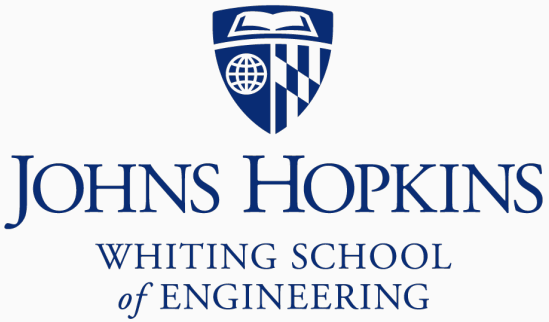
Inheritance

Ben Langmead

ben.langmead@gmail.com

www.langmead-lab.org



Source markdown available at github.com/BenLangmead/c-cpp-notes

Inheritance

C++ classes can be related to each other

`class Account {...};`, `class CheckingAccount {...};`

- "is a" relationship; a checking account is a kind of account

`class GradeList {...};`, `vector<double>`

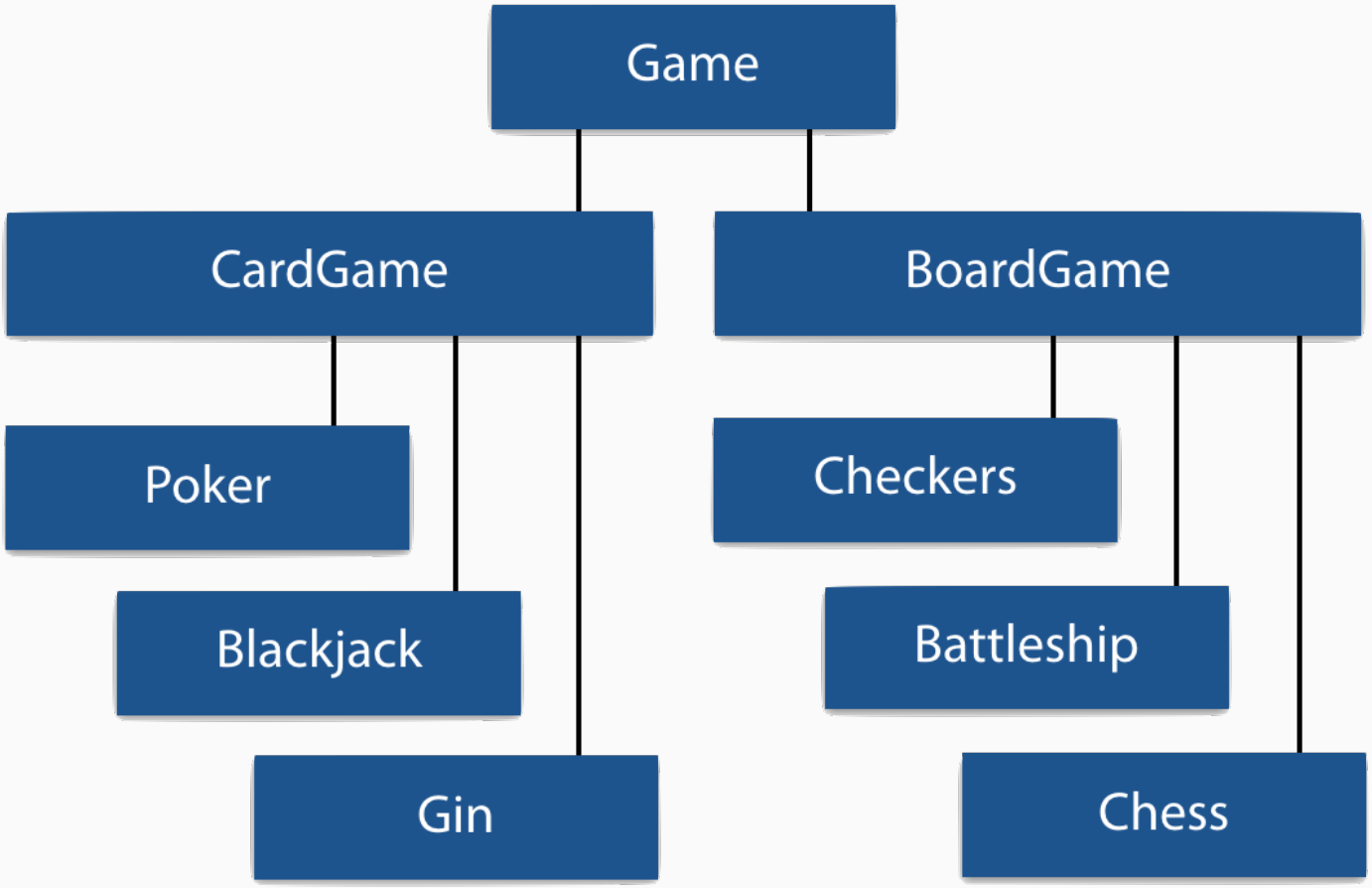
- "has a" relationship; grade list *has* vector of grades as a field

