**Mobile Makers Academy**

*Pre-course Material, Summer 2013 iOS Immersive*

All Rights Reserved. © Mobile Makers Academy

**Challenge 2: Time To Dig In**

I hope everything went well with your last challenge! Be sure to send us your responses to

the challenge questions if you haven't already!

Now you’re going to start getting your hands dirty. Go ahead and find the lynda.com video

series called: *Foundations of Programming: Object-Oriented Design.* Watch each video in the

following list and answer to the correlated question. And again, send us your answers when

you have completed them. As always, let us know if you get stuck!

**Section 0 - Introduction**

**0.1 What to expect from this course 3m 6s**

What is the intended purpose and potential advantage of learning object oriented design?

Pursuing object-oriented design allows developers to create complex applications that are more flexible, extendable and easily enhanced.

**0.2 Exploring object-oriented analysis, design, and development 1m 41s**

Why might it be advantageous to analyze and design before beginning programming?

Through analysis you understand the “what” you’re going to develop while design outlines the “how” or plan for development. Understanding these critical elements provides an easier roadmap for developers to follow and will save time and effort when they actually begin coding the application.

**0.3 Reviewing software development methodologies 4m 8s**

What is the difference between a "waterfall" and an "agile" approach to development? What is an iteration and how do we use them to build software?

A waterfall approach leverages a strict linear plan that has clearly defined steps or stages while agile development is a constant iterative process with each phase comprised of analysis, design and programming. An agile approach to software development has several advantages:

1. Shorter time to market - limiting functionality within each iterative period allows for faster app development
2. Iterative process allows for more flexible and responsive development
3. Supports continual development

An iteration is a project phase that include analysis, design and programming. When developing in an agile environment you conduct multiple iterations, making minor functional changes, enhancements and bug fixes in each phase.

**Section 1 - Core Concepts**

**1.1 Why we use object-orientation 2m 42s**

What are the various types of programming languages and in which domain is each used?

In addition to procedural and object-oriented programming languages which are found in the commercial space, you also have logic programming languages (like Prolog) and functional programming languages (like Haskell). These languages are highly specialized and typically found in education environments.

**1.2 What is an object? 5m 22s**

Describe in your own words the three properties of a computing object.

A object in programming is an item that possesses the following 3 characteristics:

1. Identity – Objects are unique, separate items and exist independently of other objects. Objects can also exist inside other objects yet they still maintain their identity.
2. Attributes – Objects have attributes or characteristics that define their current state or condition. For example, is the lamp on? Or off? Objects of the same class have the same attributes but those attributes do not have to be the same.
3. Behavior – Objects are not required to perform any particular functions or actions but they must have the capability to perform actions (via methods) should that need exist.

**1.3 What is a class? 4m 43s**

Explain how classes are analogous to blueprints. Include the relationship between a class and an object. Can you think of how the analogy breaks down?

A class describes what the object will be or do. For the blueprint analogy, a class (the blueprint) provides the plan for the object (the house). You can create multiple instances (i.e. houses) from a single class (i.e. blueprint).

This analogy breaks down when you get into subclassing. For example, when constructing a house from a blueprint, if someone wants a larger kitchen, you have to create a new blueprint. In programming, however, you can create a subclass that allows you to modify a particular “room” while still relying on the larger blueprint.

**1.4 What is abstraction? 2m 45s**

When a developer uses the term "abstraction" what are they describing?

A developer employs “abstraction” when creating a class and assigning it attributes and methods. By focusing only on the essential qualities of the object and ignoring the irrelevant or unimportant, a developer is able to create an “abstract” of an object that is the most flexible and extensible.

**1.5 What is encapsulation? 3m 45s**

What does encapsulation prevent? What does it enable?

Encapsulation restricts access (information/data hiding) to the inner workings of a class and prevents other objects or classes making changes to that class or the objects it creates.

Minimizing the dependencies between classes makes it easier to extend or enhance the functionality of an application. Successfully employing encapsulation enables the developer to safely change functionality within a single class without breaking the application.

**1.6 What is inheritance? 3m 35s**

Describe the inheritance relationship between classes. When would this relationship be advantageous to establish?

When a subclass is created it has access to, or inherits, all the attributes and methods of the superclass. Subclassing is most useful when you need to accommodate exceptions or minor differences from the superclass. The example used in the video is needing a “customer” subclass to the “person” class. “Customer” inherits all the attributes and methods of person (“name”, “age”, etc.) as well as its own unique attributes and methods – in this case “customerNumber”.

**1.7 What is polymorphism? 3m 22s**

What is the basic idea behind polymorphism? How can it make the classes we create more flexible?

