**Refactoring for Build 3**

**Potential list of Refactoring targets**

1. Use try-with-resource

* WriteMap.java 38

1. Remove Explicit Type Argument

* Continent.java 25
* Country.java 26
* GameEngine.java 314
* GameMap.java 244
* MapValidation.java 132
* MapValidationTest.java 144 152

1. Split Multiple Variable Declaration

* ReadMap.java 52
* WriteMap.java 37

1. Use Collection Singleton List

* GameStarterTest.java 5

1. Remove Collection :: addAll

* GameEngine.java 314

1. Reuse Random Objects

* GameEngine.java 29

1. Use Secure Random

* GameEngine.java 14

1. Remove Boxing for String Conversion

* GameMap.java 250 254

1. Replace assignment with Compound Operator

* Commands.java 109 118 154 163 199 209 281 290

1. Use StringBuilder::Append

* GameEngine.java 44 47 278
* GameMap.java 245
* GameStarter.java 49
* ReadMap.java 51
* WriteMap.java 40 50 62

1. Use Multi-Catch

* GameEngine.java 742
* Player.java 334

1. Use @Override Annotation
   * EditPhase.java 28 39 50 60 70 81 92 103 114 125 136 147 158 169 179 189
   * EndPhase.java 180
   * GamePhase.java 26 37 48 59 70 81 92
   * GameSetup.java 60 71 82 93 104 115 125 135 146
   * PostLoad.java 62
   * PreEdit.java 57
2. Split Multiple Variable Declarations
   * ReadMap.java 50
   * WriteMap.java 36
3. Remove Modifiers from Interface Properties
   * Ovserver.java 18
4. Reorder modifiers
   * Phase.java 29 38 47 56 abstract public -> public abstract

**Refactoring targets used**

1. **Adapter Pattern**

**Adapter** pattern is used enable the application to read or write from or to a file using the “conquest” game map format. The application decides whether to use the “domination” file reader or the “conquest” file reader, when a file is opened, depending on the file type.

When the map file is saved, the user is given the option to decide the file format to use as output.