Inheritance

Base class	Derived class
`Account`	`CheckingAccount`, `SavingsAccount`
`Shape`	`Rectangle`, `Circle`
`Document`	`WordDoc`, `WebPage`, `TextDoc`

Example “is a” relationships

Inheritance



Multiple levels of “is a” relationships

Inheritance

Derived class inherits from *base class*

Java-like vocab: *subclass* inherits from *superclass*

(We'll typically say "derived" and "base")

Inheritance

```
class BaseClass {
    // Definitions for BaseClass
    ...
};

class DerivedClass: public BaseClass {
    // Definitions for DerivedClass
    ...
};
```

This is “public inheritance” – by far the most common kind

(`protected` & `private` inheritance also possible, but rarely used)

Inheritance

Derived class *inherits all members* of base class, whether `public`, `protected` or `private`, *except*:

- Constructors
- Assignment operator (discussed later)

Derived class cannot delete things it inherited; cannot pick and choose what to inherit

- But derived class can *override* inherited member functions
- override = substitute own implementation for base class's

Inheritance

Base-class members marked `public` or `protected` can be accessed from member functions defined in the derived class

Base-class members marked `private` *cannot* be accessed from member functions defined in the derived class

- They're still there, and *base class* member functions can still use them, but *derived class* member functions can't

Inheritance

`protected` is an access modifier we haven't used yet

- `protected` fields & functions cannot be accessed except from member functions of class (like `private`)
- They *are* accessible from member functions defined in derived classes (like `public`)

Inheritance

```
class Account {
public:
    Account() : balance(0.0) { }
    Account(double initial) : balance(initial) { }

    void credit(double amt)    { balance += amt; }
    void debit(double amt)     { balance -= amt; }
    double get_balance() const { return balance; }
private:
    double balance;
};
```

Default constructor sets balance to 0; non-default constructor sets according to argument

balance is private, modified via `credit(amt)`/`debit(amt)`

Inheritance

What does this const mean?

```
class Account {
public:
    ...
    double get_balance() const { return balance; }
    //                     ^^^^^
private:
    double balance;
};
```

Means member function *does not modify any fields*

- `get_balance()` does not modify `balance`

If you have a `const Account` (or `const Account &`), `const` member functions are the *only* ones you can call

Inheritance

```
#include <iostream>
#include "account.h"

using std::cout;  using std::endl;

int main() {
    Account acct(1000.0);
    acct.credit(1000.0);
    acct.debit(100.0);
    cout << "Balance is: " << acct.get_balance() << endl;
    return 0;
}
```

```
$ g++ -c account_main1.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o account_main1 account_main1.o
$ ./account_main1
Balance is: 1900
```

Inheritance

```
class CheckingAccount : public Account {
public:
    CheckingAccount(double initial, double atm) :
        Account(initial), total_fees(0.0), atm_fee(atm) { }

    void cash_withdrawal(double amt) {
        total_fees += atm_fee;
        debit(amt + atm_fee);
    }

    double get_total_fees() const { return total_fees; }

private:
    double total_fees;
    double atm_fee;
};
```

Inheritance

```
class SavingsAccount : public Account {
public:
    SavingsAccount(double initial, double rate) :
        Account(initial), annual_rate(rate) { }

    // Not implemented here; usual compound interest calc
    double total_after_years(int years) const;

private:
    double annual_rate;
};
```

Inheritance

Syntax for declaring a `class` that derives from another:

