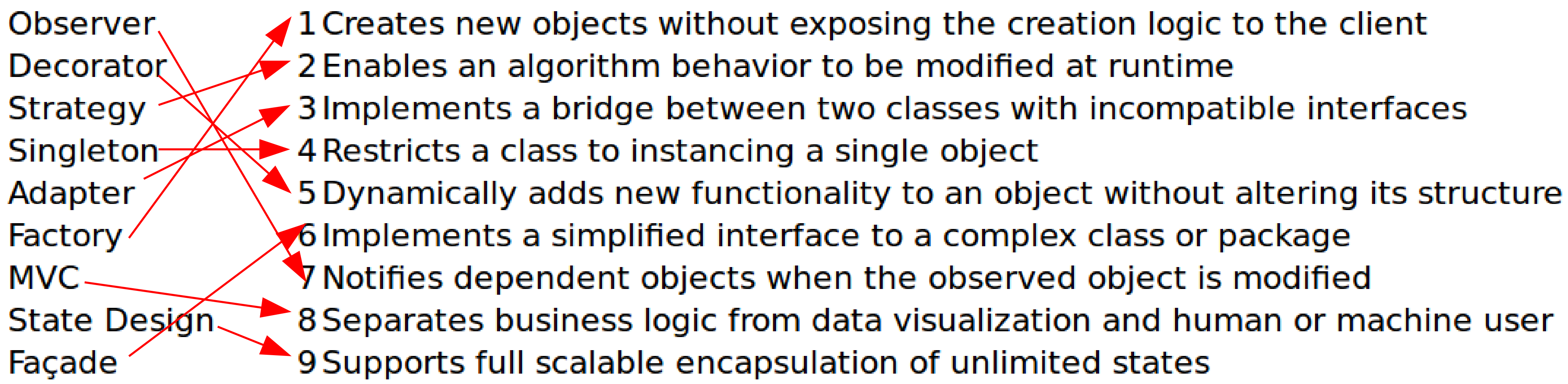
Quick Review
Lecture 22

- List 4 options for representing strings in C++ and the most significant advantage of each. (1) char* / char[ ], the “C way” (2) std::string, a true class
  (3) basic_string, a template (4) Glib::ustring, which does Unicode
- How is a map similar to a vector? What’s the most significant difference?
  Like a vector, map is a template for storing objects of any type.
  Unlike a vector, the subscript may also be of any type.
- How are key / value pairs accessed in a map? a and d
  (a) value = map[key]
  (b) map.key and map.value
  (c) iterator->key and iterator->value
  (d) iterator->first and iterator->second
- Which are common map operations? b, c, and d
  (a) navigate (b) begin and end (c) operator[ ] (d) find
- List at least 3 advantages of <random> over rand. Supports a choice of generators and distributions. Statistically valid. May be used with true random numbers.
- Which type(s) compose a <random> number? b and e
  (a) rand (b) generator (c) to_string (d) at (e) distribution

