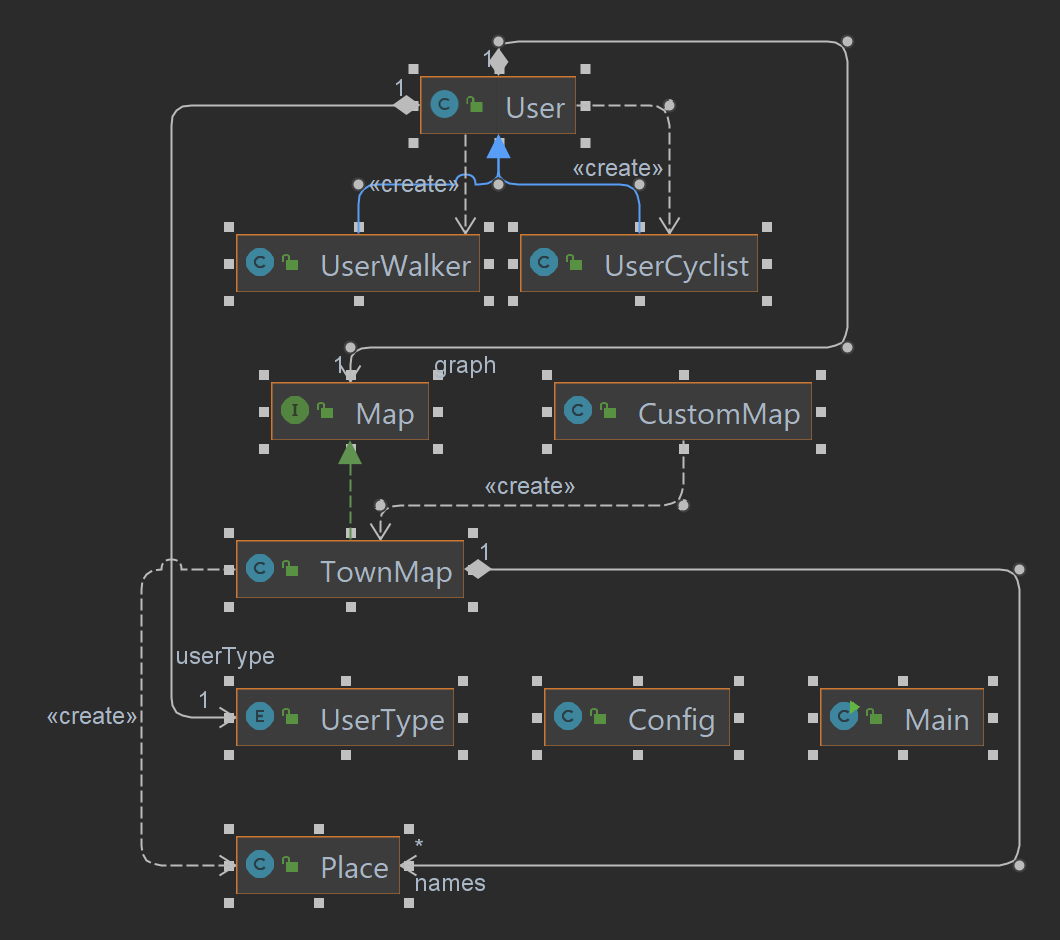
Application OOP- Fast Travel.co

Artem Vasko, 15.05.2023

Purpose of the Project:

The project "Fast Travel.co" is a Java application that calculates the best (fastest) route for walking or bicycling in a town, taking into account the user's interests and type. The main purpose of this project is to provide an efficient and personalized route planning solution for users who want to explore a town in a fun and efficient way, while also taking into account their physical abilities and preferences.



The Java application is designed to calculate the fastest route for walking or bicycling in a town, taking into consideration the interests and user type. It utilizes a graph data structure to represent the town map, with vertices representing locations and edges representing the roads between them. Each edge has a weight, which represents the distance or time it takes to traverse that road.

The application prompts the user to enter their starting location and the type of tourist attraction they are interested in. Based on this information, the application calculates the fastest route to the user's destination, taking into consideration the user's speed and the type of transportation they are using.