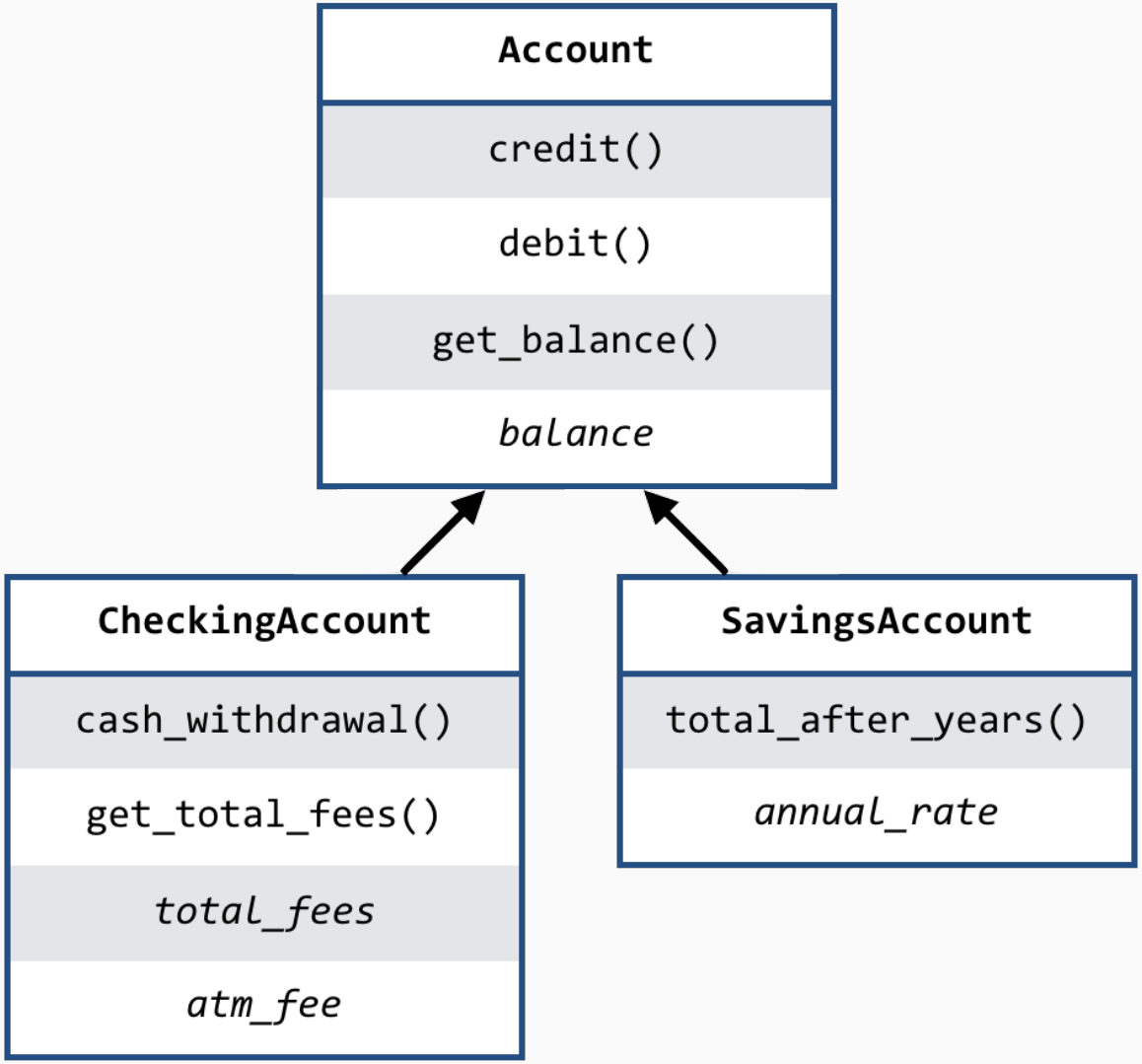
```
class Derived : public Base {
    ...
};
```

Who can use members with `public`, `protected` and `private` access modifiers?

Access modifier	Any function	Derived-class members	Same-class members
`public`	Yes	Yes	Yes
`protected`	No	Yes	Yes
`private`	No	No	Yes

Inheritance

Returning to our financial account example:



Inheritance

```cpp
#include <iostream>
#include "account.h"
using std::cout;  using std::endl;

int main() {
    Account acct(1000.0);
    acct.credit(1000.0);
    acct.debit(100.0);
    cout << "Account balance is: $" << acct.get_balance() << endl;

    CheckingAccount checking(1000.0, 2.00);
    checking.credit(1000.0);
    checking.cash_withdrawal(100.0); // incurs ATM fee
    cout << "Checking balance is: $" << checking.get_balance() << endl;
    cout << "Checking total fees is: $" << checking.get_total_fees() << endl;

    SavingsAccount saving(1000.0, 0.05);
    saving.credit(1000.0);
    cout << "Savings balance is: $" << saving.get_balance() << endl;
    return 0;
}
```

Inheritance

```
$ g++ -c account_main2.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o account_main2 account_main2.o
$ ./account_main2
Account balance is: $1900
Checking balance is: $1898
Checking total fees is: $2
Savings balance is: $2000
```