

CS 213 – Software Methodology

Spring 2017

Sesh Venugopal

Lecture 18 – Mar 28

Design Patterns – 2

State and Singleton Patterns

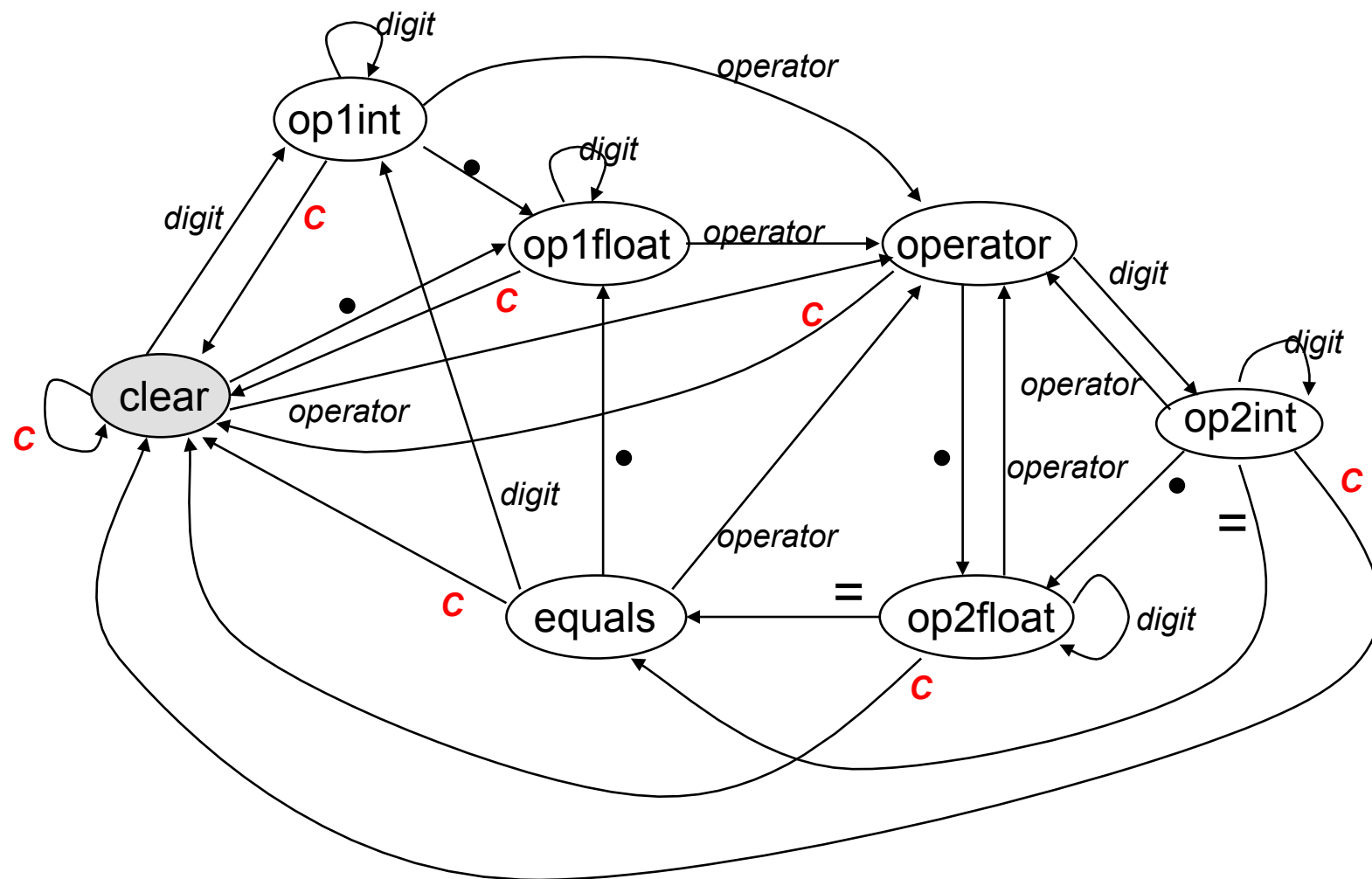
Illustration: State-Based Calculator

