CMPE 202

Gang of Four Design Patterns

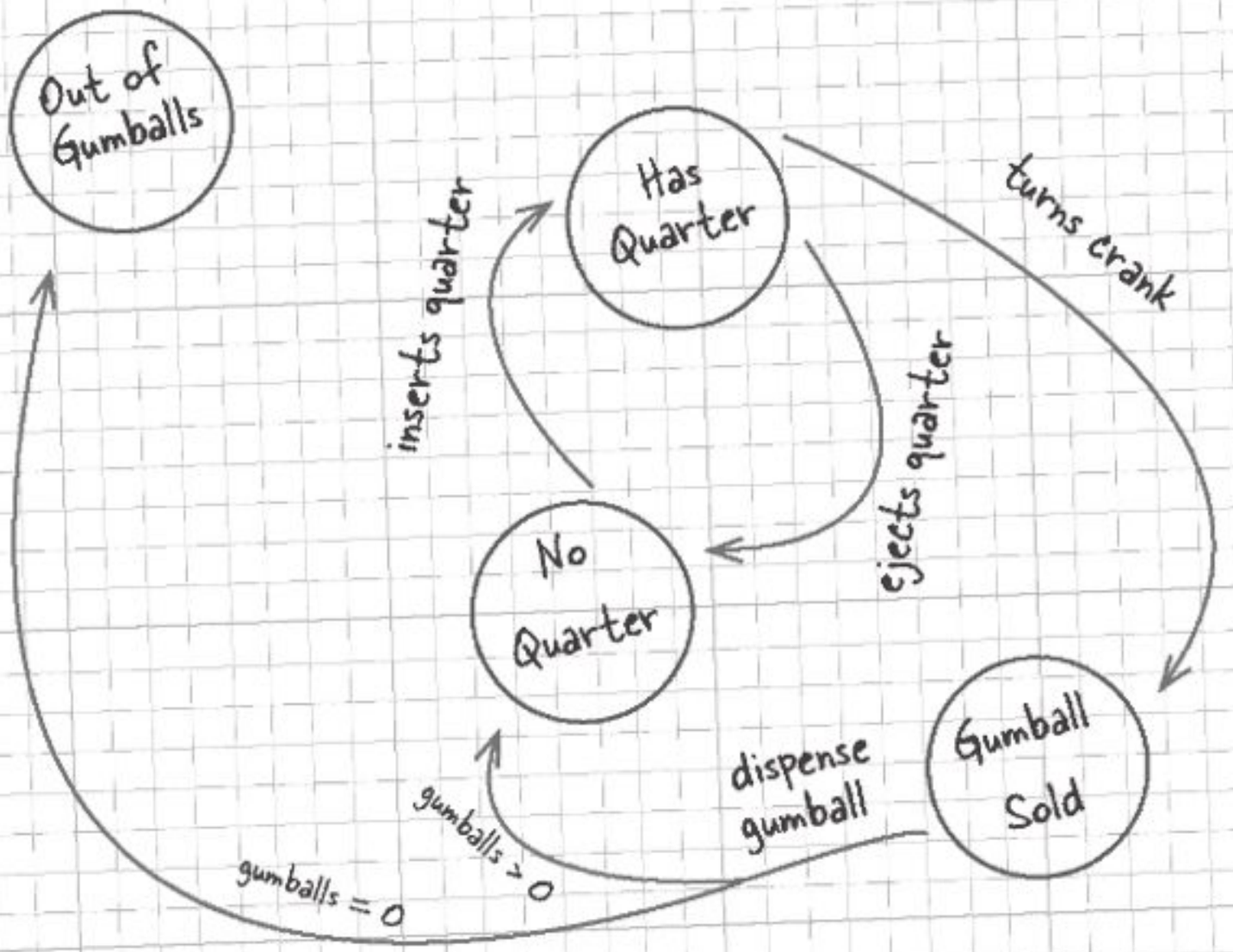
State

Mighty Gumball, Inc.

Where the Gumball Machine
is Never Half Empty

Here's the way we think the gumball machine controller needs to work. We're hoping you can implement this in Java for us! We may be adding more behavior in the future, so you need to keep the design as flexible and maintainable as possible!

– Mighty Gumball Engineers



Applicability

Use the State pattern in either of the following cases:

- An object's behavior depends on its state, and it must change its behavior at run-time depending on that state.

- Operations have large, multipart conditional statements that depend on the object's state. This state is usually represented by one or more enumerated constants. Often, several operations will contain this same conditional structure. The State pattern puts each branch of the conditional in a separate class. This lets you treat the object's state as an object in its own right that can vary independently from other objects.

Participants

- **Context** (Interface)
    - defines the interface of interest to clients.
    - maintains an instance of a ConcreteState subclass that defines the current state.

- **State** (Interface)
    - defines an interface for encapsulating the behavior associated with a particular state of the Context.

