Command Pattern Exercise

Amazon Orders

Amazon orders

- In 2014, Bernstein Research estimated that the USPS handled 40% of Amazon orders. This amounted to ~150 million items
- This means Amazon fulfills ~375 million physical orders every year, and this is just the physical orders

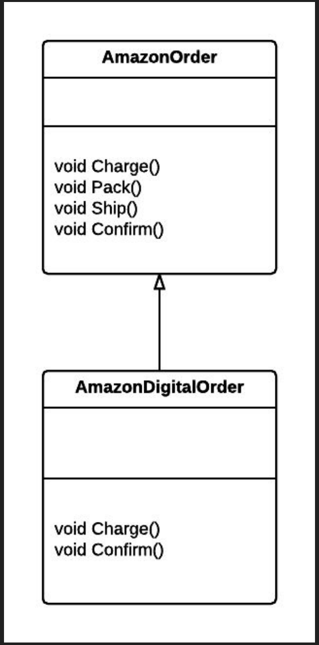
Assumptions

- Let’s assume an amazon order has (a subset of) the following steps:
    - Charge
    - Pack
    - Ship
    - Confirm

How would you build an object to track an `AmazonOrder`?

Using a single object

- However not every order has these steps, think about digital products



Using functions (or function pointers)

```
class order {
    private:
        vector<void (*step)()> order_steps;
    public:
        order() { }
        void add_step(void (*step)()) {
            order_steps.push_back(step);
        }
        void execute() {
            for (unsigned i = 0;i < order_steps.size();i++) {
                (order_steps.at(i))();
            }
        }
};
```

Using functions (or function pointers)

- `void Charge() { … }`
- `void Pack() { … }`
- `void Ship() { … }`
- `void Confirm() { … }`

```
Order* amazon_order = new Order();
amazon_order->add_step(&charge);
amazon_order->add_step(&pack);
amazon_order->add_step(&ship);
amazon_order->add_step(&confirm);
amazon_order->execute();
```

```
Order* digital_order = new Order();
digital_order->add_step(&charge);


digital_order->add_step(&confirm);
digital_order>execute();
```

What if we used objects?

Amazon order as objects

First we need some items

- A **client**,
- an **invoker**,
- the **command** interface,
- the **concrete commands**,
- and **Receivers** for each command

First let’s add an account object

There has to be someone associated with each order

```
class Account {
    private:
        string address;
        string email;
        double order_cost;
        …
    public:
        // Constructors
        // Getters and setters
};
```

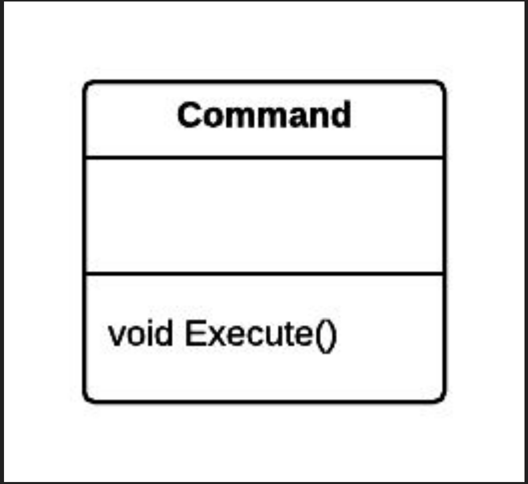
Order object

```
class Order {
    private:
        Account* account;
        // How do we interact with the order commands?
    public:
        order(Account* new_account) { account = new_account; }
};
```