Building a Calculator - The State Design Pattern

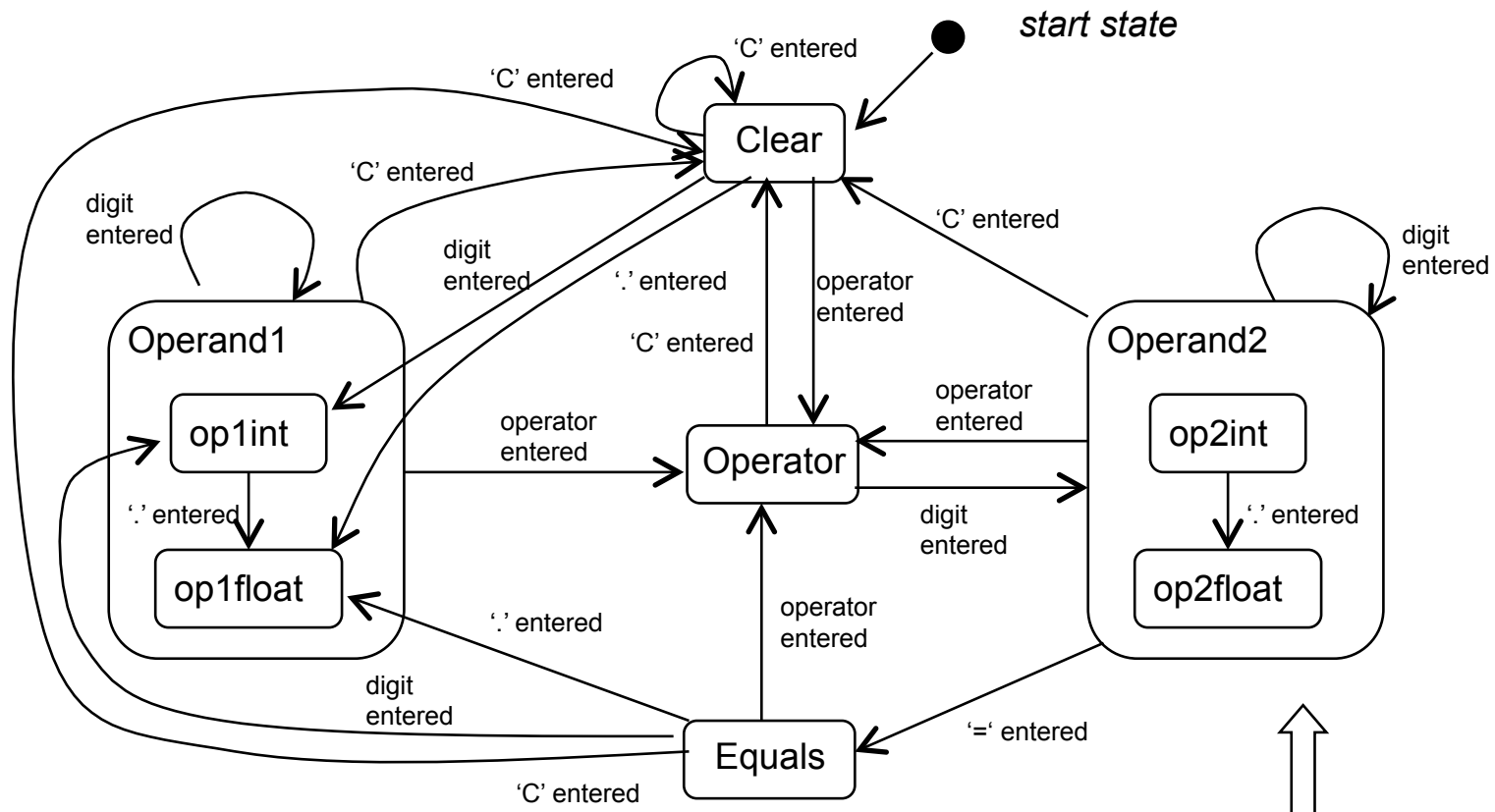
State Design Pattern

Allow an object to alter its behavior when the internal state changes.
The object will appear to change its class.

