1. Library Management System

Java Rigorous Test (LibraryRigorousTest.java)

```
import java.util.ArrayList;
import java.util.List;

public class LibraryRigorousTest {
    public static void main(String[] args) {
        // Create books (including one unavailable)
        Book book1 = new Book("The Great Gatsby", "F. Scott Fitzgerald", "ISBN001",
        Book book2 = new Book("1984", "George Orwell", "ISBN002", true);
        Book book3 = new Book("Moby Dick", "Herman Melville", "ISBN003", false);

        Library library = new Library(new ArrayList<>());

        // Test adding books
        library.addBook(book1);
        library.addBook(book2);
        library.addBook(book3);
        if (library.listAllBooks().size() != 3) {
            System.out.println("Error: Not all books added.");
        }

        // Test listing all books
        System.out.println("Listing all books:");
        for (Book b : library.listAllBooks()) {
            System.out.println("Title: " + b.getTitle() + ", Available: " + b.isAvai
        }

        // Test search method (existing and non-existing)
        Book found = library.searchBookByTitle("1984");
        System.out.println("Found '1984': " + (found != null));
        Book notFound = library.searchBookByTitle("Nonexistent");
        System.out.println("Search for nonexistent title: " + (notFound == null));

        // Test removal (successful and failure)
        boolean removedSuccess = library.removeBook("ISBN002");
        System.out.println("Removed ISBN002: " + removedSuccess);
        boolean removedFailure = library.removeBook("ISBN999");
        System.out.println("Attempted to remove non-existent ISBN: " + removedFailur

        // Final listing check
        System.out.println("Books remaining:");
        for (Book b : library.listAllBooks()) {
            System.out.println(b.getTitle());
        }
    }
```

```
}
```

Python Rigorous Test (library_rigorous_test.py)

```
def main():
    # Create books (including one unavailable)
    book1 = Book("The Great Gatsby", "F. Scott Fitzgerald", "ISBN001", True)
    book2 = Book("1984", "George Orwell", "ISBN002", True)
    book3 = Book("Moby Dick", "Herman Melville", "ISBN003", False)

    library = Library([])

    # Test adding books
    library.addBook(book1)
    library.addBook(book2)
    library.addBook(book3)
    if len(library.listAllBooks()) != 3:
        print("Error: Not all books added.")

    # Test listing all books
    print("Listing all books:")
    for b in library.listAllBooks():
        print(f"Title: {b.getTitle()}, Available: {b.isAvailable()}")

    # Test search method (existing and non-existing)
    found = library.searchBookByTitle("1984")
    print("Found '1984':", found is not None)
    not_found = library.searchBookByTitle("Nonexistent")
    print("Search for nonexistent title:", not_found is None)

    # Test removal (successful and failure)
    removed_success = library.removeBook("ISBN002")
    print("Removed ISBN002:", removed_success)
    removed_failure = library.removeBook("ISBN999")
    print("Attempted removal of non-existent ISBN:", removed_failure)

    # Final listing check
    print("Books remaining:")
    for b in library.listAllBooks():
        print(b.getTitle())

if __name__ == '__main__':
    main()
```

2. Parking Lot Management

Java Rigorous Test (ParkingLotRigorousTest.java)

```java
import java.util.ArrayList;

public class ParkingLotRigorousTest {
    public static void main(String[] args) {
        // Create car objects
        Car car1 = new Car("ABC123", "Toyota Camry", "10:00 AM");
        Car car2 = new Car("XYZ789", "Honda Accord", "10:15 AM");
        Car car3 = new Car("LMN456", "Ford Focus", "10:30 AM");

        ParkingLot lot = new ParkingLot(new ArrayList<>());

        // Test parking multiple cars
        lot.parkCar(car1);
        lot.parkCar(car2);
        lot.parkCar(car3);
        if (lot.displayCars().size() != 3) {
            System.out.println("Error: Not all cars parked.");
        }

        // Validate each car's license plate and model
        System.out.println("Parked cars:");
        for (Car c : lot.displayCars()) {
            System.out.println(c.getLicensePlate() + " - " + c.getModel());
        }

        // Test removing an existing car
        boolean removed = lot.removeCar("ABC123");
        System.out.println("Car ABC123 removed: " + removed);

        // Attempt to remove non-existent car
        boolean removedNonExistent = lot.removeCar("ZZZ999");
        System.out.println("Non-existent car removal: " + removedNonExistent);

        // Final state of parked cars
        System.out.println("Remaining cars:");
        for (Car c : lot.displayCars()) {
            System.out.println(c.getLicensePlate());
        }
    }
}
```

Python Rigorous Test (parking_lot_rigorous_test.py)

```python
def main():
    # Create car objects
    car1 = Car("ABC123", "Toyota Camry", "10:00 AM")
    car2 = Car("XYZ789", "Honda Accord", "10:15 AM")
```

```
    car3 = Car("LMN456", "Ford Focus", "10:30 AM")

    lot = ParkingLot([])

    # Test parking multiple cars
    lot.parkCar(car1)
    lot.parkCar(car2)
    lot.parkCar(car3)
    if len(lot.displayCars()) != 3:
        print("Error: Not all cars parked.")

    # Validate each car's license plate and model
    print("Parked cars:")
    for c in lot.displayCars():
        print(f"{c.getLicensePlate()} - {c.getModel()}")

    # Test removing an existing car
    removed = lot.removeCar("ABC123")
    print("Car ABC123 removed:", removed)

    # Attempt to remove non-existent car
    removed_nonexistent = lot.removeCar("ZZZ999")
    print("Non-existent car removal:", removed_nonexistent)

    # Final state of parked cars
    print("Remaining cars:")
    for c in lot.displayCars():
        print(c.getLicensePlate())

if __name__ == '__main__':
    main()
```

