CS489:
Applied Software Development

Lesson 3:
Software Development Platforms

Wholeness

- This lesson explores the range of platforms, languages, technologies, frameworks that are available for Application Software development
- Based on the requirements and constraints, the software developer chooses a suitable language, platform and tools for the application.
- Science of Consciousness: Harmony exists in diversity.

Software Platforms

- Operating systems
  - Windows,
  - MacOS/iOS,
  - Linux,
  - UNIX
  - Android etc.

Software Platforms

- Microsoft .NET Platform
  - .NET languages: C#, Managed C++, F#, VB.NET,
  - Desktop client technologies: MAUI, WPF, etc.,
  - Web App technologies: ASP.NET, Blazor etc.

Software Platforms

- JavaScript/TypeScript:
  - node.js,
  - express.js,
  - react.js,
  - angular,
  - vue,
  - next.js etc.

Software Platforms

- Other languages/platforms:
  - Go lang,
  - Rust lang,
  - Python,
  - Kotlin,
  - Scala,
  - Dart etc.

Software Platforms

- Java Platform:
  - Java SE,
  - Jakarta EE (formerly, Java EE):
    - Specifications: Servlet/JSP/EL, JPA, JTA, EJB, JAX-WS, JAX-RS etc.,
    - Glassfish Jakarta EE Application server
  - Java ME and JavaCard

Software Platforms

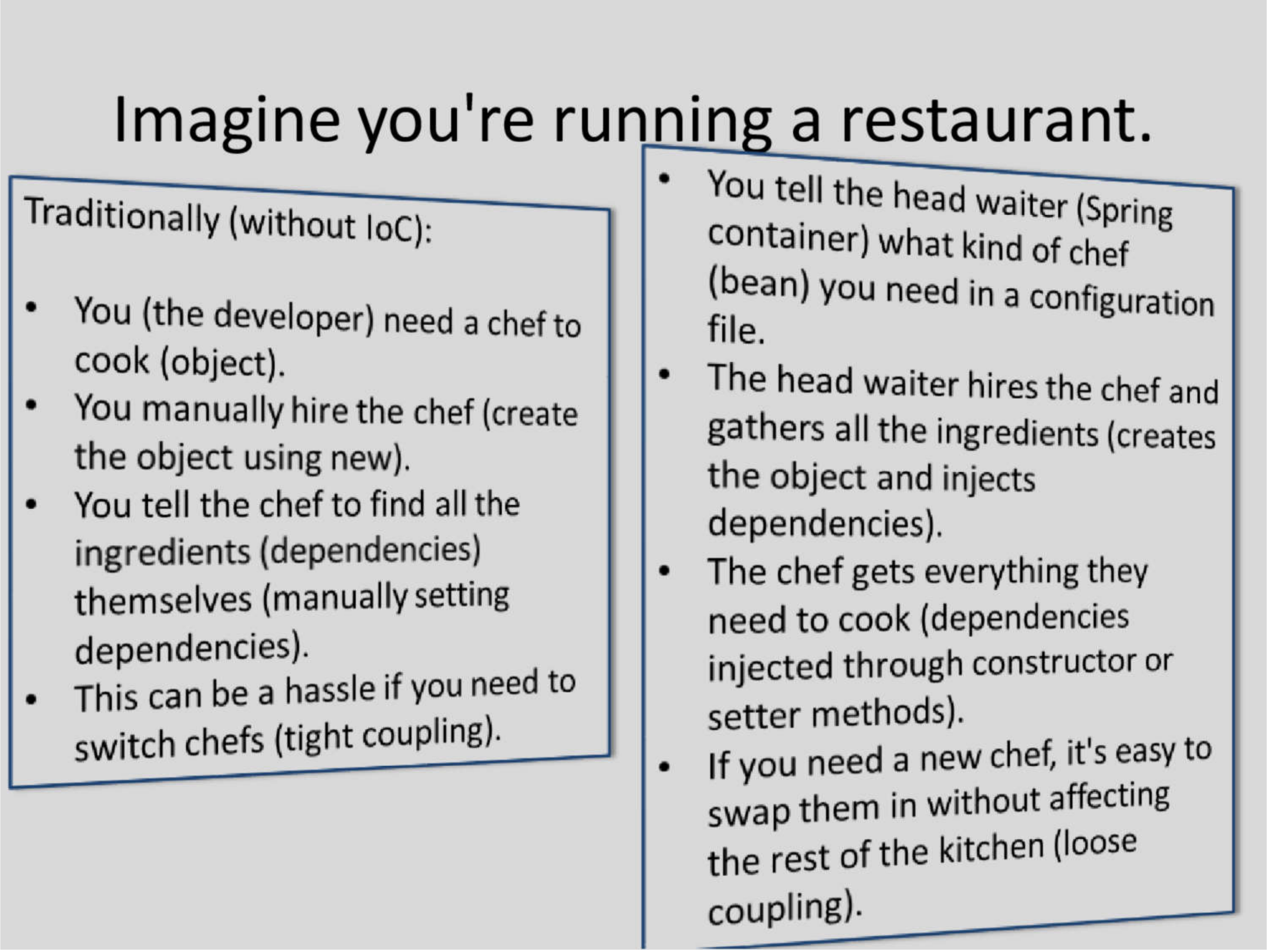
- Spring Platform:
  - Spring framework (Spring Core):
    - Inversion of Control,
    - Dependency Injection,
    - Spring Context etc.
  - Modules (Projects):
    - Spring Boot,
    - Spring Framework (contains Spring WebMVC etc.)
    - Spring Data etc.

IoC

In contrast with traditional programming, in which our custom code makes calls to a library, IoC enables a framework to take control of the flow of a program and make calls to our custom code.

Advantages

- decoupling the execution of a task from its implementation
- making it easier to switch between different implementations.
- greater modularity of a program
- greater ease in testing a program by isolating a component or mocking its dependencies, and allowing components to communicate through contracts.



Imagine you're running a restaurant.

Traditionally (without IoC):

- You (the developer) need a chef to cook (object).
- You manually hire the chef (create the object using new).
- You tell the chef to find all the ingredients (dependencies) themselves (manually setting dependencies).
- This can be a hassle if you need to switch chefs (tight coupling).

- You tell the head waiter (Spring container) what kind of chef (bean) you need in a configuration file.
- The head waiter hires the chef and gathers all the ingredients (creates the object and injects dependencies).
- The chef gets everything they need to cook (dependencies injected through constructor or setter methods).
- If you need a new chef, it's easy to swap them in without affecting the rest of the kitchen (loose coupling).

Software Solution architecture

- Application architectural considerations:
  - Component-based Architectural solutions e.g. MVC
  - Client/Server architecture
  - Rich-client/desktop application e.g. JavaFX, JavaSwing/AWT, Eclipse RCP/SWT, .NET MAUI, WPF, Qt, Electron, Flutter etc.
  - Mobile device client application
  - Web Application architecture

Software Solution architecture

- Architectural considerations:
  - Monoliths and Monolithic architecture
  - Service-Oriented Architecture (SOA)
    - Web Services
      - XML-based SOAP Web Services
      - REST and RESTful Web Services
  - Microservices architecture
  - Distributed systems architecture
  - Layers and Tiers: Data, Application/Business Logic, Presentation
  - IoT and Embedded Device applications

Software Solution architecture

- Exercise:
  - Create two possible software solution architecture diagrams for the City Library system
  - In your diagrams, show the components and layering
  - Also indicate what technology is being used for each component/layer
  - You may use the sample architecture diagram as a guide

CS425: Software Engineering