Für die Implementierung des Website-Monitoring-Systems mit dem Strategy Pattern habe ich folgende **Java-Coding-Conventions** angewendet:

**Namenskonventionen**

**Klassen und Interfaces:** Verwendung von **PascalCase** für alle Klassen und Interfaces. Beispiele: ComparisonStrategy, ContentSizeStrategy, HtmlContentStrategy, TextContentStrategy, Website, WebsiteMonitorApp. Interface-Namen sind beschreibend gewählt ohne das traditionelle "I"-Präfix.

**Methoden und Variablen:** Konsequente Anwendung von **camelCase** für alle Methoden und Variablen. Beispiele: hasChanged(), checkForUpdates(), comparisonStrategy, lastContent, removeHtmlTags(). Die Namen sind selbsterklärend und beschreiben die Funktionalität.

**Konstanten:** Würden in **UPPER\_SNAKE\_CASE** geschrieben werden (in diesem Projekt nicht verwendet).

**Code-Struktur und Organisation**

**Ein Konzept pro Datei:** Jedes Interface und jede Klasse erhält eine eigene Datei für bessere Modularität und Wartbarkeit. Das Strategy Pattern wird durch separate Dateien für ComparisonStrategy, ContentSizeStrategy, HtmlContentStrategy und TextContentStrategy umgesetzt.

**Dependency Injection:** Der Website-Konstruktor verwendet **Constructor Injection** für die ComparisonStrategy, was das Dependency Inversion Principle befolgt und Testbarkeit verbessert.

**Kapselung:** Alle Klassenattribute sind **private** deklariert und werden nur über öffentliche Methoden zugänglich gemacht. Private Hilfsmethoden wie getUrlContent() und removeHtmlTags() sind klar von öffentlichen Methoden getrennt.

**Methodendesign**

**Kurze, aussagekräftige Namen:** Methoden haben präzise Namen, die ihre Funktion beschreiben: hasChanged(), setComparisonStrategy(), removeHtmlTags(). Boolesche Methoden beginnen mit Verben wie "has" oder "is".

**Single Responsibility:** Jede Methode hat eine klar definierte Aufgabe. Die hasChanged()-Methode in den Strategy-Klassen konzentriert sich ausschließlich auf den Vergleich, während removeHtmlTags() nur für die HTML-Bereinigung zuständig ist.

**Kommentierung und Dokumentation**

**Sparsame aber präzise Kommentare:** Kommentare werden nur dort eingesetzt, wo sie echten Mehrwert bieten. Beispiel: // Neue Eigenschaft für Strategy Pattern oder // Verwende die Strategy für den Vergleich.

**Selbstdokumentierender Code:** Der Code ist so geschrieben, dass er durch aussagekräftige Namen und klare Struktur weitgehend selbsterklärend ist, wodurch übermäßige Kommentierung vermieden wird.

**Design Patterns und Architektur**

**Strategy Pattern Implementation:** Saubere Trennung zwischen Interface (ComparisonStrategy) und konkreten Implementierungen. Die Website-Klasse ist gegen das Interface programmiert, nicht gegen konkrete Klassen.

**Observer Pattern Beibehaltung:** Das bestehende Observer Pattern aus Exercise 5 wurde beibehalten und nahtlos mit dem Strategy Pattern kombiniert, ohne die bestehende Architektur zu beeinträchtigen.

**Open-Closed Principle:** Neue Vergleichsstrategien können hinzugefügt werden, ohne bestehenden Code zu ändern, was die Erweiterbarkeit und Wartbarkeit des Systems gewährleistet.