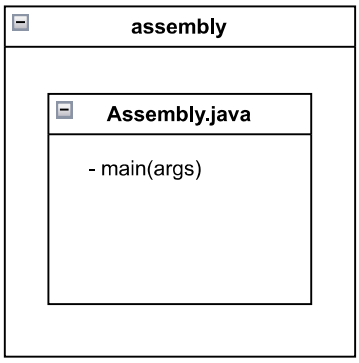
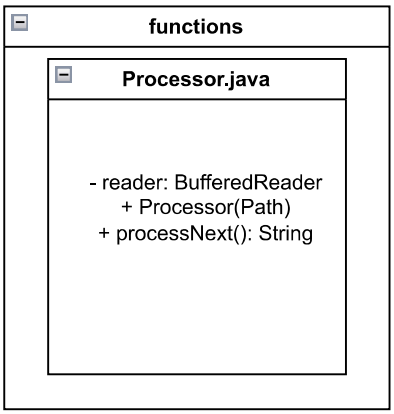




Explanation of Issues

- **Tight Coupling**: The `Processor` class depends directly on low-level classes (`BufferedReader` and `Files`),
- violating the Dependency Inversion Principle (DIP).
- **Hard to Test**: Because `Processor` directly interacts with file-handling code,
- testing it in isolation is difficult.

Low Level Classes:

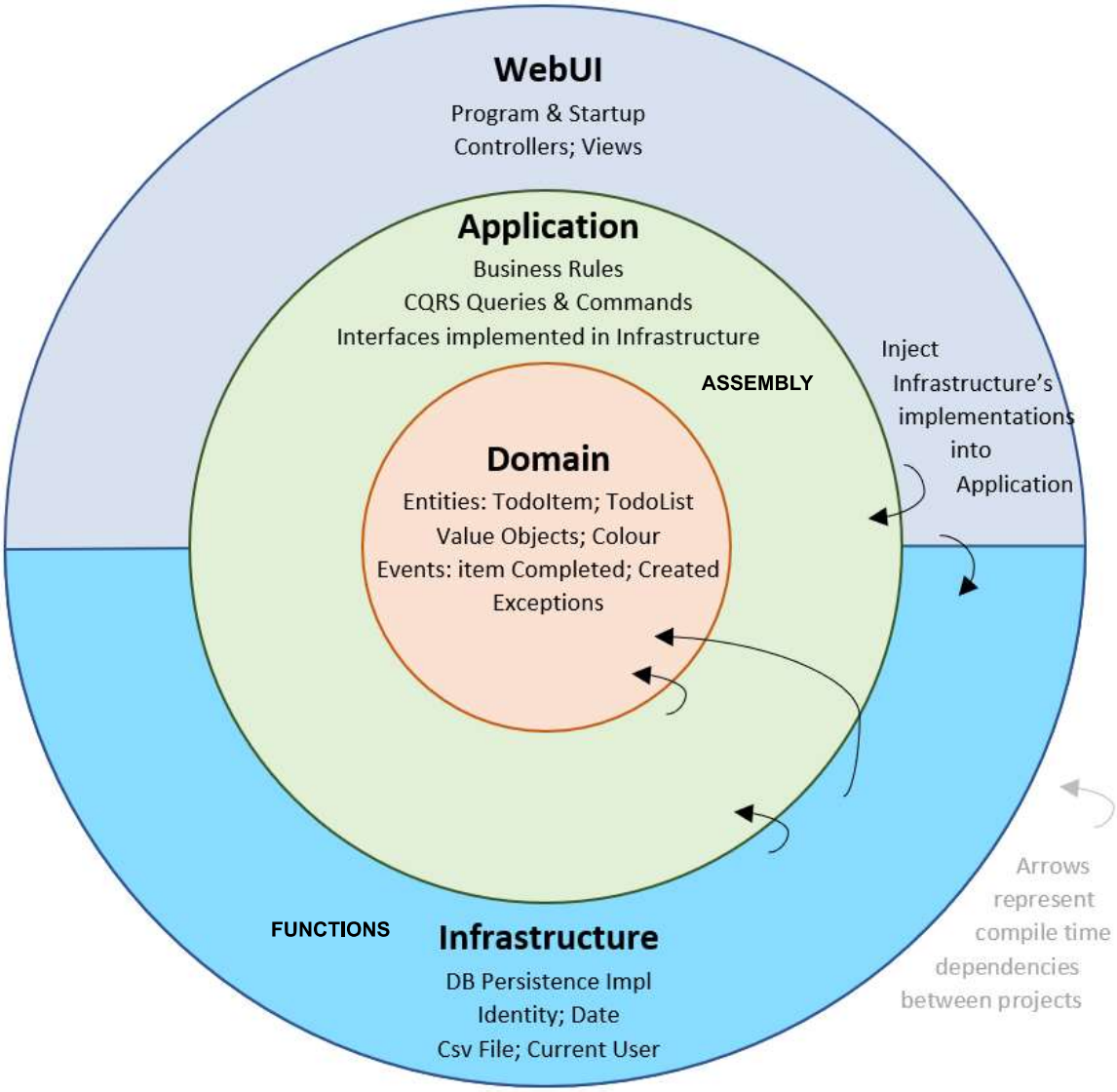
1. Direct File Interaction classes / File Handling functions -> FileInputStream, BufferedReader, Socket, Thread
2. Identify from imports -> java.io, java.nio, java.net

When is `assembly` in the Presentation Layer?

- **WebUI Components**: If `assembly` had classes that specifically dealt with user interaction (e.g., controllers for HTTP routes in a web app or graphical UI logic), it would be part of the **Presentation Layer**.
- **In This Case**: The `assembly` package acts more like a launcher or orchestrator and doesn't directly serve as a user interface. It is not part of a web framework or UI library, so it fits better as part of the **Application Layer** or as a startup component.

Final Clarification:

- **In a Web Application**: You would have a specific UI layer or module for handling web requests (e.g., a `WebController`).
- **In This Example**: `assembly` acts as an orchestrator or entry point, initializing the application and calling the processor, fitting into the **Application Layer** rather than the **Presentation Layer**.


WebUI
Program & Startup
Controllers; Views
Application
Business Rules
CQRS Queries & Commands
Interfaces implemented in Infrastructure
ASSEMBLY
Domain
Entities: TodoItem; TodoList
Value Objects; Colour
Events: item Completed; Created
Exceptions
Inject
Infrastructure's
implementations
into
Application
FUNCTIONS
Infrastructure
DB Persistence Impl
Identity; Date
Csv File; Current User
Arrows
represent
compile time
dependencies
between projects