3. Student Course Enrollment

Java Rigorous Test (EnrollmentRigorousTest.java)

```
import java.util.ArrayList;
import java.util.List;

public class EnrollmentRigorousTest {
    public static void main(String[] args) {
        // Create students
        Student student1 = new Student(1, "Alice", new ArrayList<>());
        Student student2 = new Student(2, "Bob", new ArrayList<>());

        // Create a course with a maximum enrollment of 1 (to test capacity limits)
        Course course = new Course("CS101", "Intro to CS", 1, 0);
```

```
        // Create EnrollmentManager with students and course
        List<Student> students = new ArrayList<>();
        students.add(student1);
        students.add(student2);
        List<Course> courses = new ArrayList<>();
        courses.add(course);
        EnrollmentManager em = new EnrollmentManager(students, courses);

        // Test enrolling first student (should succeed)
        boolean enroll1 = em.enrollStudent(1, "CS101");
        System.out.println("Alice enrolled in CS101: " + enroll1);

        // Test enrolling second student (should fail due to capacity)
        boolean enroll2 = em.enrollStudent(2, "CS101");
        System.out.println("Bob enrolled in CS101 (should fail): " + enroll2);

        // List students in CS101
        System.out.println("Students in CS101:");
        for (Student s : em.listStudentsInCourse("CS101")) {
            System.out.println(s.getName());
        }

        // Additional: Check student's course list
        System.out.println("Alice's enrolled courses: " + student1.getCourses());
        System.out.println("Bob's enrolled courses: " + student2.getCourses());
    }
}
```

Python Rigorous Test (enrollment_rigorous_test.py)

```
def main():
    # Create students
    student1 = Student(1, "Alice", [])
    student2 = Student(2, "Bob", [])

    # Create a course with maximum enrollment 1 to test capacity limits
    course = Course("CS101", "Intro to CS", 1, 0)

    # Create EnrollmentManager with students and the course
    em = EnrollmentManager([student1, student2], [course])

    # Test enrolling first student (should succeed)
    enroll1 = em.enrollStudent(1, "CS101")
    print("Alice enrolled in CS101:", enroll1)

    # Test enrolling second student (should fail due to capacity)
    enroll2 = em.enrollStudent(2, "CS101")
    print("Bob enrolled in CS101 (should fail):", enroll2)
```

```
    # List students in CS101
    print("Students in CS101:")
    for s in em.listStudentsInCourse("CS101"):
        print(s.name)

    # Additional: Check student's enrolled courses
    print("Alice's courses:", student1.getCourses())
    print("Bob's courses:", student2.getCourses())

if __name__ == '__main__':
    main()
```

4. Bank Customer and Account Management

Java Rigorous Test (BankRigorousTest.java)

```
import java.util.ArrayList;
import java.util.List;

public class BankRigorousTest {
    public static void main(String[] args) {
        // Create a BankAccount and test deposit/withdrawal edge cases
        BankAccount account = new BankAccount("ACC001", 1000.0, "savings");
        account.deposit(500.0);
        if (account.getBalance() != 1500.0) {
            System.out.println("Error in deposit calculation.");
        }
        // Test valid withdrawal
        boolean successWithdraw = account.withdraw(300.0);
        System.out.println("Withdrawal of 300 successful: " + successWithdraw);
        // Test invalid withdrawal (exceeding balance)
        boolean failWithdraw = account.withdraw(2000.0);
        System.out.println("Withdrawal of 2000 (should fail): " + failWithdraw);

        // Create a Customer and add accounts
        Customer customer = new Customer(1, "John Doe", new ArrayList<>());
        customer.addAccount(account);
        BankAccount retrieved = customer.getAccount("ACC001");
        System.out.println("Retrieved account balance: " + retrieved.getBalance());

        // Check non-existing account retrieval
        BankAccount nonExist = customer.getAccount("ACC999");
        System.out.println("Non-existent account retrieval: " + (nonExist == null));
    }
}
```

Python Rigorous Test (bank_rigorous_test.py)

```
def main():
    # Create a BankAccount and test deposit/withdrawal
    account = BankAccount("ACC001", 1000.0, "savings")
    account.deposit(500.0)
    if account.getBalance() != 1500.0:
        print("Error: Incorrect balance after deposit.")
    # Test valid withdrawal
    success_withdraw = account.withdraw(300.0)
    print("Withdrawal of 300 successful:", success_withdraw)
    # Test invalid withdrawal (exceeding balance)
    fail_withdraw = account.withdraw(2000.0)
    print("Withdrawal of 2000 (should fail):", fail_withdraw)

    # Create a Customer and add the account
    customer = Customer(1, "John Doe", [])
    customer.addAccount(account)
    retrieved = customer.getAccount("ACC001")
    print("Retrieved account balance:", retrieved.getBalance())

    # Test retrieval of a non-existing account
    non_exist = customer.getAccount("ACC999")
    print("Non-existent account retrieval:", non_exist is None)

if __name__ == '__main__':
    main()
```

5. Product Inventory Management

Java Rigorous Test (InventoryRigorousTest.java)

```
import java.util.ArrayList;

public class InventoryRigorousTest {
    public static void main(String[] args) {
        // Create products
        Product prod1 = new Product(1, "Laptop", 1500.0, 10);
        Product prod2 = new Product(2, "Smartphone", 800.0, 20);

        Inventory inventory = new Inventory(new ArrayList<>());

        // Test adding products
        inventory.addProduct(prod1);
        inventory.addProduct(prod2);
        if (inventory.listProducts().size() != 2) {
```

```
            System.out.println("Error: Products not added correctly.");
        }

        // Test updating stock: add and subtract quantity
        boolean updateSuccess = inventory.updateProductStock(1, 5);
        System.out.println("Updated stock for product 1 (add 5): " + updateSuccess);
        boolean updateFailure = inventory.updateProductStock(3, 5);
        System.out.println("Attempt update for non-existent product: " + updateFailu

        // Test listing products and printing product info
        System.out.println("Products in inventory:");
        for (Product p : inventory.listProducts()) {
            System.out.println(p.getProductInfo());
        }
    }
}
```

