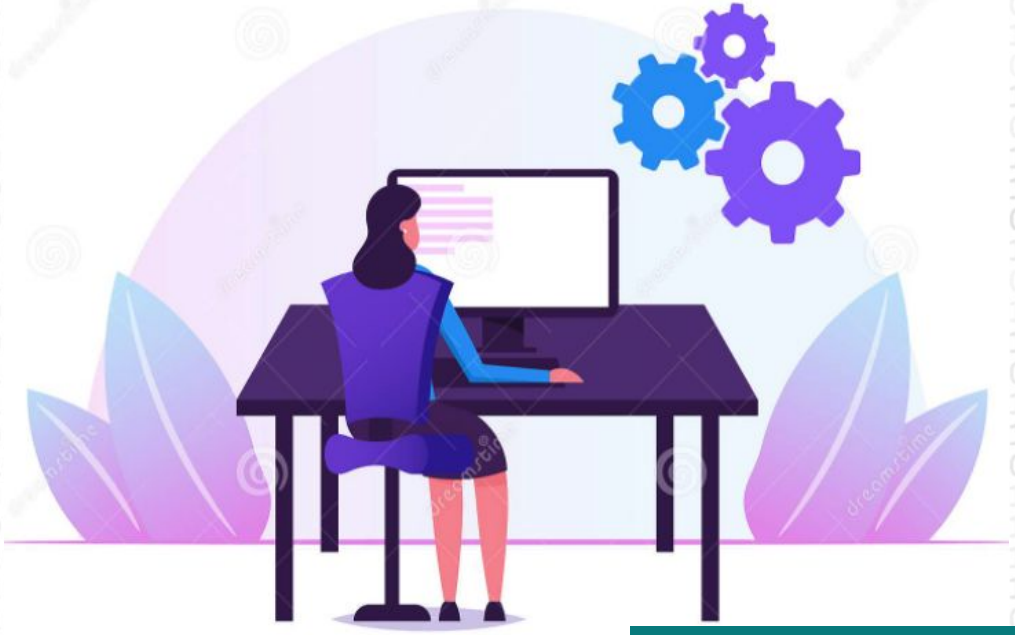
Welcome!

- We'll start in a moment :)
- We are NOT recording tonight's event. We may plan to take screenshots for social media.
    - ***If you want to remain anonymous***, change your name & keep video off.
- We'll introduce the hosts and break in-between for Q/A.
- We will make some time for Q&A at the end of the presentation as well.
- You can come prepared with questions. And, feel free to take notes.
- Online event best practices:
    - Don't multitask. Distractions reduce your ability to remember concepts.
    - Mute yourself when you aren't talking.
    - We want the session to be interactive.
    - Feel free to unmute and ask questions in the middle of the presentation.
    - Turn on your video if you feel comfortable.
    - Disclaimer: Speaker doesn't knows everything!

Check out:

- Technical Tracks and Digital Events
- Get updates – join the Digital mailing list
- Give us your feedback – take the Survey

WOMEN WHO CODE



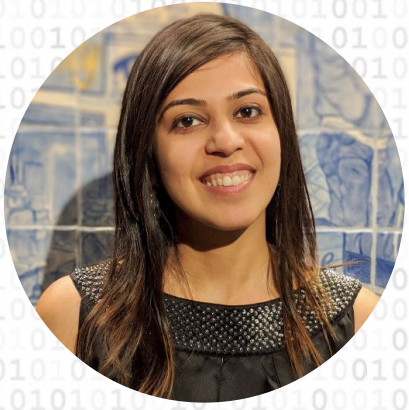
WWCode Digital + **Backend**
Backend Study Group

July 1, 2021

WOMEN WHO CODE

Introduction & Agenda

- Welcome from WWCode!
- Our mission: Inspiring women to excel in technology careers.
- Our vision: A world where women are representative as technical executives, founders, VCs, board members and software engineers.



Prachi Shah
Senior Software Engineer @ Metromile

- What is Backend Engineering?
- Software Design
- Design Patterns
- **Software Design Patterns [Part 5 of 5]**
  - Creational Design Patterns [4/22]
  - Structural Design Patterns [5/20]
  - Behavioral Design Patterns [6/3]
  - Anti-patterns [6/17]
  - **Interview Questions and Q/A [7/1]**

WOMEN WHO CODE

Backend Engineering

- What is Backend Engineering?
- Design, build and maintain server-side web applications.
- Concepts: Client-server architecture, API, micro-service, database engineering, distributed systems, storage, performance, deployment, availability, monitoring, etc.

