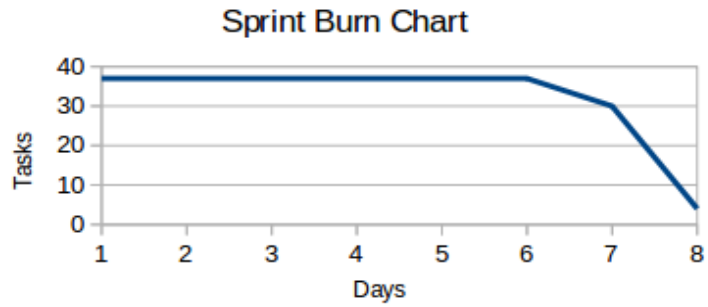
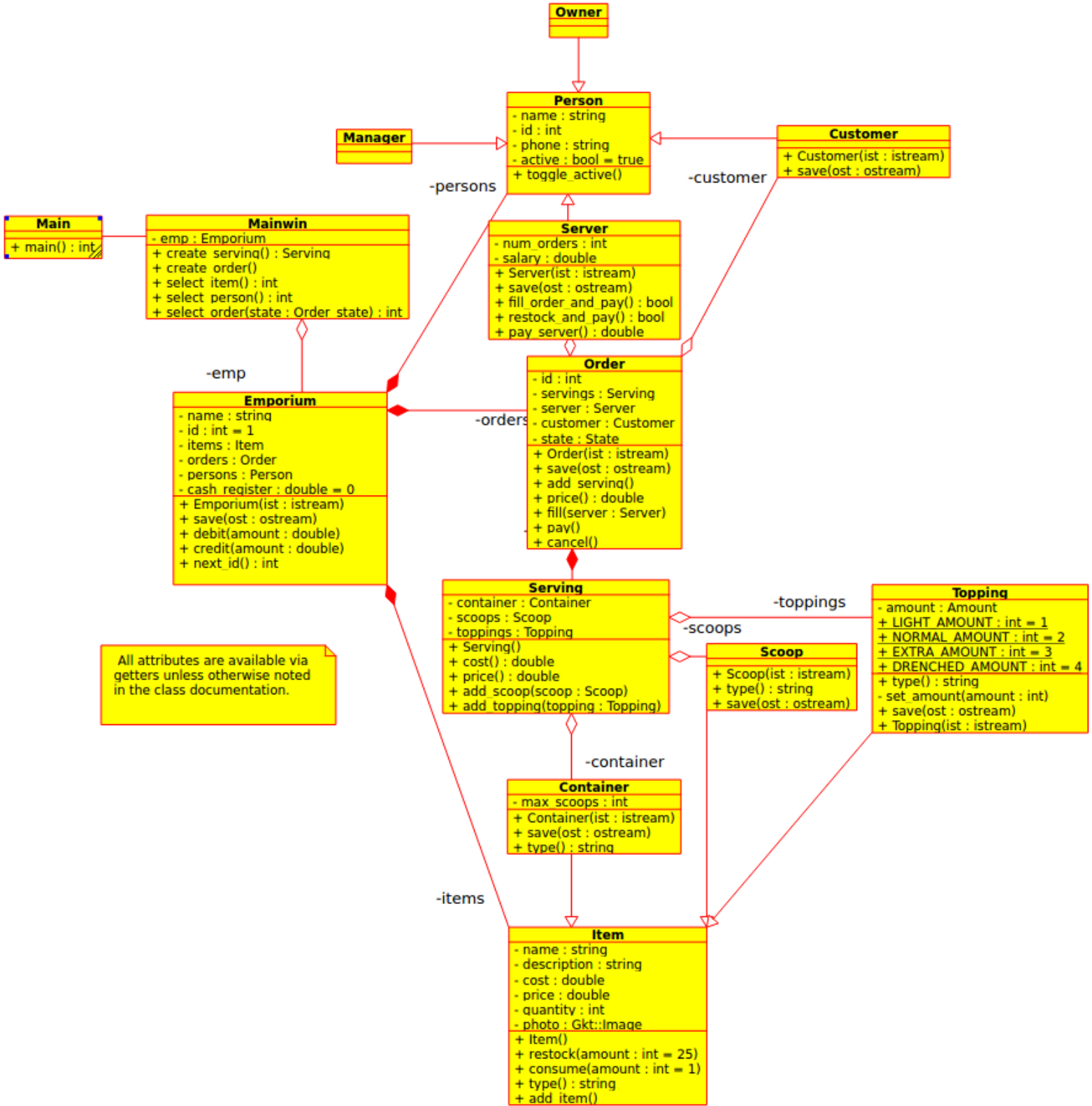
Demo on TBD