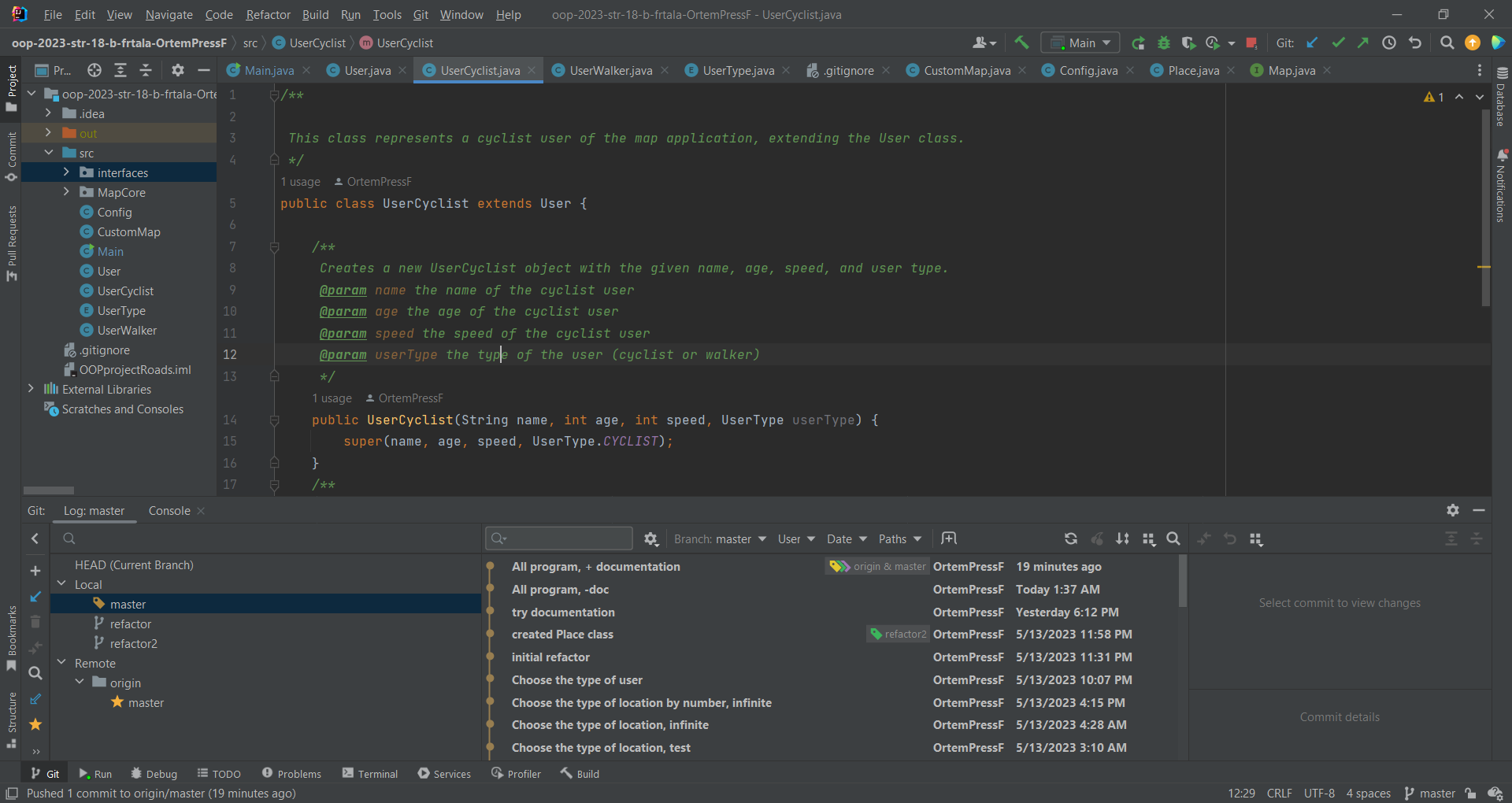
To add more convenience, after every next reached object user will decide, if he wants to continue or change the type of the next destination.  
  
  
Users can choose between two types of transportation: walking or bicycling. Cyclists have a faster speed than walkers and can cover more ground in a shorter amount of time. Users can also select their interests, which influence the recommended route. For example, if a user is interested in historical landmarks, the application will prioritize routes that take them past these landmarks.  
  
