**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?**

The intended purpose and potential advantage of learning objected oriented design is to help developers abstract code and organize their code in order to save time and improve efficiency.  Object orientation gives developers the tools to build more complex and structured code that otherwise would be messy and burdensome.  Abstraction ensures that the code can be reused often and ensures that bugs can be quickly fixed by isolating the problem.  For advanced programs with thousands of lines, this is very important.

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

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

A minute of thought today prevents an hour of fixing tomorrow.  Analyzing and design before programming helps to ensure mistakes aren't made while coding, and will save the developers countless number of hours.  By diving into coding right away, there can be mistakes made which can take hours or days to fix or possibly cause the developer to throw away a lot of code.  By spending some time to analyze and design, developers can save themselves a lot of time and headache.

**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 to use them to build software?**

Waterfall methodology is a linear, step by step process of development.  Each step is planned before the start of development, and each step has to be fully approved to specification before proceeding to the next step.  Agile development is "agile" in the sense that it has iterations over a span of a week to a few weeks which can factor in the initial design, bug fixes, and other modifications to the spec on an ad hoc basis.  This ensures the product can grow organically and optimizes for the actual best outcome instead of just following a structure for the sake of structure.

**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?**

There are logic programming languages and functional programming languages that are used mostly in academia or specialized other use cases.  Object oriented programming language are used for more practical development, such as web pages, mobile apps, and a number of other commonly used cases.

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

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

The three properties of a computing object are identity, attributes, and behavior.  Identity means that each object is separate from other objects, either similar objects or different ones.  They are inherently different items.  Attribute means that there is a way to describe the object as to the state it is in, which could be it's actions or values or it contains.  The behavior means that each object is different in the things it can do, that is self contained as part of it's own identity.

**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?**

class has 2 things - attributes and behavior

name - type

attributes - properties, data

behavior - operations

A class is similar to a blueprint in that it has the specifications, layout, and structure of what it is creating without creating the actual thing itself.  You create objects from a class, since the class gives you the framework that describes the attributes and behavior each object can have.  The actual object itself can assign values to the attributes and execute the operations, but they are defined in the class.  The analogy breaks down because the blueprint gives the exact measurements and values of each piece of wood or steel, how it connects, and what it does.  Once it's put into place, it does not change.  With programming, a class just defines what things it can or cannot do or have, and the objects can be different in their values, since they aren't exact cookie cutters of each other.

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

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

Abstraction is the idea of creating a class with just the properties it needs, not other potential attributes or behaviors.  Abstraction is about simplifying design to ensure maximum value for the specific project it is used for, and nothing more.  This allows for clean design and re-use.

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

**What does encapsulation prevent? What does it enable?**

Encapsulation prevents other parts of a program from reaching in and directly accessing or changing the attributes of an object.  This ensures that the program does not have several dependencies so any one change will not have a cascading effect on other parts of the program.  Encapsulation enables the boxing of all data and methods besides what is necessary for other parts of the program to use, keep the code cleaner to read and easier to maintain.

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

**Describe the inheritance relationship between classes. When would this relationship be**

**advantageous to establish?**

Inheritance is between a sub-class and it's super class, where the sub-class will get all of the attributes and behaviors of the parent class, and can add more of either type as desired.  This is advantageous to save a programmer time since instead of having to create several different classes with many similar attributes and behaviors, they can instead create one super class and several sub classes that create their own as necessary without having to re-write more of the same code.

**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 us to override methods defined in super classes for our own custom need.  This creates flexibility because the method will know based on the inputs which method definition it should use, so there is no need to create different names for similar functionality methods.

**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?**

The steps of analysis that come before writing code are gathering requirements, describing the app, identifying the most important objects, describing interactions between those objects and creating a class diagram.  These steps make writing code easier because it clearly defines a product, the scope, and the important parts of it.  Instead of diving in ad-hoc and making changes on the fly, these steps ensure that the app functionality and objects are understood.  This will save the developer time since most of the classes will already be defined as well as limiting product scope to the defined areas.

