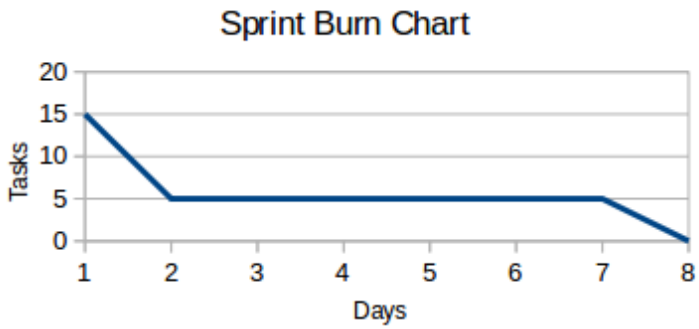
Sprint 3 Backlog

The Plan for the Suggested Solution

TBD

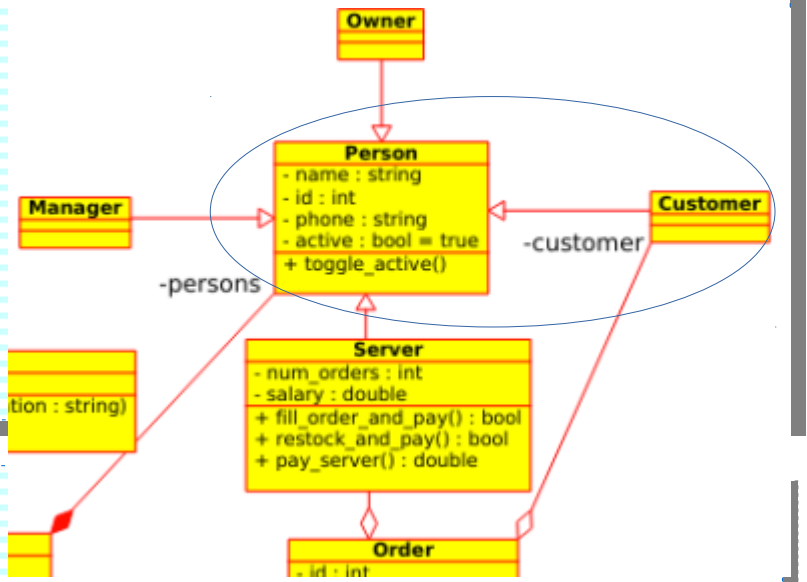
Remaining	Completed (this day)
15	
5	10
5	0
5	0
5	0
5	0
5	0
0	5



Feature ID	Assigned To	Description	Status	Notes
CTM		Add Person class	Completed Day 1	
CTM		Add test_person regression test	Completed Day 1	
CB		Add Customer class	Completed Day 1	
CB		Add test_customer regression test	Completed Day 1	
CTM		Add Server class	Completed Day 1	
CTM		Add test_server regression test	Completed Day 1	
CB		Add Create > Customer menu item	Completed Day 1	
CB		Add on_create_customer_click callback	Completed Day 1	
CTM		Add Create > Server menu item	Completed Day 1	
CTM		Add on_create_server_click callback	Completed Day 1	
CO		Add Order class	Completed Day 7	
CO		Update on_create_order_click callback	Completed Day 7	Currently creates a serving – now create an order, and add servings
CO		Add data validation and convenience shortcuts	Completed Day 7	
CSB		Create Role > Owner, Manager, Server, and Customer	Completed Day 7	
COB		Set sensitivity of menu items and tool buttons for each	Completed Day 7	Tool button sensitivity can be set to menu item's at end of handler