1. **Strategy Pattern**

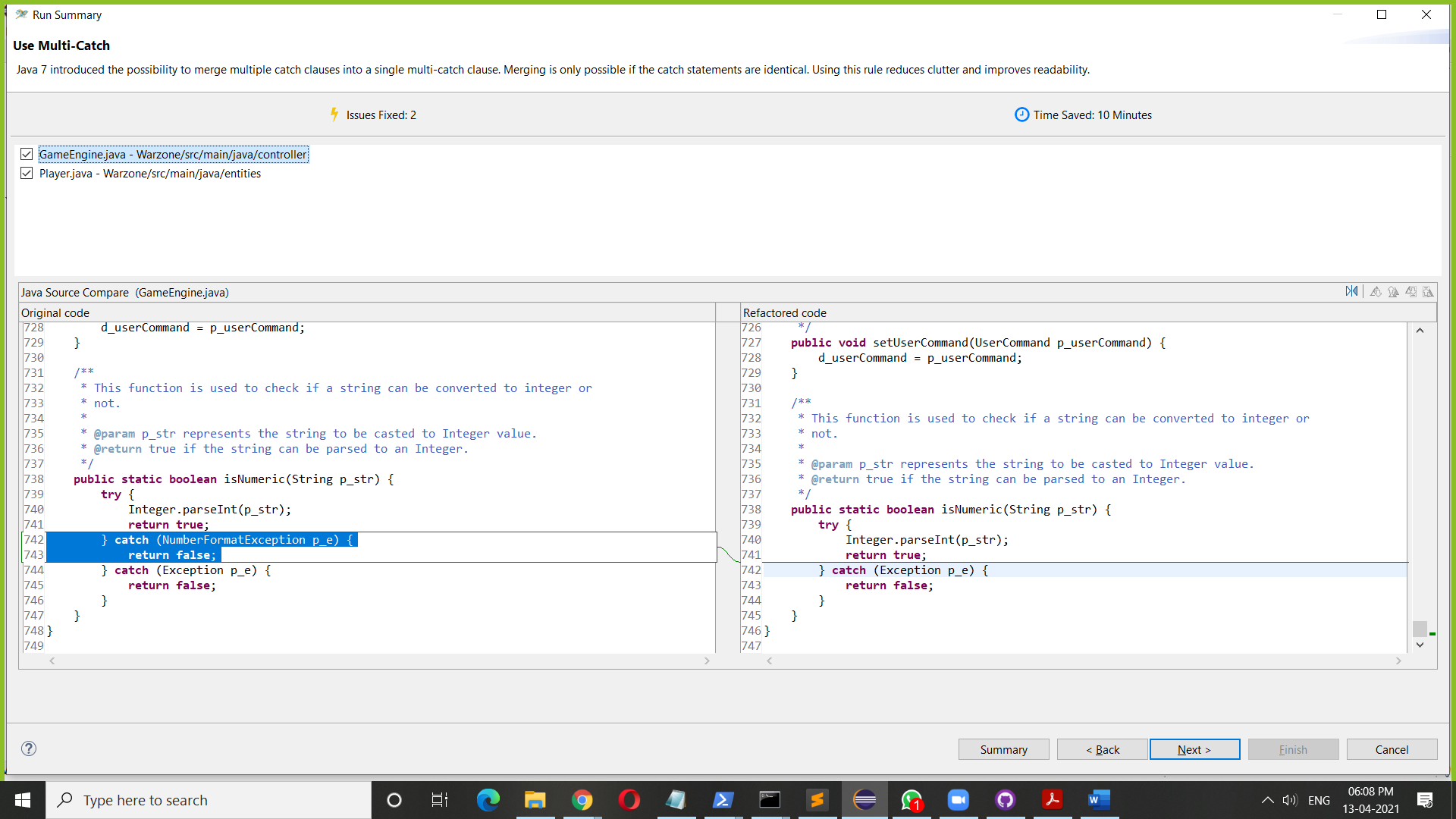
Here the player’s issueOrder() method’s behavior is altered by using the **Strategy** pattern. Implementation of different computer player behaviors is done. The strategies provide different behaviors that support the Player class to execute varying behavior when executing the issueOrders() method.

