3. **Describe the coding conventions you were using for your software.**

1. **Naming Conventions**
Classes and Interfaces: Use PascalCase (also called UpperCamelCase), e.g. WebsiteMonitor, ComparisonStrategy, TextContentComparisonStrategy.

Methods and variables: Use camelCase, e.g. checkForUpdates(), fetchWebsiteContent(), lastWebsiteContents.

Constants: Use ALL_CAPS_WITH_UNDERSCORES (though you may not have constants yet).

Packages: Use lowercase with dot notation, e.g. com.uas.websiteMonitor.service.

2. **Indentation and Braces**
Use 4 spaces for indentation (no tabs).

Opening braces { are placed on the same line as the method or class declaration.

Closing braces } are aligned with the start of the block.

3. **Method Design**
Methods should be small and focused — each method does one clear task.

Use descriptive names that clearly explain what the method does.

Avoid very long parameter lists; consider using objects or builders if many parameters are needed.

4. **Comments and Documentation**
Add JavaDoc comments for public classes and methods to describe their purpose.

Use inline comments sparingly, only to clarify complex logic.

Keep comments up-to-date with code changes.

5. **Error Handling**
Catch exceptions explicitly, log or print useful error messages.

Avoid empty catch blocks or silent failures.

Return null or default values clearly if needed, but document that behavior.

6. **Separation of Concerns**
Use interfaces (Subject, Observer, ComparisonStrategy) to decouple components.

Each class has a single responsibility:

WebsiteMonitor manages subscriptions and notifications.

ComparisonStrategy implementations handle different ways of comparing websites.

Subscription encapsulates user subscriptions.

7. **Use of Collections**
Prefer generic collections (List<Observer>, Map<String, String>) to ensure type safety.

Initialize collections on declaration or in constructor.

8. **Package Organization**
Group related classes in packages by functionality:

service for core logic (WebsiteMonitor),

observer for observer pattern interfaces/classes,

model for data classes like Notification, Subscription,

strategy for comparison strategies.