Quick Review

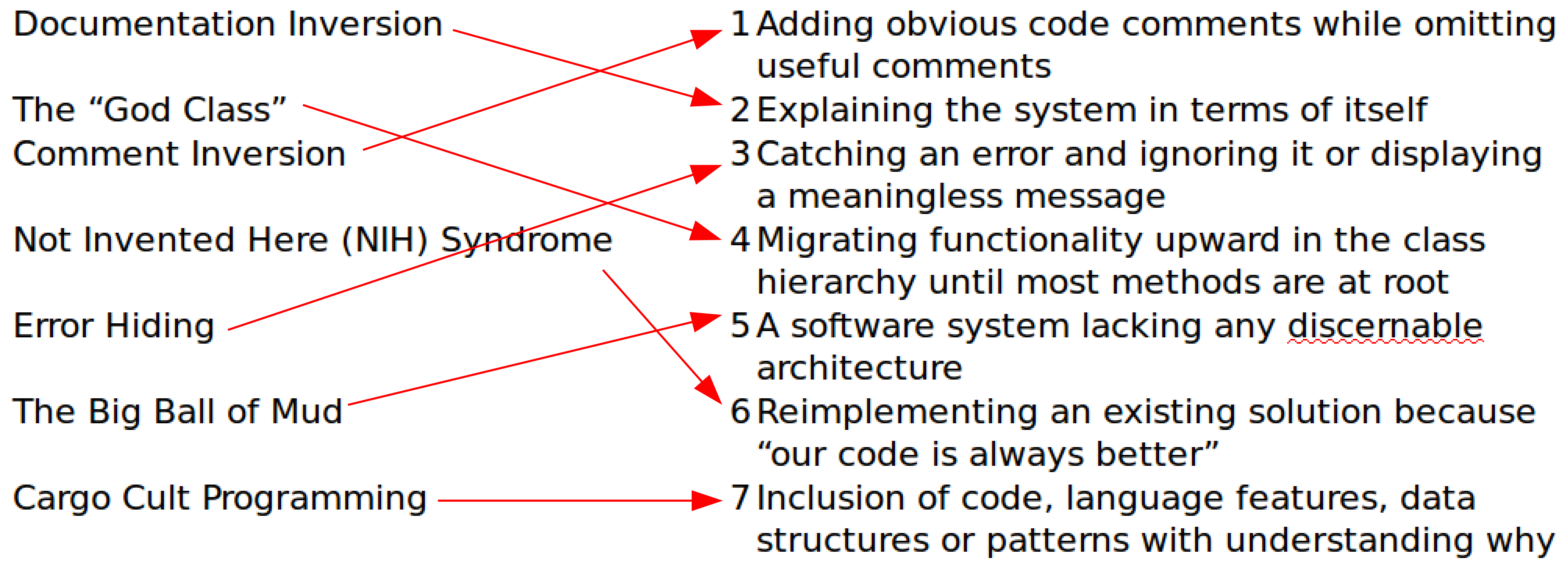
- Match Creational Pattern, Structural Pattern, and Behavioral Pattern to their definitions
  - These patterns address class and object composition, which is in general favored over inheritance Structural
  - These patterns address communication between objects Behavioral
  - These patterns address creating objects from classes via mechanisms more flexible than “new” Creational
- Match the pattern name to the application and identify its type



Observer	1 Creates new objects without exposing the creation logic to the client
Decorator	2 Enables an algorithm behavior to be modified at runtime
Strategy	3 Implements a bridge between two classes with incompatible interfaces
Singleton	4 Restricts a class to instancing a single object
Adapter	5 Dynamically adds new functionality to an object without altering its structure
Factory	6 Implements a simplified interface to a complex class or package
MVC	7 Notifies dependent objects when the observed object is modified
State Design	8 Separates business logic from data visualization and human or machine user
Façade	9 Supports full scalable encapsulation of unlimited states

Quick Review

- Match the anti-pattern name to its definition



Documentation Inversion

The “God Class”

Comment Inversion

Not Invented Here (NIH) Syndrome

Error Hiding

The Big Ball of Mud

Cargo Cult Programming

1 Adding obvious code comments while omitting useful comments

2 Explaining the system in terms of itself

3 Catching an error and ignoring it or displaying a meaningless message

4 Migrating functionality upward in the class hierarchy until most methods are at root

5 A software system lacking any discernable architecture

6 Reimplementing an existing solution because “our code is always better”

7 Inclusion of code, language features, data structures or patterns with understanding why

Quick Review

- Suggest a pattern to address each of the following concerns
  - "I want to create an object based on subjective criteria" Factory
  - "I want to switch to a more precise algorithm as my self-driving car approaches another vehicle" Strategy
  - "I want my software to be notified every time the user clicks the left mouse button" Observer
  - "I want to extend a set of Boolean methods so that each one, when called, keeps re-running until it returns true" Decorator
- Define and suggest strategies to overcome each of the following anti-patterns
  - The "God Class" Redesign using UML class hierarchy, or don't use a hierarchy
  - Not Invented Here (NIH) Syndrome Mitigate the disadvantages, e.g., escrow the code, obtain a compatible license, use Adapter or Facade pattern
  - The Big Ball of Mud Write regression tests, design target architecture, migrate slowly
  - Comment and Documentation Inversion Write for the target audience, test documentation
  - Error Hiding If you can't handle an exception well, don't handle it at all
  - Cargo Cult Programming Use only language features you understand well, keep learning!