Python Rigorous Test (inventory_rigorous_test.py)

```
def main():
    # Create products
    prod1 = Product(1, "Laptop", 1500.0, 10)
    prod2 = Product(2, "Smartphone", 800.0, 20)

    inventory = Inventory([])

    # Test adding products
    inventory.addProduct(prod1)
    inventory.addProduct(prod2)
    if len(inventory.listProducts()) != 2:
        print("Error: Products not added correctly.")

    # Test updating stock
    update_success = inventory.updateProductStock(1, 5)
    print("Updated stock for product 1 (add 5):", update_success)
    update_failure = inventory.updateProductStock(3, 5)
    print("Attempt update for non-existent product:", update_failure)

    # Test listing products
    print("Products in inventory:")
    for p in inventory.listProducts():
        print(p.getProductInfo())

if __name__ == '__main__':
    main()
```

6. Movie Ticket Booking System

Java Rigorous Test (MovieTicketRigorousTest.java)

```java
import java.util.ArrayList;

public class MovieTicketRigorousTest {
    public static void main(String[] args) {
        // Create a movie and multiple tickets
        Movie movie = new Movie(1, "Inception", 148);
        Ticket ticket1 = new Ticket(101, movie.getMovieID(), "A10", 12.5);
        Ticket ticket2 = new Ticket(102, movie.getMovieID(), "A11", 12.5);

        Booking booking = new Booking(new ArrayList<>());

        // Test booking tickets
        booking.bookTicket(ticket1);
        booking.bookTicket(ticket2);
        if (booking.getBookingDetails().size() != 2) {
            System.out.println("Error: Tickets not booked correctly.");
        }

        // Test ticket cancellation: valid and invalid cancellation
        boolean cancelValid = booking.cancelTicket(101);
        System.out.println("Ticket 101 cancelled: " + cancelValid);
        boolean cancelInvalid = booking.cancelTicket(999);
        System.out.println("Attempt cancellation of non-existent ticket: " + cancelI

        // Final booking details
        System.out.println("Remaining tickets in booking:");
        for (Ticket t : booking.getBookingDetails()) {
            System.out.println(t.getTicketInfo());
        }
    }
}
```

Python Rigorous Test (movie_ticket_rigorous_test.py)

```python
def main():
    # Create a movie and multiple tickets
    movie = Movie(1, "Inception", 148)
    ticket1 = Ticket(101, movie.movieID, "A10", 12.5)
    ticket2 = Ticket(102, movie.movieID, "A11", 12.5)

    booking = Booking([])

    # Test booking tickets
    booking.bookTicket(ticket1)
```

```
    booking.bookTicket(ticket2)
    if len(booking.getBookingDetails()) != 2:
        print("Error: Tickets not booked correctly.")

    # Test ticket cancellation
    cancel_valid = booking.cancelTicket(101)
    print("Ticket 101 cancelled:", cancel_valid)
    cancel_invalid = booking.cancelTicket(999)
    print("Attempt cancellation of non-existent ticket:", cancel_invalid)

    # Final booking details
    print("Remaining tickets:")
    for t in booking.getBookingDetails():
        print(t.getTicketInfo())

if __name__ == '__main__':
    main()
```

7. Employee Payroll System

Java Rigorous Test (PayrollRigorousTest.java)

```
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

public class PayrollRigorousTest {
    public static void main(String[] args) {
        // Create employees with varying hourly rates
        Employee emp1 = new Employee(1, "Alice", 20.0);
        Employee emp2 = new Employee(2, "Bob", 25.0);
        Employee emp3 = new Employee(3, "Charlie", 30.0);

        Payroll payroll = new Payroll(Arrays.asList(emp1, emp2, emp3));

        // Create hours worked map (including an edge case: zero hours)
        Map<Integer, Double> hoursMap = new HashMap<>();
        hoursMap.put(1, 40.0);
        hoursMap.put(2, 35.0);
        hoursMap.put(3, 0.0);

        Map<Integer, Double> payResults = payroll.processPayroll(hoursMap);

        // Validate the calculated pay values
        System.out.println("Payroll results:");
        for (Map.Entry<Integer, Double> entry : payResults.entrySet()) {
            System.out.println("Employee ID " + entry.getKey() + " earned: " + entry
```

```
        }
    }
}
```

Python Rigorous Test (payroll_rigorous_test.py)

```
def main():
    # Create employees
    emp1 = Employee(1, "Alice", 20.0)
    emp2 = Employee(2, "Bob", 25.0)
    emp3 = Employee(3, "Charlie", 30.0)

    payroll = Payroll([emp1, emp2, emp3])

    # Hours mapping including an edge case with zero hours
    hours_map = {1: 40.0, 2: 35.0, 3: 0.0}
    pay_results = payroll.processPayroll(hours_map)

    print("Payroll results:")
    for emp_id, pay in pay_results.items():
        print(f"Employee ID {emp_id} earned: {pay}")

if __name__ == '__main__':
    main()
```

