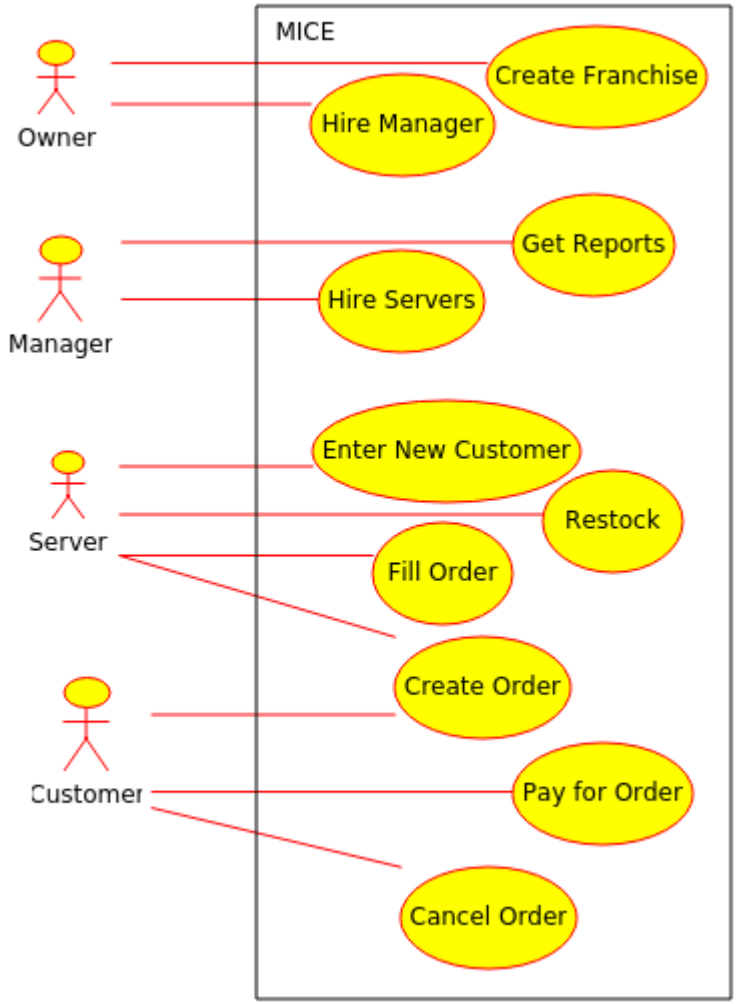
