Comma-separated list of classes to inherit from:

```
#include "BaseClass.h"

class Name : public BaseClass {
  ...
};
```

- Focus on single inheritance, but *multiple inheritance* possible

complicated : if a class inherits from both Cowboy and Artist, what should .draw() do?

❖ Almost always you will want public inheritance

· Acts like extends does in Java.
· Any member that is non-private in base class is the same in the derived class; both interface and implementation inheritance.
Except ctor, dtor, cctor, op=. ← Never inherited

Polymorphism in C++

- In Java: `PromisedType var = new ActualType();`
  - `var` is a reference (different term than C++ reference) to an object of `ActualType` on the Heap
  - `ActualType` must be the same class or a subclass of `PromisedType`

- In C++: `PromisedType *var_p = new ActualType();`
  - `var_p` is a *pointer* to an object of `ActualType` on the Heap
  - `ActualType` must be the same or a derived class of `PromisedType`
  - (also works with references)

  - `PromisedType` defines the *interface* (*i.e.* what can be called on `var_p`), but `ActualType` may determine which *version* gets invoked

Dynamic dispatch : a run-time decision of what code to invoke
- A member function invoked on an object should be the most derived function accessible to the object's visible type. Can determine what to invoke from the object itself.

- Example:
  - `void PrintStock(Stock *s) { s->Print(); }`
  - Calls the appropriate `Print()` without knowing the actual type of `*s`, other than it is some sort of `Stock`

Key words:
virtual , derived child doesn't need to repeat but

good style to do so.
override (C++11): annotation that compiler will check
not effect on output; prevents overriding/overloading bugs

```
double DividendStock::GetMarketValue() const {
  return get_shares() * get_share_price() + dividends_;
}

double "DividendStock"::GetProfit() const {   // inherited
  return GetMarketValue() - GetCost();
}
```

want this to invoke DS::GetMarketValue()

inherited

DividendStock.cc

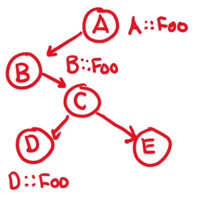
```
double Stock::GetMarketValue() const {
  return get_shares() * get_share_price();
}

double Stock::GetProfit() const {
  return GetMarketValue() - GetCost();
}
```

Stock.cc

```
#include "Stock.h"
#include "DividendStock.h"

DividendStock dividend;
DividendStock* ds = &dividend;
Stock* s = &dividend;    // why is this allowed?

// Invokes DividendStock::GetMarketValue()
ds->GetMarketValue();

// Invokes DividendStock::GetMarketValue()
s->GetMarketValue();

// invokes Stock::GetProfit(), since that method is inherited.
// Stock::GetProfit() invokes DividendStock::GetMarketValue(),
// since that is the most-derived accessible function.
s->GetProfit();
```

Stock is base class of DividendStock, so everything in Stock interface must be in DividendStock interface

- Whose Foo() is called?



	Q1	Q2
A.	A	B
B.	A	D
[✓ C.]	B	B
D.	B	D
E.	We're lost...	

```
void Bar() {
  A* a_ptr;

  // Q1:
  a_ptr = new C;
  a_ptr->Foo();
              B::Foo
  // Q2:
  a_ptr = new E;
  a_ptr->Foo();
}             B::Foo
```

```
class A {
 public:
  virtual void Foo();
};

class B : public A {
 public:
  virtual void Foo();
};

class C : public B {
};

class D : public C {
 public:
  virtual void Foo();
};

class E : public C {
};
```

can send other encodings

[ask

ascii → plain text

8.14 CSE 461

找一下 mac version of \r 和 \n

\r\n → windows version

---

header & body 中间有一行空行.

Saga..

teaching skills

resume could be ...

333 homepage

- --

. ~ ~

↓

specific term.