8. Flight Reservation System

Java Rigorous Test (FlightReservationRigorousTest.java)

```
import java.util.ArrayList;
import java.util.Arrays;

public class FlightReservationRigorousTest {
    public static void main(String[] args) {
        // Create a flight with limited seats
        Flight flight = new Flight("FL123", "New York", "London", 2);

        // Reserve seats until full
        System.out.println("Seat reservation 1: " + flight.reserveSeat());
        System.out.println("Seat reservation 2: " + flight.reserveSeat());
        System.out.println("Seat reservation 3 (should fail): " + flight.reserveSeat

        // Create reservations for a flight
        Reservation reservation1 = new Reservation(1, flight.getFlightNumber(), "Joh
        Reservation reservation2 = new Reservation(2, flight.getFlightNumber(), "Jar
```

```
        // Create ReservationManager and add flight
        ReservationManager rm = new ReservationManager(new ArrayList<>(Arrays.asList
        rm.makeReservation(flight.getFlightNumber(), "John Smith");
        rm.makeReservation(flight.getFlightNumber(), "Jane Doe");

        // List and then cancel a reservation
        System.out.println("Reservation details for ID 1: " + reservation1.getReserv
        boolean cancelResult = rm.cancelReservation(1);
        System.out.println("Reservation 1 cancelled: " + cancelResult);
    }
}
```

Python Rigorous Test (flight_reservation_rigorous_test.py)

```
def main():
    # Create a flight with limited seats
    flight = Flight("FL123", "New York", "London", 2)

    # Reserve seats until flight is full
    print("Seat reservation 1:", flight.reserveSeat())
    print("Seat reservation 2:", flight.reserveSeat())
    print("Seat reservation 3 (should fail):", flight.reserveSeat())

    # Create reservations
    reservation1 = Reservation(1, flight.flightNumber, "John Smith")
    reservation2 = Reservation(2, flight.flightNumber, "Jane Doe")

    # Create ReservationManager and add the flight
    rm = ReservationManager([flight], [])
    rm.makeReservation(flight.flightNumber, "John Smith")
    rm.makeReservation(flight.flightNumber, "Jane Doe")

    # Display and cancel reservation
    print("Reservation details for ID 1:", reservation1.getReservationDetails())
    cancelled = rm.cancelReservation(1)
    print("Reservation 1 cancelled:", cancelled)

if __name__ == '__main__':
    main()
```

9. Restaurant Order Management

Java Rigorous Test (RestaurantOrderRigorousTest.java)

```
import java.util.ArrayList;

public class RestaurantOrderRigorousTest {
    public static void main(String[] args) {
        // Create menu items
        MenuItem item1 = new MenuItem(1, "Burger", 8.5);
        MenuItem item2 = new MenuItem(2, "Fries", 3.0);
        MenuItem item3 = new MenuItem(3, "Soda", 2.0);

        // Create an order and add items
        Order order = new Order(101, new ArrayList<>(), 5);
        order.addItem(item1);
        order.addItem(item2);
        order.addItem(item3);
        double total = order.calculateTotal();
        System.out.println("Calculated total: " + total);

        // Test removal of an item
        boolean removeItem = order.removeItem(2);
        System.out.println("Item with ID 2 removed: " + removeItem);
        System.out.println("New total after removal: " + order.calculateTotal());

        // Create OrderManager and test order retrieval and cancellation
        OrderManager om = new OrderManager(new ArrayList<>());
        om.createOrder(order);
        Order retrievedOrder = om.getOrder(101);
        System.out.println("Retrieved order for table " + retrievedOrder.getTableNum
        boolean cancelOrder = om.cancelOrder(101);
        System.out.println("Order cancellation status: " + cancelOrder);
    }
}
```

Python Rigorous Test (restaurant_order_rigorous_test.py)

```
def main():
    # Create menu items
    item1 = MenuItem(1, "Burger", 8.5)
    item2 = MenuItem(2, "Fries", 3.0)
    item3 = MenuItem(3, "Soda", 2.0)

    # Create an order and add items
    order = Order(101, [], 5)
    order.addItem(item1)
    order.addItem(item2)
    order.addItem(item3)
    total = order.calculateTotal()
    print("Calculated total:", total)

    # Remove an item and recalc total
```

```
    removed = order.removeItem(2)
    print("Item 2 removed:", removed)
    print("New total after removal:", order.calculateTotal())

    # Test OrderManager functionality
    om = OrderManager([])
    om.createOrder(order)
    retrieved_order = om.getOrder(101)
    print("Retrieved order for table", retrieved_order.tableNumber)
    cancelled = om.cancelOrder(101)
    print("Order cancellation status:", cancelled)

if __name__ == '__main__':
    main()
```

10. Hospital Patient Management

Java Rigorous Test (HospitalRigorousTest.java)

```
import java.util.ArrayList;

public class HospitalRigorousTest {
    public static void main(String[] args) {
        // Create a patient and two appointments
        Patient patient = new Patient(1, "Emma", 30);
        Appointment app1 = new Appointment(101, patient.getPatientID(), "Dr. Brown",
        Appointment app2 = new Appointment(102, patient.getPatientID(), "Dr. White",

        PatientManager pm = new PatientManager(new ArrayList<>(), new ArrayList<>())
        pm.getPatients().add(patient);
        pm.getAppointments().add(app1);
        pm.getAppointments().add(app2);

        // Test listing appointments for patient
        System.out.println("Appointments for Emma:");
        for (Appointment app : pm.listAppointmentsForPatient(1)) {
            System.out.println(app.getAppointmentDetails());
        }

        // Cancel an appointment and re-check
        boolean cancelStatus = pm.cancelAppointment(101);
        System.out.println("Appointment 101 cancelled: " + cancelStatus);
        System.out.println("Remaining appointments for Emma:");
        for (Appointment app : pm.listAppointmentsForPatient(1)) {
            System.out.println(app.getAppointmentDetails());
        }
    }
```

```
}
```

