HW14 – Design Pattern

Strategy Pattern - CountryRace

**What is a design pattern?**

A design pattern in software engineering is a reusable solution to a commonly occurring problem within a given context in software design. Design patterns are standard reference points for the experienced software developer. These patterns are not complete templates but rather guidelines on how to solve a problem in different situations. They provide a blueprint for how to structure or solve complex software design issues.

**What is the Strategy Pattern?**

The Strategy Pattern, a type of behavioural design pattern, enables an object to have some of its behaviour defined by a separate object, which follows a particular interface. In simpler terms, it allows an object to change its behaviour dynamically by switching out a part of itself (the strategy) with another that conforms to a common interface.

**Strategy Pattern in CountryRace:**

For the CountryRace java application, I implemented the Strategy Pattern, a behavioural design pattern that defines a family of algorithms, encapsulates each one, and makes them interchangeable. This pattern lets the algorithm vary independently from clients that use it. It is particularly useful in scenarios where you need to dynamically change the behaviour of an application.

**Implementation Details:**

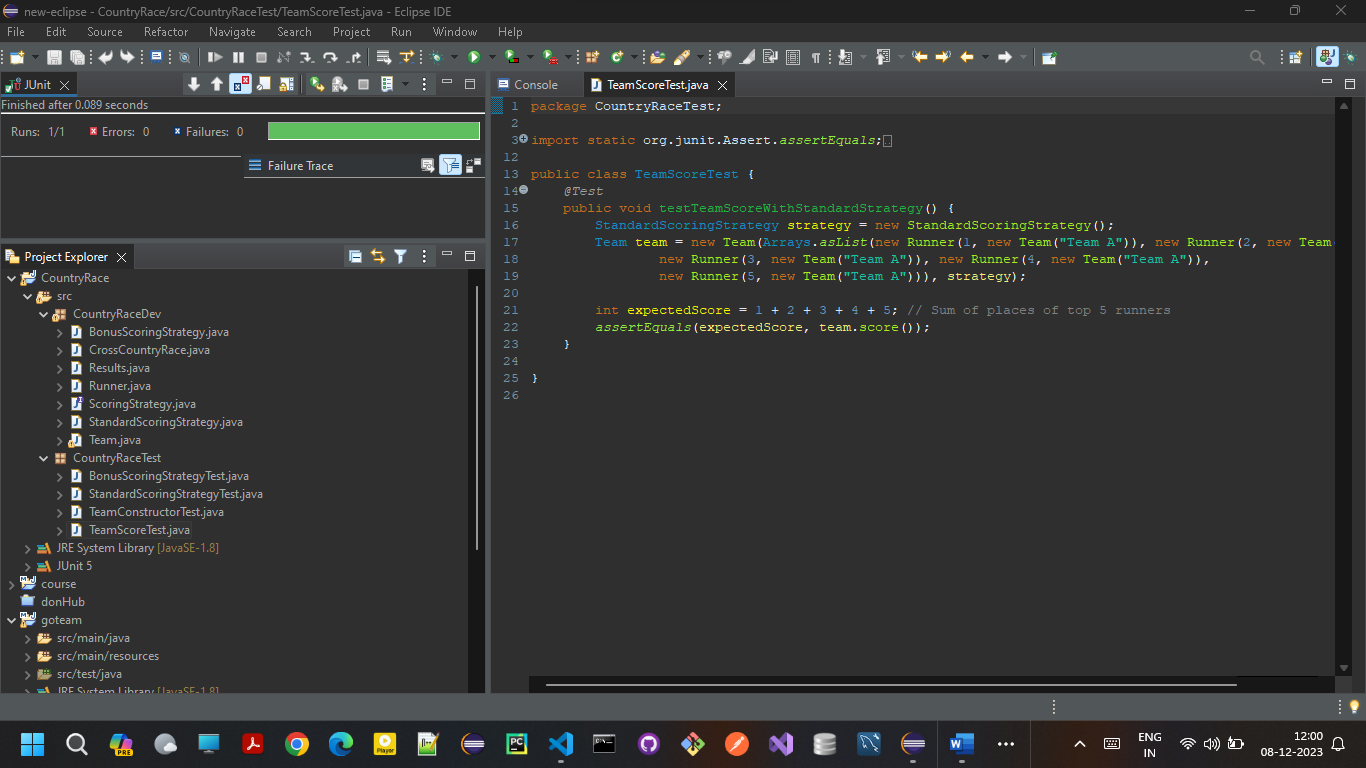
1. **Scoring Strategy Interface:** The core of the Strategy Pattern implementation involved defining a **ScoringStrategy** interface. This interface declared a **calculateScore** method, which different scoring strategies must implement. By doing this, we encapsulated the scoring algorithm, making it easily interchangeable and extendable.
2. **Concrete Strategy Classes:** Two concrete classes, **StandardScoringStrategy** and **BonusScoringStrategy**, were created as implementations of the **ScoringStrategy** interface. Each class provided a different way of calculating scores for the teams. **StandardScoringStrategy** might implement the traditional scoring method, while **BonusScoringStrategy** could introduce bonus points or other variations in the scoring mechanism.
3. **Integration with Team Class:** The Team class was modified to accept a **ScoringStrategy** object. This integration allowed each **Team** instance to calculate its score based on the provided strategy. As a result, it became possible to assign different scoring behaviours to different teams, if required, without modifying the internal logic of the **Team** class.
4. **Flexible and Maintainable:** Applying the Strategy Pattern to the **CrossCountryRace** application introduced a high level of flexibility. It allowed the scoring logic to be extended or modified without altering the core application logic. For instance, introducing a new scoring method would simply involve creating a new class that implements the **ScoringStrategy** interface.