	Remaining	Completed (this day)
Total Tasks	**37**	
Day 1 Left	**37**	**0**
Day 2 Left	**37**	**0**
Day 3 Left	**37**	**0**
Day 4 Left	**37**	**0**
Day 5 Left	**37**	**0**
Day 6 Left	**30**	**7**
Day 7 Left	**4**	**26**



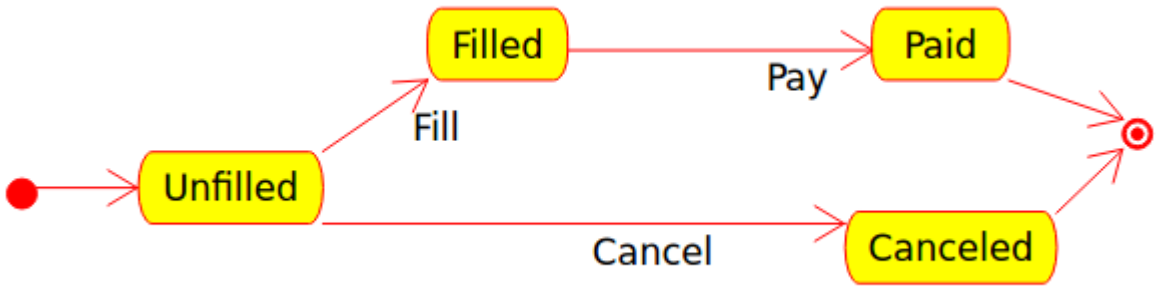
Task ID	Feature ID	Assigned To	Description	Status	Notes
1	CE		Create Emporium class	Completed Day 6	
2	CE		Create Emporium class regression test		
3	CE		Migrate the emporium data to the Emporium class	Completed Day 6	Eventually a vector, so multiple emporiums can be open
4	CE		Redirect all references to emporium data to the Emporium insta	Completed Day 6	
5	CE		Add File > New Emporium command	Completed Day 6	Implement and label as a state machine
6	CE		Add File > Switch Emporium command	Completed Day 6	
7	MST		Implement Order state machine	Completed Day 6	
8	MST		Add Process > Fill Order…	Completed Day 6	
9	MST		Add Process > Payment	Completed Day 7	
10	MST		Add Process > Cancel Order...	Completed Day 7	
11	MST		select_order by state (so only valid orders are listed)	Completed Day 7	
12	SAVD		Container: Create save to stream method	Completed Day 7	
13	LOAD		Container: Create stream constructor	Completed Day 7	
14	LOAD		Container: Update test_container	Completed Day 7	
15	SAVD		Scoop: Create save to stream method	Completed Day 7	
16	LOAD		Scoop: Create stream constructor	Completed Day 7	
17	LOAD		Scoop: Update test_scoop	Completed Day 7	
18	SAVD		Topping: Create save to stream method	Completed Day 7	
19	LOAD		Topping: Create stream constructor	Completed Day 7	
20	LOAD		Topping: Update test_topping	Completed Day 7	
21	SAVD		Customer: Create save to stream method	Completed Day 7	
22	LOAD		Customer: Create stream constructor	Completed Day 7	
23	LOAD		Customer: Update test_customer	Completed Day 7	
24	SAVD		Server: Create save to stream method	Completed Day 7	
25	LOAD		Server: Create stream constructor	Completed Day 7	
26	LOAD		Server: Update test_server	Completed Day 7	
27	SAVD		Serving: Create save to stream method	Completed Day 7	
28	LOAD		Serving: Create stream constructor	Completed Day 7	
29	LOAD		Serving: Update test_serving		
30	SAVD		Order: Create save to stream method	Completed Day 7	
31	LOAD		Order: Create stream constructor	Completed Day 7	
32	LOAD		Order: Update test_order		
33	SAVD		Emporium: Create save to stream method	Completed Day 7	
34	LOAD		Emporium: Create stream constructor	Completed Day 7	
35	LOAD		Emporium: Update test_emporium		
36	SAVD		Implement File > Save	Completed Day 7	Baseline code from CSE1325 Paint
37	LOAD		Implement File > Open	Completed Day 7	Baseline code from CSE1325 Paint


