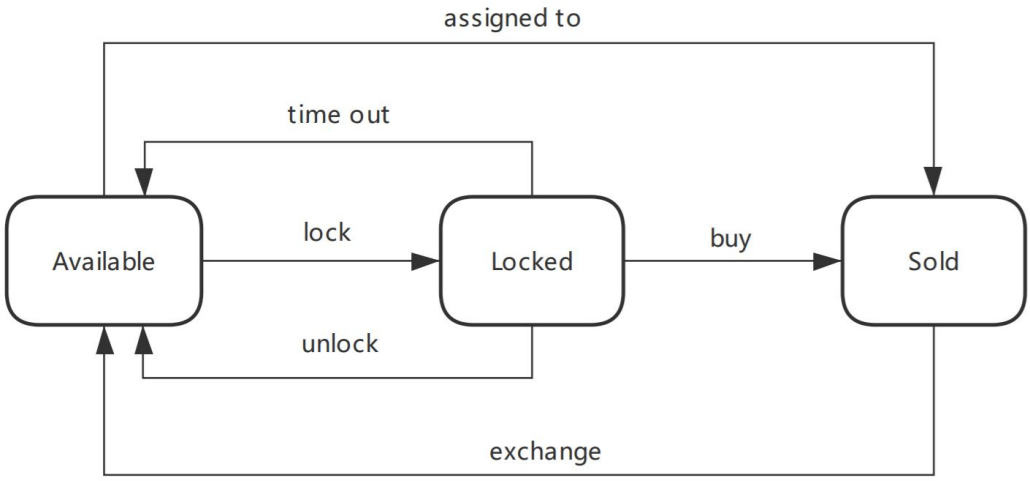
Object-Oriented Methodology HW 12

2024 Fall Semester

21 CST H3Art

Design Problem



Air Ticket State Diagram

Using the GoF State Pattern, write a complete Java program to simulate the Air Ticket System represented by the state diagram above.

The Java simulation code is as follows:

```java
public class AirTicketStatePattern {
    public static void main(String[] args) {
        Ticket ticket = new Ticket();

        // Locking the ticket
        ticket.lock();

        // Buying the ticket
        ticket.buy();

        // Trying to lock again after being sold
        ticket.lock();

        // Exchange the ticket
        ticket.exchange();

        // Lock the ticket again
        ticket.lock();
```

```
    }
}

interface State {
    void lock(Ticket ticket);

    void unlock(Ticket ticket);

    void buy(Ticket ticket);

    void timeout(Ticket ticket);

    void exchange(Ticket ticket);
}

class AvailableState implements State {
    @Override
    public void lock(Ticket ticket) {
        System.out.println("Ticket is now locked.");
        ticket.setState(new LockedState());
    }

    @Override
    public void unlock(Ticket ticket) {
        System.out.println("Ticket is already available.");
    }

    @Override
    public void buy(Ticket ticket) {
        System.out.println("Ticket has been bought.");
        ticket.setState(new SoldState());
    }

    @Override
    public void timeout(Ticket ticket) {
        System.out.println("Ticket is already available.");
    }

    @Override
    public void exchange(Ticket ticket) {
        System.out.println("Cannot exchange an available ticket.");
    }
}

class LockedState implements State {
    @Override
    public void lock(Ticket ticket) {
        System.out.println("Ticket is already locked.");
    }

    @Override
    public void unlock(Ticket ticket) {
```

```
        System.out.println("Ticket has been unlocked and is now available.");
        ticket.setState(new AvailableState());
    }

    @Override
    public void buy(Ticket ticket) {
        System.out.println("Ticket has been bought.");
        ticket.setState(new SoldState());
    }

    @Override
    public void timeout(Ticket ticket) {
        System.out.println("Lock timed out. Ticket is now available.");
        ticket.setState(new AvailableState());
    }

    @Override
    public void exchange(Ticket ticket) {
        System.out.println("Cannot exchange a locked ticket.");
    }
}

class SoldState implements State {
    @Override
    public void lock(Ticket ticket) {
        System.out.println("Cannot lock a sold ticket.");
    }

    @Override
    public void unlock(Ticket ticket) {
        System.out.println("Cannot unlock a sold ticket.");
    }

    @Override
    public void buy(Ticket ticket) {
        System.out.println("Ticket is already sold.");
    }

    @Override
    public void timeout(Ticket ticket) {
        System.out.println("Cannot timeout a sold ticket.");
    }

    @Override
    public void exchange(Ticket ticket) {
        System.out.println("Ticket exchange is processed.");
        ticket.setState(new AvailableState());
    }
}

class Ticket {
    private State state;
```

```
    public Ticket() {
        state = new AvailableState();
    }

    public void setState(State state) {
        this.state = state;
    }

    public void lock() {
        state.lock(this);
    }

    public void unlock() {
        state.unlock(this);
    }

    public void buy() {
        state.buy(this);
    }

    public void timeout() {
        state.timeout(this);
    }

    public void exchange() {
        state.exchange(this);
    }
}
```

The running result of above Java code: