UML and design patterns

...

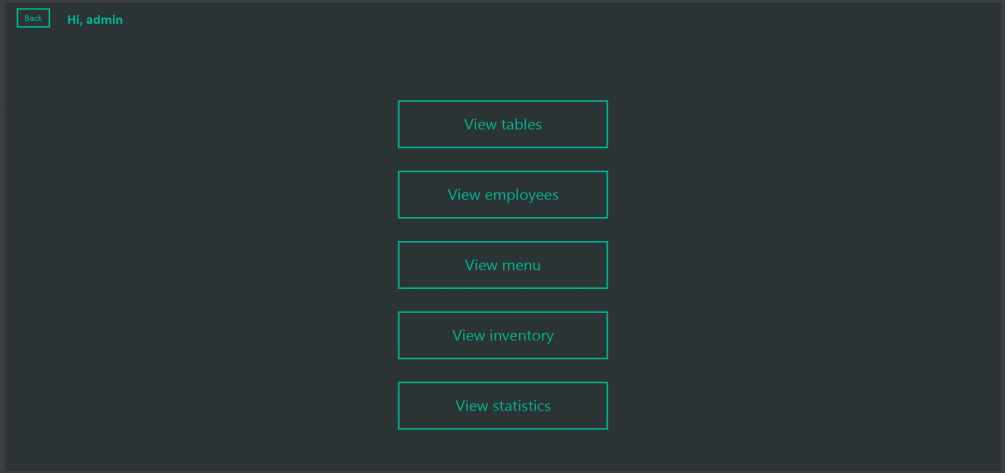
... To aid with marking

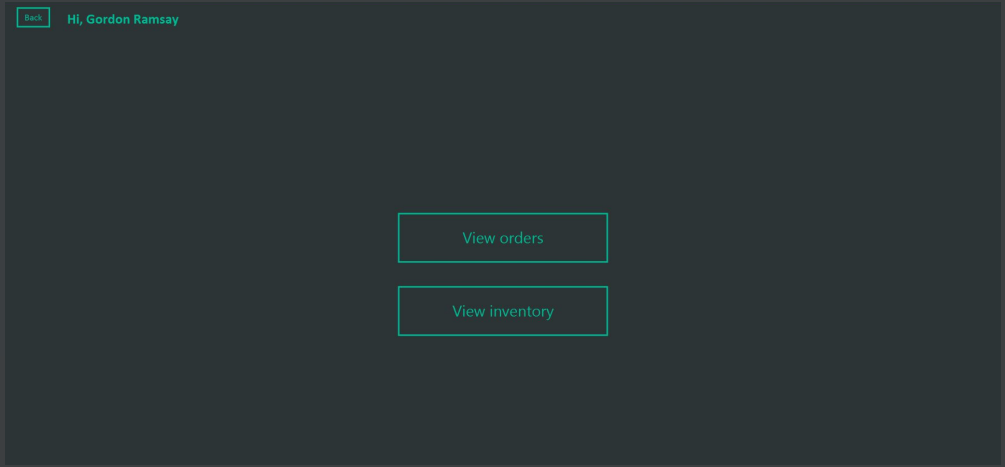
backend, controller, layouts

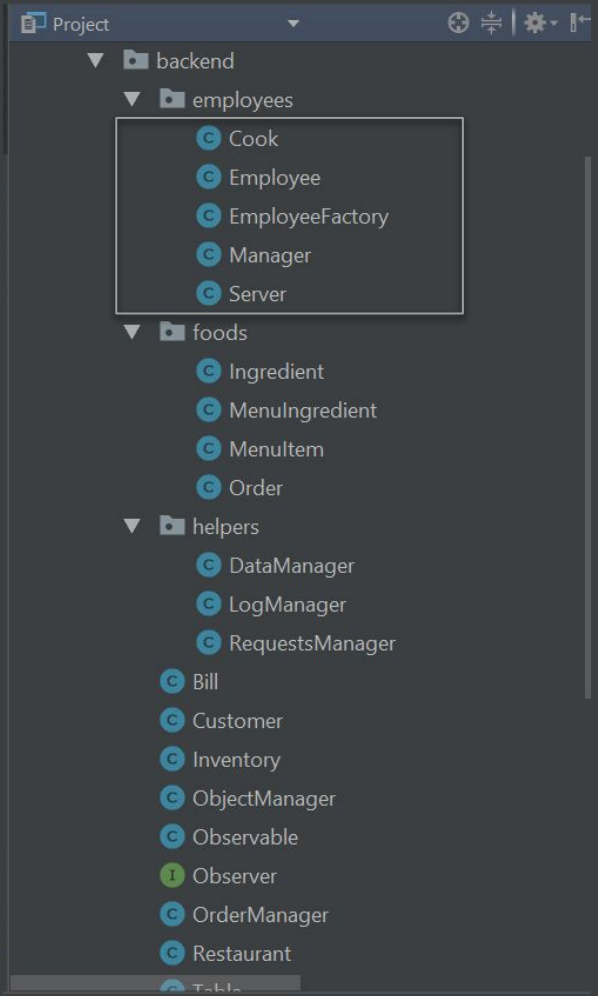
...

Three major packages

- layouts
  - cells
  - CookOrders.fxml
  - DataManager.fxml
  - Home.fxml
  - InventoryManager.fxml
  - ListAccordion.fxml
  - MonoBox.fxml
  - Pickup.fxml
  - SplitList.fxml
  - styles.css







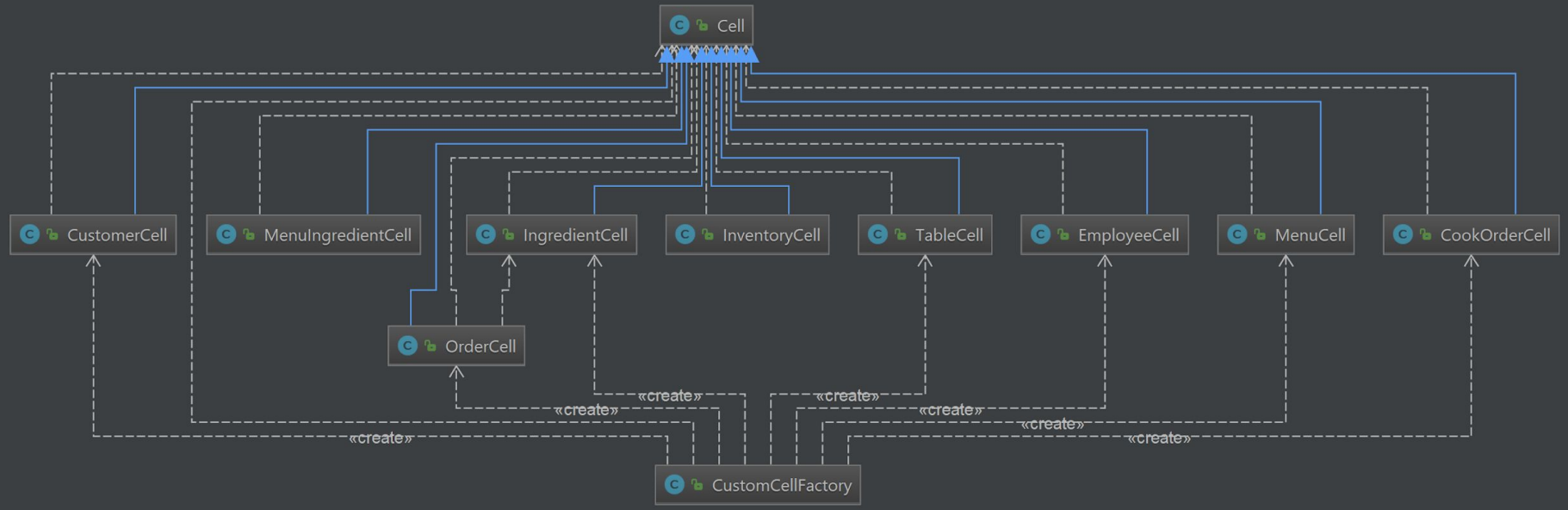
Encapsulation
Open-close
Liskov-substitution
Single-responsibility

Singleton

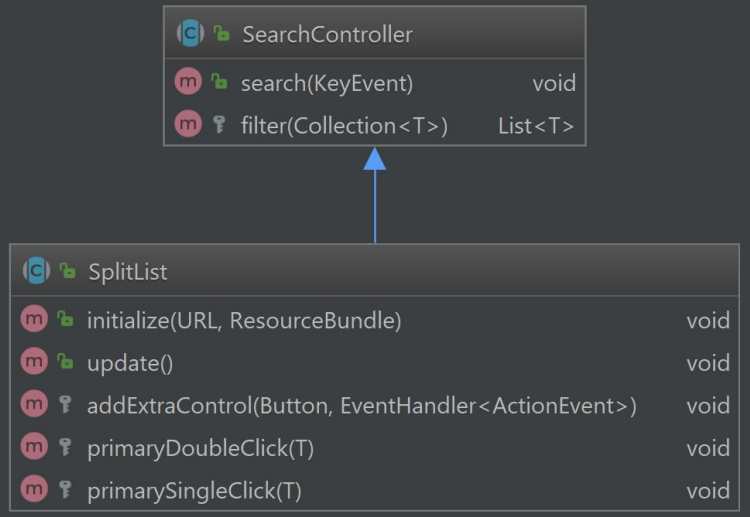
RequestsManager	
getInstance()	RequestsManager
removeIngredient(String)	boolean
removeIngredient(String, double)	boolean
addIngredient(String, double)	boolean

LogManager	
getInstance()	LogManager
log(Order, Employee)	boolean
log(Ingredient, double, double)	boolean
log(int, int)	boolean

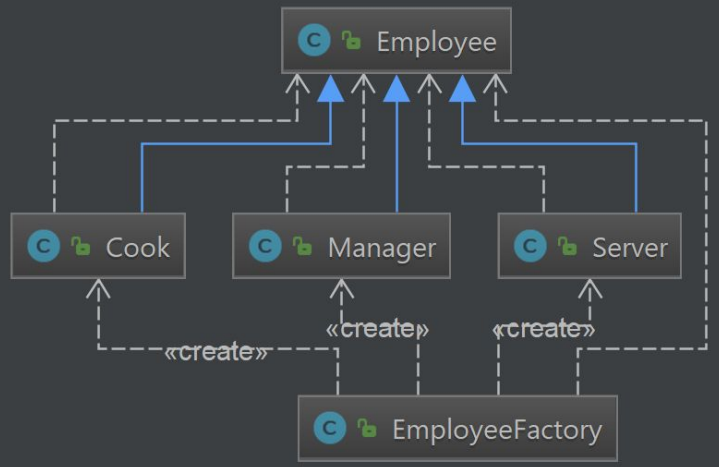
Hierarchy and avoiding duplication



More composite design

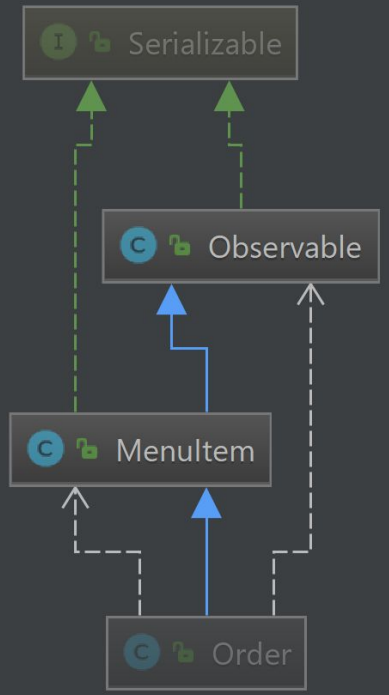


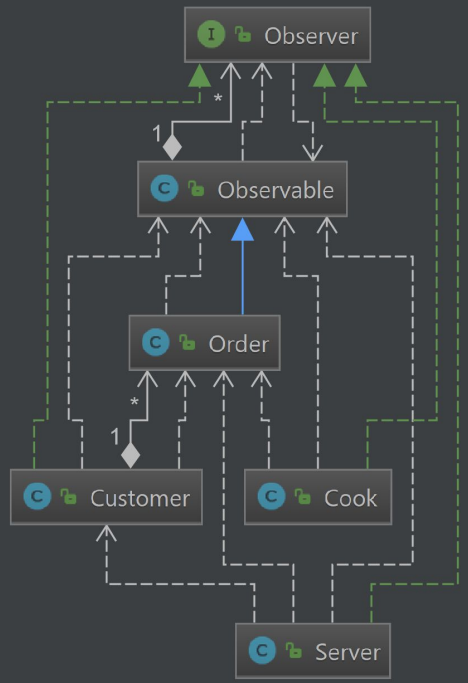
Factory (⇒ dependency injection)