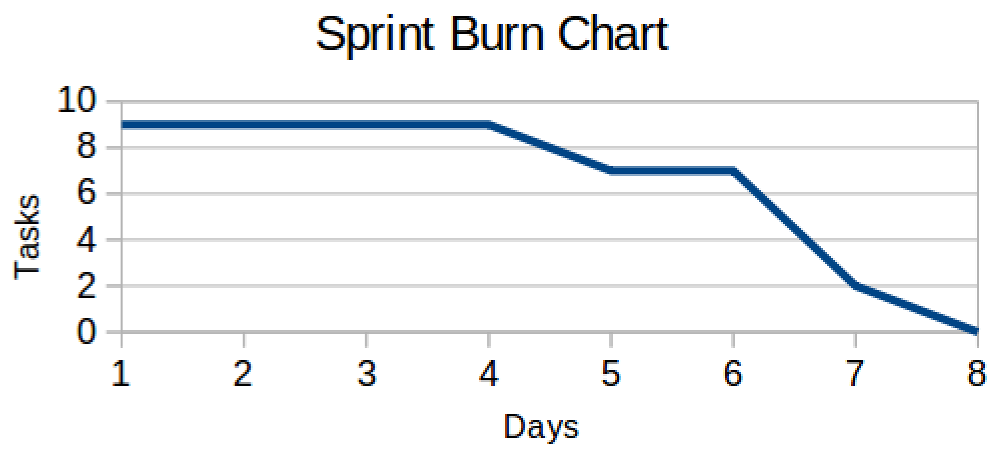
Sprint 5 Backlog

The Plan for the Suggested Solution

The workload was shorter this week due to the pending Thanksgiving Day holiday.

TBD

	Remaining	Completed (this day)
t	9	
t	9	0
t	9	0
t	9	0
t	7	2
t	7	0
t	2	5
t	0	2



	Feature ID	Assigned To	Description	Status	Notes
1	POC		Add cost to the customer’s receipt	Completed Day 4	
2	POC		Add Report > Customer	Completed Day 4	
3	POS		Add a serving report in Pango suitable for servers	Completed Day 6	
4	POS		Build the server-centric order report in Pango	Completed Day 6	
5	POS		Collect amount of topping during order	Completed Day 6	
6	POS		Insert report in a MessageDialog	Completed Day 6	
7	POS		Add Report > Server	Completed Day 6	
8			Use REGEX to validate phone numbers	Completed Day 7	
9			Use TEMPLATE to test people-related classes	Completed Day 7	

Displaying a Receipt

```
void Mainwin::on_receipt_click() {on_display_receipt_click(select_order());}
void Mainwin::on_display_receipt_click(int order) {
    // Data validation!
    if (0 > order || order >= _emp->num_orders()) return;

    // Convert the order to text using a string stream
    std::ostringstream os;
    os << _emp->order(order) << std::endl;

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

Callback for Report > Receipt – let the user select an order

The (reusable) method to display the receipt.

