HW13 - Refactoring

CorrectedClock.java

Refactoring the **CorrectedClock** class from the **com.test.Junit** package has a common code smell known as **Feature Envy.** This issue arises when a method frequently uses data from another object rather than its data, indicating that it might belong better in a different class. Within the **CorrectedClock** class, the **pad**, **getSuffix** methods, and the formatting processes in **get12HourFormat** and **get24HourFormat** demonstrate characteristics of Feature Envy.

**Refactoring Steps:**

1. **Shifting the Formatting to a new class:** The formatting logic within **get12HourFormat** and **get24HourFormat** could be moved to a new class, **TimeFormatter**, which would take responsibility for all time-related formatting. This makes **CorrectedClock** focused solely on time manipulation.
2. **Move Related Helpers Over:** Those little helper methods, **pad** and **getSuffix**, which help in formatting the time, will also be moved to **TimeFormatter**. They’re really more about making the time look right rather than figuring out what the time is.
3. **Make Sure Everything Still Works:** After we’ve made our changes, we’ll run these tests again to make sure we haven’t messed anything up and that **CorrectedClock** still works as it should.

**TimeFormatter.java:**

package com.test.JUnit;

public class TimeFormatter {

public static String format12Hour(int hours, int minutes, int seconds) {

int formattedHours = hours;

if (formattedHours == 0) {

formattedHours = 12;

} else if (formattedHours > 12) {

formattedHours -= 12;

}

return *pad*(formattedHours) + ":" + *pad*(minutes) + ":" + *pad*(seconds) + " " + *getSuffix*(hours);

}

public static String format24Hour(int hours, int minutes, int seconds) {

return *pad*(hours) + ":" + *pad*(minutes) + ":" + *pad*(seconds);

}

private static String pad(int value) {

return (value < 10) ? "0" + value : String.*valueOf*(value);

}

private static String getSuffix(int hours) {

return (hours < 12) ? "AM" : "PM";

}

}

**CorrectedClock.java (after refactoring):**

package com.test.JUnit;

public class CorrectedClock {

// ... existing fields and constructors ...

// Methods for adding time remain the same

// Modified get12HourFormat, get24HourFormat

// Removed pad, and getSuffix methods

}

public String get24HourFormat() {

return TimeFormatter.*format24Hour*(hours, minutes, seconds);