Person and Customer

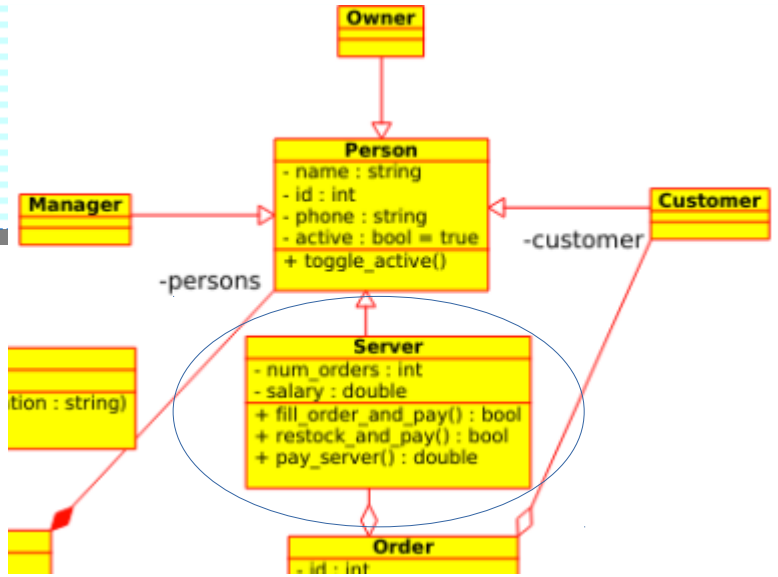
```
namespace Mice {
    class Person {
      public:
        Person(std::string name, std::string id, std::string phone);
        std::string name();
        std::string id();
        std::string phone();
        bool is_active();
        void set_active(bool active);
      private:
        std::string _name;
        std::string _id;
        std::string _phone;
        bool _active;
    };
}
```




```
#include "person.h"
#include <string>

namespace Mice {
    class Customer : public Person {
      public:
        Customer(std::string name, std::string id, std::string phone);
    };
}
```

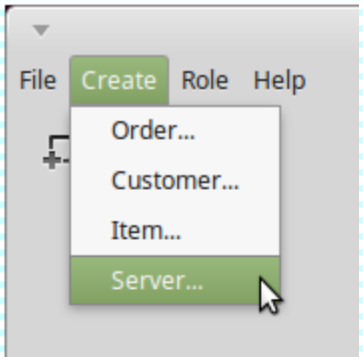
Server

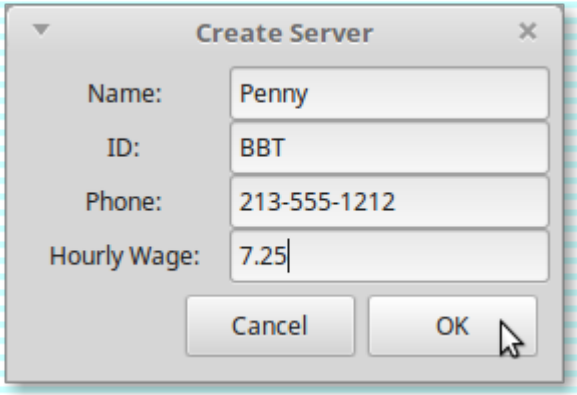
```
#include "person.h"
#include <string>

namespace Mice {
    class Server : public Person {
      public:
        Server(std::string name, std::string id, std::string phone, double salary);
        bool fill_order_and_pay();     // True if server was paid for this order
        bool restock_and_pay();        // True if server was paid for this restock
        double pay_server();           // Returns server's salary for this paycheck
        int num_orders();              // Returns number of orders filled by this server
        int salary();                  // Returns salary of this server per hour
        static const int PAY_PERIOD = 10; // Number of orders between paychecks
        static const int RESTOCK_PAY = 2; // Equivalent number of orders per restock
      private:
        int _num_orders;
        double _salary;
    };
}
```