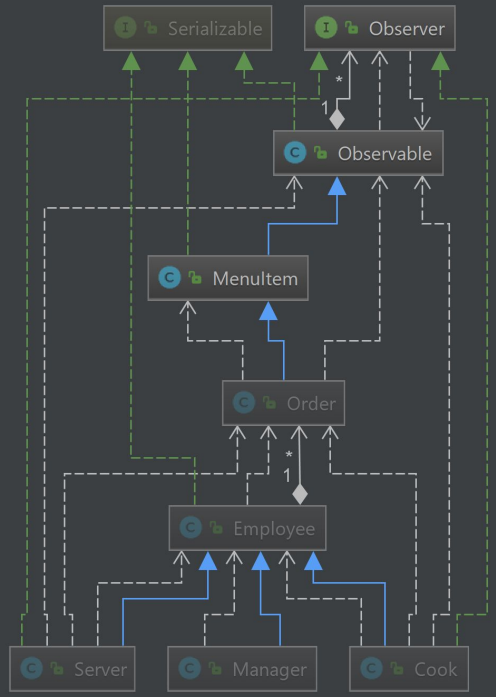


InputDialogFactory	
InputDialogFactory(String, String, String)	
getConfirmation()	boolean
getString()	String
getInteger()	Integer
getDouble()	Double
getChoice(Collection<T>)	T

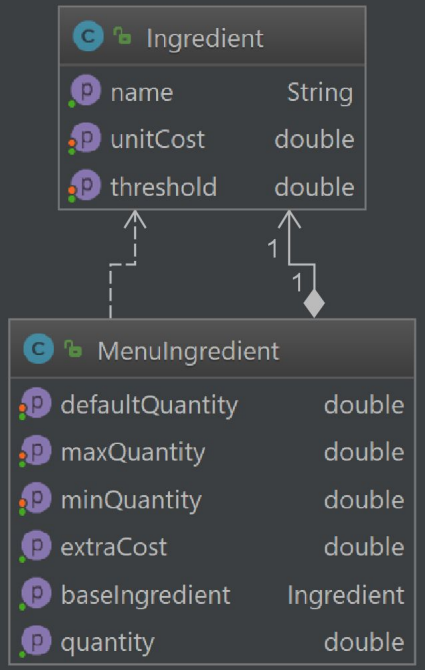
Observer (and who stores what)

