Challenge 2: Time To Dig In

By Steven Lytovka

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?

A: Object oriented design helps to build more complex application with more clarity, agility, accuracy, and maintainability.

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

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

A: In the process of building application first step is to understand what do you need to do and how you are going to execute it – make a plan. Breaking down complex task into smaller ones, analyzing your resources, initial conditions, instruments, expected outcomes and system designing would definitely make programming easier in future.

0.3 Reviewing software development methodologies 4m 8s

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

A: “Waterfall” approach – is a strict linear plan with several distinct steps (step-by-step methodology). With “agile/iterative” approach development will involve several incremental cycles (iterations), each including analysis, design and programming. Since software development must be responsive, programmer must be able to add a new features, support continuing development “agile” approach is more flexible and better in this case. With this approach we don’t have to know everything upfront, developer can create different iterations, and then improve them moving along the process.

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?

A: There are thousands of different programming languages, usually classified by programming paradigm or intended domain of use. There are four main paradigms: imperative, functional, object-oriented and logical. Logical (such Prolog) and functional (such Haskell) program languages are mostly used in academic world for providing some scientific researches, computation linguistics etc. But in practice for creating web or mobile application, web development programmers use object-oriented languages.

1.2 What is an object? 5m 22s

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

A: There are three properties of computing object:

1) Identity – property of an object that distinguishes it from other objects. That’s the name of the object.

2) State – property that describes the data stored in the object. Object can have different states, like apple can be green or red; it can have different weight etc.

3) Behavior – property that describes the methods in the object’s interface by which the object can be used. Like a payment can be received, processed, airplane can fly, phone can ring, send messages… Behavior describes what an object can do.

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: To create an object developer needs a class, which describes what an object will be. To build the house we would need a blueprint, which describes everything how the house will be. With one blueprint construction company can create hundreds of houses, the same developer with one class can create hundreds of objects. Each class has a name and describes attributes and behavior of the object. With blueprint company would probably create all the same houses, with class developer most likely will create many objects that will have different properties and executing different operations.

1.4 What is abstraction? 2m 45s

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

A: Using abstraction developer describes essential qualities of something, ignoring irrelevant and unimportant information. Abstraction reduces details, so programmer can focus on a few concepts at a time.

1.5 What is encapsulation? 3m 45s

What does encapsulation prevent? What does it enable?

A: Encapsulation prevent from other parts of application to access to inner working of the object we want encapsulate. So developer can hide all data and information about particular object except of information that is necessary for other parts of application. Encapsulation allows safely make some changes to object without changing all system, reducing dependency between different parts of application.

1.6 What is inheritance? 3m 35s

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

A: Inheritance is a great form of code reuse. Sometimes instead of writing class from scratch developer can create it based on another class. So new class inherits all attributes and behavior from existing class without writing any code. Then developer can just add some changes to it if necessary. In this case existing class is a Superclass or Parent and a new one is Subclass or Child. If we make a change in Superclass all Subclasses will automatically inherit all these changes, which might be very helpful feature.

1.7 What is polymorphism? 3m 22s

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

A: Polymorphism is the ability to create a variable, a function or on an object that has more than one form. It let’s automatically have the correct behavior of the variable or object in different situations. If program has many subclasses which inherited behavior from superclass, but developer needs different behavior in one of them, polymorphism allows fix this problem, making our classes more flexible.

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?

A: Steps of analysis before writing a code:

1) Gather requirements;

2) Describe the application;

3) Identify the most important objects;

4) Describe the interactions between those objects;

5) Create a class diagram.

Completing those steps will help developer to understand what application need to do, what problems need to be solved, responsibilities of different objects and their behavior. The most important outcome that we will get class diagram, which will help to write classes we need for this project. So, it will be much easier to write the code having all this information, knowing project’s classes and their behavior.

2.2 Defining requirements 6m 9s

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

A: Developer will have:

1) Functional requirements (features and capabilities of the system);

2) Non-functional requirements (documentation that need to be provided, legal requirements, performance, support and security requirements)

So, developer will gather information what future application MUST do.

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?

A: Unified Modeling Language is a graphical notation specifically for drawing diagrams of object-oriented systems; it’s a simple representation of a class, including its name, attributes and behavior. Using UML allows developer quickly to draw an idea that is readable and understandable despite language used for programming.

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.

A*: Title:* Creating an event

*Actor*: User

*Scenario*: User chooses the date in calendar. User inputs title, location of Event, starts and ends time, attendees. User confirms information. Event created.

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: When application interacts with other members, organizations, visitors the actor can be different than just user. For example in app for big corporation HR can be an actor (sending schedule), Security can be an actor (sending alerts), CEO can be an actor (approving some operation, giving access).

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.

A: *Title:* Creating an event

*Actor*: User

*Scenario*: User chooses the date in calendar. User inputs title, location of Event, starts and ends time, attendees, and alert. User confirms information. Event created.

