W04 Team Activity: Foundation Programs Design

Date: 2025-09-26

Student: Olakunle Obademi

Course: CSE 210 — Programming with Classes (BYU‑I)

**Overview**

This document contains the design for two foundation programs:  
1) Abstraction with YouTube Videos  
2) Encapsulation with Online Ordering  
  
Each design includes: responsibilities, candidate classes, attributes, methods, and a simple run flow.

**Foundation Program #1 — Abstraction with YouTube Videos**

What does the program do?

Models a tiny YouTube-like experience where a user searches videos, builds a playlist, and plays it—without knowing how search/storage/streaming are implemented. The complexity is abstracted behind simple methods.

Candidate classes & responsibilities:

**• Video —** holds metadata about a single video.

**• Playlist —** manages an ordered list of videos (add, remove, total duration, play order).

**• User —** owns playlists; can create/select a playlist.

**• YouTubeService —** abstracts backend search/fetch; hides data source and algorithm details.

**• Player —** abstracts playback controls; hides buffering/decoding.

Class diagrams (text):

Class: Video  
Attributes:  
• \_id : string  
• \_title : string  
• \_channel : string  
• \_durationSeconds : int  
• \_tags : List<string>  
• \_views : long  
Methods:  
• GetDisplayTitle() : string

Class: Playlist  
Attributes:  
• \_name : string  
• \_videos : List<Video>  
Methods:  
• Add(Video v) : void  
• Remove(string videoId) : bool  
• GetTotalDuration() : int  
• ListVideos() : IEnumerable<Video>

Class: User  
Attributes:  
• \_name : string  
• \_playlists : List<Playlist>  
Methods:  
• CreatePlaylist(string name) : Playlist  
• GetPlaylist(string name) : Playlist?

Class: YouTubeService (abstraction boundary)  
Attributes:  
• \_catalog : List<Video>  
Methods:  
• Search(string query) : List<Video>  
• GetById(string id) : Video?

Class: Player (abstraction boundary)  
Attributes:  
• \_current : Video?  
• \_queue : Queue<Video>  
Methods:  
• Load(IEnumerable<Video> items) : void  
• Play() : void  
• Pause() : void  
• Next() : void  
• IsPlaying() : bool

Run flow: 1) Search → 2) Build playlist → 3) Load player → 4) Play.

Abstraction highlight: UI talks only to YouTubeService/Player; storage and playback are hidden.

**Foundation Program #2 — Encapsulation with Online Ordering**

What does the program do?

Implements a minimal online ordering flow (cart, discount, tax, checkout) while protecting internal state through encapsulation (private members + controlled methods).

Candidate classes & responsibilities:

• Product — immutable product info (id, name, unit price, tax rate).

• CartItem — product + quantity; computes line totals.

• ShoppingCart — holds items; exposes subtotal/discount/tax via methods; hides internal list.

• DiscountPolicy — validates & computes discount from codes (encapsulates rules).

• Order — snapshot of cart at checkout; immutable post‑creation.

• Inventory — tracks stock; validates & reserves/commits.

• PaymentProcessor — abstracts payment authorization/capture.

• CheckoutService — orchestrates validate → reserve → pay → commit.

Key class sketches (text):

Class: Product  
Attributes:  
• \_id : string  
• \_name : string  
• \_unitPrice : decimal  
• \_taxRate : decimal  
Methods:  
• GetPrice() : decimal  
• GetTaxRate() : decimal

Class: CartItem  
Attributes:  
• \_product : Product  
• \_quantity : int  
Methods:  
• SetQuantity(int qty) : void  
• GetLineSubtotal() : decimal  
• GetLineTax() : decimal  
• GetLineTotal() : decimal

Class: ShoppingCart  
Attributes:  
• \_items : Dictionary<string, CartItem>  
• \_discountPolicy : DiscountPolicy  
Methods:  
• Add(Product p, int qty=1) : void  
• Update(string productId, int qty) : void  
• Remove(string productId) : void  
• Clear() : void  
• GetItems() : IReadOnlyCollection<CartItem>  
• GetSubtotal() : decimal  
• GetDiscount(string code) : decimal  
• GetTaxTotal(string code) : decimal  
• GetGrandTotal(string code) : decimal

Run flow: Add → PlaceOrder → Inventory reserve → compute totals → payment → commit → Order.

Encapsulation highlight: Private members prevent invalid external mutation; policies & inventory hide internal rules.

**Appendix — Quiz Answers (for your notes)**

• 1) Simplify by removing implementation details.

• 2) Classes.

• 3) Class = template; Instance = object from it.

• 4) True.

• 5) Member function of a class.

• 6) Hiding implementation/protecting data.

• 7) Other parts of the program.

• 8) True.

• 9) Getters/setters can expose representation.

• 10) Default no‑arg constructor is generated in C#.