Owner
Person
- name : string
- id : int
- phone : string
- active : bool = true
+ toggle_active()
Manager
Customer
+ Customer(ist : istream)
+ save(ost : ostream)
-persons
-customer
Main
+ main() : int
Mainwin
- emp : Emporium
+ create_serving() : Serving
+ create_order()
+ select_item() : int
+ select_person() : int
+ select_order(state : Order_state) : int
Server
- num_orders : int
- salary : double
+ Server(ist : istream)
+ save(ost : ostream)
+ fill_order_and_pay() : bool
+ restock_and_pay() : bool
+ pay_server() : double
-emp
Order
- id : int
- servings : Serving
- server : Server
- customer : Customer
- state : State
+ Order(ist : istream)
+ save(ost : ostream)
+ add_serving()
+ price() : double
+ fill(server : Server)
+ pay()
+ cancel()
Emporium
- name : string
- id : int = 1
- items : Item
- orders : Order
- persons : Person
- cash_register : double = 0
+ Emporium(ist : istream)
+ save(ost : ostream)
+ debit(amount : double)
+ credit(amount : double)
+ next_id() : int
-orders
Serving
- container : Container
- scoops : Scoop
- toppings : Topping
+ Serving()
+ cost() : double
+ price() : double
+ add_scoop(scoop : Scoop)
+ add_topping(topping : Topping)
-toppings
-scoops
Topping
- amount : Amount
+ LIGHT_AMOUNT : int = 1
+ NORMAL_AMOUNT : int = 2
+ EXTRA_AMOUNT : int = 3
+ DRENCHED_AMOUNT : int = 4
+ type() : string
- set_amount(amount : int)
+ save(ost : ostream)
+ Topping(ist : istream)
Scoop
+ Scoop(ist : istream)
+ type() : string
+ save(ost : ostream)
All attributes are available via getters unless otherwise noted in the class documentation.
-container
Container
- max_scoops : int
+ Container(ist : istream)
+ save(ost : ostream)
+ type() : string
-items
Item
- name : string
- description : string
- cost : double
- price : double
- quantity : int
- photo : Gkt::Image
+ Item()
+ restock(amount : int = 25)
+ consume(amount : int = 1)
+ type() : string
+ add_item()

Emporium Class

```
class Emporium {
    public:
        Emporium(std::string name);
        Emporium(std::istream& ist);
        void save(std::ostream& ost);

        std::string name();
        double cash_register();
        void debit(double amount);
        void credit(double amount);
        int next_id();

        int num_containers();  // and scoops, toppings,
                               //orders, servers, and customers
        Container& container(int index);         // ditto
        void add_container(Container container); // ditto

    private:
        std::string _name;
        double _cash_register;
        int _id;

        std::vector<Mice::Container> _containers;
        std::vector<Mice::Scoop> _scoops;
        std::vector<Mice::Topping> _toppings;
        std::vector<Mice::Order> _orders;
        std::vector<Mice::Server> _servers;
        std::vector<Mice::Customer> _customers;
};
```