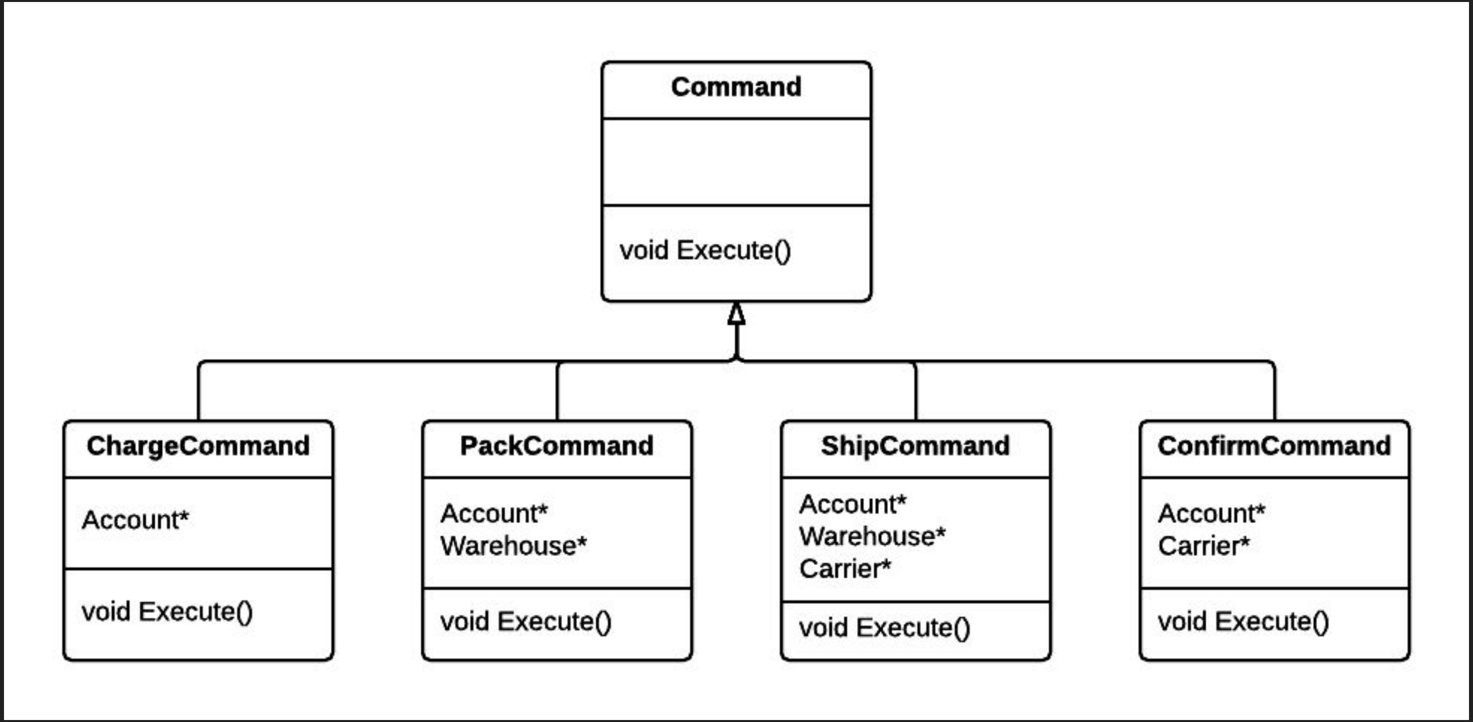
Command Interface

Let’s create the inherited interface (`Command` is an abstract base class)