```
// OPERATOR OVERLOADING for class Order
std::ostream& operator<<(std::ostream& os, const Mice::Order& order) {
    double total = 0;
    for (Mice::Serving s : order.servings()) {
        os << s << std::endl << std::endl;
        total += s.price();
    }
    os << std::setw(40) << "Total: "<< " $" << std::setprecision(2) << std::fixed << total;
    return os;
}
```

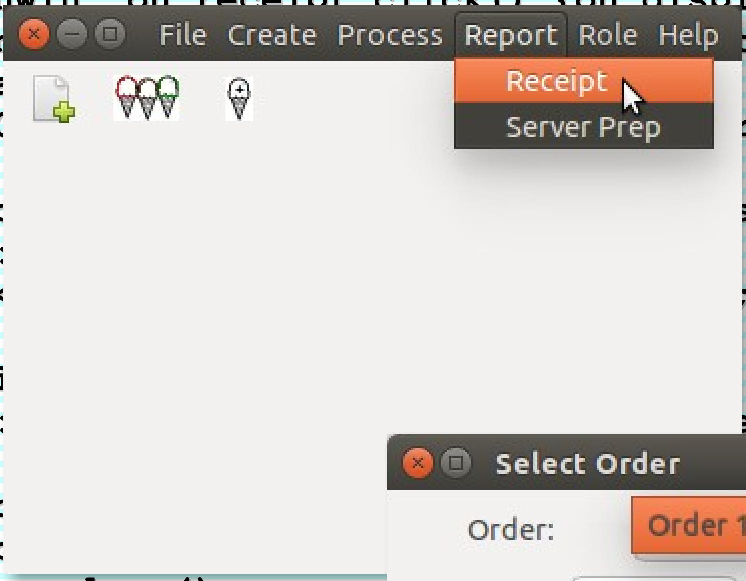
The actual receipt is generated by streaming out the Order object.

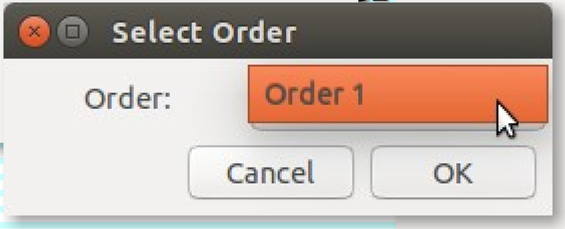
Displaying a Receipt

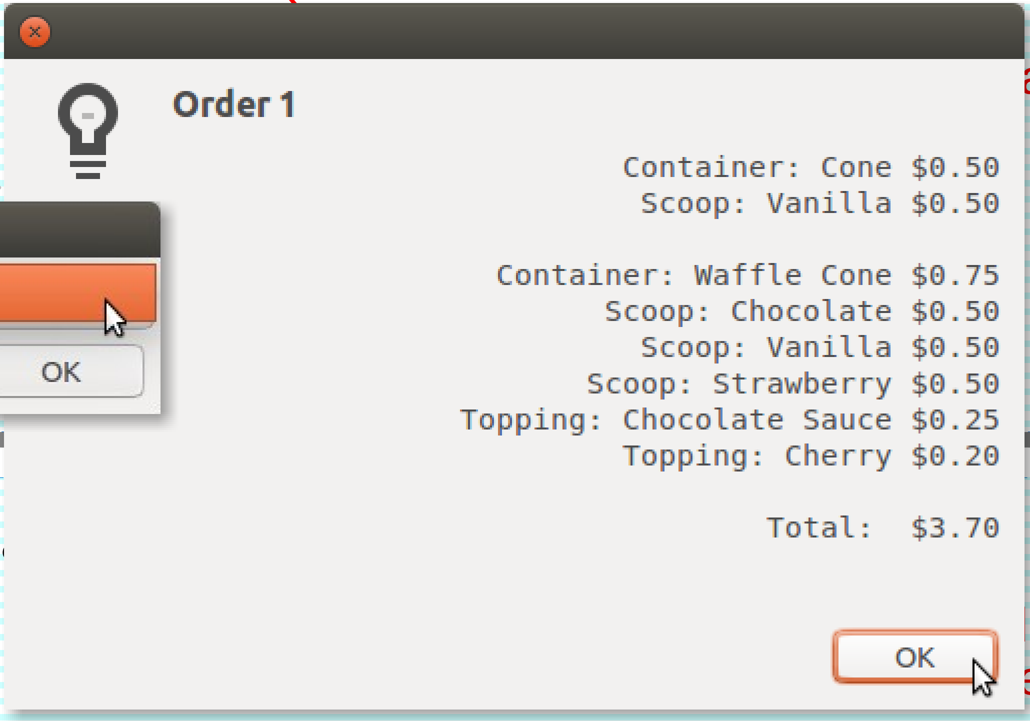
```
void Mainwin::on_receipt_click() {on_display_receipt_click(select_order());}
void Mai                                               order) {
    // Da
    if ((                                             rders()) return;

    // Co                                             string stream
    std::
    os <<                                             ;

    // Di
    Gtk::                                             er

    dial
    dial
    dialog.close();
}
```

Callback for Report > Receipt – let the user select an order

ay







Order 1

```
                   Container: Cone $0.50
                    Scoop: Vanilla $0.50

            Container: Waffle Cone $0.75
                  Scoop: Chocolate $0.50
                    Scoop: Vanilla $0.50
                 Scoop: Strawberry $0.50
         Topping: Chocolate Sauce $0.25
                   Topping: Cherry $0.20

                         Total:  $3.70
```

ect.