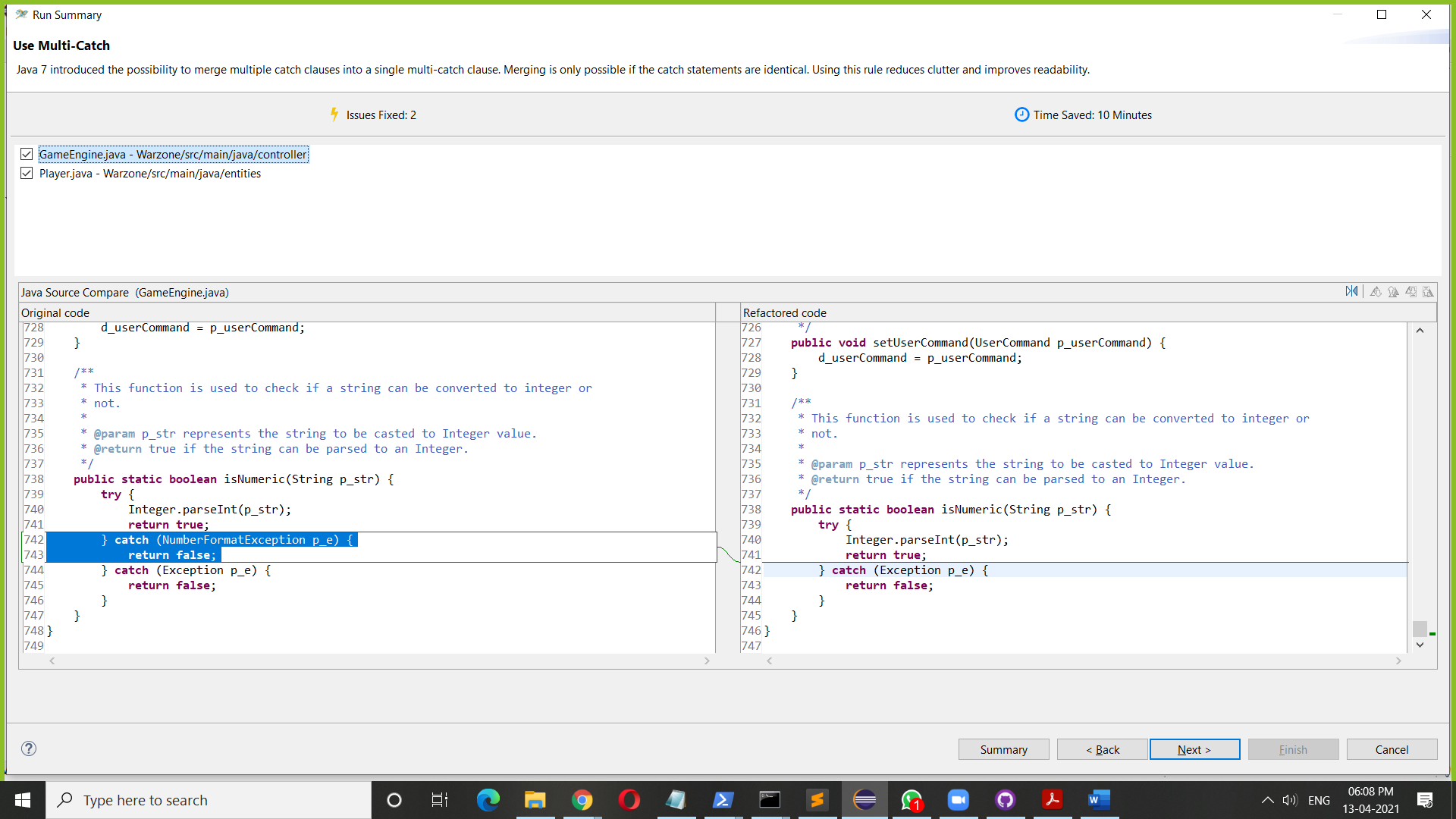
Player types are:

1. A **human** player requires user interaction to make decisions.
2. An **aggressive** player strategy i.e. deployment on its strongest country, attacking with its strongest country, moving its armies in order to maximize aggregation of forces in one country.
3. A **benevolent** player strategy i.e. deployment on its weakest country, never attacks, then moving its armies in order to reinforce its weaker country
4. A **random** player strategy i.e. deployment on a random country, attacking random neighboring countries, moving armies randomly between its countries.
5. A **cheater** player strategy whose issueOrder() method conquers all the immediate neighboring enemy countries, and then doubles the number of armies on its countries that have enemy neighbors. The cheater’s strategy implementation is still called when the issueOrder() method is called, but it won’t end up creating orders, but rather implement this behavior by directly affecting the map during the order creation phase.

1. **Use Multi catch**

Merge multiple catch clauses into a single multi-catch clause. Merging is done if catch statements are identical, in turn reducing clutter and improved readability.