The application uses a factory method to create user objects, which are subclasses of the User class. The User class stores information about the user, such as their name, age, speed, and user type. The UserCyclist and UserWalker classes are subclasses of User, with the former having a faster speed than the latter.  
  
The Map interface defines methods for adding edges to the graph, getting the weight of an edge, setting the name of a vertex, finding the fastest path to all vertices of a certain type. The Graph class implements the Map interface and contains a two-dimensional array to represent the graph.

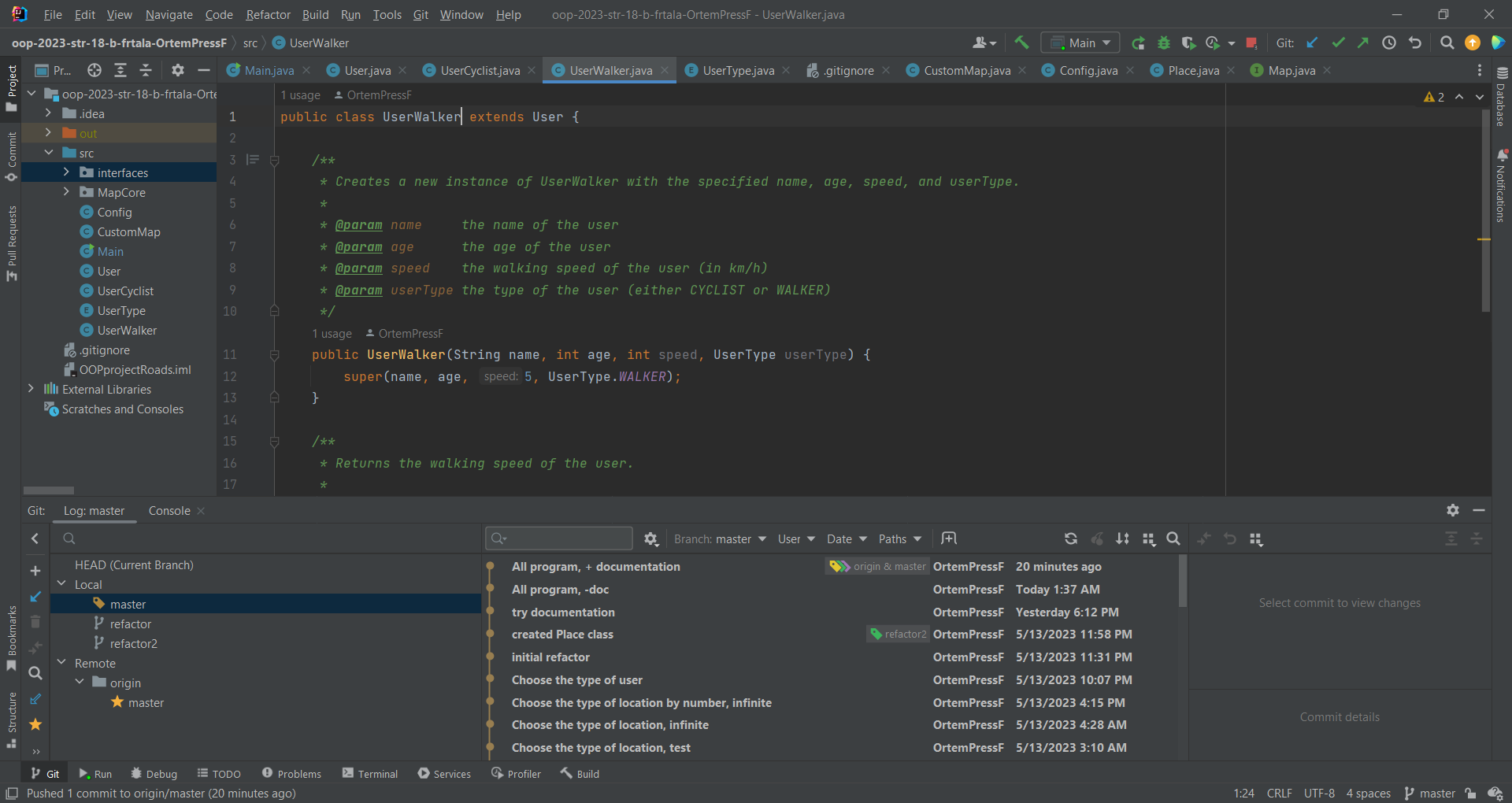
The application provides a user-friendly interface for entering input and viewing the recommended route. If the user enters invalid input, such as a non-integer value for their starting location, the application handles the exception and prompts the user to enter valid input.  
  