```
// OPERATOR OVERLOADING for class Order
std::ostream& operator<<(std::ostream& os,
    double total = 0;
    for (Mice::Serving s : order.servings()
        os << s << std::endl << std::endl;
        total += s.price();
    }
    os << std::setw(40) << "Total: "<< " $" << std::setprecision(2) << std::fixed << total;
    return os;
}
```

What to Prepare

```
void Mainwin::on_server_prep_click() {on_display_server_prep_click(select_order());}
void Mainwin::on_display_server_prep_click(int order) {
    // Data validation!
    if (0 > order || order >= _emp->num_orders()) return;

    try {
        // Display the server prep report in a dialog
        Gtk::MessageDialog dialog{*this, "Order " +
            std::to_string(_emp->order(order).id())};
        dialog.set_secondary_text("<tt>" + _emp->order(order).show_server()+"</tt>", true);
        dialog.run();
        dialog.close();
    } catch (std::exception e) {
        std::cerr << "Exception: " << e.what() << std::endl;
    }
}
```

Callback for Report > Server Prep – let the user select an order

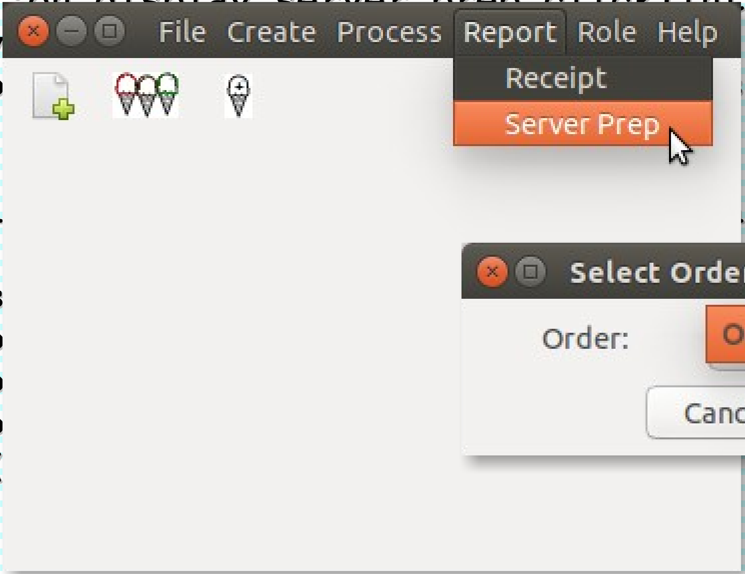
The (reusable) method to display the server prep report.

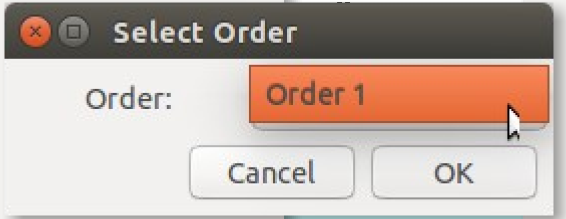
```
std::string Order::show_server() const {
    std::string result;
    for(int i=0; i < _servings.size(); ++i) {
        result += "<b>Serving " + std::to_string(i+1) + "</b>\n";
        result += _servings[i].show_server() + '\n';
    }
    return result;
}
```

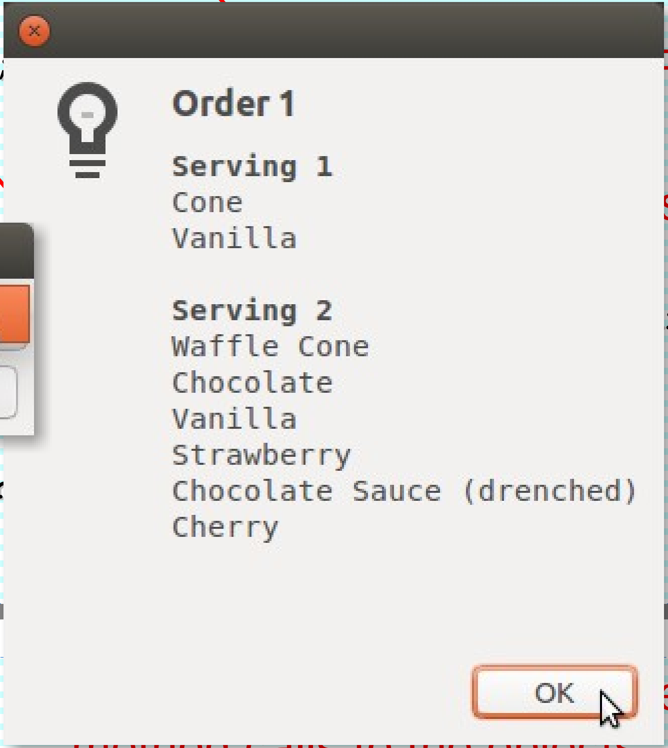
The actual report is generated via method calls to the objects.

What to Prepare