**Impact of implementation:**

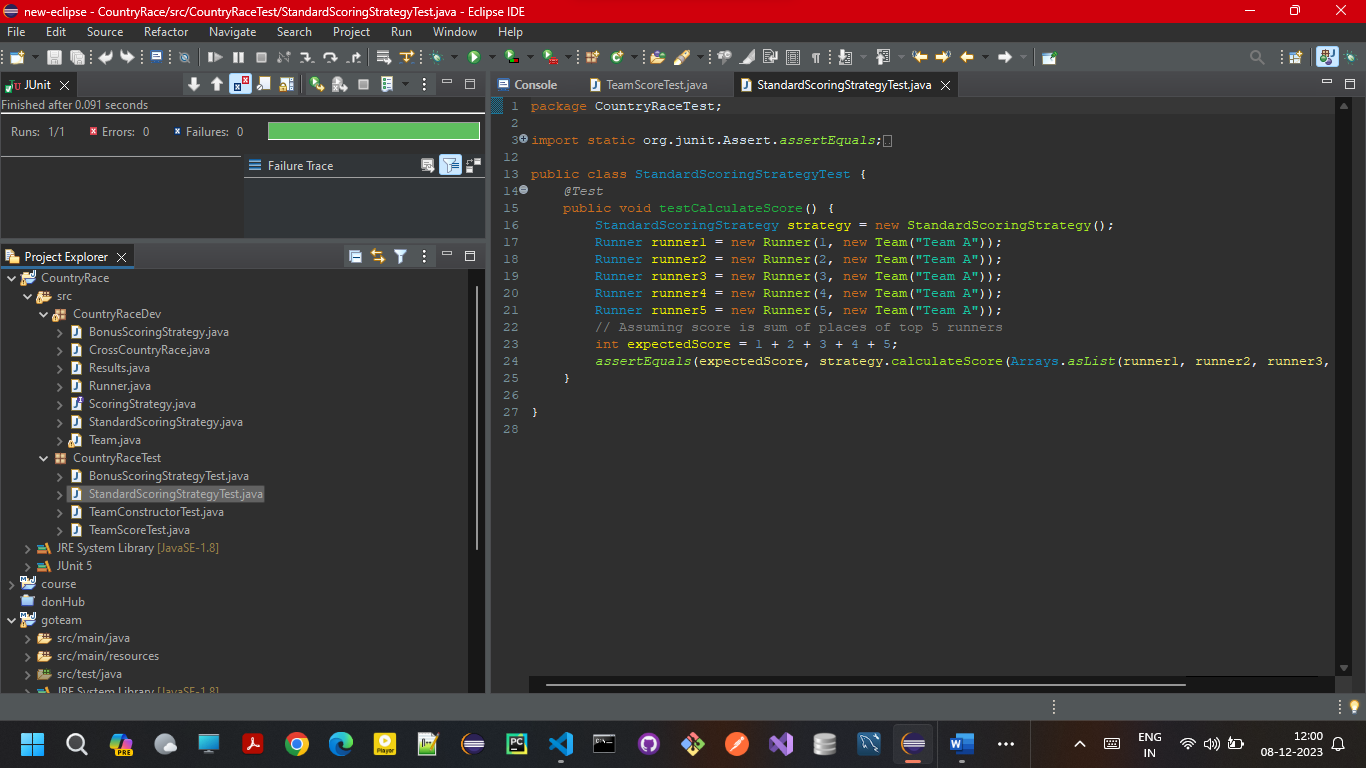
1. **Enhanced Flexibility:** The use of the Strategy Pattern made it easy to introduce new scoring algorithms without changing the existing classes. This is particularly useful for adapting to new requirements or rules in the cross-country race scoring system.
2. **Improved Maintainability:** By encapsulating the scoring logic within separate strategy classes, the application's maintainability improved. Each scoring algorithm could be developed, tested, and debugged independently of others, reducing complexity and potential for errors.
3. **Ease of Extension:** Adding a new scoring strategy became a straightforward process, enhancing the application's ability to evolve. This is beneficial in a dynamic environment where scoring rules might change frequently.
4. **Decoupled Code:** The Strategy Pattern helped in decoupling the scoring logic from the Team class, following the principle of separation of concerns. This decoupling not only made the code more modular but also easier to understand and manage.