Calculator: State Diagram



Calculator: UML State(chart) Diagram



Every state is responsible for enabling and disabling buttons as appropriate



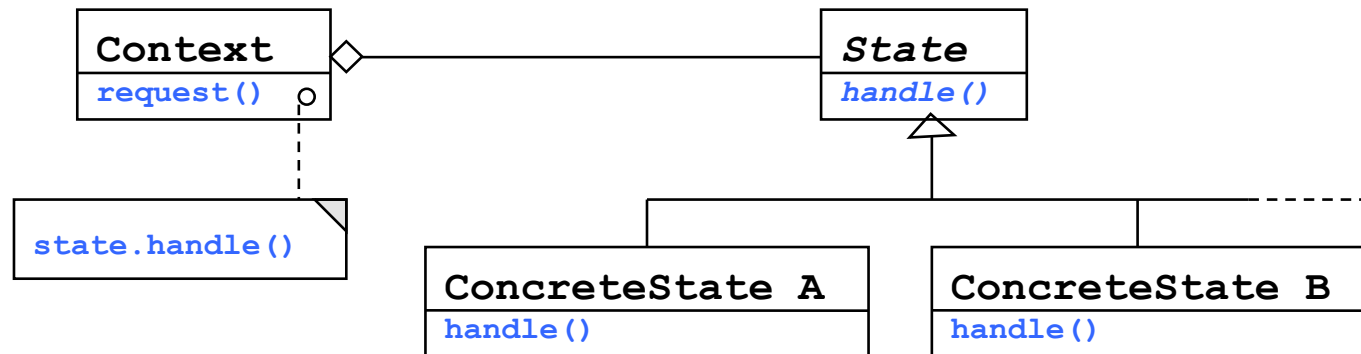
end state

(all states above can go to end state – transitions not shown because all transitions are same, and happen on exiting the application)

composite
state with
Internal sub states

State Design Pattern: Behavioral

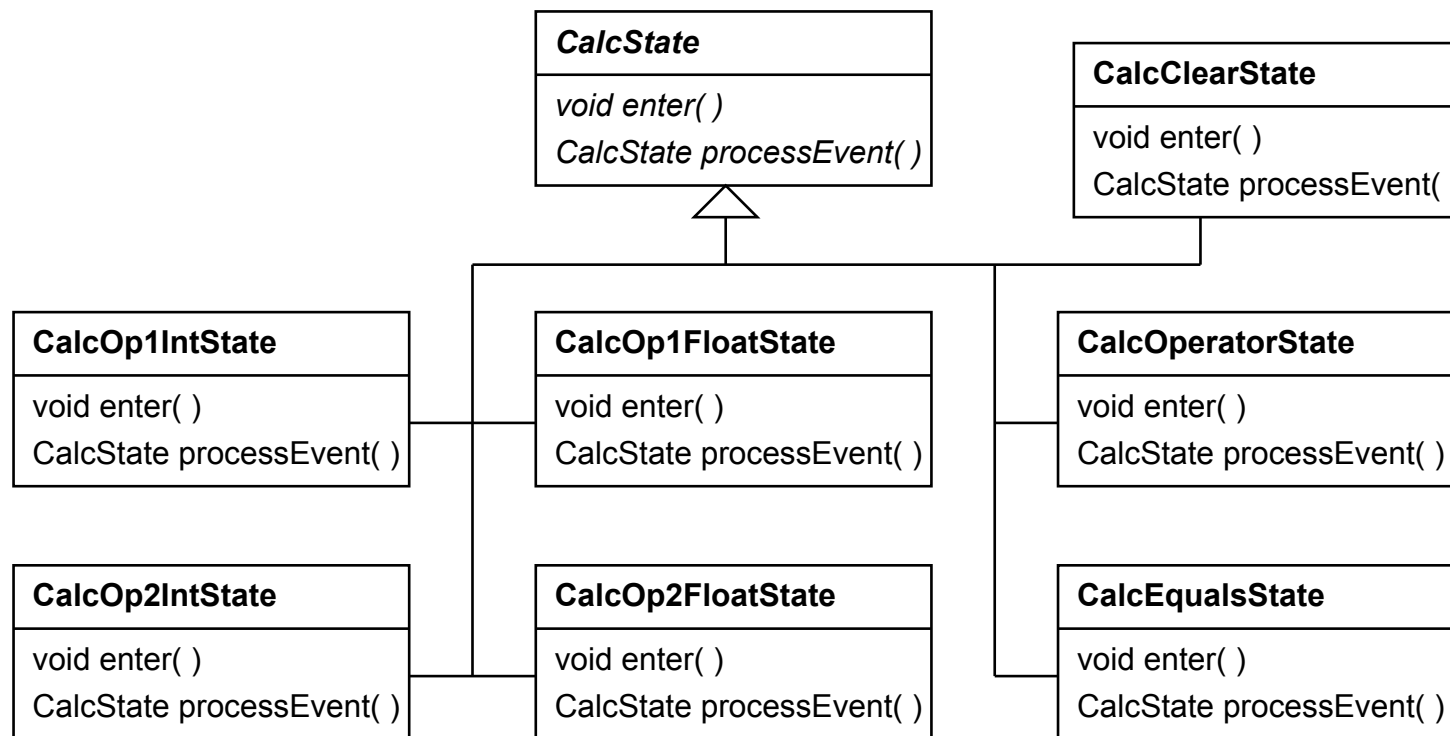
- Allow an object to change its behavior when its internal state changes – the “object” is a subclass of an abstract class, thus polymorphism