Python Rigorous Test (hospital_rigorous_test.py)

```
def main():
    # Create a patient and two appointments
    patient = Patient(1, "Emma", 30)
    app1 = Appointment(101, patient.patientID, "Dr. Brown", "2:00 PM")
    app2 = Appointment(102, patient.patientID, "Dr. White", "3:00 PM")

    pm = PatientManager([], [])
    pm.patients.append(patient)
    pm.appointments.extend([app1, app2])

    # List appointments for patient
    print("Appointments for Emma:")
    for app in pm.listAppointmentsForPatient(1):
        print(app.getAppointmentDetails())

    # Cancel an appointment and check again
    cancelled = pm.cancelAppointment(101)
    print("Appointment 101 cancelled:", cancelled)
    print("Remaining appointments for Emma:")
    for app in pm.listAppointmentsForPatient(1):
        print(app.getAppointmentDetails())

if __name__ == '__main__':
    main()
```

11. Gym Membership System

Java Rigorous Test (GymRigorousTest.java)

```
import java.util.ArrayList;

public class GymRigorousTest {
    public static void main(String[] args) {
        // Create members and membership plans
        Member member1 = new Member(1, "David", "monthly");
        Member member2 = new Member(2, "Linda", "yearly");
        MembershipPlan plan1 = new MembershipPlan(101, "Standard", 50.0);
        MembershipPlan plan2 = new MembershipPlan(102, "Premium", 80.0);

        Gym gym = new Gym(new ArrayList<>(), new ArrayList<>());
        gym.registerMember(member1);
```

```
        gym.registerMember(member2);

        // Test assigning plans (assume assignPlan uses IDs to match a plan to a mem
        boolean assign1 = gym.assignPlan(member1.getMemberID(), plan1.getPlanID());
        boolean assign2 = gym.assignPlan(member2.getMemberID(), plan2.getPlanID());
        System.out.println("Plan assigned to David: " + assign1);
        System.out.println("Plan assigned to Linda: " + assign2);

        // Attempt assigning plan to non-existent member
        boolean assignInvalid = gym.assignPlan(999, plan1.getPlanID());
        System.out.println("Plan assignment to non-existent member: " + assignInvali
    }
}
```

Python Rigorous Test (gym_rigorous_test.py)

```
def main():
    # Create members and membership plans
    member1 = Member(1, "David", "monthly")
    member2 = Member(2, "Linda", "yearly")
    plan1 = MembershipPlan(101, "Standard", 50.0)
    plan2 = MembershipPlan(102, "Premium", 80.0)

    gym = Gym([], [])
    gym.registerMember(member1)
    gym.registerMember(member2)

    # Test assigning membership plans
    assign1 = gym.assignPlan(member1.memberID, plan1.planID)
    assign2 = gym.assignPlan(member2.memberID, plan2.planID)
    print("Plan assigned to David:", assign1)
    print("Plan assigned to Linda:", assign2)

    # Attempt assignment for a non-existent member
    assign_invalid = gym.assignPlan(999, plan1.planID)
    print("Plan assignment to non-existent member:", assign_invalid)

if __name__ == '__main__':
    main()
```

12. Hotel Room Booking

Java Rigorous Test (HotelBookingRigorousTest.java)

```
import java.util.ArrayList;
```

```
public class HotelBookingRigorousTest {
    public static void main(String[] args) {
        // Create rooms
        Room room1 = new Room(101, "double", false);
        Room room2 = new Room(102, "single", false);

        // Book room1 successfully, then try to book again
        boolean booked1 = room1.bookRoom();
        boolean bookedAgain = room1.bookRoom();
        System.out.println("Room 101 first booking: " + booked1);
        System.out.println("Room 101 second booking (should fail): " + bookedAgain);

        // Create bookings
        Booking booking1 = new Booking(1, room1.getRoomNumber(), "Laura", 3);
        Booking booking2 = new Booking(2, room2.getRoomNumber(), "Mark", 2);

        Hotel hotel = new Hotel(new ArrayList<>(), new ArrayList<>());
        hotel.getRooms().add(room1);
        hotel.getRooms().add(room2);
        hotel.getBookings().add(booking1);
        hotel.getBookings().add(booking2);

        // List available rooms (only room2 should be available if room1 is booked)
        System.out.println("Available rooms:");
        for (Room r : hotel.listAvailableRooms()) {
            System.out.println("Room " + r.getRoomNumber());
        }

        // Cancel a booking and then check available rooms again
        boolean cancelBooking = hotel.cancelBooking(1);
        System.out.println("Booking 1 cancelled: " + cancelBooking);
        System.out.println("Available rooms after cancellation:");
        for (Room r : hotel.listAvailableRooms()) {
            System.out.println("Room " + r.getRoomNumber());
        }
    }
}
```

Python Rigorous Test (hotel_booking_rigorous_test.py)

```
def main():
    # Create rooms
    room1 = Room(101, "double", False)
    room2 = Room(102, "single", False)

    # Book room1 and test booking again
    booked1 = room1.bookRoom()
    booked_again = room1.bookRoom()
    print("Room 101 first booking:", booked1)
```

```
    print("Room 101 second booking (should fail):", booked_again)

    # Create bookings
    booking1 = Booking(1, room1.roomNumber, "Laura", 3)
    booking2 = Booking(2, room2.roomNumber, "Mark", 2)

    hotel = Hotel([], [])
    hotel.rooms.extend([room1, room2])
    hotel.bookings.extend([booking1, booking2])

    # List available rooms (room2 should be available if room1 booked)
    print("Available rooms:")
    for r in hotel.listAvailableRooms():
        print("Room", r.roomNumber)

    # Cancel a booking and check available rooms again
    cancelled = hotel.cancelBooking(1)
    print("Booking 1 cancelled:", cancelled)
    print("Available rooms after cancellation:")
    for r in hotel.listAvailableRooms():
        print("Room", r.roomNumber)

if __name__ == '__main__':
    main()
```