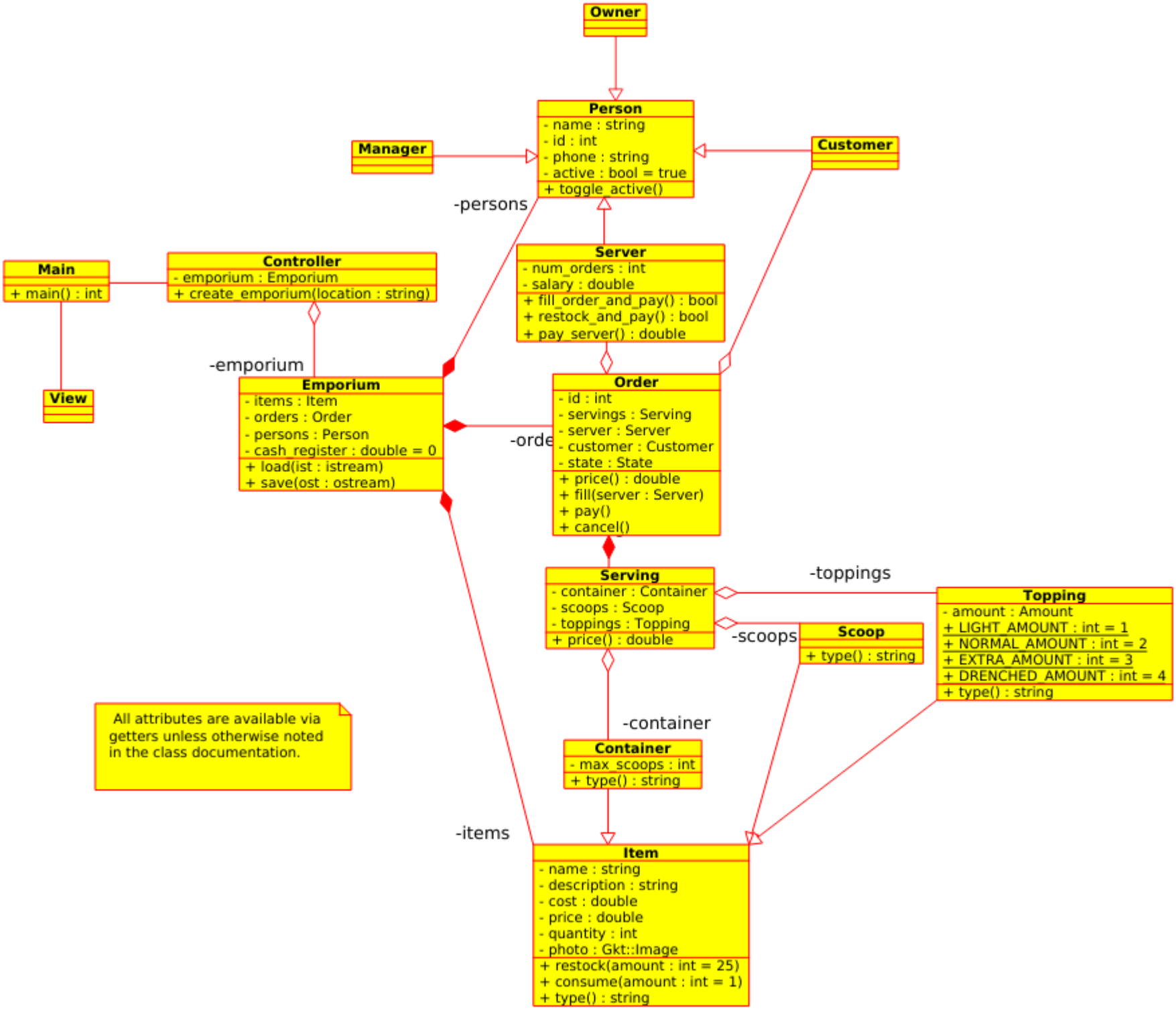
Suggested Solution for Sprint 1