Creating Persons' Dialogs

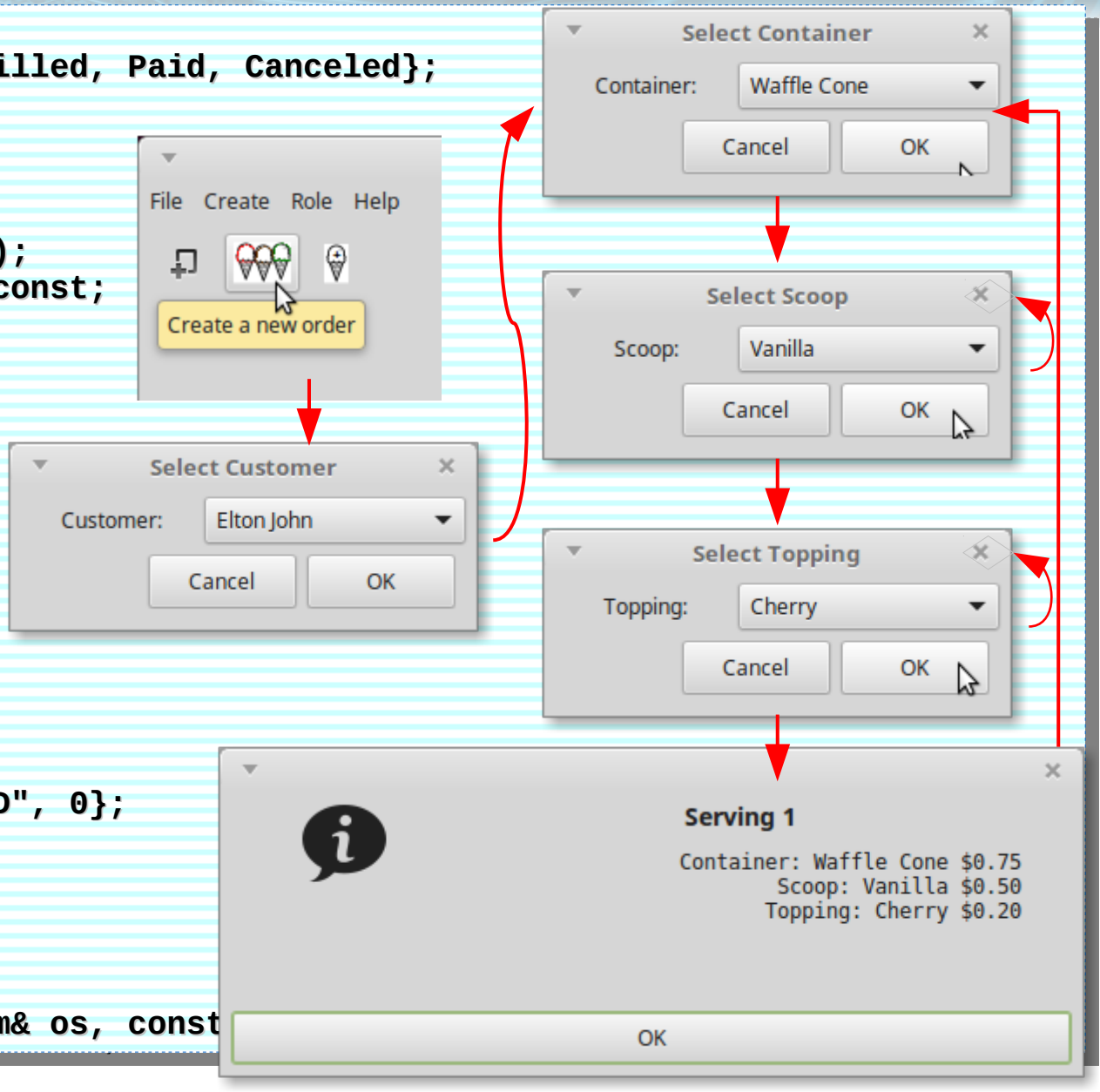
```
#include "mainwin.h"
#include <exception>
#include <stdexcept>

void Mainwin::on_create_server_click() {
    on_create_person_click("Server");
}

void Mainwin::on_create_customer_click() {
    on_create_person_click("Customer");
}

void Mainwin::on_create_person_click(std::string role) {
    const int WIDTH = 15;
    Gtk::Dialog dialog{"Create " + role, *this};

// Create and run the dialog here...

    // Instance person
    if (role == "Server") {
        Mice::Server c{e_name.get_text(), e_id.get_text(), e_phone.get_text(), d_salary};
        _servers.push_back(c);
    } else if (role == "Customer") {
        Mice::Customer c{e_name.get_text(), e_id.get_text(), e_phone.get_text()};
        _customers.push_back(c);
    }

    dialog.close();
}
```





