JAVA Mini-Project Assignment: Ride Booking System

📝 Problem Statement:

You are tasked to develop a **Ride Booking Management System** using Java. You must apply **object-oriented principles** such as **abstraction, inheritance, polymorphism, exception handling, annotations, reflection, file handling, and streams**.
Follow the detailed class and method guidelines below.

---

1. User Class

Create an **abstract class** named `User`. This class should have two protected fields: `id` (String) and `name` (String).
You must define a **constructor** `User(String id, String name)` that initializes these fields.
Also, declare an **abstract method** named `showProfile()` which should not accept any parameters and should have a void return type.

---

2. Customer Class

Create a class named `Customer` that **extends** `User`.
Implement a **constructor** `Customer(String id, String name)` that calls the superclass constructor.
Override the `showProfile()` method to print a message displaying the customer's name.

---

3. Driver Class

Create a class named `Driver` that **extends** `User`.
Introduce a private field `available` of type boolean, initialized to `true`.
Implement a **constructor** `Driver(String id, String name)` to initialize id, name, and set available to true.
Create a **method** `isAvailable()` that returns a boolean indicating driver availability.

Create another **method** `setAvailable(boolean available)` to update the driver's availability status.
Override the `showProfile()` method to display the driver's name along with their availability.

---

4. Ride Class

Create a class called `Ride` that should have three private fields: a `Customer` object, a `Driver` object, and a `status` (String).
Implement a **constructor** `Ride(Customer customer, Driver driver)` that sets the customer, driver, and initializes the status as "Booked".
Create a **method** `completeRide()` which should set the status to "Completed" and make the driver available again.
Create a **method** `String getStatus()` that returns the current ride status.
Also, implement a **method** `String rideDetails()` which should return a formatted string showing customer name, driver name, and current status.

---

5. InvalidRideException Class

Create a **custom exception class** `InvalidRideException` that **extends RuntimeException**.
Implement a **constructor** `InvalidRideException(String message)` which simply calls the superclass constructor with the given message.

---

6. SecurityCheck Annotation

Define a **custom annotation** `SecurityCheck`.
It must have `@Retention(RetentionPolicy.RUNTIME)` and `@Target(ElementType.TYPE)`.
Inside the annotation, define a field `String role()` which will specify the security role required.

---

7. Admin Class

Create a class `Admin` that **extends** `User` and is annotated with `@SecurityCheck(role = "Admin")`.

Implement a **constructor** `Admin(String id, String name)` to initialize the admin's id and name using the superclass.
Override the `showProfile()` method to display the admin's name and role.
Implement a **method** `removeDriver(List<Driver> drivers, String driverId)` which removes a driver from the list based on ID.
Inside this method, you must use **reflection** to check if the Admin class has the `SecurityCheck` annotation and whether the role matches "Admin".
Use an **Iterator** to safely remove the driver while iterating the list.

---

8. RideBookingSystem Class

Create a class named `RideBookingSystem` which will maintain three lists: `List<Customer> customers`, `List<Driver> drivers`, and `List<Ride> rides`.
Provide the following methods:

- `void registerCustomer(Customer customer)`: Adds a customer to the customers list.
- `void registerDriver(Driver driver)`: Adds a driver to the drivers list.
- `Ride bookRide(Customer customer)`: Finds an available driver using **Stream API with filter()**, books the ride by setting driver as unavailable, and adds the ride to the rides list. If no driver is available, throw `InvalidRideException`.
- `void saveRides()`: Saves the ride details into a file called "rides.txt" (one line per ride).
- `void loadDriversFromFile()`: Loads drivers from an external file, reads id and name, and registers them.
- `void saveDriversToFile()`: Saves the registered drivers into a file.
- `void showAllDrivers()`: Displays all drivers by calling each driver's `showProfile()` method.
- Provide **getter and setter** methods for the customers list as needed.

All file operations should use **BufferedReader** and **BufferedWriter** inside try-with-resources blocks.

---

9. Main Class

Create the `Main` class containing the `public static void main(String[] args)` method.
Inside main, create an instance of `RideBookingSystem` and immediately call `loadDriversFromFile()` to load existing drivers.
Display a menu with options:

- Register a Customer
- Register a Driver
- Book a Ride
- Show All Drivers
- Save Data and Exit

Based on user input, call appropriate methods.
When booking a ride, accept customer ID input, search for the customer in the list using **Stream API and filter()**, and then proceed.
If an invalid customer ID is entered, display an appropriate message.
Before exiting, save all rides and drivers to their respective files.

Use **try-catch blocks** wherever necessary, especially when catching `InvalidRideException`.