Use Case Diagram

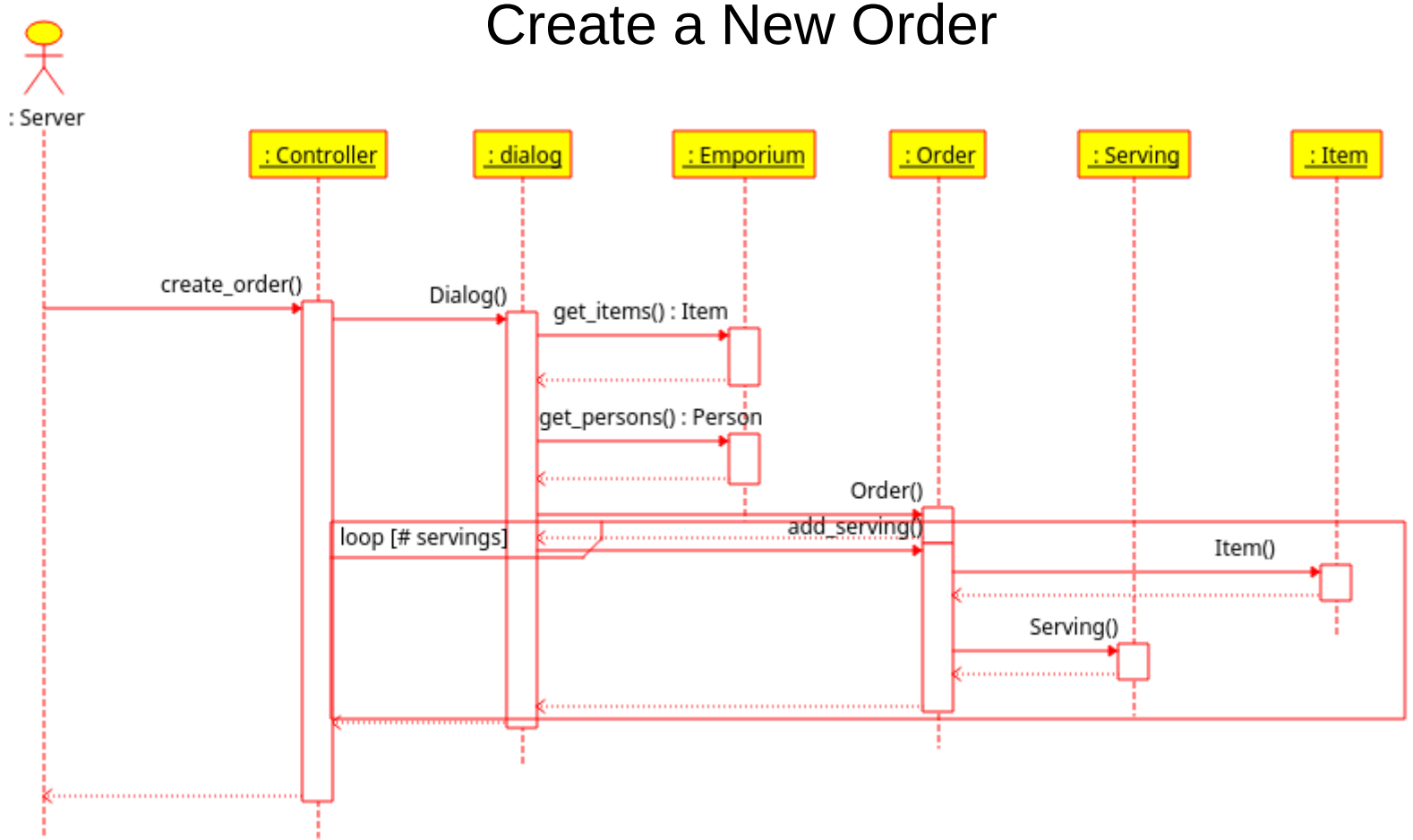



Owner
Person
- name : string
- id : int
- phone : string
- active : bool = true
+ toggle_active()
Manager
Customer
-persons
Server
- num_orders : int
- salary : double
+ fill_order_and_pay() : bool
+ restock_and_pay() : bool
+ pay_server() : double
Main
+ main() : int
Controller
- emporium : Emporium
+ create_emporium(location : string)
-emporium
View
Emporium
- items : Item
- orders : Order
- persons : Person
- cash_register : double = 0
+ load(ist : istream)
+ save(ost : ostream)
-orde
Order
- id : int
- servings : Serving
- server : Server
- customer : Customer
- state : State
+ price() : double
+ fill(server : Server)
+ pay()
+ cancel()
Serving
- container : Container
- scoops : Scoop
- toppings : Topping
+ price() : double
-toppings
Topping
- amount : Amount
+ LIGHT_AMOUNT : int = 1
+ NORMAL_AMOUNT : int = 2
+ EXTRA_AMOUNT : int = 3
+ DRENCHED_AMOUNT : int = 4
+ type() : string
-scoops
Scoop
+ type() : string
All attributes are available via getters unless otherwise noted in the class documentation.
-container
Container
- max_scoops : int
+ type() : string
-items
Item
- name : string
- description : string
- cost : double
- price : double
- quantity : int
- photo : Gkt::Image
+ restock(amount : int = 25)
+ consume(amount : int = 1)
+ type() : string