Overall, this Java application provides a useful tool for users who want to explore a town by foot or bicycle and discover interesting tourist attractions along the way. Its intuitive interface, customizable options, and efficient routing algorithm make it a valuable resource for tourists and locals alike.

In the GitHub you can see a list of the main versions of the program that I’ve submitted, and the names describe the most important changes for each version.

I think my project fulfills the most of evaluation criteria:

**Inheritance and Polymorphism:**

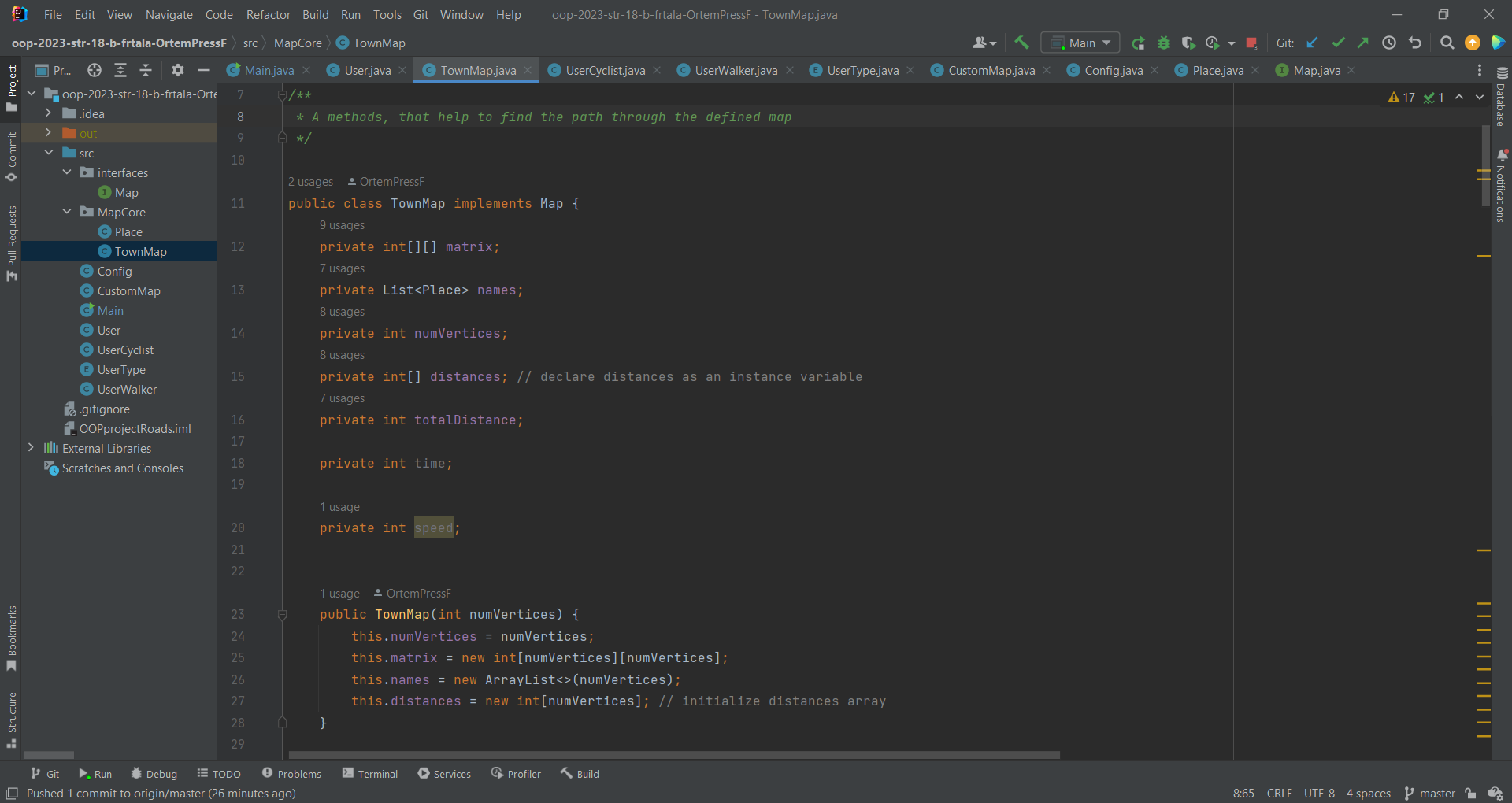
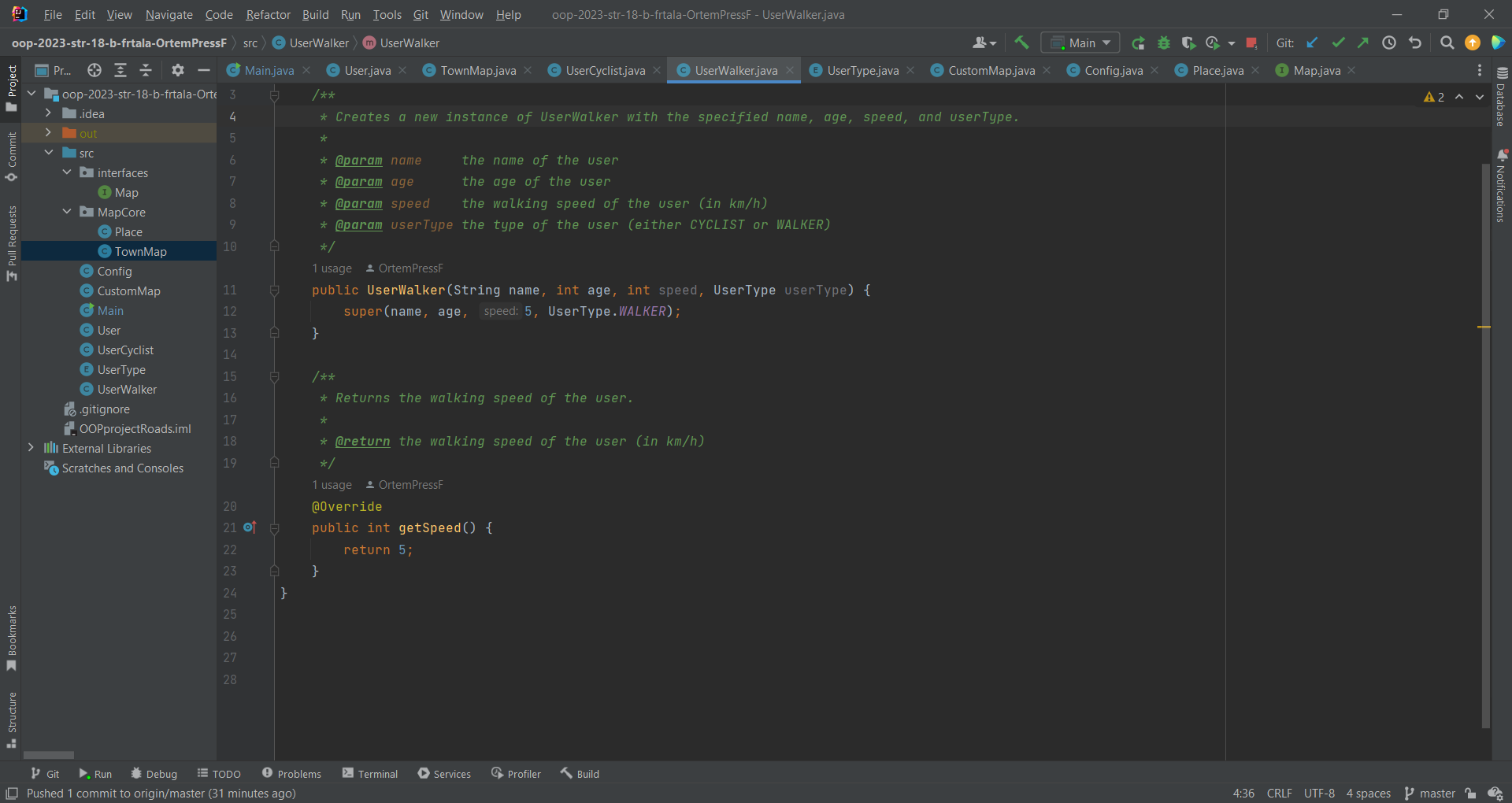


