Let's analyze the provided **TimeBlock** class in terms of cohesion, coupling, refactoring, and design patterns:

Cohesion:

1. **Functional Cohesion:**
    - The class seems to have functional cohesion, as it focuses on time-related functionality, such as setting start and end times, calculating duration, and determining the price based on the hourly rate.

Coupling:

1. **Low Coupling:**
    - The class has low coupling, as it primarily depends on standard Java libraries (**ZonedDateTime**, **Duration**). The **getPrice** method mentions querying a staff member's hourly rate, but the implementation details are not shown, implying that the coupling with other components is deferred.

Refactoring:

1. **Method Extraction:**
    - The **setStartAndEnd** method is a good example of method extraction, separating the responsibility of setting start and end times.
2. **Encapsulation:**
    - Private fields (**staffID**, **startTime**, **endTime**, **durationHours**) are encapsulated, and access is controlled through getter and setter methods.

Design Patterns:

1. **Builder Pattern:**
    - Although not explicitly implemented, the constructor and the **setStartAndEnd** method provide a way to construct a **TimeBlock** object in a step-by-step manner. This is reminiscent of the Builder Pattern.
2. **Strategy Pattern:**
    - The **getPrice** method mentions that the implementation is not shown, but it implies the use of a strategy pattern where different strategies could be used for calculating the price based on the staff member's hourly rate.

Other Software Engineering Techniques:

1. **Use of Java Date and Time API:**
    - The class uses the **ZonedDateTime** and **Duration** classes from the Java Date and Time API, which is a good practice for handling time-related operations.
2. **Consistent Naming:**
    - Methods and variables are named consistently and descriptively, enhancing code readability.

In summary, the **TimeBlock** class exhibits functional cohesion, low coupling, and employs refactoring techniques like method extraction and encapsulation. It hints at the possible use of design patterns such as the Builder Pattern and Strategy Pattern. The use of Java Date and Time API and consistent naming contributes to good software engineering practices.