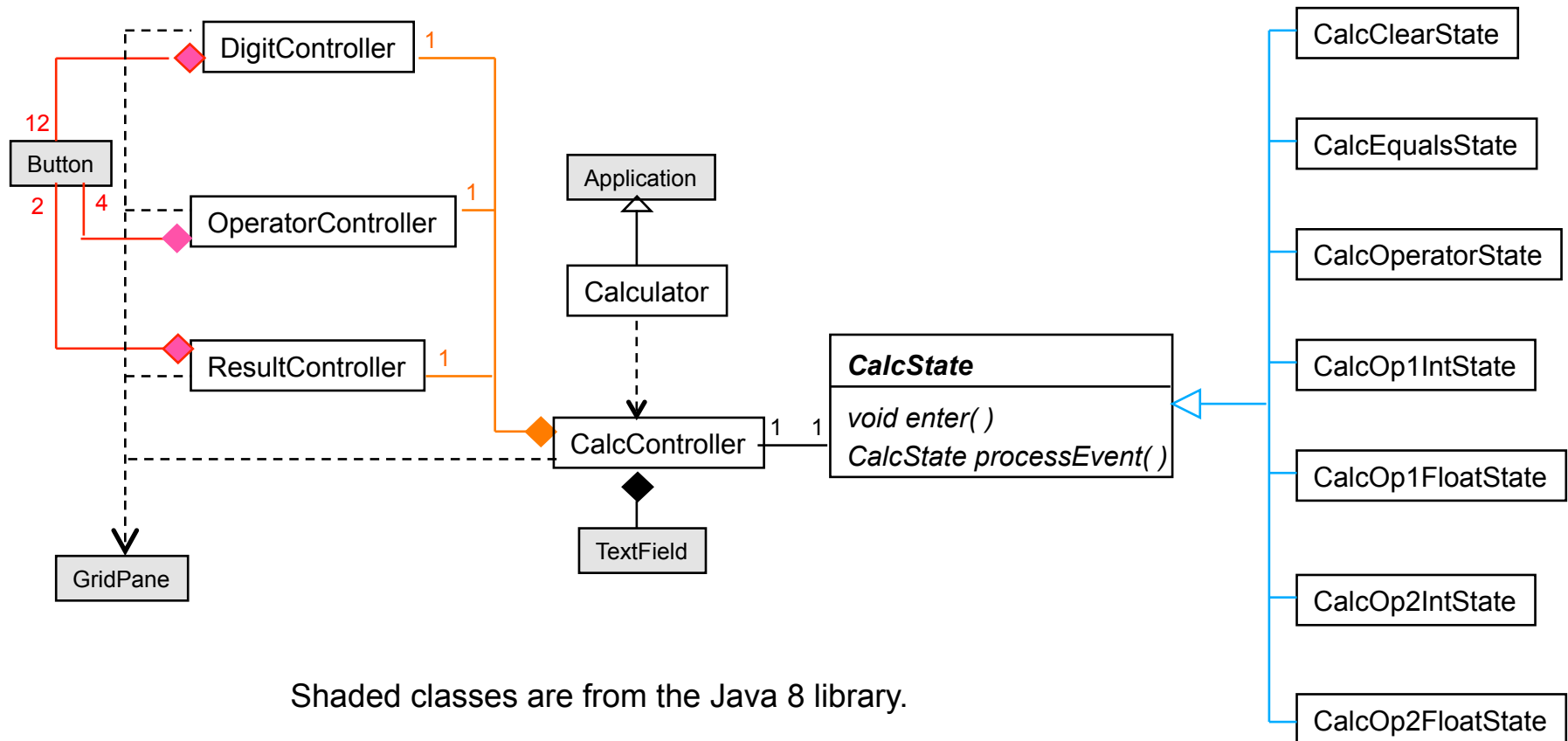
- Context (client code) has a state object that is one of the concrete instances: the request method executes `handle` on this concrete instance, dynamically binding the appropriate concrete class method – neat use of polymorphism
- Example: State classes `CalcState` (abstract) and `CalcClearState`, `CalcEqualsState`, etc. (concrete) in the state-based calculator application. The context is the `CalcController` class.