Before

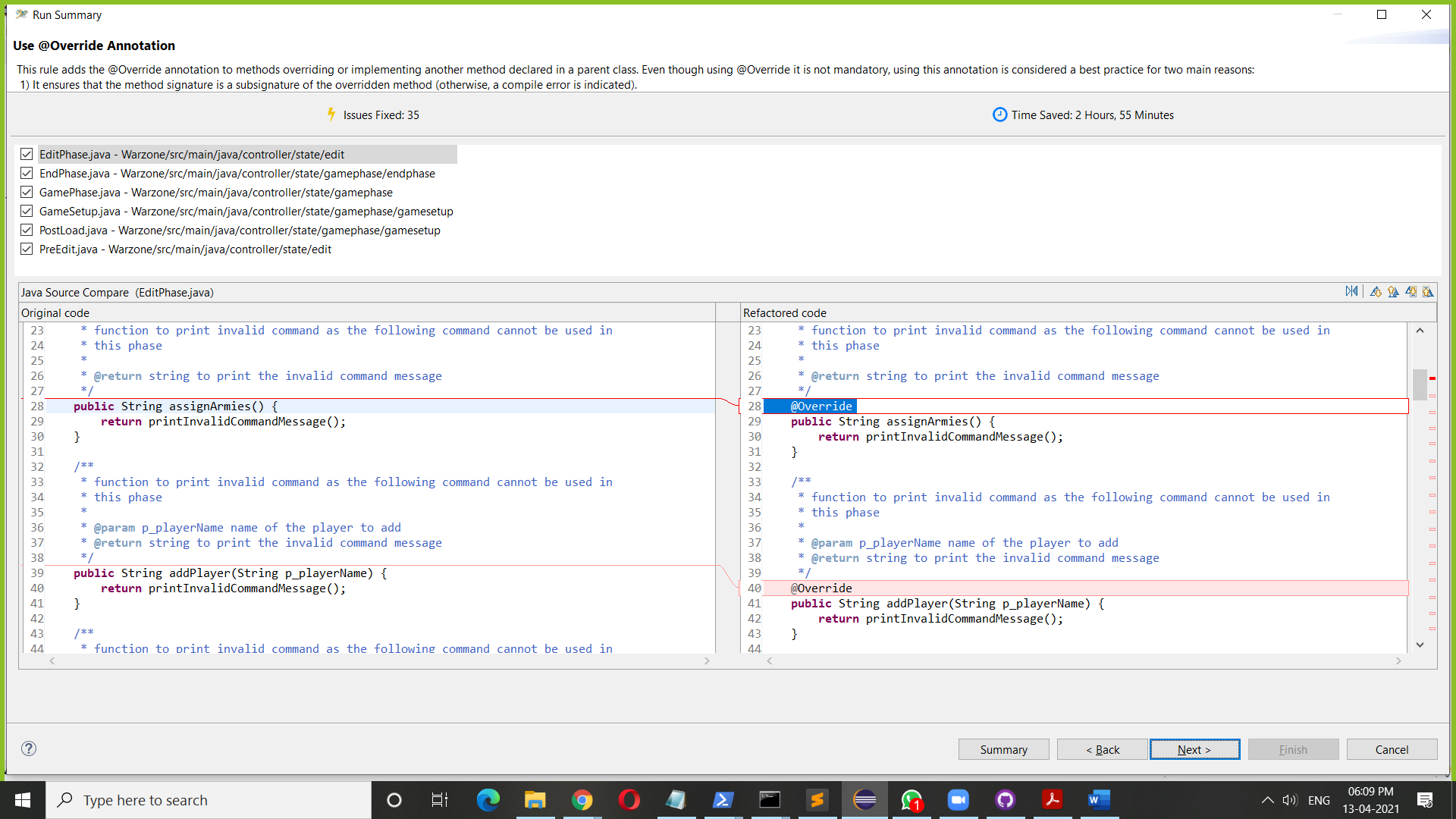
After

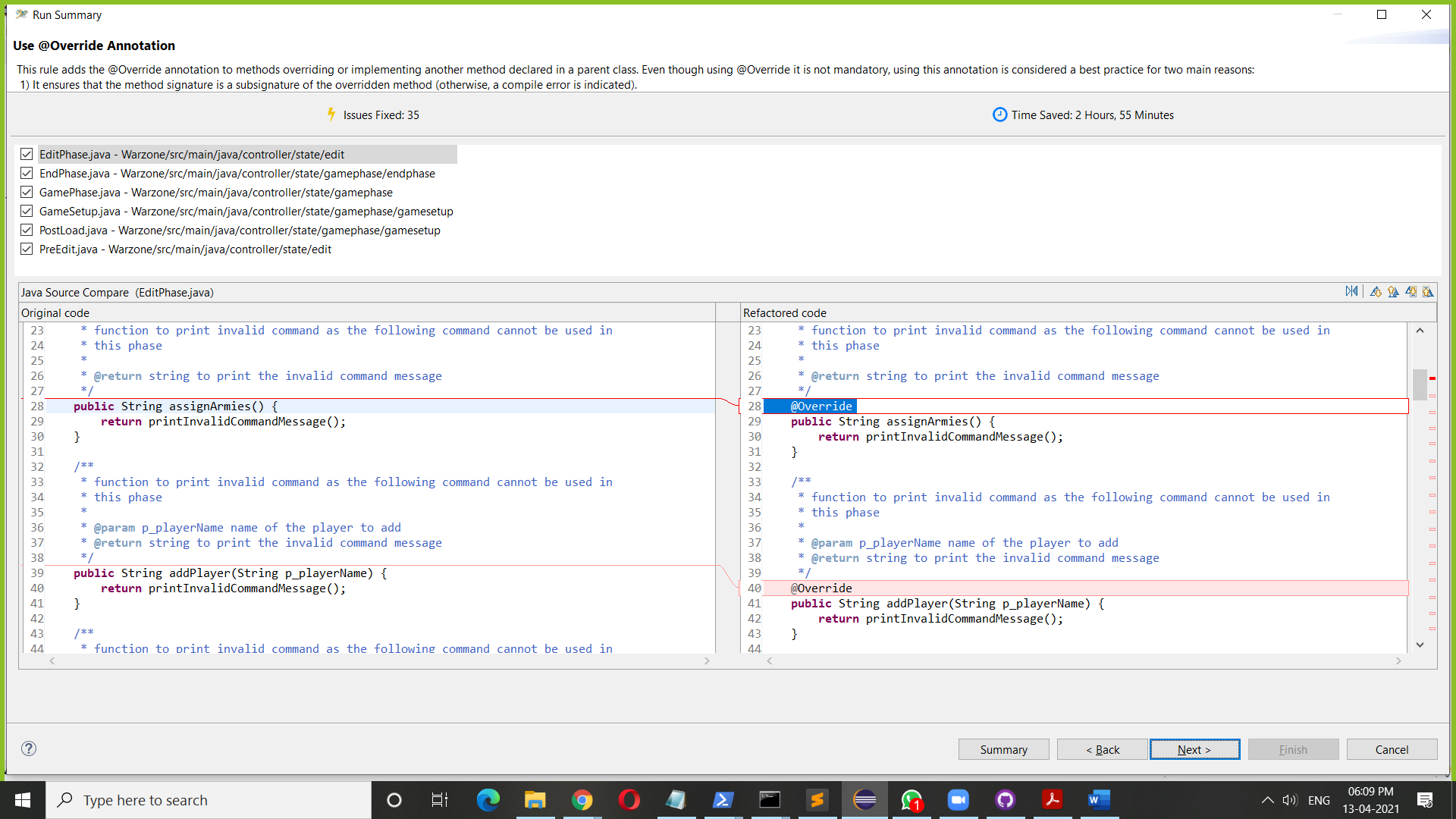
1. **Using @Override notations**

This rule adds the @Override annotation to methods overriding or implementing another method declared in a parent class.

1) It ensures that the method signature is a sub-signature of the overridden method (otherwise, a compile error is indicated)

2) It improves the readability.