Order State Machine



Order State Machine

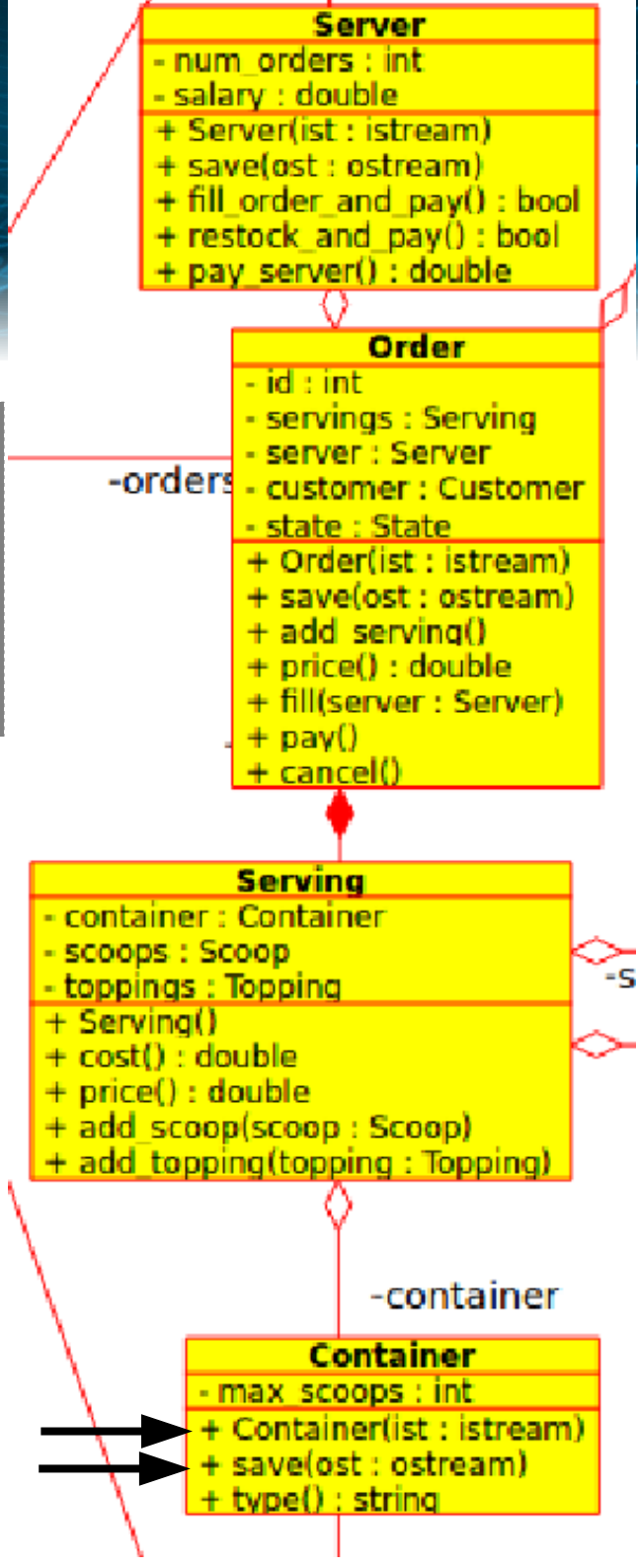
```
// STATE MACHINE that manages Order's state
void Order::process_event(Order_event event, Server server) {
    if (_state == Order_state::Unfilled) {
        if (event == Order_event::Fill) {
            _state = Order_state::Filled;
            _server = server;
        } else if (event == Order_event::Cancel) {
            _state = Order_state::Canceled;
        } else {
            throw std::runtime_error("Invalid state transition in Unfilled");
        }
    } else if (_state == Order_state::Filled) {
        if (event == Order_event::Pay) {
            _state = Order_state::Paid;
        } else {
            throw std::runtime_error("Invalid state transition in Filled");
        }
    } else if (_state == Order_state::Paid) {
        throw std::runtime_error("State transition attempted in Paid");
    } else if (_state == Order_state::Canceled) {
        throw std::runtime_error("State transition attempted in Canceled");
    } else {
        throw std::runtime_error("Invalid state");
    }
}
```

File Save and Load

```
class Container : public Item {
  public:
    Container(std::string name, std::string description,
          double cost, double price, int max_scoops);
    Container(std::istream& ist);
    void save(std::ostream& ost);
// continued...
```

```
#
CONTAINER
Waffle Cone
0.35
0.75
0
3
Crunchy wrapped waffle cake
```

Each class is responsible for saving and loading itself to / from an iostream. Each field is on a different line.