}

public String get12HourFormat() {

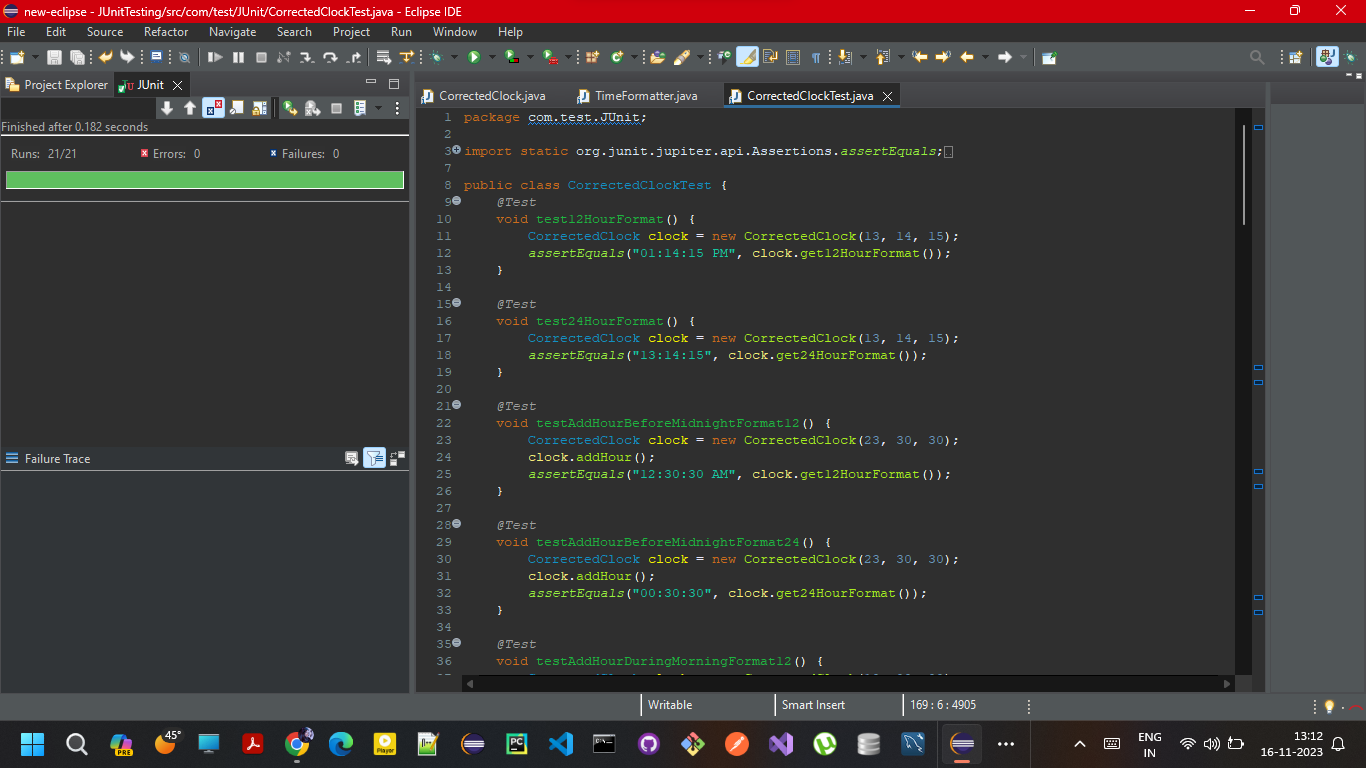
return TimeFormatter.*format12Hour*(hours, minutes, seconds);

}

**Explanation**

* The f**ormat12Hour** and **format24Hour** methods in **TimeFormatter** handle the formatting of time based on the given hours, minutes, and seconds.
* The **pad** method adds a leading zero to single-digit numbers, ensuring a consistent two-digit format.
* The **getSuffix** method determines whether the time is AM or PM based on the hour.

By moving these methods to **TimeFormatter**, we're separating the concerns of time calculation and time display, which makes both **CorrectedClock** and **TimeFormatter** easier to understand and maintain. The **CorrectedClock** class now focuses solely on managing time, while **TimeFormatter** takes care of how the time is presented.



**How Feature Envy Manifests:**

1. **Accessing Data of Another Class:** The method frequently calls methods or accesses data fields of another class, more so than its own class.
2. **Lack of Cohesion:** There's a lack of cohesion within the class as the method relates more to the data it's envying than the data in its own class.
3. **Poor Encapsulation:** It indicates potential issues with encapsulation where classes are not adequately encapsulating their data and behaviour, leading to tightly coupled designs.

**Impact on Implementation:**

1. **Refactoring Requirement:** To address this, refactoring was needed to extract the formatting logic into a separate class (**TimeFormatter**). This aligns with the Single Responsibility Principle, where each class should only have one reason to change.
2. **Separation of Concerns:** By moving the formatting logic to **TimeFormatter**, the separation of concerns is achieved. **CorrectedClock** now focuses solely on time manipulation, while **TimeFormatter** handles the presentation of time.
3. **Enhanced Readability and Maintenance:** This makes both classes cleaner and more maintainable. Changes in time formatting no longer require modifications in the **CorrectedClock** class, and vice versa.
4. **Improved Testability:** It also improves testability, as each class can now be tested independently for its specific set of responsibilities. **CorrectedClock** can be tested for its time calculation accuracy, while **TimeFormatter** can be tested for correct time formatting.
5. **Design Flexibility:** The change increases the design's flexibility. If new formats for time representation are needed, they can be added to **TimeFormatter** without affecting **CorrectedClock**.

In summary, addressing the Feature Envy smell in **CorrectedClock** led to a cleaner, more modular design where each class has a clear, distinct responsibility. This improves the overall maintainability and scalability of the code.