The State Pattern

"Scheme"-like Lists

Overview

February 9, 2000
... {
    public class Cons extends AbstractList {

    ... {
    public class Empty extends AbstractList {

    protected abstract String toString();
    public abstract AbstractList cdr();
    public abstract List car();

    "Scheme¨-like Lists"

    See the handout for a complete implementation...
```java
{
    System.out.println(ans1 + anotherRevered); // What is printed?
}
```

```java
System.out.println(list[1]);
list = new Cons(13, list[1]); // Add "13" to the list.
list = new Cons(7, list[1]); // Add "7" to the list.
```

```java
Alice: list = new Cons(7, list[1]); // Create an empty list.
```

```java
} public static void main(String[] args)
{ public class Test
```

**Consider**

```
A Problem
```

February 9, 2000

Comp 212
The key is to encapsulate the states of a system as classes.

- That can change states dynamically.
- That have states and

View the list as a collection of objects.

A Solution
A General-Purpose Container

{ }
This instance represents the current state of the system.

Represent the system by a class containing an instance of a concrete state.

- Each concrete state must implement its own concrete methods.

Define a concrete subclass of the abstract class for each state of the system.

All of the concrete subclasses:
- This abstract state class should provide all the abstract methods for

The State Pattern
- Since this instance can change dynamically, the system will behave as
- Delegate all requests made to the system to the current state instance.
- Define methods for the system to return the current state and to change state.

The State Pattern (cont.)