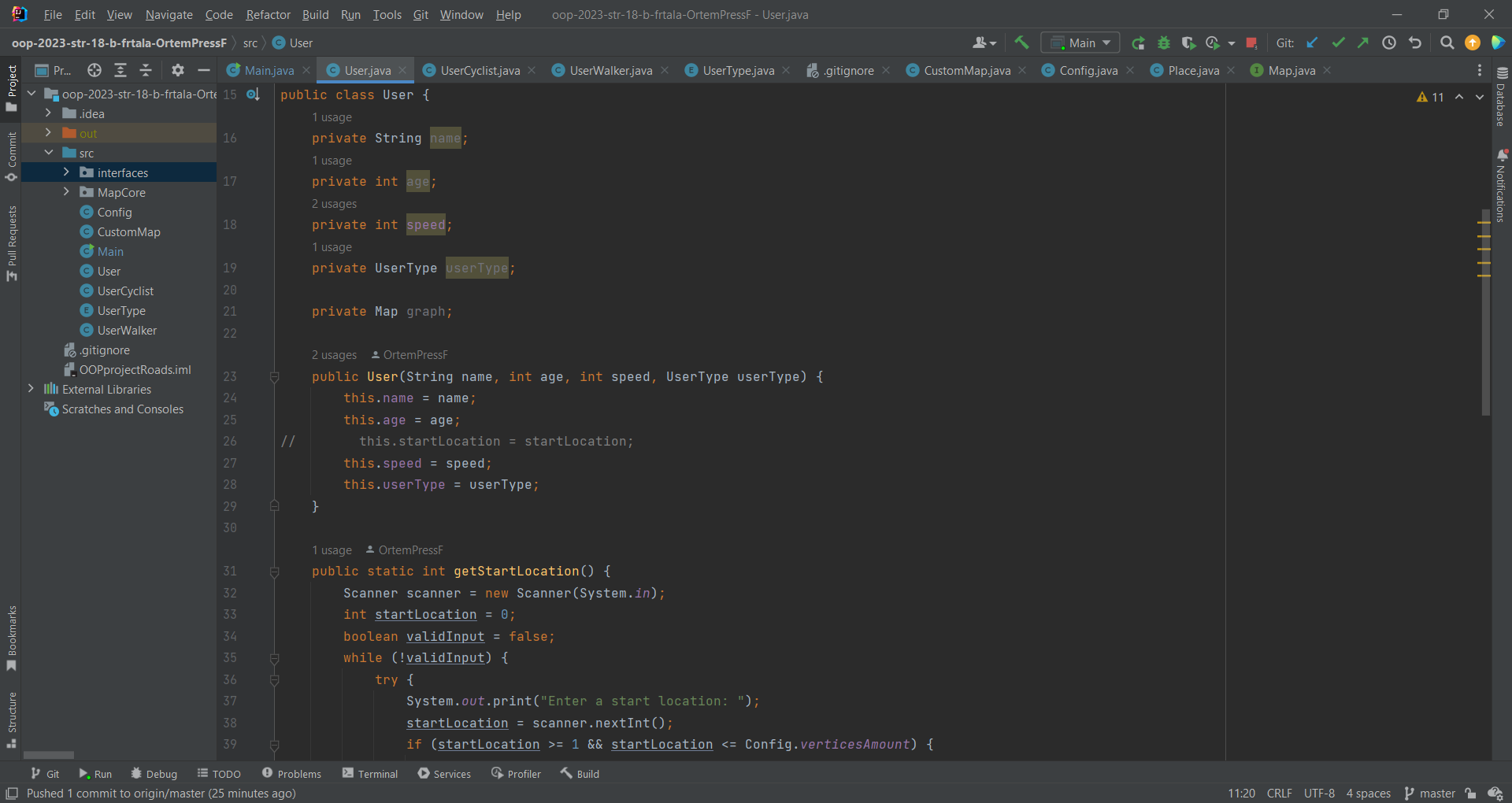


The project includes inheritance between classes, such as the User class and its subclasses UserCyclist and UserWalker.; Additionally, the Map interface is implemented by the TownMap class, which can be extended further to create more specialized maps.