Suggested Solution for Sprint 1

Sequence Diagram

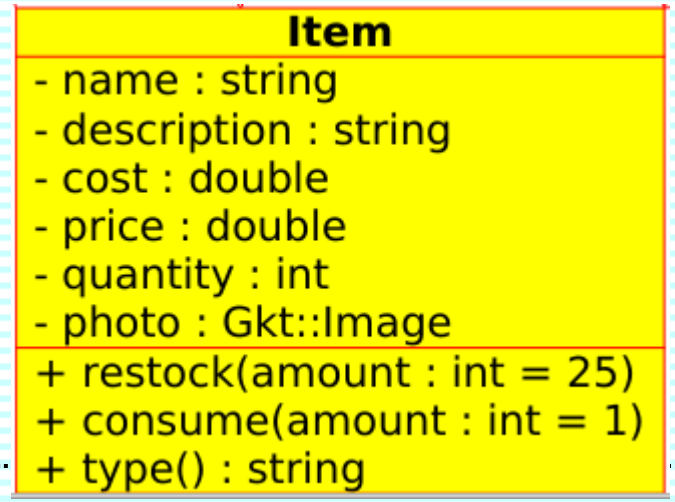
Create a New Order



Suggested Solution to Sprint 1

Item

```
#include <string>
class Item {
  public:
    Item(std::string name, std::string description, double cost, double price);
    void restock(int quantity = 25);
    void consume(int quantity = 1);
    virtual std::string type();
    std::string name();
    std::string description();
    double cost();
    double price();
    int quantity();
  private:
    std::string _name;
    std::string _description;
    double _cost;
    double _price;
    int _quantity;
    // Gtk::Image _photo;
};
```




```
#include "item.h"
Item::Item(std::string name, std::string description,
           double cost, double price)
    : _name{name}, _description{description},
      _cost{cost}, _price{price}, _quantity{0} { }
std::string Item::type() {return "Item";}
void Item::restock(int quantity) {_quantity  = quantity;}
void Item::consume(int quantity) {_quantity -= quantity;}
std::string Item::name() {return _name;}
std::string Item::description() {return _description;}
double Item::cost() {return _cost;}
double Item::price() {return _price;}
int Item::quantity() {return _quantity;}
```

Suggested Solution to Sprint 1
Test Item

```
// Returns true if test passes
bool test_item();
```

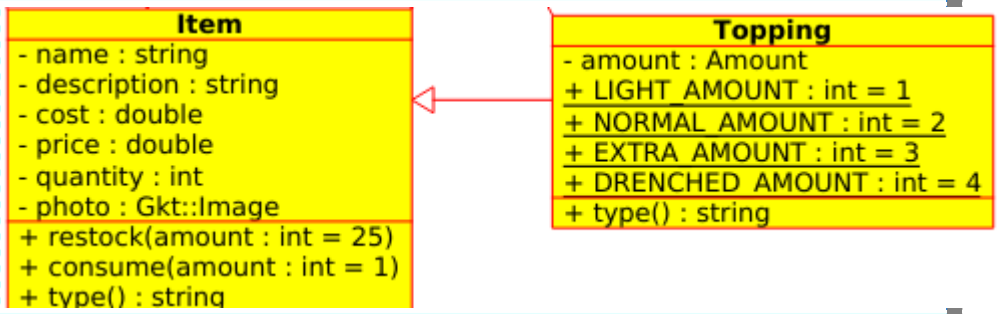
Selected exerpts

```
#include "test_item.h", "item.h", <iostream>

bool test_item() {
  bool passed = true; // Optimist!
  //
  // Test constructor
  std::string x_name = "Fudge Ripple";
  std::string x_description = "Chocolatey goodness in vanilla swirl";
  double x_cost = 0.75;    double x_price = 1.50;

  Item item{x_name, x_description, x_cost, x_price};

  if (item.name() != x_name || item.description() != x_description || ...
    std::cerr << "#### Item constructor fail" << std::endl;
    std::cerr << "Expected: " << x_name << ',' << x_description << ','...
    std::cerr << "Actual:   " << item.name() << ',' << item.description() << ','...
    passed = false;
  }
  //
  // Test restock
  item.restock();
  if (item.quantity() != 25) {
    std::cerr << "#### Item: Restock failed" << std::endl;
    std::cerr << "Expected: 25  Actual: " << item.quantity() << std::endl;
    passed = false;
  }
```