Software Design

- Defining the architecture, modules, interfaces and data.
- Solve a problem or build a product.
- Define the input, output, business rules, data schema.
- Design patterns solve common problems.
- 3 Types:
  - UI design: Data visualization and presentation.
  - Data design: Data representation and storage.
  - Process design: Validation, manipulation and storage of data.

WOMEN WHO CODE

Backend Engineering

Previously, we discussed...

- Prerequisites for design patterns:
  Basics of programming & OOP
- Necessity of design patterns:
  - Template solutions/ shared vocabulary.
  - Build code on-top of a pattern solution.
  - Maintainability: Easy to maintain code.
  - Reusability: Easy to reuse code for new features.
  - Scaling: Large-scale reuse of architectures.
- Types of design patterns: Creational, Structural, Behavioral
- Solving problems using design patterns and code demos, and real-life applications.
  Examples: Order generation [Bridge/Structural],
  create different shapes [Factory/Creational],
  iterate over a collection [Iterator/Behavioral], etc.
- Anti-patterns

You can unmute and talk or use the chat.

WOMEN WHO CODE

Backend Engineering

Design Patterns

- Set of template solutions that can be reused.
- Improved code maintainability, reusability and scaling.
- Leverages Object-oriented programming (OOP) principles for flexible & maintainable designs.
- Shared pattern vocabulary. Relationship between objects, loosely coupling, security.
- Not a library or framework, but recommendations for code structuring and problem solving.
- Adapt a pattern and improve upon it to fit application needs.

Types of Design Patterns

- **Creational**:
  - Initialize a class and instantiate the objects.
  - Decoupled from implementing system.
  - Singleton, Factory, Builder, Abstract Factory, Prototype.
- **Structural**:
  - Class structure and composition.
  - Increase code reusability and functionality.
  - Create large objects relationships.
  - Adapter, Facade, Decorator, Bridge, Composite, Flyweight, Proxy.
- **Behavioral**:
  - Relationship and communication between different classes.
  - Observer, Strategy, Iterator, etc.

		Purpose		
		Creational	Structural	Behavioral
Scope	Class	Factory Method	Adapter	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Facade Proxy	Chain of Responsibility Command Iterator Mediator Memento Flyweight Observer State Strategy Visitor

WOMEN WHO CODE

Backend Engineering

Object Oriented Programming Principles: Create objects with data (attributes) & functions (methods)

Abstraction:	**Encapsulation**:	**Inheritance**:	**Polymorphism**:
• Hide complexity and details • Caller does not need to know the details • Each class has its own abstraction • Easy to maintain code and add features	• Bundle data and methods into one unit (class) • Entity: data and operations match real-world scenario • Hide data from users • Declare attributes as **private** • **get()** and **set()** methods to access data • Better control over data access and methods • Improved security of data • Data can be read-only or write-only	• Derive a class from another class • Classes share attributes & methods • Hierarchy of super class, sub class • `extends` keyword • Reusability	• Use same interface for different classes • `is-a` relationship • `implements` keyword • Interface has public methods without implementation • Implementing class overrides all of these methods • Implementing class provides own function/ logic

WOMEN WHO CODE

Backend Engineering

Creational Design Patterns:

Singleton:	Factory:	Builder:
• One instance only • Instance variable is static • private constructor • Caller gets the instance from a getInstance() • Lazy initialization: Instance is created and initialized on-demand • Eager initialization: Instance is created and initialized on class load • One instance per singleton per Java Virtual Machine (JVM) • Example: Company has one CEO; University has one Proctor; Log4j logging program	• Create an object by hiding the creation logic • Use a common interface to create objects • Create a new instance on-demand and initializes fields • Reduces code duplication, provides consistent behavior • Easy to maintain classes as creation is centralized • Loosely coupled classes • Example: Uber users	• Build custom objects of a class • Objects can be different • Use the same creation logic • Seperate the construction and representation • Flexible design, readable code, complete objects • Example: Ordering food from DoorDash

WOMEN WHO CODE

Backend Engineering

Structural Design Patterns:

Adapter:
- Wrapper pattern.
- Incompatible objects can interact.
- Object adapts to interface of another object.
- Reusability of functionality.
- Seperate the interface from business logic.
- New adapters can be introduced for different client integrations.
- Adapter: Object that connects two different interfaces.Wraps an object to hide the implementation complexity. Object can use the interface, to call adapter methods.
- Example: Connect your phone to Alexa, Fitbit, Apple Watch

Bridge:
- Separate abstraction from implementation.
- Independent development, loosely-coupled, hierarchical and hide details. Client accesses abstraction, agnostic of implementation.
- Abstraction: Interface declare operations and delegates.References the implementation. *abstract* class and concrete class.
- Implementor: Operations are implemented.*interface* and concrete implementor class that implements the interface.
- Example: Lyft app has *driver* login and *rider* login.