File Save and Load

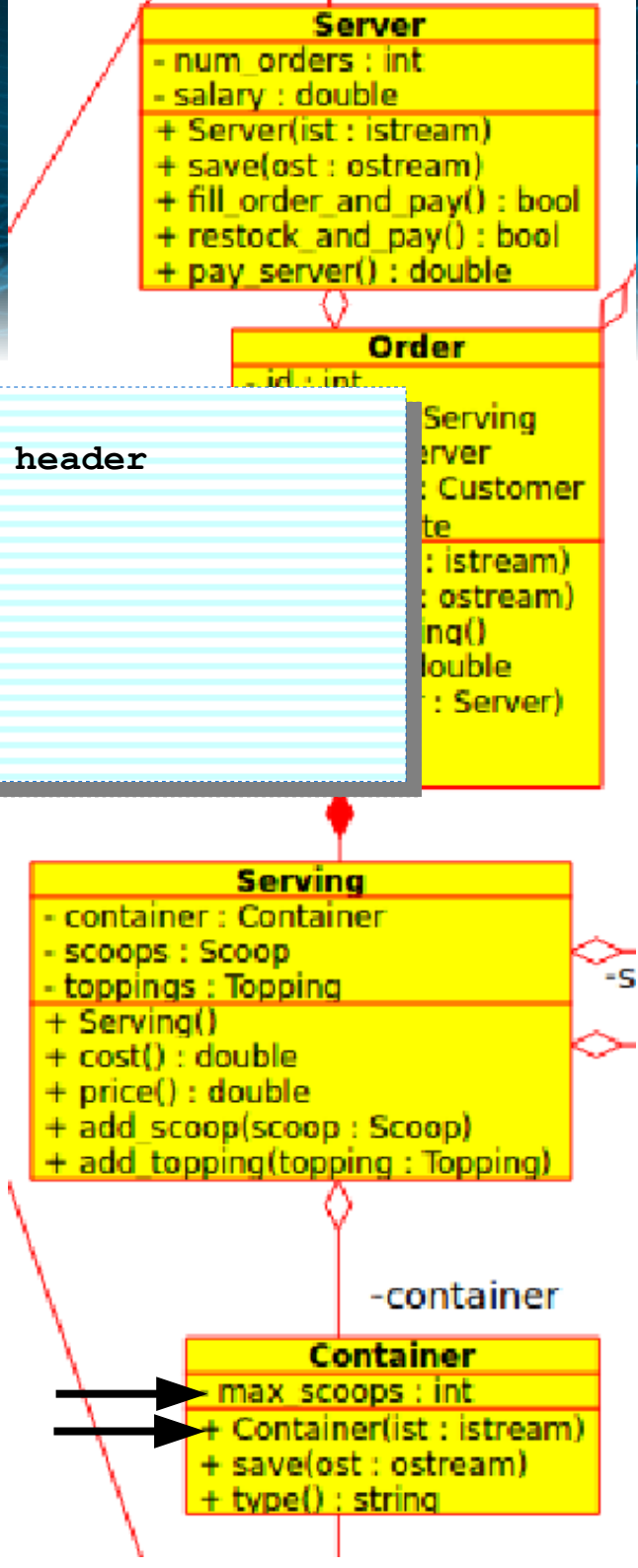
```
void Container::save(std::ostream& ost) {
    ost << "#" << std::endl << "CONTAINER" << std::endl; // header
    ost << _name << std::endl;
    ost << _cost << std::endl;
    ost << _price << std::endl;
    ost << _quantity << std::endl;
    ost << _max_scoops << std::endl;
    ost << _description << std::endl;
}
```

A header consisting of “#” (to mark start of record) and “CONTAINER” (to identify the class) is added to output.

```
Container::Container(std::istream& ist) {
    // The header must have been stripped from
    // the incoming stream at this point
    getline(ist, _name);
    ist >> _cost; ist.ignore();
    ist >> _price; ist.ignore();
    ist >> _quantity; ist.ignore();
    ist >> _max_scoops; ist.ignore();
    getline(ist, _description);
}
```

```
#
CONTAINER
Waffle Cone
0.35
0.75
0
3
Crunchy wrapped waffle cake
#
```

We assume the header is stripped on input (that’s how the Open method knew what was next in the stream!)



Testing Save and Load for Each Class

```
// Test I/O
std::ostringstream ost;
container.save(ost);  // Save to a stringstream to save the object

std::istringstream ist{ost.str()};
std::string header1, header2;
getline(ist, header1);
getline(ist, header2);
if (header1 != "#" && header2 != "CONTAINER") {
    // Error message on failure
}

Mice::Container clone{ist}; // Read the stringstream to reconstruct the object

if (container.name() != clone.name() ||
  container.description() != clone.description() ||
  container.cost() != clone.cost() ||
  container.price() != clone.price() ||
  container.type() != clone.type() ||
  container.max_scoops() != clone.max_scoops()) {
    // Error message on failure
    passed = false;
}
```

Composite File Save and Load

Serving and Order

```
void Serving::save(std::ostream& ost) {
    ost << "#" << std::endl << "SERVING" << std::endl; // header
    _container.save(ost);
    for (Scoop& s : _scoops) s.save(ost);
    for (Topping& t : _toppings) t.save(ost);
    ost << "#" << std::endl
        << "END SERVING" << std::endl; // footer
}
```