Suggested Solution to Sprint 1

Topping

```
#include "item.h"
class Topping : public Item {
  public:
    Topping(std::string name, std::string description, double cost, double price,
                int amount);
    std::string type() override;
    int amount();
    const static int LIGHT_AMOUNT = 1;
    const static int NORMAL_AMOUNT = 2;
    const static int EXTRA_AMOUNT = 3;
    const static int DRENCHED_AMOUNT = 4;
  private:
    void _set_amount(int amount);
    int _amount;
};
```




```
#include "topping.h"
#include <stdexcept>

Topping::Topping(std::string name, std::string description, double cost, double price,
                int amount) :
    Item(name, description, cost, price), _amount{amount} { }
std::string Topping::type() {return "Topping";}
int Topping::amount() {return _amount;}
void Topping::_set_amount(int amount) {
    if (0 < amount && amount < 5) _amount = amount;
    else throw std::runtime_error("Invalid topping amount");
}
```

Suggested Solution to Sprint 1

Test Topping

```
// Returns true if test passes
bool test_topping();
```

Selected exerpts

```
#include "test_topping.h", "topping.h", <iostream>

bool test_topping() {
  bool passed = true; // Optimist!
  //
  // Test constructor
  //
  std::string x_name = "Maraschino Cherry";
  std::string x_description = "A sweet, plump cherry preserved in maraschino syrup";
  double x_cost = 0.10;
  double x_price = 0.50;
  int x_amount = Topping::EXTRA_AMOUNT;

  Topping topping{x_name, x_description, x_cost, x_price, x_amount};

  if (topping.name() != x_name || topping.description() != x_description || ...
    std::cerr << "#### Topping constructor fail" << std::endl;
    std::cerr << "Expected: " << x_name << ',' << x_description << ','...
    std::cerr << "Actual:   " << topping.name() << ',' << topping.description()...
    passed = false;
  }

  return passed;
}
```

Suggested Solution to Sprint 1

test.cpp

```
#include "test_item.h"
#include "test_container.h"
#include "test_scoop.h"
#include "test_topping.h"
#include <iostream>

int main() {
  if (!(test_item() &&
        test_container() &&
        test_scoop() &&
        test_topping()))
    std::cerr << "fail" << std::endl;
}
```

Suggested Solution to Sprint 1
(Almost) Universal Makefile

```
CXXFLAGS+=--std=c++11

#source files
SOURCES=$(wildcard *.cpp)

#all, Main, and Test object files
OBJECTS=$(SOURCES:.cpp=.o)
MOBJECTS=$(filter-out test%,$(OBJECTS))
TOBJECTS=$(filter-out main.o,$(OBJECTS))

#included libraries (gtkmm goes here once we get to GUI code!)
INCLUDE=

#executable filename
EXECUTABLE=mice

$(EXECUTABLE): $(MOBJECTS)
	$(CXX) $(CXXFLAGS) $^ -o $@ $(INCLUDE)

test: CXXFLAGS+= -g
test: $(TOBJECTS)
	$(CXX) $(CXXFLAGS) $^ -o $@ $(INCLUDE)

debug: CXXFLAGS+= -g
debug: $(EXECUTABLE)

%.o: %.cpp *.h
	$(CXX) $(CXXFLAGS) $(INCLUDE) -c $< -o $@
clean:
	-rm -f $(EXECUTABLE) test $(OBJECTS)
```

$(wildcard *.cpp) expands to every file ending in .cpp
$(SOURCES:.cpp=.o) changes all .cpp to .o in $SOURCES
$(filter-out test%, $(OBJECTS)) removes all object files that begin with “file” from $OBJECTS

#Special symbols used below:
#$^ - is all the dependencies (in this case =$(MOBJECTS))
#$@ - is the result name (in this case =$(EXECUTABLE))

#Compile all non-test files into executable “mice”

#Compile all test files (with debug) into executable “test”

#Compile all non-test files (with debug)

#This is the “generic make rule” where the magic happens

#Delete all of the products, as usual