The good

1. gather requirements

     why does the app need to do, what problem is it trying to solve.  define what app should do instead of what it could do

2. describe app

   use case and user stories, not exhaustive but it is the smallest set of user stories that make the app - may be inaccurate or change - mockup or prototype

3. identify the most important objects

     most importnat concepts, pick a lot of interactions that are classes

4. describe interactions between those objects

     how do things interact with each other

     what they do and what order they do it in

5. create a class diagram

     visual rep of the classes you need

     inhereitance, polymorphism

**2.2 Defining requirements 6m 9s**

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

functional requirements - feature capabilities and non functional requirements - laws, compliance, etc performance requirements, support requirements, security

After completing the first phase of defining my requirements, I should have the minimal viable function and nonfunctional description of the application.  This includes the features, capabilities, and tangentially related things like performance requirements, support, and security.  This is not meant to be exhaustive or finalized, but a good starting point for reference that will evolve.

**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?**

UML stands for unified modeling language and is useful for mapping out class hierarchies and it's attributes and behaviors.  This is useful so the overall class structure and methods are defined so we account for all of an app's functionality instead of having to change and rewrite code during the coding process itself.

**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.**

Use case: Creating an event on my phone's calendar

Title: Creating an event on my phone's calendar

Actor: Device owner/user

Scenario:

1. User goes to calendar app on home screen

2. User taps the + icon to create a new event

3. User enters in information such as the title, location, and time

4. User taps the "done" button to confirm the event

**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?**

There are many use cases where the actor is not the user of the mobile device.  One such thing is notifications on the Facebook app.  Someone else on Facebook likes my post, and so I receive a notification about it.  The primary actor is someone else, but I received the action on my device.

**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.**

Use case: Creating an event on my phone's calendar

Title: Creating an event on my phone's calendar

Actor: Device owner/user

Scenario:

1. User enters the calendar app

2. User creates a new event

3. User enters in information such as the title, location, time, and reminder settings

4. User confirms the event information

5. User sees the event on the calendar

Extensions:

1. User wants to cancel the app

2. User wants to edit information

**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?**

All of them in general have an actor (usually represented by a stick figure), actions of an app in a box, and arrows connecting the actor with the actions.

**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 user I want to tap the location icon so that I can see where I am.

2. As a user I want to enter in an address so I can see where it is

3. As a user I want to be able to get directions to an address from my location so that I won't get lost

4. As a user I want to see traffic on my map so that I can save time

5. As a user I want to see alternate routes so that I can save time.

**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.**

* event
  + title
  + location
  + time
  + reminder settings
* user

**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.**

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

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

* event
  + enters in information of title, location, time, and reminder settings
* user
  + confirms

**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.**

**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?**

It's easier to come up with attributes rather than actions.  I define classes based on an object, and that object has known characteristics of what it contains.  What actions it takes with relation to other objects is harder to perceive.

**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?**

The separation is an advantage for programs with many lines and a lot of complexity.  Though dynamic languages like Ruby seem easier to read and use, by defining the methods and attributes in the interface, we clearly abstract what information has to be used in a class.  The actual definition can be kept in the implementation file, but we now clearly know what things will be used in a program.  For a class diagram specifically, we only need the interface file, and so we don't have to actually define the implementation during the class diagram phase.

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

**What are the constructors and destructors in Objective-C? Why do we use them?**

Constructors are methods in a class with the same name as the class that are used to give new instances of the class defined attributes during the initiation process.  Rather than repeatedly creating empty instances and then defining it, we can define it during initialization.  Destructors are important to dealloc memory.  Because the initialization allocates memory and then initializes it, once an object is done being used we should add it back to the heap, and so we use a destructor to do that.

**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.**

1. the half life of a class of a type of element

2. the rate of expiration of a milk class

3. the warranty on an iPhone class

**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.**

Inheritance allows for classes with many similar attributes and behaviors to have a parent class where they pull in those similar things.  Instead of defining the same attributes and behaviors in all of the classes, they just get the same from the single definition of the parent class.  Any individual characteristics can then be defined in the actual class itself.  This saves developers time while making the flow of information much clearer.

**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.**

1. Angry Birds.  The different types of birds all share many characteristics with their physics engine and thus may have shared methods from their super class.

2. Sportstacular.  The different leagues (NFL, MLB, etc) all share different actions from their parent class, team.

3. Clock.  The stop watch, timer, and world clocks all inherent methods from the clock class itself since they are time related actions.