Composite classes also use a footer to signal the end of the object.

```
Serving::Serving(std::istream& ist) {
    // The header must have been stripped from the incoming stream at this point
    std::string header1, header2;
    while (true) {
        std::getline(ist, header1); // header
        std::getline(ist, header2);
        if (header1 != "#") throw std::runtime_error("missing # during input");
        if (header2 == "END SERVING") break;  // footer
        else if (header2 == "CONTAINER") _container = Mice::Container{ist};
        else if (header2 == "SCOOP") _scoops.push_back(Mice::Scoop{ist});
        else if (header2 == "TOPPING") _toppings.push_back(Mice::Topping{ist});
        else throw std::runtime_error("invalid item type in Serving");
    }
}
```

Emporium Save

```
void Emporium::save(std::ostream& ost) {
        ost << "MICE" << std::endl << "0.1" << std::endl; // magic cookie
        ost << "#" << std::endl << "EMPORIUM" << std::endl; // header
        ost << _name << std::endl;
        ost << _cash_register << std::endl;
        ost << _id << std::endl;

        for (Mice::Container c : _containers) c.save(ost);
        for (Mice::Scoop s : _scoops) s.save(ost);
        for (Mice::Topping t : _toppings) t.save(ost);
        for (Mice::Order o : _orders) o.save(ost);
        for (Mice::Server s : _servers) s.save(ost);
        for (Mice::Customer c : _customers) c.save(ost);
        ost << "#" << std::endl << "END EMPORIUM" << std::endl; // footer
}
```

The Emporium class initiates the save, with a “magic cookie” (MICE followed by a version number), and then a header (“#” followed by “EMPORIUM”). But it otherwise follows the same pattern, telling each of its composite objects to save itself to the output stream.

Emporium Load
(1 of 2)

```
Emporium::Emporium(std::istream& ist) {
    // WARNING: Do NOT strip the header - pass the FULL FILE to Emporium!
    std::string header1, header2;

    std::getline(ist, header1); // magic cookie
    std::getline(ist, header2);
    if (header1 != "MICE")
        throw std::runtime_error("NOT an Emporium file");
    if (header2 != "0.1")
        throw std::runtime_error("Incompatible file version");

    std::getline(ist, header1); // header
    std::getline(ist, header2);
    if (header1 != "#")
        throw std::runtime_error("No Emporium records in file");
    if (header2 != "EMPORIUM")
        throw std::runtime_error("Malformed Emporium record");

    std::getline(ist, _name);
    ist >> _cash_register; ist.ignore();
    ist >> _id; ist.ignore();
```

We carefully perform data validation on the first part of the file.
If that passes, we throw caution to the wind and try to load it.

Emporium Load
(1 of 2)

```
    while(ist) {
        std::getline(ist, header1); // header
        std::getline(ist, header2);

        if (header1 != "#") throw std::runtime_error("missing # during input");
        if (header2 == "CONTAINER") _containers.push_back(Container{ist});
        else if (header2 == "SCOOP") _scoops.push_back(Scoop{ist});
        else if (header2 == "TOPPING") _toppings.push_back(Topping{ist});
        else if (header2 == "ORDER") _orders.push_back(Order{ist});
        else if (header2 == "SERVER") _servers.push_back(Server{ist});
        else if (header2 == "CUSTOMER") _customers.push_back(Customer{ist});
        else if (header2 == "END EMPORIUM") break;
        else throw std::runtime_error("invalid item type in Emporium");
    }
}
```

The order of objects in the file overall is irrelevant. We simply check the header, and then instance the type of object the header specifies and push it on its vector.

Gtkmm Event Handlers

```
void Mainwin::on_file_open_click() {
    try {
        std::ifstream ifs{"emporium.emp", std::ifstream::in};
        _emp = new Mice::Emporium{ifs};
    } catch (std::exception& e) {
        Gtk::MessageDialog dialog{*this, "Unable to open emporium.emp"};
        dialog.set_secondary_text(e.what());
        dialog.run();
        dialog.close();
    }
}

void Mainwin::on_file_save_click() {
    try {
        std::ofstream ofs{"emporium.emp", std::ofstream::out};
        _emp->save(ofs);
    } catch (std::exception& e) {
        Gtk::MessageDialog dialog{*this, "Unable to save emporium.emp"};
        dialog.set_secondary_text(e.what());
        dialog.run();
        dialog.close();
    }
}
```

The event handlers require only 2 lines to load or save a file – one to create the stream, and the other to tell the emporium to save or load itself. The rest handles any exceptions.