Order

```
namespace Mice {
 enum class Order_state {Unfilled, Filled, Paid, Canceled};

 class Order {
  public:
    Order(Customer customer);
    void add_serving(Serving serving);
    std::vector<Serving> servings() const;

    void fill(Server server);
    void pay();
    void cancel();

    int id() const;
    double cost() const;
    double price() const;
    Order_state state() const;
  private:
    int _id;
    static int _next_id;
    Customer _customer;
    Server _server{"TBD", "TBD", "TBD", 0};
    Order_state _state;
    std::vector<Serving> _servings;
 };

}
std::ostream& operator<<(std::ostream& os, const
```

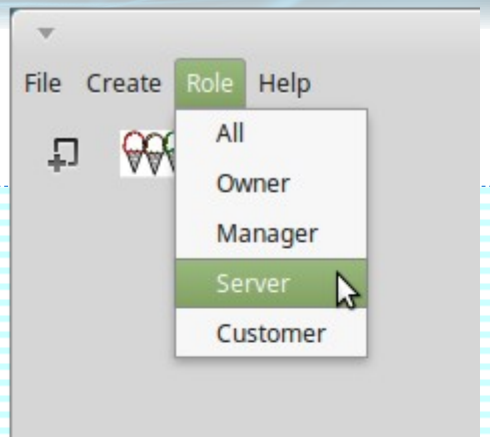


Operator Overloading

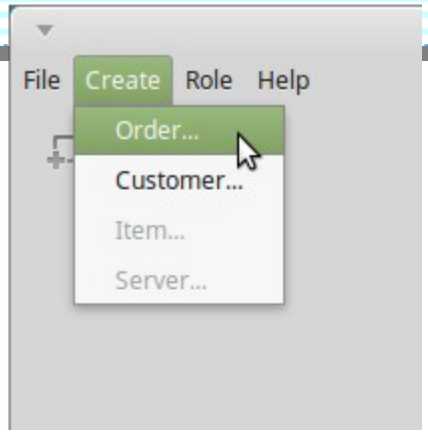
```
// OPERATOR OVERLOADING for Item and its derived classes Container, Scoop, and Topping
std::ostream& operator<<(std::ostream& os, const Mice::Item& item) {
    os << std::setw(40) << item.type() + ": " + item.name() << " $"
       << std::setprecision(2) << std::fixed << item.price();
    return os;
}
// OPERATOR OVERLOADING for class Serving
std::ostream& operator<<(std::ostream& os, const Mice::Serving& serving) {
    os << serving.container();
    for (Mice::Scoop s : serving.scoops()) os << std::endl << s;
    for (Mice::Topping t : serving.toppings()) os << std::endl << t;
    return os;
}
// OPERATOR OVERLOADING for class Order
std::ostream& operator<<(std::ostream& os, const Mice::Order& order) {
    std::string nlnl = "";
    for (Mice::Serving s : order.servings()) {os << nlnl << s; nlnl = "\n\n";}
    return os;
}
        // Convert the order to text using a string stream
        std::ostringstream os;
        os << order << std::endl;

        // Display the receipt in a dialog
        Gtk::MessageDialog dialog{*this, "Order " + std::to_string(order.id())};
        dialog.set_secondary_text("<tt>" + os.str() + "</tt>", true);
        dialog.run();
        dialog.close();
    }
```

The overloaded stream out operators makes creating a receipt order a snap!

Creating Roles




```
void Mainwin::on_server_click() {
    menuitem_new->set_sensitive(false);
    menuitem_easteregg->set_sensitive(false);
    menuitem_quit->set_sensitive(true);
    menuitem_order->set_sensitive(true);
    menuitem_customer->set_sensitive(true);
    menuitem_item->set_sensitive(false);
    menuitem_server->set_sensitive(false);

    new_emporium_button->set_sensitive(menuitem_new->get_sensitive());
    create_order_button->set_sensitive(menuitem_order->get_sensitive());
    create_item_button->set_sensitive(menuitem_item->get_sensitive());
}
```



We'll eventually want accounts with logins and passwords. For now, this demonstrates the required feature.