Intro To Object OOP

December 27, 2020

0.1 Overview

- object oriented programming is a process on how to discover and implement a conceptual machine to realize that desired output

0.1.1 Goals Of OOP

- Modularity
  - goal is to increase code modularity
  - so a change in one module shouldn't break things in other modules
  - improves code maintenance -adding or removing features does nto break thing, and you can change faster and more reliably
  - enables code re-use: meaning you can package bits of code
  - enable team collaboration: different people can work on different parts
  - enable easier integration: easily abstract and isolate large systems
- Improve Process of Analysis, Design and Communication
  - this has to do with improving the process of breaking down a problem domain and designing a candidate solution and communicating it to others
- a software machinery is simmilar to physical machinery. A software application is constructed from parts. These parts – software objects – interact by sending messages to request information or action from others. Through its lifetime, each object remains resposible for responding to a fixed set of requests.
- To fulfull these requests, objects encapsulate scripted responses and the information that they base them on. If an object is designed to emember certain facts, it can use them to respond differently to future requests
- building an object-oriented application means inventing appropriate machinery. We represent real-world information, processes, interactions, relationships, even errors, by inventing objecects that dont exist in the real world
- we give life and intelligence to inanimate things. We take difficult-to-comprehent real-world objects and split them into simpler, more managable software ones. We invent new objects
- Each has a specific role to play in the application. Our measure of success lies in how clearly we invent a software reality that satisfies our applications requirements –and not in how closely it resembles the real world
- Beacuse they have machine-like behaviors and because they can be plugged togheter to work in concert, objects used to build very complex machines
- to manage this compelxity, we divvy the systems behaviors into objects that play well-defined roles. If we keep out focus on the behavior, we can design the application using several complementary perspectives

How to invent new software machines

- we invent new objects. Each object has a specific role in the application. Out measure of success lies in how clearly we invent a software reality that satisfies our application's requirements –and not in how closely it resembles the realy world

0.2 Application lifecycles

- Big design upfront is a bad stragety
- design only takes into account the "happy path"
- 80% of code is desgined to take care of 20% of the edge cases
- edge case may take entire redesgin of application
- took forver to build the application and business changes by the time software developers get it
- keeping design documentation updated is too hard and abandoned

0.3 Agile Methodology

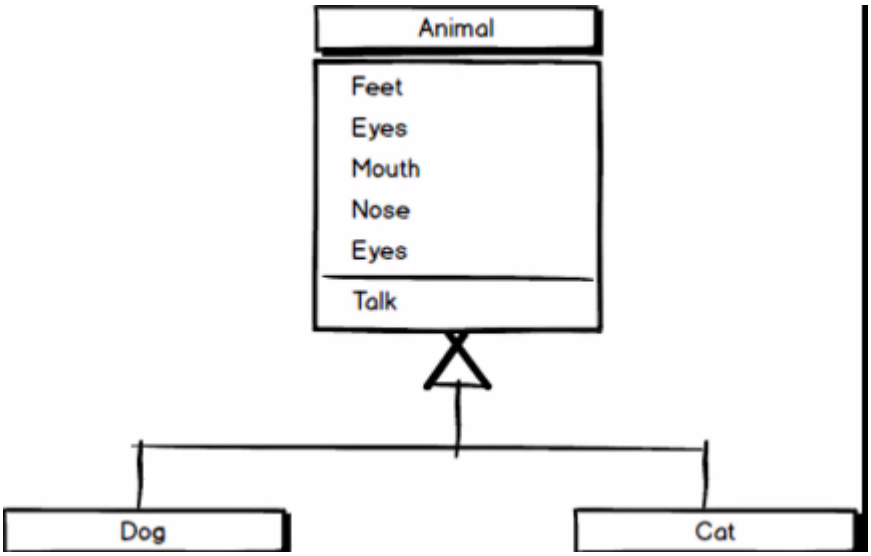
- oppsite of big desgin upfront up
- development team was invloved in the design
- the business domain experts were in same office as developers
- the PM would outline "sprints" which are boxes of time in which work is performed
- you see things like continous builds or continous deployments
- at the end of the day, you would have to come with working peice of software that can excuete
- unit testing made it possible do multiple changes without major errors