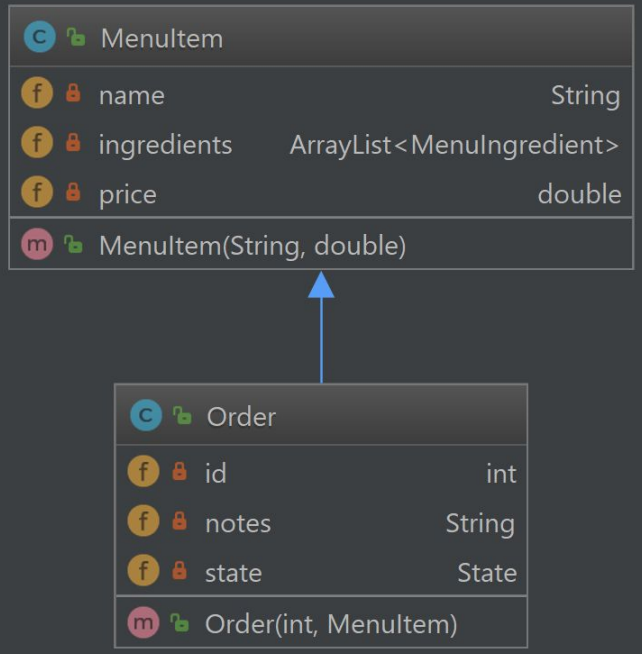




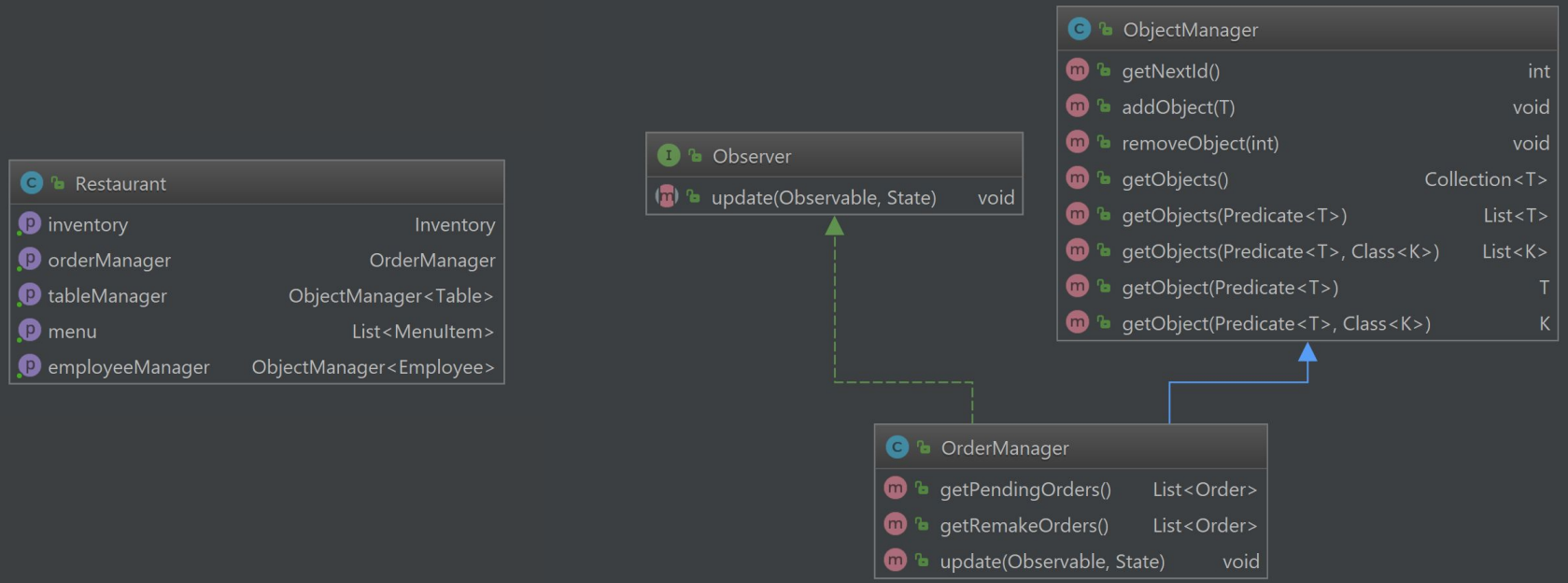


Decorator versus Inheritance





Generic object management



Model-like behaviour

- Queries

```
employeeManager.getObjects(employee -> employee.getOrders().size() > 10);
employeeManager.getObjects(employee -> employee.getName().length() < 12 && employee.getId() > 2);
```

- Centralized downcasting

```
Server server = employeeManager.getObject(employee -> employee.getId() == 2, Server.class);
Cook cook = employeeManager.getObject(employee -> employee.getId() == 1, Cook.class);

public <K extends T> K getObject(Predicate<T> predicate, Class<K> type)
```

- Less duplication (counting, storing, retrieving)

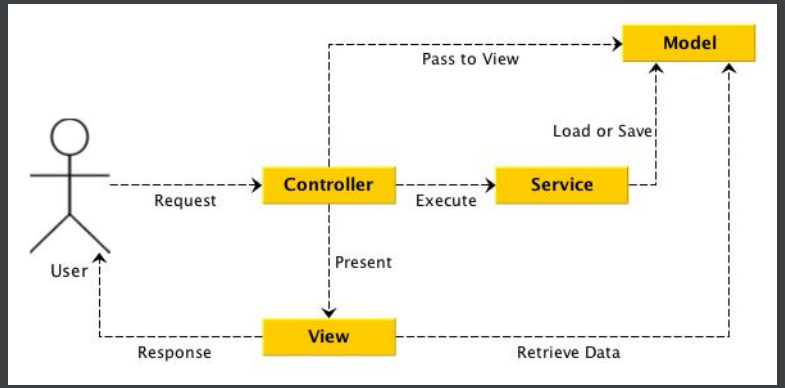
```
private int count = 0;  // Static counts don't get serialized
```

To MVC or not to MVC

- Why our design is *partially* MVC
  - + Segregation between backend, layouts and controllers
  - + “Templating”-kind structure used for layout reusability
  - - MVC typically more useful when some form of database involved
  - - Application logic at backend, not controller
    - - interaction between backend classes
    - - but less duplication in controller classes
    - - more extendable - new controllers require less effort

To MVC or not to MVC

- Real world apps - layer between model and controller
  - Model-View-Controller-Service: business logic in service
  - Think MEAN stack
  - Model-Service as our *backend*, serves as an API



Other features

Serializer	
deserialize()	T
serialize(Object)	boolean

StringHelper	
isNumeric(String)	boolean
isNumeric(String[])	boolean
isAlpha(String)	boolean
isAlpha(String[])	boolean
capitalize(String)	String

BaseController	
setup(Stage, Restaurant)	void
show()	void
navigate(BaseController)	void
back()	void
update()	void
initialize(URL, ResourceBundle)	void