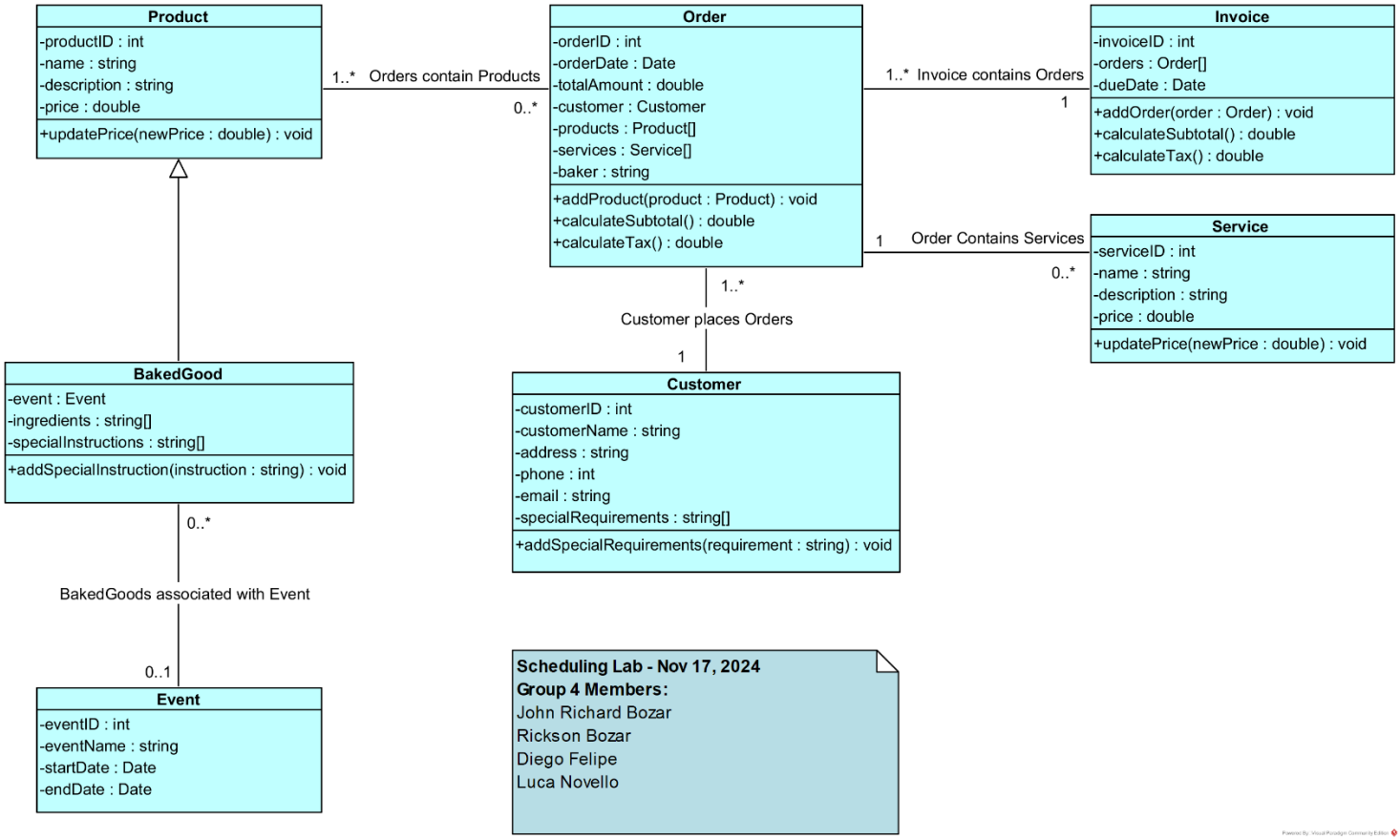
Group 4 – Scheduling Lab

John Richard Bozar | Rickson Bozar | Diego Felipe | Luca Novello

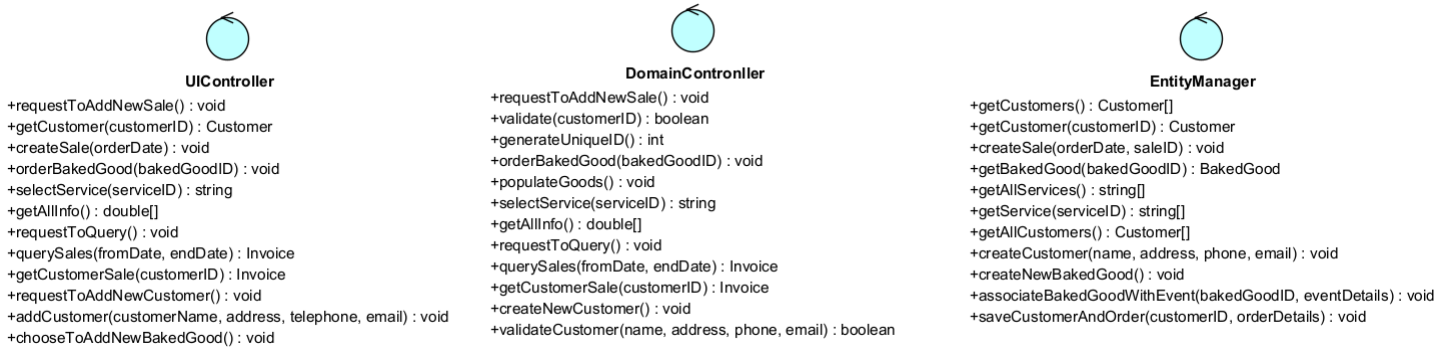
November 17, 2024

Classes:

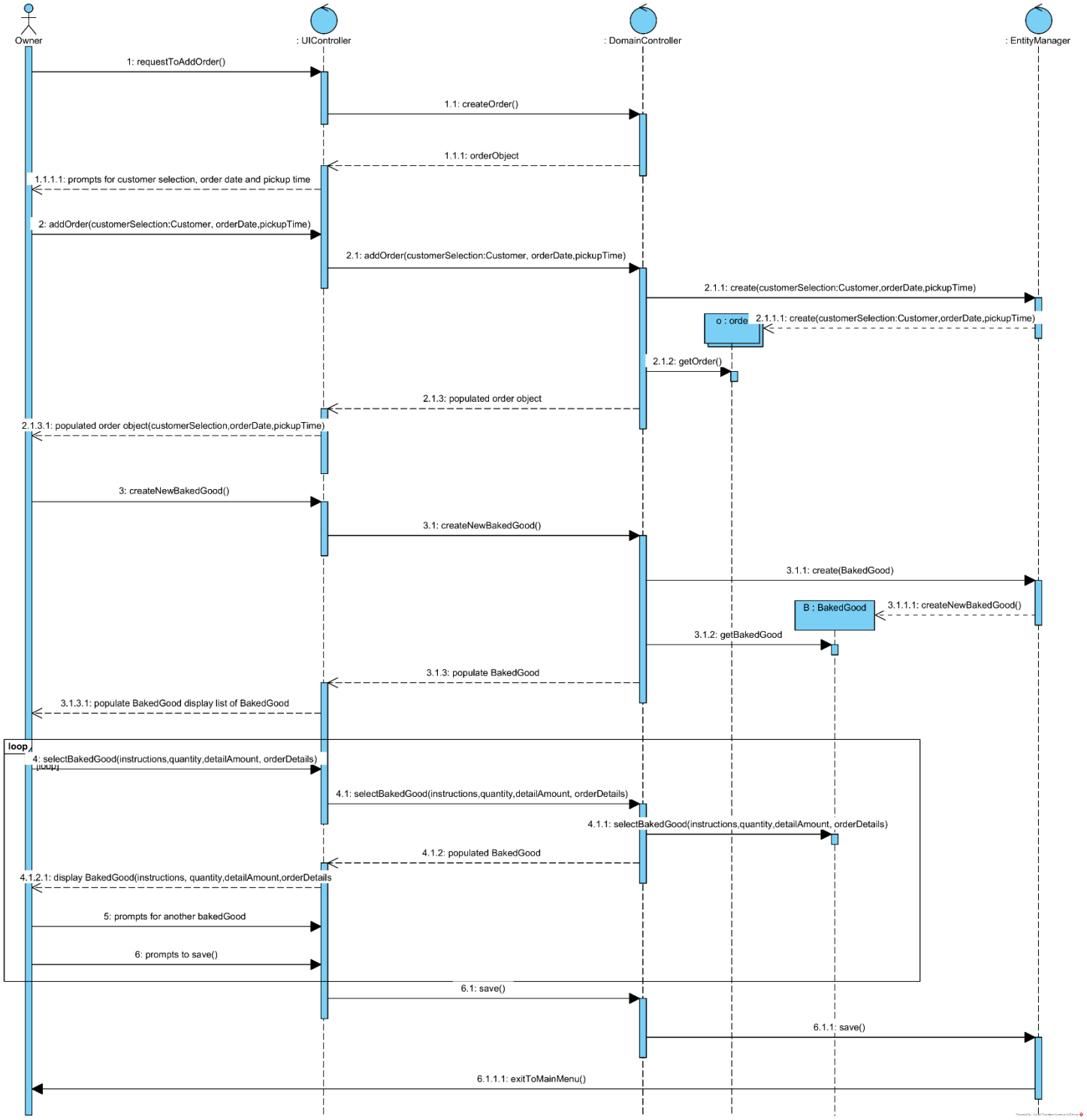


The Invoice and Service classes were added from the previous lab.

Controller Classes:



Sequence Diagram - Create Customer Order:



Wireframe – Create Customer Order

Menu: Account Information and additional options.