Before

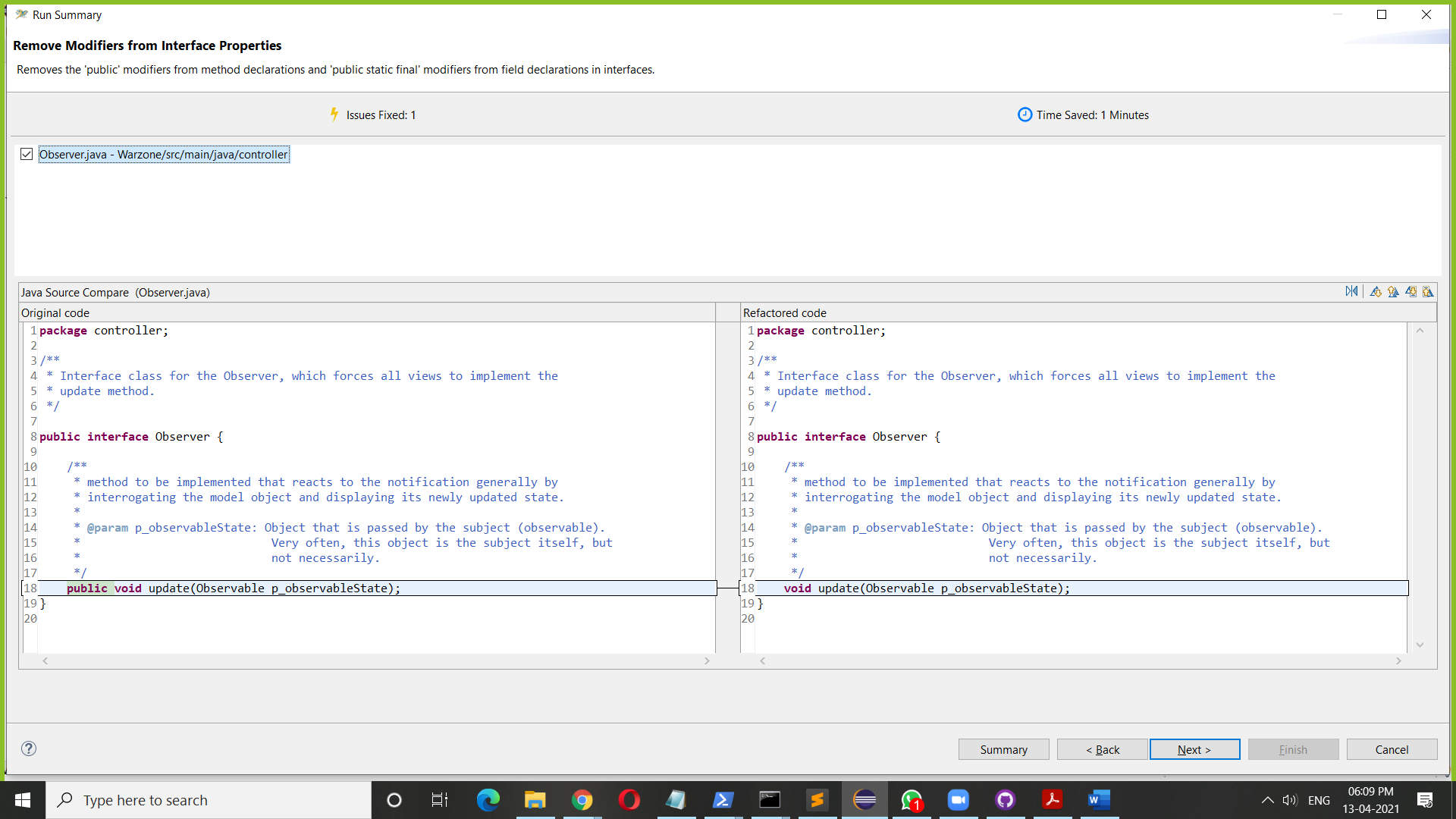
After

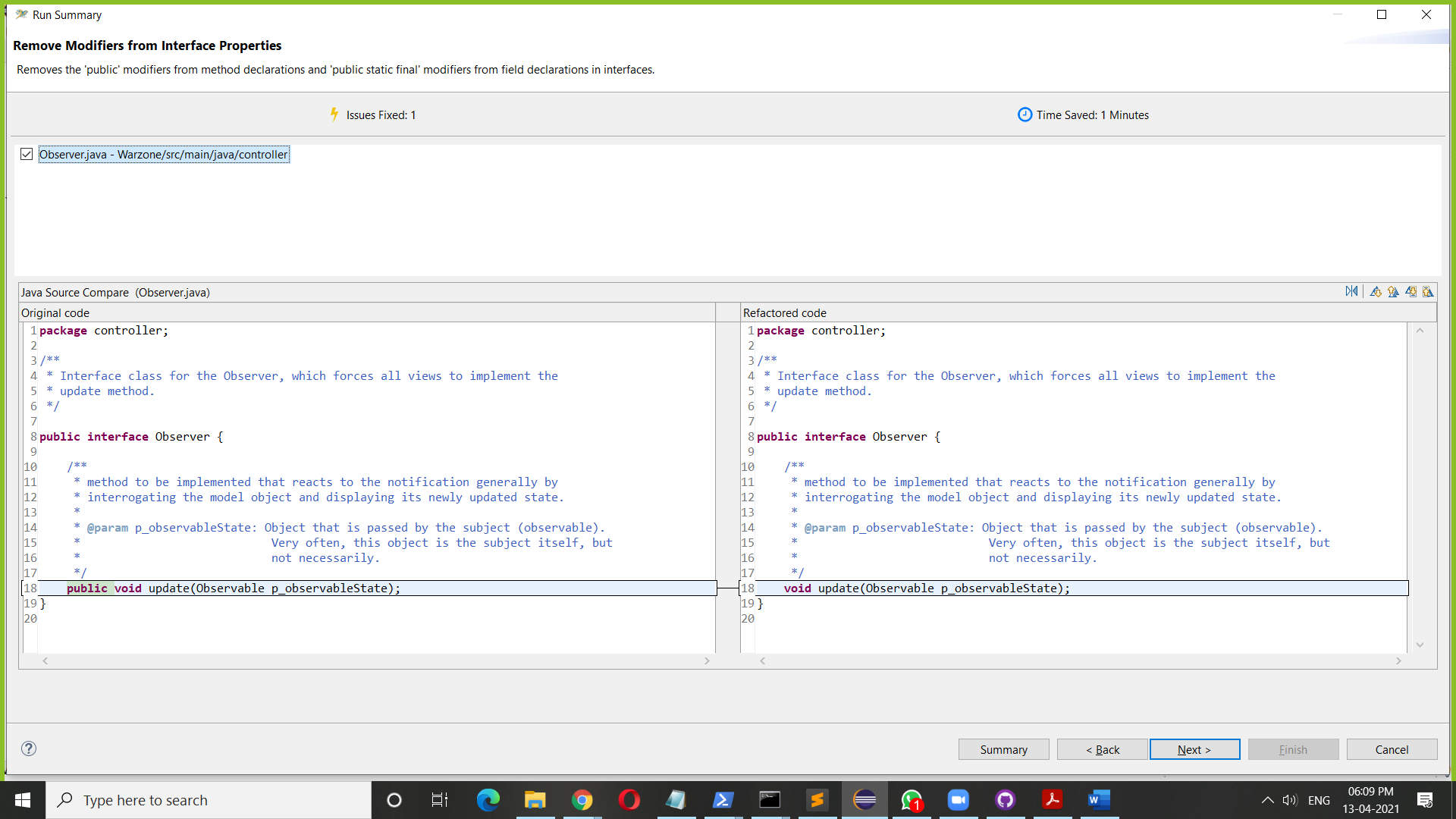
1. **Removal of Modifiers from Interface properties**

Remove Modifiers from Interface Properties – Removes the ‘public’ modifiers from method declarations and ‘public static final’ modifiers from field declarations in interfaces.

Removes the clutter and unifies the attributes of Java interfaces

Before



After