**Encapsulation:**

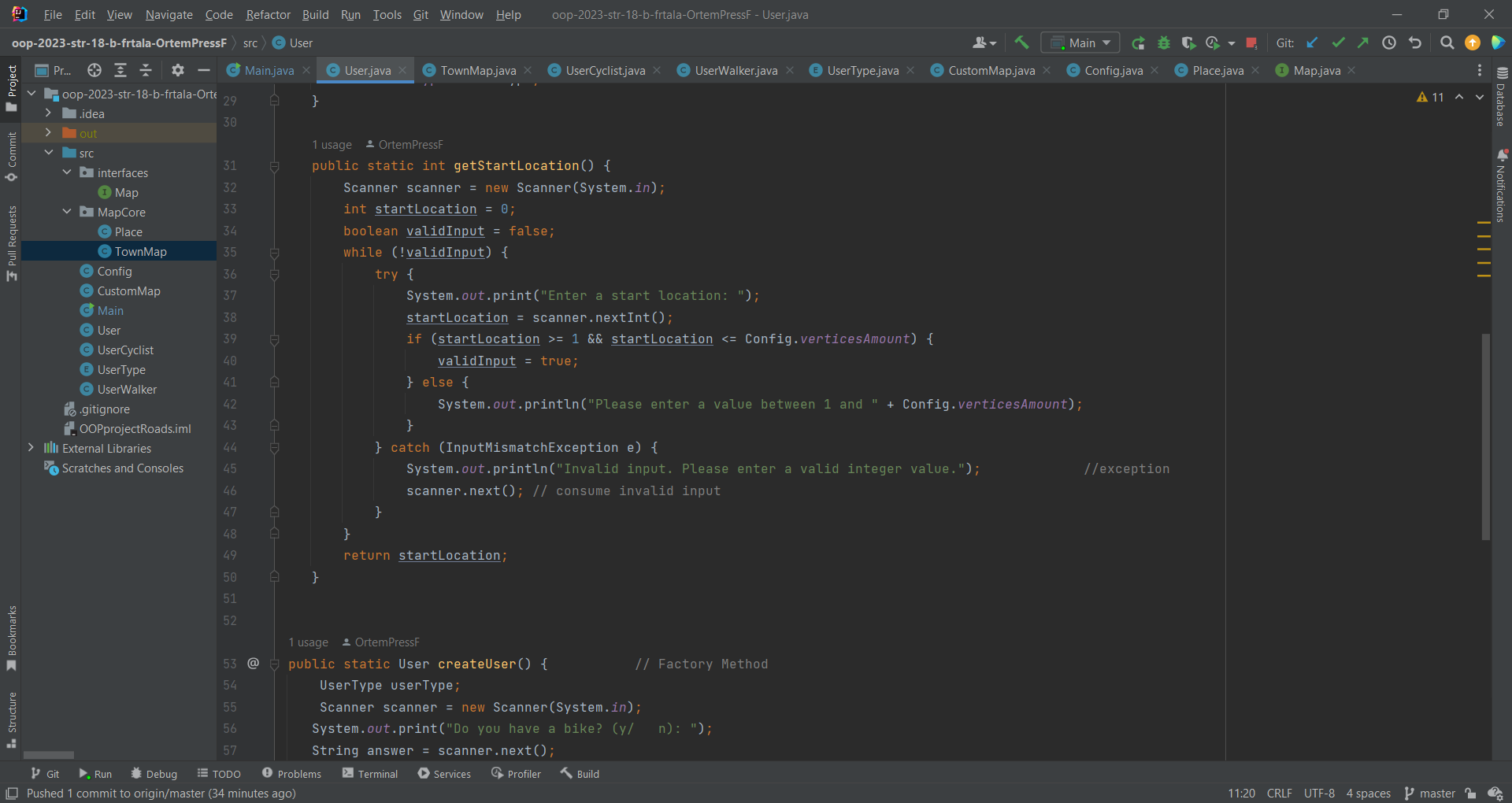
is used in my project to hide the implementation details of the classes from external code. For example, the User class has private fields that can only be accessed through public getter and setter methods.





