**Assignment: Implement a Ride-Sharing System in Java**

**Objective: Develop a system to manage a ride-sharing service where users can book rides, drivers can accept or cancel bookings, and both users and drivers can view ongoing and completed rides. This assignment will focus on Java OOP, multithreading, and exception handling in a real-time application.**

**System Requirements:**

1. **Ride Class**
   * **Attributes:**
     + **rideId (int): Unique identifier for each ride.**
     + **passengerName (String): Name of the passenger.**
     + **driverName (String): Name of the assigned driver.**
     + **pickupLocation (String): Location where the passenger will be picked up.**
     + **dropLocation (String): Destination location.**
     + **status (String): Current status of the ride (e.g., "Requested," "Accepted," "In Progress," "Completed," "Cancelled").**
   * **Override Methods:**
     + **Override toString() to display ride details.**
     + **Override equals() and hashCode() to manage rides by rideId.**
2. **Driver Class**
   * **Attributes:**
     + **driverId (int): Unique identifier for each driver.**
     + **name (String): Name of the driver.**
     + **isAvailable (boolean): Indicates whether the driver is available for new rides.**
     + **currentRide (Ride): The ride currently being handled by the driver (if any).**
3. **RideSharingManager Class**
   * **Manages a collection of drivers and rides using HashMap<Integer, Driver> and HashMap<Integer, Ride>.**
   * **Core Operations:**
     + **Request Ride: Allow a passenger to request a ride. If no drivers are available, throw a NoDriverAvailableException.**
     + **Accept Ride: Allow an available driver to accept a requested ride. If the ride has already been accepted by another driver or does not exist, throw a RideNotAvailableException.**
     + **Cancel Ride: Allow a driver or passenger to cancel a ride by rideId. If the ride is already completed or canceled, throw a RideNotCancelableException.**
     + **View Active Rides: Display all rides that are currently "In Progress."**
4. **Custom Exception Handling**
   * **Define custom exceptions:**
     + **NoDriverAvailableException: Thrown when no drivers are available to accept a ride request.**
     + **RideNotAvailableException: Thrown when a requested ride has already been accepted or is unavailable.**
     + **RideNotCancelableException: Thrown when an attempt is made to cancel a ride that is already completed or canceled.**
5. **Multithreading**
   * **Implement separate Runnable classes for each core operation (RequestRide, AcceptRide, CancelRide, ViewActiveRides).**
   * **Each operation should be executed in its own thread. Use join() in the main thread to wait for each operation’s completion.**
6. **Synchronization**
   * **Synchronize access to shared resources, especially when multiple drivers are trying to accept the same ride request simultaneously, to avoid conflicts.**

**Bonus Tasks:**

1. **Rate Driver and Passenger: After a completed ride, allow both the driver and passenger to rate each other.**
2. **Search Rides by Status: Enable viewing rides based on status (e.g., "Requested," "In Progress").**
3. **Earnings Calculation: Calculate total earnings for each driver based on completed rides.**