Execute is declared pure virtual



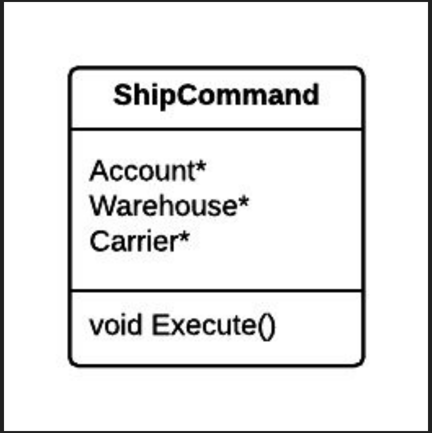
And now some concrete commands



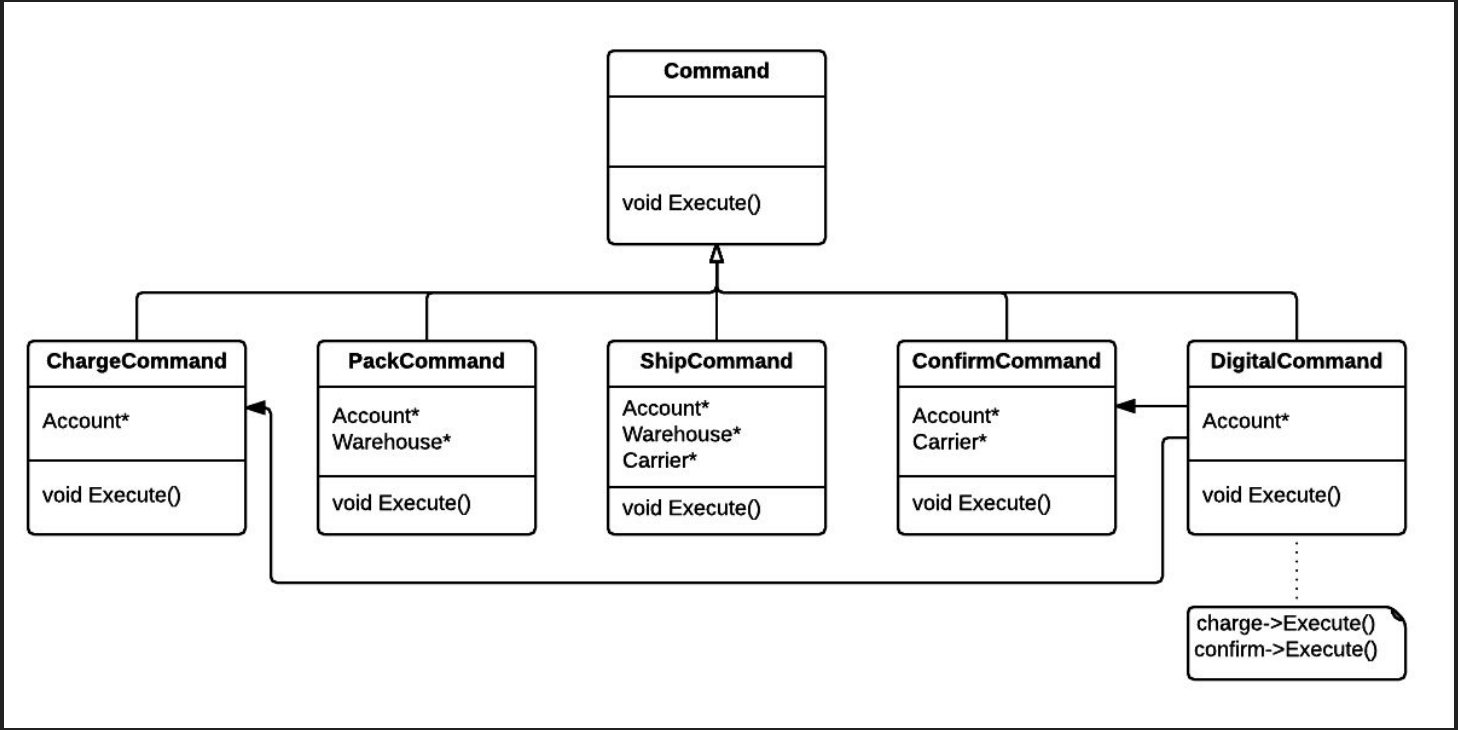
Notice the `Receiver` objects

```
Void ShipCommand::Execute() {
    OrderDetails* order = account->get_details();
    Shipment* shipping = carrier->set_pickup(warehouse,order);
    Tracker* tracker = shipping->get_tracking();
    account->set_tracking(tracker);
}
```

- `Execute()` delegates some of it’s work to different `Receiver` objects



And now for digital orders



Amazon order as objects

- **client** - server taking amazon orders
- **invoker** - system processing amazon orders
- **command** interface - base class for defining order interface
- **concrete commands** - different types of order (digital, physical, etc.)
- **Receivers** for each command - warehouse, account, carrier, etc.