13. Online Shopping Cart

Java Rigorous Test (ShoppingCartRigorousTest.java)

```
import java.util.ArrayList;

public class ShoppingCartRigorousTest {
    public static void main(String[] args) {
        // Create several products
        Product prod1 = new Product(1, "Headphones", 100.0);
        Product prod2 = new Product(2, "Keyboard", 75.0);
        Product prod3 = new Product(3, "Mouse", 40.0);

        // Create shopping cart and add products
        ShoppingCart cart = new ShoppingCart(new ArrayList<>());
        cart.addItem(prod1);
        cart.addItem(prod2);
        cart.addItem(prod3);
        double total = cart.calculateTotal();
        System.out.println("Total cart price: " + total);

        // Remove a product and recalc
```

```
        boolean removed = cart.removeItem(2);
        System.out.println("Product 2 removed: " + removed);
        System.out.println("New total: " + cart.calculateTotal());

        // Attempt to remove a product not in the cart
        boolean removeInvalid = cart.removeItem(999);
        System.out.println("Removal of non-existent product: " + removeInvalid);
    }
}
```

Python Rigorous Test (shopping_cart_rigorous_test.py)

```
def main():
    # Create products
    prod1 = Product(1, "Headphones", 100.0)
    prod2 = Product(2, "Keyboard", 75.0)
    prod3 = Product(3, "Mouse", 40.0)

    # Create shopping cart and add products
    cart = ShoppingCart([])
    cart.addItem(prod1)
    cart.addItem(prod2)
    cart.addItem(prod3)
    total = cart.calculateTotal()
    print("Total cart price:", total)

    # Remove a product and recalc
    removed = cart.removeItem(2)
    print("Product 2 removed:", removed)
    print("New total:", cart.calculateTotal())

    # Attempt to remove non-existent product
    removed_invalid = cart.removeItem(999)
    print("Attempted removal of non-existent product:", removed_invalid)

if __name__ == '__main__':
    main()
```

14. University Admission System

Java Rigorous Test (AdmissionRigorousTest.java)

```
import java.util.ArrayList;

public class AdmissionRigorousTest {
```

```
    public static void main(String[] args) {
        // Create applicants and applications
        Applicant applicant1 = new Applicant(1, "Sara", 92.5);
        Applicant applicant2 = new Applicant(2, "Tom", 85.0);
        Application application1 = new Application(101, 1, "Computer Science");
        Application application2 = new Application(102, 2, "Mathematics");

        AdmissionOffice office = new AdmissionOffice(new ArrayList<>(), new ArrayLis
        office.getApplicants().add(applicant1);
        office.getApplicants().add(applicant2);
        office.getApplications().add(application1);
        office.getApplications().add(application2);

        // Submit an application (even if already added, simulate submission)
        office.submitApplication(application1);
        System.out.println("Application submitted for " + applicant1.getName());

        // Review applications for a valid and an invalid applicant ID
        String review1 = office.reviewApplication(1);
        String reviewInvalid = office.reviewApplication(999);
        System.out.println("Review outcome for applicant 1: " + review1);
        System.out.println("Review outcome for non-existent applicant: " + reviewInv
    }
}
```

Python Rigorous Test (admission_rigorous_test.py)

```
def main():
    # Create applicants and applications
    applicant1 = Applicant(1, "Sara", 92.5)
    applicant2 = Applicant(2, "Tom", 85.0)
    application1 = Application(101, 1, "Computer Science")
    application2 = Application(102, 2, "Mathematics")

    office = AdmissionOffice([], [])
    office.applicants.extend([applicant1, applicant2])
    office.applications.extend([application1, application2])

    # Submit an application
    office.submitApplication(application1)
    print("Application submitted for", applicant1.name)

    # Review applications
    review1 = office.reviewApplication(1)
    review_invalid = office.reviewApplication(999)
    print("Review outcome for applicant 1:", review1)
    print("Review outcome for non-existent applicant:", review_invalid)

if __name__ == '__main__':
    main()
```

15. Event Management System

Java Rigorous Test (EventManagementRigorousTest.java)

```
import java.util.ArrayList;

public class EventManagementRigorousTest {
    public static void main(String[] args) {
        // Create an event with capacity 2
        Event event = new Event(1, "Tech Conference", "Hall A", 2);

        // Create participants
        Participant participant1 = new Participant(101, "Mike", "mike@example.com");
        Participant participant2 = new Participant(102, "Anna", "anna@example.com");
        Participant participant3 = new Participant(103, "John", "john@example.com");

        EventManager em = new EventManager(new ArrayList<>(), new ArrayList<>());
        em.getEvents().add(event);

        // Register participants (first two should succeed; third should fail if cap
        boolean reg1 = em.registerParticipant(1, participant1);
        boolean reg2 = em.registerParticipant(1, participant2);
```

```
        boolean reg3 = em.registerParticipant(1, participant3);
        System.out.println("Participant 101 registered: " + reg1);
        System.out.println("Participant 102 registered: " + reg2);
        System.out.println("Participant 103 registered (should fail): " + reg3);

        // List participants
        System.out.println("Participants for event 1:");
        for (Participant p : em.listParticipants(1)) {
            System.out.println(p.getName());
        }
    }
}
```

Python Rigorous Test (event_management_rigorous_test.py)

```
def main():
    # Create an event with capacity 2
    event = Event(1, "Tech Conference", "Hall A", 2)

    # Create participants
    participant1 = Participant(101, "Mike", "mike@example.com")
    participant2 = Participant(102, "Anna", "anna@example.com")
    participant3 = Participant(103, "John", "john@example.com")

    em = EventManager([], [])
    em.events.append(event)

    # Register participants
    reg1 = em.registerParticipant(1, participant1)
    reg2 = em.registerParticipant(1, participant2)
    reg3 = em.registerParticipant(1, participant3)
    print("Participant 101 registered:", reg1)
    print("Participant 102 registered:", reg2)
    print("Participant 103 registered (should fail):", reg3)

    # List participants
    print("Participants for event 1:")
    for p in em.listParticipants(1):
        print(p.name)

if __name__ == '__main__':
    main()
```