State Design Pattern: Applied to Calculator

- The general implementation of **State** pattern:
 - There is an abstract class that specifies state methods: in general these could be entry/body/exit methods
 - Subclasses of this abstract class define different specific states

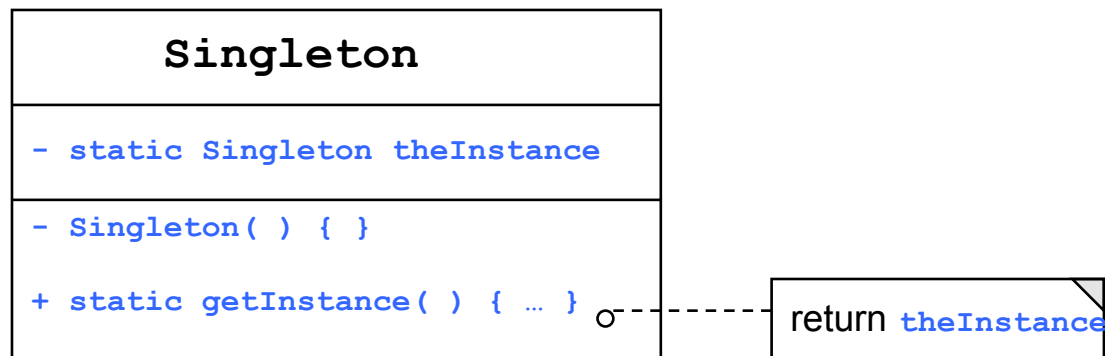


State-based Calculator – UML Class Diagram



Singleton Design Pattern: Creational

- Ensure that a class has only one object (instance) and provide a global point of access to this single instance



- The single private constructor ensures that an instance of Singleton cannot be created using `new`

Singleton Design Pattern: Applied to Calculator

- Each of the concrete state classes implements the [Singleton](#) pattern. For instance, the `calcClearState` class:

```
class calcClearState {  
  
    . . .  
    private static calcClearState instance = null;  
    . . .  
    private calcClearState() {  
  
    }  
  
    . . .  
    public static calcClearState getInstance() {  
        if (instance == null) {  
            instance = new calcClearState();  
        }  
        return instance;  
    }  
    . . .  
}
```