Order Main Section:

- Form to enter details such as:
  - Customer
  - Baked Goods
  - Date and Time
  - Order Details
  - FAQ section

.h File

UIController Class:

```
#ifndef SCHEDULING_LAB_H
#define SCHEDULING_LAB_H
#include <string>
#include <vector>

using namespace std;

class UIController {
public:
    void requestToAddNewSale(); // Initiates a new sale.
    Customer getCustomer(int customerID); // Retrieve customer details based on customerID.
    void createSale(const string& orderDate); // Creates a new sale with the given date.
    void orderBakedGood(int bakedGoodID); // Orders a baked good using its ID.
    string selectService(int serviceID); // Selects a service by its ID.
    vector<double> getAllInfo(); // Fetches Totals.
    void requestToQuery(); // Initiates a sales query.
    Invoice querySales(const string& fromDate, const string& endDate); // Queries sales within a date range.
    Invoice getCustomerSale(int customerID); // Gets sales associated with a specific customer.
    void requestToAddNewCustomer(); // Initiates the process to add a new customer.
    void addCustomer(const string& customerName, const string& address, const string& phone, const string& email); // Adds a new customer.
    void chooseToAddNewBakedGood(); // Offers to add a new baked good.
};
```

DomainController Class:

```
class DomainController {
public:
    void requestToAddNewSale(); // Handles requests to add a new sale.
    bool validate(int customerID); // Validates customer.
    int generateUniqueID(); // Generates a unique ID.
    void orderBakedGood(int bakedGoodID); // Places an order for a baked good.
    void populateGoods(); // Populates available goods.
    string selectService(int serviceID); // Selects a service.
    vector<double> getAllInfo(); // Retrieve all relevant information.
    void requestToQuery(); // Handles query requests.
    Invoice querySales(const string& fromDate, const string& endDate); // Queries sales withn a date range.
    Invoice getCustomerSale(int customerID); // Retrieve sales for a specific customer.
    void createNewCustomer(); // initiates a new customer creation.
    bool validateCustomer(const string& name, const string& address, const string& phone, const string& email); // Validates new customer details.
};
```

EntityManager Class:

```
class EntityManager {
public:
    vector<Customer> getCustomers(); // Retrieve all customers.
    Customer getCustomer(int customerID); // Gets details of a specific customer.
    void createSale(const string& orderDate, int saleID); // Creates a sale with the given date and ID.
    BakedGood getBakedGood(int bakedGoodID); // Retrieve details of a baked good.
    string getAllServices(); // Fetches all available services.
    string getService(int serviceID); // Gets details of a specific service.
    vector<Customer> getAllCustomers(); // Retrieve all customer records.
    void createCustomer(const string& name, const string& address, const string& phone, const string& email); // Adds a new customer.
    void createNewBakedGood(); // Initiates creation of a new baked good.
    void associateBakedGoodWithEvent(int bakedGoodID, const string& eventDetails); // Associates a baked good with an event.
    void saveCustomerAndOrder(int customerID, const string& orderDetails); // Saves a customer and their associated order.
};
```

Order Class:

```
class Order {
private:
    int orderID;
    int orderDate;
    double totalAmount;
    Customer customer;
    vector<Product> products;
    vector<Service> services;
    string baker;
public:
    void addProduct(const Product& product); // Adds a product to the order.
    double calculateSubtotal() const; // Calculates the subtotal of the order.
    double calculateTax() const; // Calculates the tax on the order.
};
```

Invoice Class:

```
class Invoice {
private:
    int invoiceID;
    vector<Order> orders;
    int dueDate;
public:
    void addOrder(const Order& order); // Adds an order to the invoice.
    double calculateSubtotal() const; // Calculates the subtotal of the invoice.
    double calculateTax() const; // Calculates the tax on the invoice.
};
```

Customer Class:

```
class Customer {
private:
    int customerID;
    string customerName;
    string address;
    string phone;
    string email;
    vector<string> specialRequirements;
public:
    void addSpecialRequirements(const string& requirement); // Adds a special requirement.
};
```

Service Class:

```
class Service {
private:
    int serviceID;
    string name;
    string description;
    double price;
public:
    void updatePrice(double newPrice); // Updates the price of the service.
};
```

Product Class:

```
class Product {
private:
    int productID;
    string name;
    string description;
    double price;
public:
    void updatePrice(double newPrice); // Updates the price of the product.
};
```

BakedGood Class:

```
class BakedGood : public Product {
private:
    Event event;
    vector<string> ingredients;
    vector<string> specialInstructions;
public:
    void addSpecialInstruction(const string& instruction); // Adds a special instruction.
};
```

Event Class:

```
class Event {
private:
    int eventID;
    string eventName;
    string startDate;
    string endDate;
};

#endif // SCHEDULINGLAB_H
```