```
void Mainwin::on_server_prep_click() {on_display_server_prep_click(select_order());}
void Mainwin::on_display_server_prep_click(int order) {
    // Data v
    if (0 > o                                   ()) return;

    try {
        // Di                                   dialog
        Gtk::
            s
        dialo                                                                   rue);
        dialo
        dialo
    } catch (
        std::                                   << std::er
    }
}
```







Order 1

Serving 1
Cone
Vanilla

Serving 2
Waffle Cone
Chocolate
Vanilla
Strawberry
Chocolate Sauce (drenched)
Cherry

Prep –

splay

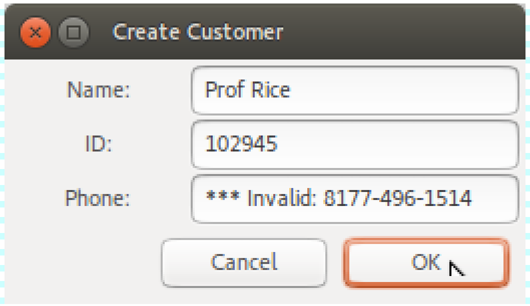
ed via

method calls to the objects.

```
std::string Order::show_server() const {
    std::string result;
    for(int i=0; i < _servings.size(); ++i) {
        result += "<b>Serving " + std::to_string(i+1) + "</b>\n";
        result += _servings[i].show_server() + '\n';
    }
    return result;
}
```

Regular Expressions (REGEX)

```
#include <regex>
// ...
    while(!valid_data) {
        if (dialog.run() != 1) {
            dialog.close();
            return;
        }

        // Data validation
        valid_data = true;

        // OTHER DATA VALIDATION GOES HERE...

        // REGEX - Supports US phone numbers only, with optional area code
        // delimited with parentheses, a dash, or no delimiters at all,
        // e.g., (817) 555-1212, 817-555-1212, 8175551212, or 555-1212

        std::regex r_phone{"(\\(?\\d{3}\\)?\\s*\\-?)?\\d{3}\\-?\\d{4}"};
        std::string phone = e_phone.get_text(); // because g++ complains otherwise
        if (!std::regex_match(phone, r_phone)) {
            e_phone.set_text("*** Invalid: " + e_phone.get_text());
            valid_data = false;
        }

        // INSTANCE THE PERSON HERE...
```


Create Customer

Name:	Prof Rice
ID:	102945
Phone:	*** Invalid: 8177-496-1514

WARNING: The regex library spontaneously segfaults in g++ 4.8.5. (Happily we're using 5.4. *whew*)

Candidate for Template

- A logical candidate for a template is selecting an object from a vector by name
  - In the suggested solution, mainwin-select.cpp
  - Select a container, ice cream scoop flavor, topping, order, customer, server...
  - This greatly reduces the code in each select method
- I didn’t take this route, because the vectors are private to Emporium (emporium.h)
  - Exposing the vectors damages the value of encapsulation
    - We would have to expose an implementation detail (our vector definitions) such that it can’t be changed later
  - The template is on the next slide for reference, though
    - It was test compiled successfully but not executed

Example Template
(would be in mainwin-select.cpp)