16. Vehicle Rental System

Java Rigorous Test (VehicleRentalRigorousTest.java)

```java
import java.util.ArrayList;
import java.util.Arrays;

public class VehicleRentalRigorousTest {
    public static void main(String[] args) {
        // Create a vehicle and test rental/return functionality
        Vehicle vehicle = new Vehicle(1, "Sedan", 40.0, false);
        System.out.println("Vehicle rented (should be true): " + vehicle.rentVehicle
        System.out.println("Vehicle rented again (should be false): " + vehicle.rent
        vehicle.returnVehicle();
        System.out.println("Vehicle available after return: " + (!vehicle.isRented()

        // Create rental records
        Rental rental = new Rental(1, vehicle.getVehicleID(), "Anna", 3);
        RentalService service = new RentalService(new ArrayList<>(Arrays.asList(vehi
        boolean createRental = service.createRental(rental);
        System.out.println("Rental created: " + createRental);

        // End the rental and check vehicle availability
        boolean endRental = service.endRental(1);
        System.out.println("Rental ended: " + endRental);
        System.out.println("Vehicle available after ending rental: " + (!vehicle.isR
    }
}
```

Python Rigorous Test (vehicle_rental_rigorous_test.py)

```python
def main():
    # Create a vehicle and test rental and return
    vehicle = Vehicle(1, "Sedan", 40.0, False)
    print("Vehicle rented (should be True):", vehicle.rentVehicle())
    print("Vehicle rented again (should be False):", vehicle.rentVehicle())
    vehicle.returnVehicle()
    print("Vehicle available after return:", not vehicle.isRented)

    # Create a rental record and service
    rental = Rental(1, vehicle.vehicleID, "Anna", 3)
    service = RentalService([vehicle], [])
    created = service.createRental(rental)
    print("Rental created:", created)

    # End the rental and verify
    ended = service.endRental(1)
    print("Rental ended:", ended)
    print("Vehicle available after ending rental:", not vehicle.isRented)

if __name__ == '__main__':
    main()
```

17. E-commerce Review System

Java Rigorous Test (ReviewSystemRigorousTest.java)

```java
import java.util.ArrayList;

public class ReviewSystemRigorousTest {
    public static void main(String[] args) {
        // Create reviews for a product
        Review review1 = new Review(1, 101, "Great product!", 5);
        Review review2 = new Review(2, 101, "Not bad", 4);
        Review review3 = new Review(3, 102, "Poor quality", 2);

        // Create product objects
        Product product1 = new Product(101, "Smart Watch", 199.99);
        Product product2 = new Product(102, "Fitness Tracker", 99.99);

        // Create ReviewManager and add reviews
        ReviewManager rm = new ReviewManager(new ArrayList<>());
        rm.addReview(review1);
        rm.addReview(review2);
        rm.addReview(review3);

        // Display review summaries for product1 and product2
        System.out.println("Reviews for product 101:");
        for (Review r : rm.getReviewsByProduct(101)) {
            System.out.println(r.getReviewSummary());
        }
        System.out.println("Reviews for product 102:");
        for (Review r : rm.getReviewsByProduct(102)) {
            System.out.println(r.getReviewSummary());
        }

        // Display product info
        System.out.println("Product 101 info: " + product1.getProductInfo());
        System.out.println("Product 102 info: " + product2.getProductInfo());
    }
}
```

Python Rigorous Test (review_system_rigorous_test.py)

```python
def main():
    # Create reviews
    review1 = Review(1, 101, "Great product!", 5)
    review2 = Review(2, 101, "Not bad", 4)
    review3 = Review(3, 102, "Poor quality", 2)
```

```
    # Create products
    product1 = Product(101, "Smart Watch", 199.99)
    product2 = Product(102, "Fitness Tracker", 99.99)

    # Create ReviewManager and add reviews
    rm = ReviewManager([])
    rm.addReview(review1)
    rm.addReview(review2)
    rm.addReview(review3)

    # Display reviews for product1 and product2
    print("Reviews for product 101:")
    for r in rm.getReviewsByProduct(101):
        print(r.getReviewSummary())
    print("Reviews for product 102:")
    for r in rm.getReviewsByProduct(102):
        print(r.getReviewSummary())

    # Display product info
    print("Product 101 info:", product1.getProductInfo())
    print("Product 102 info:", product2.getProductInfo())

if __name__ == '__main__':
    main()
```

18. Todo List Application

Java Rigorous Test (TodoListRigorousTest.java)

```
import java.util.ArrayList;

public class TodoListRigorousTest {
    public static void main(String[] args) {
        // Create tasks with varying statuses
        Task task1 = new Task(1, "Complete assignment", false);
        Task task2 = new Task(2, "Read chapter 5", false);
        Task task3 = new Task(3, "Go for a run", false);

        // Create TodoList and add tasks
        TodoList todo = new TodoList(new ArrayList<>());
        todo.addTask(task1);
        todo.addTask(task2);
        todo.addTask(task3);

        // Mark some tasks as completed
        task1.markCompleted();
```

```
        task3.markCompleted();

        // Check each task status
        System.out.println("Task 1 status: " + task1.getTaskStatus());
        System.out.println("Task 2 status: " + task2.getTaskStatus());
        System.out.println("Task 3 status: " + task3.getTaskStatus());

        // List all tasks
        System.out.println("All tasks:");
        for (Task t : todo.listAllTasks()) {
            System.out.println("Task " + t.getTaskID() + ": " + t.getTaskStatus());
        }

        // Remove a task and verify removal
        boolean removed = todo.removeTask(2);
        System.out.println("Task 2 removed: " + removed);
        System.out.println("Remaining tasks:");
        for (Task t : todo.listAllTasks()) {
            System.out.println("Task " + t.getTaskID() + ": " + t.getTaskStatus());
        }
    }
}
```