0.4 How the process can affect the design

- in agile approach, the code is the desgin
- and the desgin is discovered with coding
- agile methodology is incooperated with OOP
- OOP is about creating flexible, interchangable objects that decrease the complexity and implementation and decrease the resistance of the application to change, eases the maintance of the system, makes it more resileance to change long term

0.5 Abstracting The Real World Into A Domain Model

- OOP models the real world
- think about a model of an Amimal



- you dont model the real world directly, you create an abstraction of the real world that satisfies the requirements of the application
- OOP is about abstracting the real world into a bunch of subsets or supersets of objects that we need to consider for a specific problem domain and how to implement it
- with OOP we are building conceptional machines

APPLICATION DESIGN

Using several different complimentary perspectives

PERSPECTIVE	
APPLICATION	A set of interacting objects
OBJECT	An implementation of one or more roles
ROLE	A set of related responsibilities
RESPONSIBILITY	An obligation to perform a task or know information
COLLABORATION	An interaction of objects or roles (or both)
CONTRACT	An agreement outlining the terms of a collaboration

0.6 Establishing Roles and Delegating Responsibilities to Objects

0.6.1 Role

- each object plays a part of a larger conceptual machine
- we delegate one or more responsibilites to an object
- that object has to take care of that task or hold that information
- think about roles as different machines such as a giant robot ARM that moves things
- if one machine fails, the assambly line fails
- maintaining a machine that does alot of things might not do each task well

- we should be able to replace that object if need be

Contracts

- what makes OOP intresting is the collobration aspect
- objects need to collobrate, thing of a cell, it needs to collobrate with others cells
- think of a method signatures
- you need to give me these and I return this to you
- if i cant figure this out, then i give an error

- these is also a concept of interface, these are things that a class must implement
- an interface is like receptors on cells

0.7 Collaborations and Object Role Sterotypes

OBJECT ROLE STEREOTYPES

Oversimplifying and characterizing to understand an object's role more easily

PERSPECTIVE	
INFORMATION HOLDER	Knows and provides information
STRUCTURER	Maintains relationships between objects and information about those relationships
SERVICE PROVIDER	Performs work and, in general, offers computing services
COORDINATOR	Reacts to events by delegating tasks to others
CONTROLLER	Makes decisions and closely directs others' actions
INTERFACER	Transforms information and requests between distinct parts of our system.

- objects are partners, not isolated
- steriotypes are common roles
- one object may have one or more of these roles, but too many roles and it may lose its effiecny
- if an object has too little role, then it may be anemic, for instance, if it is just an information holder

A well defined Object supports a clearly defined role. Design is an interative and incremental process of envisioning objects and their responsibilities and inventing flexible collaborations

0.8 Collaborations: Conditions of Use and After Effect Guarantees

- fine print can be handled through exceptions
- if you get wrong data types then you return error
- in typed language, you verify the data type
- you can check if data is null or not
- you can check if data is between boundaries
- you can have unit tests

- each object should garuntee an output

- it should return the correct type
- how much should the object trust its collobarator
- best way to verify the return is to use unit tests

0.9 Domain vs. Application Specific Objects

- there are domain specific objects and there are application specific objects
- domain specific objects are objects that model the real world into obstractions that we will use in our conceptional machinary to create a solution
- application specific objects are stuff that performs system specific tasks
- use them for persistance, display, communication, etc

0.9.1 Domain Specific Objects

- Domain Specific Obejcts are born from an analysis of our problem domain
- Domain is made up of information and services user needs along with structures that can relate the two
- there are containers for domain/business logic
- imagine the problem is tax collection
- we might need to collect tax depending on state, country, object, etc.
- a tax expert can understand this domain
- another domain could be books
- which has author, date published, etc.