Polymorphism allows the application to perform the correct operation depending on the input or variables. For example, when presented with two integers, the plus sign (+) adds the values but if presented with two strings, the plus sign (+) concatenates the two strings.

By combining polymorphism with subclassing, you are able to override a method in the superclass with your own custom method in the subclass and have that applied anywhere in the application that the subclass is referenced. For example, say you have a superclass called “Account” with a method “withdrawalFromAccount”. You also have a subclass called “InvestmentAccount” which has a unique requirement to add a penalty to any withdrawal. With polymorphism, you can create a new “withdrawalFromAccount” method in your “InvestmentAccount” subclass that OVERRIDES the “withdrawalFromAccount” method in “Account”. Then, anytime and anywhere “withdrawalFromAccount” is called within the application, the correct method will be called based on the variables or class being acted upon – no penalty will be charged if you are withdrawing from the “CheckingAccount” but a penalty will be assessed if you are withdrawing from the “InvestmentAccount”.

**Section 2 - Object-Oriented Analysis and Design**

**2.1 Understanding the object-oriented analysis and design processes 4m 13s**

What are the steps of analysis that come before writing code for an application? Why do you think these steps make writing the code easier?

There are 5 steps of analysis in object-oriented design:

1. Gather REQUIREMENTS – What problem are you trying to solve or functionality are you trying to provide?
2. DESCRIBE the app – Simple, conversational narrative about the applications and its functionality.
   1. Sometimes create a mock-up in this step.
3. IDENTIFY the main objects – The starting point for identifying the classes.
4. Describe the INTERACTIONS – How do objects interact and what do they need to do, in what order, etc.
5. Create the CLASS DIAGRAM – A visual representation of the classes needed.

Outlining the functionality and defining the classes ensures that the development team understands what the goals of the app (or that iteration) are and it lays out how the team is going to get there. Creating the class diagram and defining the attributes and methods for each class also ensures that the pieces of the app can be more easily combined or integrated since the classes are standardized.

**2.2 Defining requirements 6m 9s**

What should you have after you've completed the first phase of defining your requirements?

The format and detail of the final deliverable from this process will depend on the environment in which you’re working but at a minimum you should have a list of the functional (features, capabilities) and non-functional (performance standards, legal requirements, etc.) MUST HAVES for that iteration.

FURPS

* Functional reqs
* Usability
* Reliability
* Performance
* Supportability

FURPS+

+ Design reqs (iPhone, Android, etc.)

+ Implementation reqs (language used, standards, etc.)

+ Interface reqs (3rd party systems, data, etc.)

+ Physical reqs (runs on device w/camera, requires DVD, etc.)

**2.3 Introduction to the Unified Modeling Language (UML) 1m 54s**

What is UML? Why is it useful to visualize your application before coding it?

Unified Modeling Language (UML) is used to define classes and interactions.

Using UML to visualize an application allows you to quickly sketch out an idea that is quickly readable and understandable regardless of the language being employed.

**Section 3 - Utilizing Use Cases**

**3.1 Understanding use cases 6m 11s**

Write a use case for creating an event on your phone's calendar.

Title: Add an event to calendar

Actor: iPhone owner

Scenario: Phone owner selects the add event button on the calendar and enters the pertinent details, including:

1. Owner enters a title
2. Owner enter the location
3. Owner selects the start/end dates and times.
4. Owner sets event as recurring.
5. The owner sets an alert reminder.
6. The owner chooses to which calendar they prefer to add the event
7. The owner chooses whether the calendar shows them as “Free” or “Busy”.
8. The owner associates an URL with the event.
9. The owner adds additional notes to the event.
10. The owner saves the event.
11. The device saves the event and associated details and returns the owner to the calendar view.

**3.2 Identifying the actors 4m 16s**

Can you think of a use case for a mobile application in which the actor is not the user of the mobile device?

A device tracking app might qualify for this category. It is installed and runs on a device but it really interacts with a service that tracks or logs its location. The user rarely, if ever, interacts with the application on the device. Instead, the user interacts with a website or similar service.

**3.3 Identifying the scenarios 5m 7s**

Write another use case for a mobile device user interacting with a calendar application. This

Time, include a couple extensions when crafting your scenario.

Title: Add an event to calendar

Actor: iPhone owner

Scenario: Phone owner selects the add event button on the calendar and enters the pertinent details, including:

1. Owner enters a title
2. Owner enter the location
3. Owner selects the start/end dates and times.
4. Owner sets event as recurring.
5. The owner sets an alert reminder.
6. The owner chooses to which calendar they prefer to add the event
7. The owner chooses whether the calendar shows them as “Free” or “Busy”.
8. The owner associates an URL with the event.
9. The owner adds additional notes to the event.
10. The owner saves event.
11. The device saves the event and associated details and returns the owner to the calendar view.