**Screenshots of the test cases:**

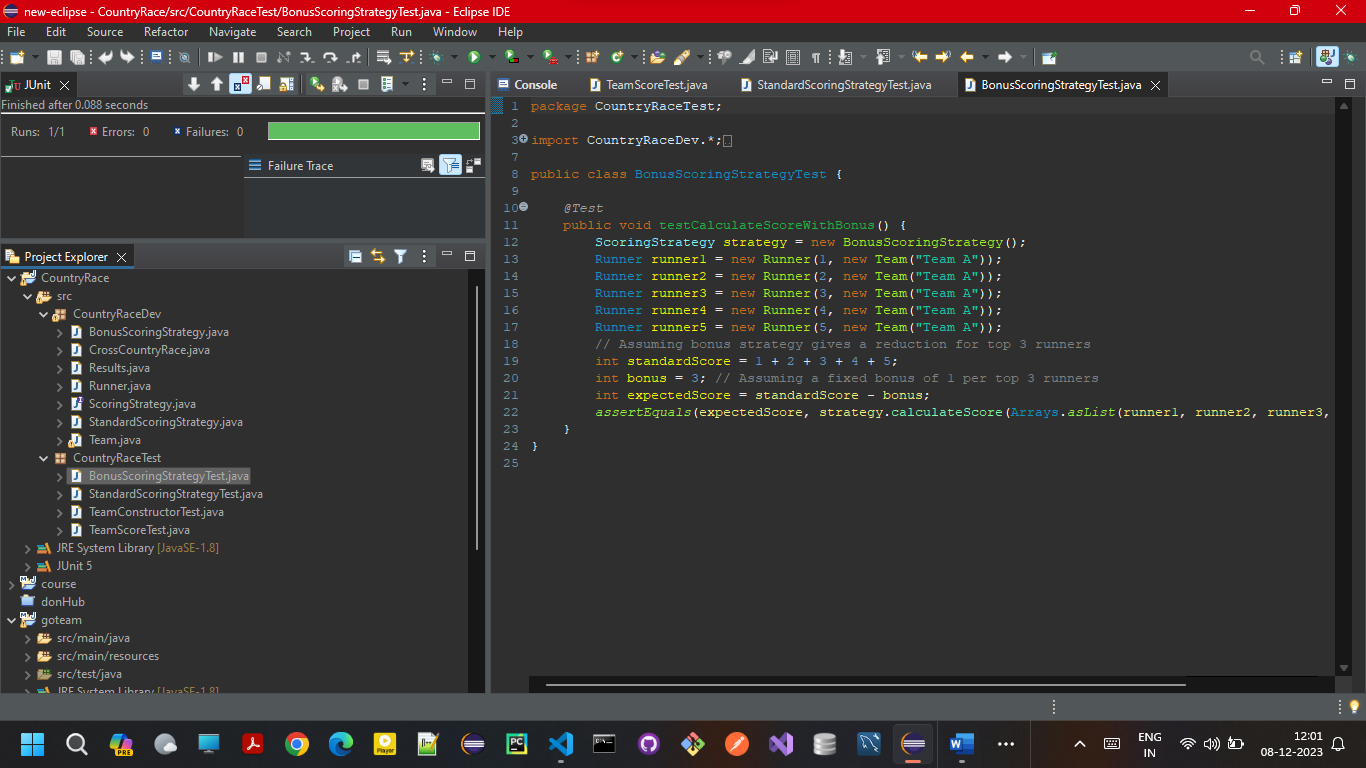
1. **TeamScoreTest.java:**

****

1. **StandardScoringStrategyTest.java:**

****

1. **BonusScoringStrategyTest.java:**

****