Extension: If user inputs time before current time, send alert “Check the time!”

Extension: If user doesn’t input title of event, alert “Please, input title of your Event!” …

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?

A: All use case diagrams have Actors (represented as a stick figures), use cases, usually combined in a box representing boundary of system and also links between actors and use cases.

3.5 Employing user stories 3m 43s

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

A: - As a user

I want to know how many miles from my current location to destination

So that I can calculate how many stops I will need;

- As a user

I want to see whole route step-by-step

So that I can arrange my next stop

- As a user

I want to switch to bike mode

So I can bike to my destination

- As a user I want to have alert 1 mile before turning points

So I can change the line

- As a user

I want to know if destination on my left or right

So I can find parking place

Section 4 - Domain Modeling (Modeling the App)

4.1 Creating a conceptual model 1m 59s

Just let it soak in. No questions here.

Thank you! ☺

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.

A: I identified next classes:

Calendar

User

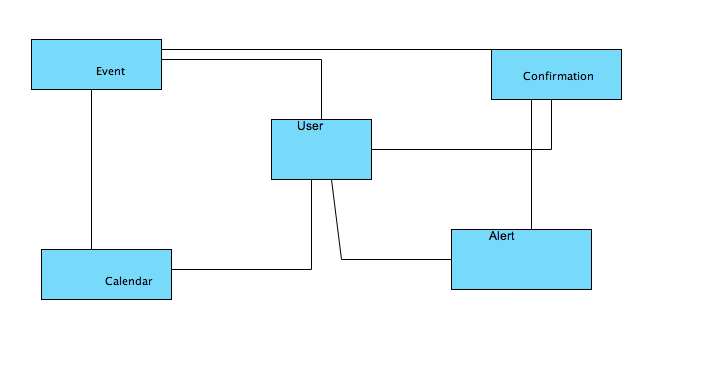
Confirmation

Alert

Event

4.3 Identifying class relationships 2m 38s

A: 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.

A: User – creates and manages Event

Calendar – display events, dates

Confirmation – check if event is correct, stores it in Calendar

Event – process event, holds data

Alert – sending notifications, remind user about Event

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?

A: I displayed two Class Diagrams of calendar app: Event and Calendar

|  |
| --- |
| Calendar |
| id: String  eventList: struct  eventMonth: String  eventDay: Integer  eventYear: Integer |
| getEventMonth()  getEventDay()  getEventYear()  cancelEvent()  updateEvent()  markEvent()  … |

|  |
| --- |
| Event |
| name: String  isComplete: Boolean  startDate: Date  endDate: Date  numberInvitees: Integer  eventLocation: String |
| getName()  checkComplete(Boolean)  displayEvent()  getTime()  getInvitees()  getEventDetails()  … |

It’s much easier to come up with attributes than with behaviors, because attributes are basically what we have, we just need to name it, but behaviors display list of operations for achieving the goal, so we need to know exact responsibilities of each class and how classes interact between each other.

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?

A: Separation allows programmer to change implementation independently of the interface, this helps with changing requirements. Multiple classes can implement the same interface, and classes can implement more than one interface, allowing developer to mix different parts of a program instead of completely inheriting classes. Separation provides achieving flexible, extensible and portable software.

5.3 Exploring object lifetime 5m 55s

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

A: Constructor is special method that exists to construct the object; it's called when object is created. In objective-C developer creates constructor by creating an "init" method in the class. We use constructor to set initial state of the object with specific variables we need.

Destructor is a method that is called when an object is no longer needed and is being disposed off. This method is automatically called when the object is destroyed. Typically destructor is used when there is an object that is holding some recourse and it's necessary to make sure that the object is released any connection that it has before it's destroyed.

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.

A: My examples of data that would be the same for all instances of a class:

1) Number of engines for calculation thrust for different types of aircrafts;

2) USD/EUR rate exchange for SWIFT transfer operations;

3) Tax rate for a particular tax bracket for TurboTax application.

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.

A: Inheritance is a key feature in object-oriented languages. It allows creating classes, which are derived from other classes, so that they automatically include some attributes and behavior, plus its own features. While identifying our classes, developer might notice that some classes share the same attributes and behavior then we can create superclass using inheritance. The simple way to define inheritance using "is a" or "is a type of" words. For example, meeting is an event, date is an event, Birthday is an event etc. Inheritance is very useful for code reuse, flexibility and controllability of the program. We can make changes in Superclass and those changes will be inherited by subclasses.

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.

A: My examples:

1) In "US Open" app I have subclasses Men's Draw, Women's Draw, Double's Draw inherited from Superclass Draw;

2) In BankOfAmerica app I have subclasses "Money Market Account", "Checking Account", "Saving Account" inherited from superclass "Bank Account";

3) In VK (Facebook prototype) I have subclasses "Music", "Videos", "Groups", which are inherited from superclass "Profile's Features"