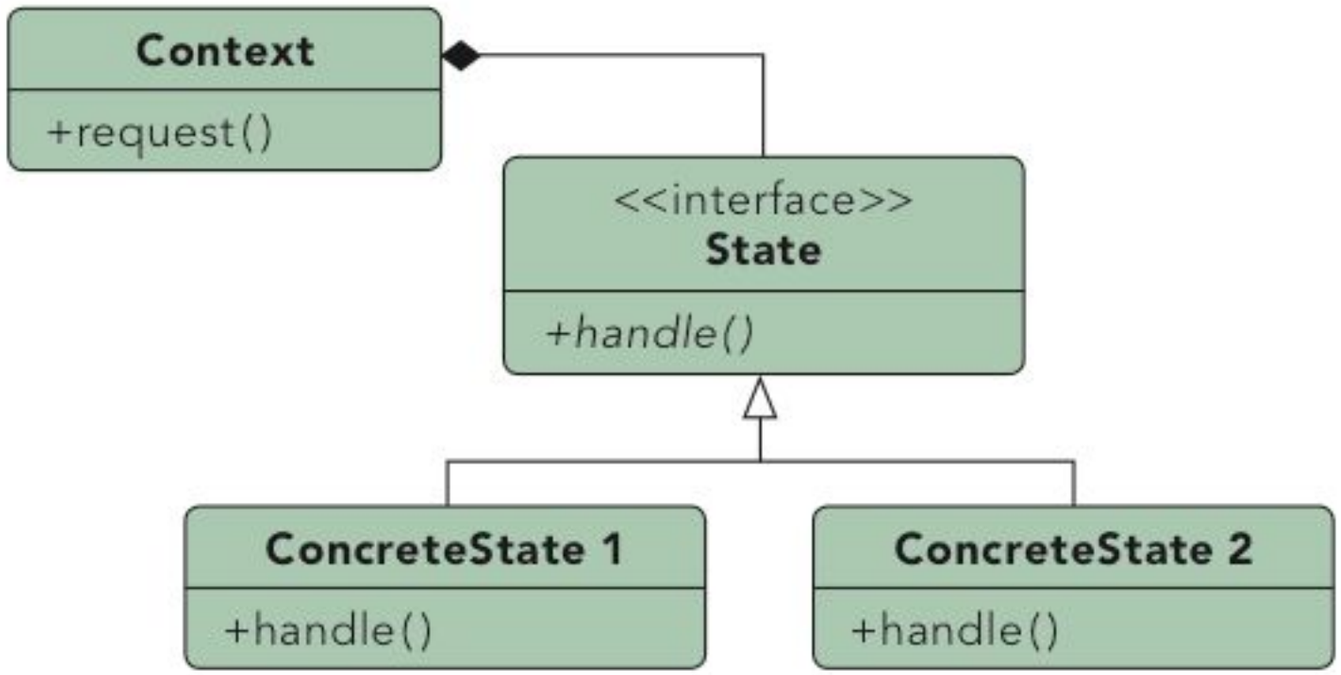
- **ConcreteState subclasses**
    - each subclass implements a behavior associated with a state of the Context.

Collaborations

- Context delegates state-specific requests to the current ConcreteState object.