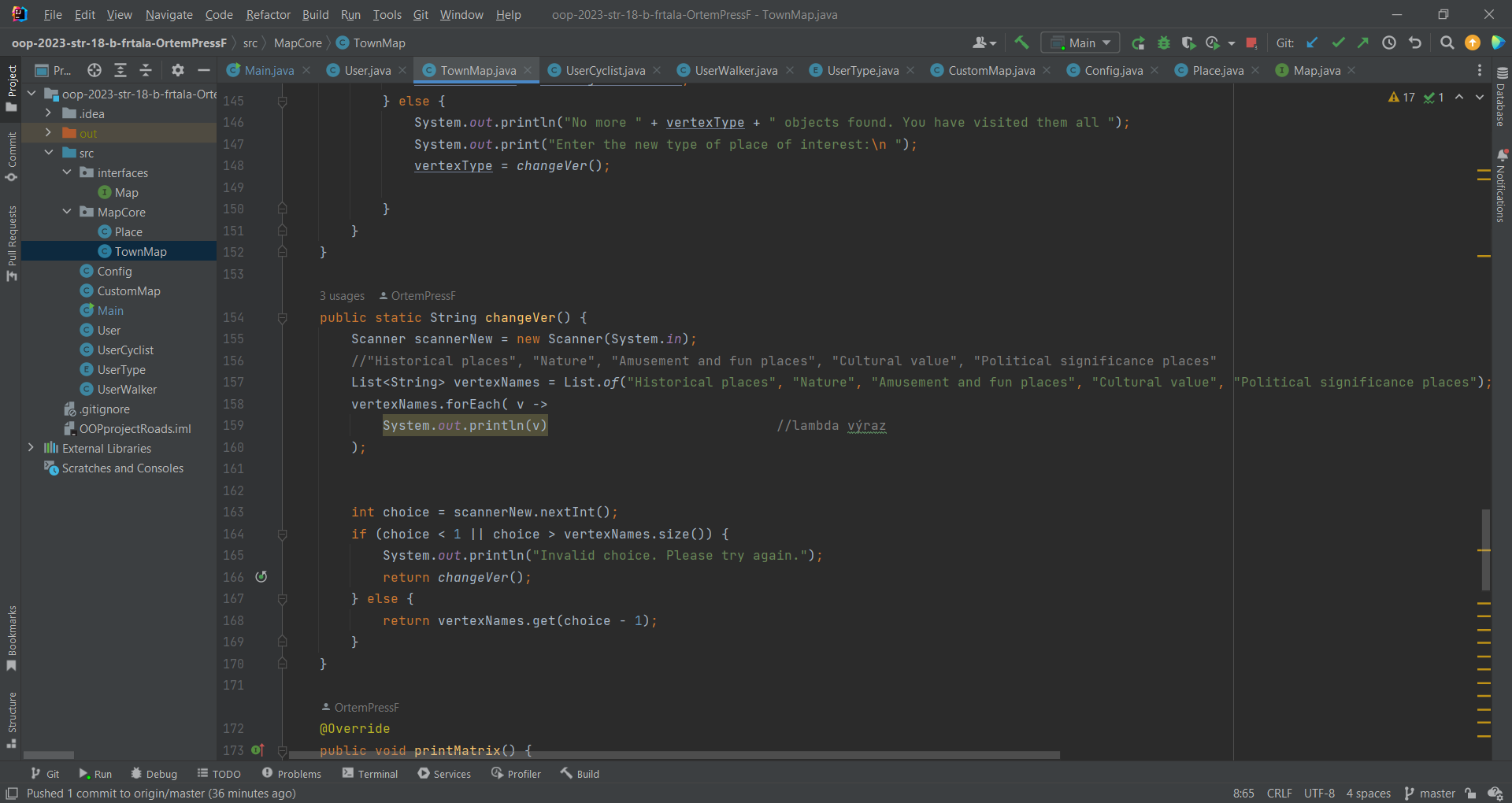
**Handling custom exceptions:**

My project may use custom exceptions to handle errors or unexpected situations. For example, if a user tries to find a path to a location that doesn't exist on the map, a custom exception can be thrown and handled appropriately.



**Multithreading**: project may use multithreading to perform tasks concurrently. For example, if a user is searching for the fastest path to a location, the application may use a separate thread to perform the search in the background.

**Lambda expressions:**



**Use of design patterns:**

Factory pattern provides a way to create objects in a flexible and decoupled manner, making it easier to maintain and extend the code over time.