Extensions: Owner cancels the entry.

1. The device closes the event detail view and returns the calendar view.

Extensions: Owner enters invalid date

1. Device alerts owner that date is invalid.
2. Owner enters correct date/time.
3. Owner cancels the entry.

**3.4 Diagramming use cases 4m 18s**

Do a google image search for "use case diagram." Notice how many variations there are.

What do they all generally have in common?

Show all actors

Show goals

**3.5 Employing user stories 3m 43s**

Write 5 user stories to describe a mobile user interacting with his or her maps application.

1. As a Lunch Planner I want to center the map on my current location so that I can find nearby restaurants.
2. As Salesperson I want to search for an address so that I know how far I have to travel for my meeting.
3. As a Partygoer I want to get directions to a contact’s location so that I can attend a party.
4. As a Driver I want to get driving directions to an address so that I don’t get lost.
5. As a Pedestrian I want to get directions to an address so that I can walk through the most parks.

**Section 4 - Domain Modeling (Modeling the App)**

**4.1 Creating a conceptual model 1m 59s**

Just let it soak in. No questions here.

**4.2 Identifying the classes 2m 27s**

Identify the classes in the use case you constructed for a user interacting with his or her calendar application in chapter 3.

Date

Event

Alert

Calendar

**4.3 Identifying class relationships 2m 38s**

Identify the relationships among the classes you found above. Create a conceptual model where you diagram these relationships and then upload a picture of your model below.

Calendar

Alert

Event

Date

**4.4 Identifying class responsibilities 6m 43s**

Identify the responsibilities of the classes you found above. List them here.

**4.5 Using CRC cards 2m 49s**

If you'd like, try creating CRC cards for the model you made above. There's no need to respond here, just try it out and see if you like this form of organization.

Class Responsibility Collaboration (CRC)

Class Name

Responsibilities | Collaborators

**Section 5 - Creating Classes**

**5.1 Creating class diagrams 6m 11s**

Construct Class Diagrams for the classes you imagine exist in a twitter app, a maps app, a

calendar app, or any other app you would like to make. Do you find that it is easier to come

up with the attributes or with the behaviors? Why do you think that is?

I think it is easier to come up with attributes because these are elements that are logically related to the class like name, latitude, longitude, buildingType, etc. Behaviors are more difficult because you often won’t know what behaviors are required until you begin to develop the app and understand how the classes interact and identify what information needs to be shared.

**5.2 Converting class diagrams to code 4m 57s**

How might the separation of interface and implementation in Objective-C be an advantage when working with class diagrams?

Because class diagrams illustrate the attributes and behaviors of a class, it is easier to visualize what exists within the class (the attributes or interface) and how they do what they do (the behaviors or the implementation).

**5.3 Exploring object lifetime 5m 55s**

Recall from Stage 1: What are the constructors and destructors in Objective-C? Why do we use them?

In Objective-C, the constructor takes the [[<object> alloc] init]; format and this allocates memory and instantiates the object. The destructor in Objective-C is [dealloc] which releases the reserved memory. ARC has eliminated the need to dealloc in current apps but [dealloc] must still be used when working on legacy apps.

**5.4 Using static or shared members 5m 22s**

Like the interest rate example in the video, give three additional examples of data that would be the same for all instances of a class.

transitBenefitPayment – If a company offers a public transit benefit for its employees, the payment might be the same for all people, regardless of their position within the company or income.

duesPayment – A gym may have a universal charge for its members regardless of which location members attend.

transactionFee – A company may charge a single transaction fee regardless of the type of purchase made.

**Section 6 - Inheritance and Composition**

**6.1 Identifying inheritance situations 6m 49s**

Describe in your own words what inheritance is and how it is useful when constructing classes.

In simplest terms, inheritance describes a relationship between two objects where one object (the subclass) is a form of another object (the superclass).

Inheritance is useful in the construction of classes because it allows you to extend the functionality of your classes or accommodate exceptions. For example, the basic components of a user might be in the class “USER” but a subclass of “USER” called “CUSTOMER” might contain an additional element like “CustomerID”. This way, “CUSTOMER” has access to anything in “USER” but it also contains the requisite “CustomerID”.

**6.2 Using inheritance 2m 43s**

Referring to the apps on your phone, come up with three examples where you believe methods are being inherited from superclasses and called by subclasses.

In my fantasy baseball app, there are custom cells in a table view which are subclassed from UITableViewCell.

In my poker app, there are different types of poker games available so these are likely subclassed by type: Hold ‘Em, Jump, Tourney.

ZipCar lists the available vehicles. These are probably subclassed from a vehicle superclass.