Exercise: Product Delivery Pipeline

Design an object-oriented solution and write a (Python) program that manages product pipelines for the company.

Problem description:

> We are building a product deployment pipeline to automatically deliver our products to various repositories. The pipeline needs to be easily extensible to new **repository targets**, new **notification channels**, and future **actions.** The extensibility should be expressed and written using object oriented design and language features.

Requirements

1. Each product has the following attributes and actions: scheduled time, build(), deploy() and notify().

2. Deploy may be to one or more of the following targets: Artifactory, Nexus, S3.

3. Notification can be done to one or more channels and user groups (Example: Mail, Slack, ...).

4. The main program should run a product pipeline with the following stages: build, deploy, notify at the scheduled time each day.*

Assignment

1. Design and Document the solution.
2. Define the input for the program.
3. Write a program that runs the product deployment pipelines based on the input.

**** Build, Deploy and Notify no need for true implementation can be printing the time, product and action to stdout/log file.***

Elie's comments

Assignment Requirements and Solution Coverage

1. Each product has the following attributes and actions: scheduled time, `build()`, `deploy()`, and `notify()`.

- **Covered by**: The `Product` class in the solution.
- **Explanation**:
  - The `Product` class includes a `scheduled_time` attribute to define when the pipeline should start.
  - The `build()`, `deploy()`, and `notify()` methods are implemented within the `Product` class as functions, each one handling a specific stage of the pipeline.
  - The methods here print simple statements (for educational purposes) rather than actually performing the operations.

2. Deploy may be to one or more of the following targets: Artifactory, Nexus, S3.

- **Covered by**: Separate deployer classes (`ArtifactoryDeployer`, `NexusDeployer`, `S3Deployer`).
- **Explanation**:
  - Each target has its own deployer class implementing the `Deployer` interface.
  - The `deploy()` method in each deployer class prints a message, simulating deployment to the target.
  - The list of deployers for each product is determined by parsing the JSON configuration, allowing for multiple deploy targets to be specified and easily extended.

3. Notification can be done to one or more channels and user groups (Example: Mail, Slack, ...).

- **Covered by**: Separate notifier classes (`EmailNotifier`, `SlackNotifier`).
- **Explanation**:
  - Similar to deployers, each notification channel has a dedicated class implementing a `Notifier` interface.
  - The `notify()` method in each notifier class prints a message, simulating a notification being sent to a specified channel.
  - The list of notifiers for each product is dynamically created based on the JSON input configuration, supporting multiple channels and extensibility for future channels.

4. The main program should run a product pipeline with the following stages: build, deploy, notify at the scheduled time each day.

- **Covered by**: `Product.run_pipeline()` method in combination with the main `load_config_and_run_pipeline` function.
- **Explanation**:
  - `run_pipeline()` orchestrates the `build()`, `deploy()`, and `notify()` stages in sequence for each product.
  - The main function `load_config_and_run_pipeline` loads the configuration and schedules each product pipeline based on the `scheduled_time` attribute.
  - For simplicity, we simulate immediate execution rather than waiting for specific scheduling. For real-time scheduling, a scheduler library like `schedule` or `cron` would be added.

Assignment Requirements and Solution Coverage

1. Design and Document the Solution

- **Covered by**: The solution design provided in our initial answer, explaining how each requirement is fulfilled using object-oriented design.
- **Explanation**:
  - The design leverages encapsulation for each pipeline stage, with dedicated classes for deployment targets and notification channels. The extensibility requirement is achieved by using interfaces (`Deployer` and `Notifier`), allowing easy addition of new targets or channels.

2. Define the Input for the Program

- **Covered by**: JSON input file `config.json`.
- **Explanation**:
  - The JSON configuration specifies the products, their scheduled times, deploy targets, and notification channels.
  - The input format allows defining multiple products with different configurations, giving flexibility to control each product's deployment pipeline independently.

3. Write a Program that Runs the Product Deployment Pipelines Based on the Input

- **Covered by**: `load_config_and_run_pipeline` function, `Product.run_pipeline` method, and deployers/notifiers.
- **Explanation**:

    - The main function loads the JSON input and dynamically instantiates the appropriate `Deployer` and `Notifier` objects for each product.
    - `run_pipeline()` runs each stage in sequence, simulating the product pipeline by printing actions and timestamps.

Notes for Further Development (if required)

- **Scheduling**: For a true scheduling mechanism, consider adding a scheduling library to trigger `run_pipeline()` at specific times.
- **Logging**: Add logging instead of `print` statements for a real deployment environment.
- **Testing**: Write unit tests to verify each pipeline stage independently, ensuring modularity and robustness.