```
template <class T>
int select_from_vector(std::vector<T> names, std::string title) {
    Gtk::Dialog dialog_index{"Select " + title, *this};
    const int WIDTH = 15;

    // Container
    Gtk::HBox b_index;

    Gtk::Label l_index{title + ":"};
    l_index.set_width_chars(WIDTH);
    b_index.pack_start(l_index,
        Gtk::PACK_SHRINK);

    // Create dropdown list
    Gtk::ComboBoxText c_index;
    c_index.set_size_request(WIDTH*10);
    for (T s : names) c_index.append(s.name());
    b_index.pack_start(c_index, Gtk::PACK_SHRINK);
    dialog_index.get_vbox()->pack_start(b_index, Gtk::PACK_SHRINK);

    // Show dialog_index
    dialog_index.add_button("Cancel", 0);
    dialog_index.add_button("OK", 1);
    dialog_index.show_all();
    if (dialog_index.run() != 1) return -1;

    int index = c_index.get_active_row_number();
    dialog_index.close();
    return index;
}
```

```
int Mainwin::select_scoop() {
    if (_emp->num_scoops() == 0) {
        Gtk::MessageDialog dialog{*this,
            "At least 1 scoop must be created first"};
        dialog.run();
        dialog.close();
        return -1;
    }
    return select_from_vector<Scoop>(names, "Scoop");
}
```

Candidate for Template

- Instead, I chose to implement the “people tests” (for the Person, Customer, and Server classes – and soon Manager and Owner)
  - Highly redundant code that’s hard to generalize because of intermixed type references an no hierarchical relationship
- This makes the actual test code almost trivial in most cases

Template test_people
(in test_people.h, 1 of 2)

```
// TEMPLATE for testing people classes (Person, Manager, Customer, Server, etc.)
// Some additional testing may be required for some classes
template<class T>
bool test_people(std::string class_type) {
  std::string expected = "";
  bool passed = true; // Optimist!

  std::string x_name = "Charlie Chaplin";
  std::string x_id = "tramp";
  std::string x_phone = "555-1212";

  T person{x_name, x_id, x_phone};

  if (person.name() != x_name ||
      person.id() != x_id ||
      person.phone() != x_phone ||
     !person.is_active()) {
    std::cerr << "#### " << class_type << " constructor fail" << std::endl;
    std::cerr << "Expected: " << x_name << ','
                              << x_id << ','
                              << x_phone << ','
                              << "is active" << std::endl;
    std::cerr << "Actual:   " << person.name() << ','
                              << person.id() << ','
                              << person.phone() << ','
                              << (person.is_active() ? "is active" : "is not active") <<
std::endl;
    passed = false;
  }
```

Template test_people
(in test_people.h, 2 of 2)

```
  // Test set_active and is_active

  person.set_active(false);
  if (person.is_active()) {
    std::cerr << "#### " << class_type << ": setting inactive failed" << std::endl;
    std::cerr << "Expected: is not active  Actual: "
              << (person.is_active() ? "is active" : "is not active") << std::endl;
    passed = false;
  }

  person.set_active(true);
  if (!person.is_active()) {
    std::cerr << "#### " << class_type << ": setting active failed" << std::endl;
    std::cerr << "Expected: is active  Actual: "
              << (person.is_active() ? "is active" : "is not active") << std::endl;
    passed = false;
  }

  return passed;
}
```

test_person.cpp

```
#include "test_person.h"
#include "person.h"
#include "test_people.h"
#include <iostream>

bool test_person() {

  // Test constructor and is_active using TEMPLATE
  return test_people<Mice::Person>("Person");
}
```

NOTE: Regression tests may be run for the suggested solution using “make test”, then “./test”