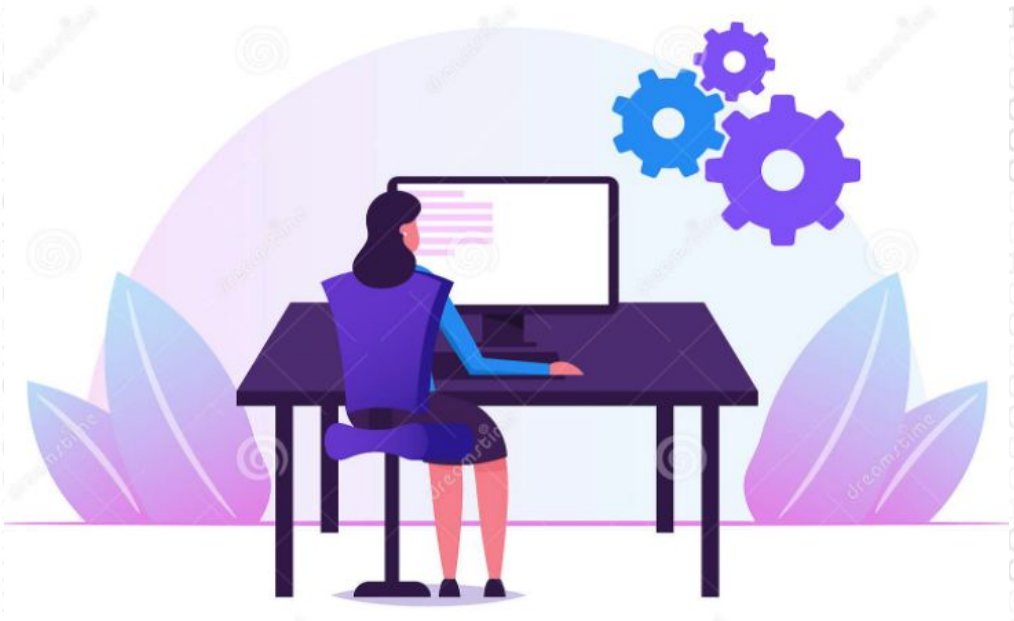
Welcome!



- 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 & might 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.
- 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 & ask questions.
- Turn on your video if you feel comfortable.

- *Disclaimer: Speaker doesn't knows everything!*

WWCode Digital + **Backend**

Backend Study Group

August 31, 2022

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.

- OOP & Design Principles (using Java).
- What is OOP (Object Oriented Programming)?
- Design Principles.
- Q & A and open discussion.

Prachi Shah
Instructor,
Senior Software Engineer.
Director, WWCode SF

Harini Rajendran
Co-host,
Software Engineer, Confluent.
Lead, WWCode SF

Disclaimer:
- Sessions can be heavy!
- Lots of acronyms.
- Speaker doesn't know everything.

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

Object-Oriented Programming Concepts:

- **Inheritance and *super()***: subclass inherits methods and attributes of superclass.
  super() inside subclass constructor to pass attributes.
- **Polymorphism**: same method but different behavior.
  - Overloading: Same name method but different parameters.
  - Overriding: Method signature (name & parameters) same in subclass and superclass.
- **Abstract class**:
  - Has *abstract* and concrete methods. Declare vs. Define variable and method.
  - Objects that extend (one) class have same/default behavior & can be overridden.
- **Interface**:
  - Has abstract methods only. Uses *implements* keyword.
  - Polymorphic behavior for objects that implement Interface(s).

WOMEN WHO CODE

Backend Engineering

Object-Oriented Programming Concepts:

- **Composition**:
  - Subclass cannot exists without superclass (conceptually). Strong association.
  - *Tree* superclass has *branch*, *leaves*, *fruit* subclasses.
- **Aggregation**:
  - Subclass can exists without superclass (conceptually). Weak association.
  - *Vehicle* superclass has *driver* subclass.
- ***is-a* and *has-a***:
  - is-a: Inheritance where subclass is-a superclass. *Mango* is a *Fruit.*
  - has-a: Composition where an object has-a another object. *Bookshelf* has *Book.*
- **Unified modeling language (UML) diagram**: Visualize design of a software.

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

Creational Design Patterns:

Singleton: • 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	**Factory:** • 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	**Builder:** • 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: • 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.	**God Object**: • 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.	**Boat Anchor**: • 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.	**Hard Coding**: • 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

Backend Study Group:

- Presentations on GitHub and session recordings are found on WWCode YouTube channel
- Upcoming session:
  - September 29th, 2022 about API Design
  - October 27th, 2022 about Database Design
  - November 3rd, 2022 about Improve your code debugging skills
  - December 8th, 2022 about Git and Version Control System

- Technical Tracks and Digital Events.
- Get updates – join the Digital mailing list!
- Have questions?
  - Contacts us at: contact@womenwhocode.com
  - Join our Slack workspace and join *#backend-study-group*!

You can unmute and talk or use the chat.

WOMEN WHO CODE