Python Rigorous Test (todo_list_rigorous_test.py)

```
def main():
    # Create tasks
    task1 = Task(1, "Complete assignment", False)
    task2 = Task(2, "Read chapter 5", False)
    task3 = Task(3, "Go for a run", False)

    # Create TodoList and add tasks
    todo = TodoList([])
    todo.addTask(task1)
    todo.addTask(task2)
    todo.addTask(task3)

    # Mark some tasks as completed
    task1.markCompleted()
    task3.markCompleted()

    # Check status of tasks
    print("Task 1 status:", task1.getTaskStatus())
    print("Task 2 status:", task2.getTaskStatus())
    print("Task 3 status:", task3.getTaskStatus())

    # List all tasks
    print("All tasks:")
    for t in todo.listAllTasks():
        print("Task", t.taskID, ":", t.getTaskStatus())
```

```
    # Remove a task and list remaining tasks
    removed = todo.removeTask(2)
    print("Task 2 removed:", removed)
    print("Remaining tasks:")
    for t in todo.listAllTasks():
        print("Task", t.taskID, ":", t.getTaskStatus())

if __name__ == '__main__':
    main()
```

19. Music Playlist Manager

Java Rigorous Test (PlaylistRigorousTest.java)

```
import java.util.ArrayList;

public class PlaylistRigorousTest {
    public static void main(String[] args) {
        // Create songs
        Song song1 = new Song(1, "Imagine", "John Lennon", 183);
        Song song2 = new Song(2, "Hey Jude", "The Beatles", 431);
        Song song3 = new Song(3, "Bohemian Rhapsody", "Queen", 354);

        // Create a playlist and add songs
        Playlist playlist = new Playlist(1, "Favorites", new ArrayList<>());
        playlist.addSong(song1);
        playlist.addSong(song2);
        playlist.addSong(song3);

        // Validate total duration calculation
        System.out.println("Total duration: " + playlist.getTotalDuration() + " secc

        // Remove a song and re-calculate duration
        boolean removed = playlist.removeSong(2);
        System.out.println("Song 2 removed: " + removed);
        System.out.println("Total duration after removal: " + playlist.getTotalDurat
    }
}
```

Python Rigorous Test (playlist_rigorous_test.py)

```
def main():
    # Create songs
    song1 = Song(1, "Imagine", "John Lennon", 183)
```

```
    song2 = Song(2, "Hey Jude", "The Beatles", 431)
    song3 = Song(3, "Bohemian Rhapsody", "Queen", 354)

    # Create a playlist and add songs
    playlist = Playlist(1, "Favorites", [])
    playlist.addSong(song1)
    playlist.addSong(song2)
    playlist.addSong(song3)

    # Validate total duration
    print("Total duration:", playlist.getTotalDuration(), "seconds")

    # Remove a song and recalc
    removed = playlist.removeSong(2)
    print("Song 2 removed:", removed)
    print("Total duration after removal:", playlist.getTotalDuration(), "seconds")

if __name__ == '__main__':
    main()
```

20. Banking Transaction Log

Java Rigorous Test (TransactionLogRigorousTest.java)

```
import java.util.ArrayList;

public class TransactionLogRigorousTest {
    public static void main(String[] args) {
        // Create a bank account and test transactions
        BankAccount account = new BankAccount("ACC001", 1000.0, "checking");

        // Create transactions: deposit and withdrawal
        Transaction deposit = new Transaction(1, "ACC001", 250.0, "deposit");
        Transaction withdrawal = new Transaction(2, "ACC001", 100.0, "withdrawal");
        Transaction invalidWithdrawal = new Transaction(3, "ACC001", 2000.0, "withdr

        // Process deposit
        boolean depositProcessed = account.performTransaction(deposit);
        System.out.println("Deposit processed: " + depositProcessed);
        System.out.println("Balance after deposit: " + account.getBalance());

        // Process valid withdrawal
        boolean withdrawalProcessed = account.performTransaction(withdrawal);
        System.out.println("Withdrawal processed: " + withdrawalProcessed);
        System.out.println("Balance after withdrawal: " + account.getBalance());

        // Process invalid withdrawal (should fail)
```

```
        boolean invalidProcessed = account.performTransaction(invalidWithdrawal);
        System.out.println("Invalid withdrawal processed (should be false): " + inva

        // List all transactions
        System.out.println("Transaction log:");
        for (Transaction t : account.listTransactions()) {
            System.out.println(t.getTransactionDetails());
        }
    }
}
```

Python Rigorous Test (transaction_log_rigorous_test.py)

```
def main():
    # Create a bank account
    account = BankAccount("ACC001", 1000.0, "checking")

    # Create transactions
    deposit = Transaction(1, "ACC001", 250.0, "deposit")
    withdrawal = Transaction(2, "ACC001", 100.0, "withdrawal")
    invalid_withdrawal = Transaction(3, "ACC001", 2000.0, "withdrawal")

    # Process deposit
    processed_deposit = account.performTransaction(deposit)
    print("Deposit processed:", processed_deposit)
    print("Balance after deposit:", account.getBalance())

    # Process valid withdrawal
    processed_withdrawal = account.performTransaction(withdrawal)
    print("Withdrawal processed:", processed_withdrawal)
    print("Balance after withdrawal:", account.getBalance())

    # Process invalid withdrawal (should fail)
    processed_invalid = account.performTransaction(invalid_withdrawal)
    print("Invalid withdrawal processed (should be False):", processed_invalid)

    # List transactions
    print("Transaction log:")
    for t in account.listTransactions():
        print(t.getTransactionDetails())

if __name__ == '__main__':
    main()
```