Decorator:
- Modify an object's behavior at runtime without modifying the structure.
- Does not affect other object instances.
- Removes need for subclassing, therefore more flexible than inheritance.
- Extendible and easy to maintain code.
- Decorator: Class that encapsulates concrete class to provide modified functionality. Wrapper linked to a target class. Implements the same *interface* as the target class.
- Example: Java IO classes like FileReader.

WOMEN WHO CODE

Backend Engineering

Behavioral Design Patterns:

Chain of Responsibility:

- Sender object sends request to a chain of receiving objects to eventually reach the receiver object. This avoids coupling between sender object and receiver.
- Once an object independently handles the request, it is sent to the next object in the chain.
- *Handler*: Interface that receives a request and sends it to the next handler object.
- Example: Shipment delivery of packages.

Iterator:

- Traverse a collection of objects in a specific manner. AKA *cursor.*
- Access elements without revealing the implementation.
- Iterator: Interface with methods to iterate over a collection (of any type). Different simultaneous iterations: one-way and bi-directional.
- Example: Directory of names: Search alphabetically, search from start or from end.

Observer:

- Define 1-1 dependency between objects.
- On change of state in one object, dependant objects are notified and updated.
- AKA broadcast communication or subscribe-publish.
- Observable: Objects state change is of interest.
- Observer: Registered objects that are notified on Observable' state change.
- Example: Marketing & new products notifications. Kafka Pub/Sub.

Strategy:

- Select one out of different strategies/ algorithms/ implementations at runtime.
- Add strategies in separate classes that the client references w.r.t. the context. Strategy: *Interface* with methods to implement the strategy (Example: Sorting). Run various *Strategy* implementations (Example: Merge, Quick, etc.).
- Example: Sort algorithms a collection of objects (List, Set, etc.).

WOMEN WHO CODE

Backend Engineering

Anti-Patterns: Process or action that doesn't solve a problem and has bad consequences.

Big Ball of Mud:	**God Object**:	**Boat Anchor**:	**Hard Coding**:
• Application lacks architecture and isn't cohesive. • Code is old/obsolete, not suitable for optimization, highly buggy, etc. • AKA Spaghetti code (unstructured code) or technical debt (need to rewrite the code). • Examples: Small set of services dependent on each other. Over time, more dependencies, more path flows, and tight coupling.	• An entity/object has many functions that complicate implementation. • Inefficient bifurcation of a large problem into smaller problems. • Tight coupling with an object for all functionalities and data. • Object exclusively stores state management. • Single point of failure. • Example: For cars, flights, hotels.	• Throw-away or obsolete code is retained. • Difficulty differentiating between working and obsolete code. • Either delete the code or mark it as deprecated, or move/isolate the code. • Examples: Poor/no documentation, convoluted implementation, C# *Obsolete* attribute, Python *@deprecated* decorator. • A metaphor to throwing an anchor in the water.	• Embedding data into the program instead of fetching at runtime. • Any change in values requires source code changes, recompilation and retesting. • End-user or downstream system needs to be made aware of the changes. • Backdoor: Security concern if hard-coded credentials. • Magic number/string: If hard-coded value is repeated then it is hard to update instances.

Backend Engineering

Common Interview Questions:

- What is a design pattern? Why use patterns? What are the benefits?
- What are the types of design pattern? Can you give examples?
- Can you solve a problem using a design pattern? If yes, what pattern(s) will you use and why?
- What are anti-patterns? Can you give examples?

Coding Problems:

- Design DoorDash order management? What design pattern(s) will you use and why?
- Design Uber user management? What design pattern(s) will you use and why?
- Design a TrackMyHealth app? What design pattern(s) will you use and why?
- Given a codebase, can you identify anti-patterns?

You can unmute and talk or use the chat.

WOMEN WHO CODE

Backend Study Group

- WWCode Presentation and Demo
- WWCode YouTube channel:
  - March 25, 2021 session recording: Backend Engineering
  - April 8, 2021 session recording: Java Microservice and REST API Demo
  - April 22, 2021 session recording: Creational Design Patterns
  - May 20, 2021 session recording: Structural Design Patterns
  - June 3, 2021 session recording: Behavioral Design Patterns
  - June 17, 2021 Anti-Patterns [No recording]
- **Resources**:
  - Software design pattern
  - Design Patterns in Java
  - Design Patterns in Python and Ruby
  - Head First Design Patterns book
  - Anti-pattern

NEXT SESSION on 7-15-2021: Data Engineering & Data Science. Come prepared with questions!

You can unmute and talk or use the chat.

WOMEN WHO CODE