0.9.2 Application Specific Objects

- born purely from imagination
- they are bridges between the domain world and software world
- they are objects
- it displays things, it interacts with the database, it interacts with file systems

0.10 Components As Neighborhoods Of Objects

- a component is a reuseable package of objects
- components should have either domain specific or application specific objects
- interfaces should allow other services to inject dependencies into the application
- birds of a feather should stick together
- classes that do simmilar things should stick together

0.11 Architectural Layers of Responsibility using Objects

- a layer is composed of several collobrating components that are tasked with the same type of work but more from a system perspective
- you have objects related to business functions (accounting, inventroy, etc)
- those layers fit into domain layer
- application layer is the presistance layer, presentation layer, web server layer, etc

- inner faces exits between all of these squares, such as between accounting presentation or accounting domain
- it can also be presentation accounting and presentation marketing
-

ARCHITECTURAL LAYERS OF RESPONSIBILITY

Matrix of Application-Specific and Domain-Specific Objects by Architectural Layer

	Accounting	Marketing	Retail	Tax
Presentation	Application Objects	Application Objects	Application Objects	Application Objects
Domain	Domain Objects	Domain Objects	Domain Objects	Domain Objects
Persistence	Application Objects	Application Objects	Application Objects	Application Objects

- information ususally flows from persistence to presentation while clien objects are usually above domain/web server objects and flow downword
- call backs are needed (think asynchronous requests) to tell the layers above them that they are working on it and will get back to them later

0.12 Why Encapsulation

- a mechnism to hide implementation details of a class
- a class is made up of information and functions (methods)
- the methods operate on the information/fields
- properties have getters/setters and we use these as gatekeepers because they safeguard the data
- encapsulation is for security concerns
- object shares as little as possible

0.13 Why Implementation Inheritance?

- there is both implementation inhertiance and interface inhertiance
- there is the parent class and then their is the child class
- the child class gets its inhertance from its parent
- good for code reuse and allows you to override
- this also gives you flexibility for better collobaration
- makes our collobratation flexiable because we are able to generalize our collobrations between objects allowing for specialized child versions for child objects in liu of parent objects
- generalization and spelization of collobarating objects

0.14 Why Interface Inheritance?

- when you have multiple objects with same shape, they can be using interchangablly
- interfaces only define public memebrs of a class that must be implemented by a class inorder to satisify a contract between two or more objects
- no implementation where you can rely on parent
- this is because the purpose of interface is to define an abstraction that breaks a dependency
- interface is a contract for services that I need rendered
- i need something done i put a contract that other can satisfy
- the other object can accept contract and
- with interface inheritance we remove tight coupling that our object has to other objects
- since object is not coupled, any obeject can implement interface inheritance we can test any object more throughly
- inheritance is a tenent because it allows up to decouple where you do not have to make a change in more than one or more place

0.15 Why Polymorphism

- means many forms or shapes
- allows you to call same method on different objects as long as it really is the same objects
- methods can be owerwritten

0.16 Understanding Relationships: Coupling and Dependency

- when one object is creating an instance to another object inorder to collobrate with it
- that is good only if the two objects live in the same neighborhood
- if they are not in the same layer, it can be a problem
- cross domains is also bad
- dont go from account object to the tax object
- an example of this is when our domain layer interacts with out perisitance layer
- if tax domain creates a class using the new keyword, then saves it to DB, it is bad
- it makes it hard to unit test, etc.
- high coupling makes out appliation brittle
- S in solid means solid and that means seperation of services

0.17 Understanding Relationships: Inversion Of Control and Dependency Injections

- to decouple, you simply flip dependencies
- thus you need to get an outsider to coordinate the dependency and inject that dependecy inside of it

0.18 Understanding Relationships: Aggregation

- “has a” style
- aggeration is when one object keeps refrences to other instances of objects
- this object is a part of a larger object

- you can have teachers in different departments
- some teaches can exist in multiple departments
- thus if you eliminate a teacher, it does not eliminate the department

- there we are talking about the lifecycle (when they life/die) and ownership
- objects should live and die seprately

- aggeration means there is ownership but there is no life/death situation
- one can surive even if another is destroyed

0.19 Understanding Relationships: Composition

- stronger version of aggergation
- in composition the child cannot exist without parent
- think about a house with room
- if you destroy the house, you destory the rooms

- you can have a class inside of a class and initilaize a class inside of another class
- Composition over inheritance
- better something has a method then to extend it
- the child does not have to worry about parents or grandparents funky behavior
- use inhertance sparingly
- in domain context, less inhertance, the better