STATE

Object Behavioral

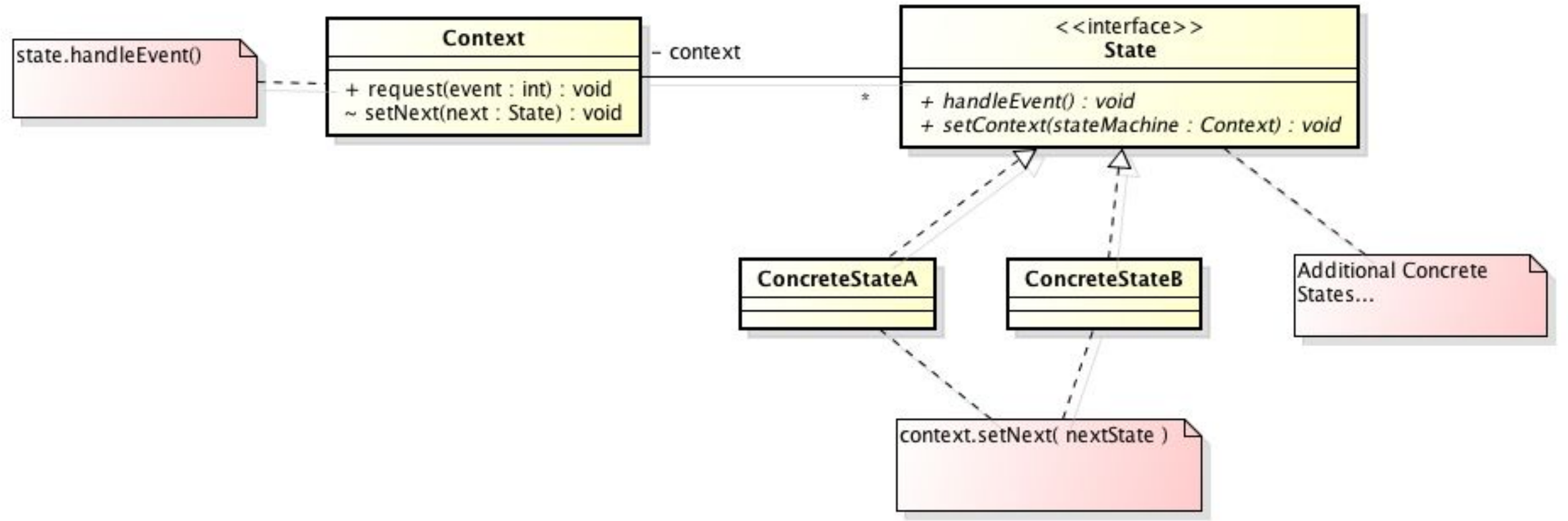


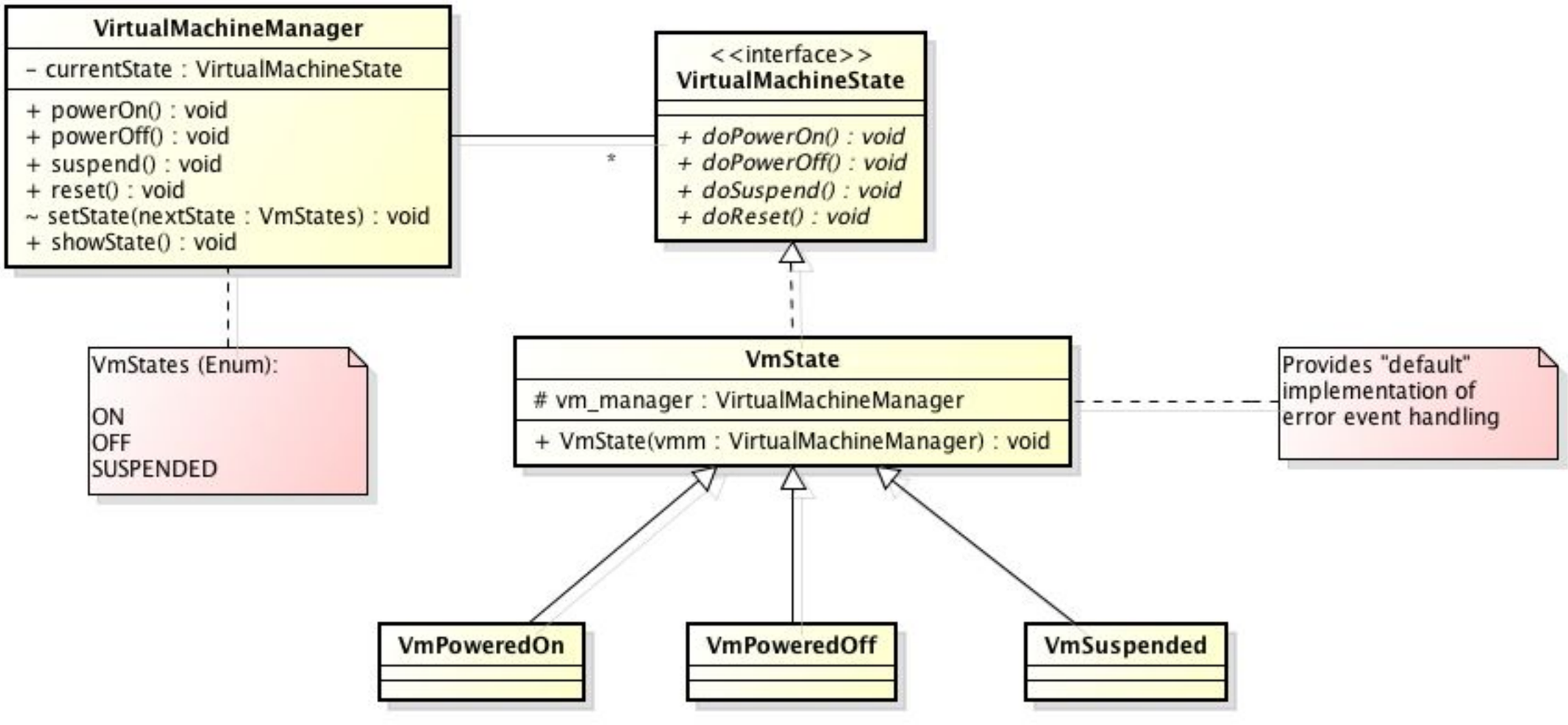
Purpose

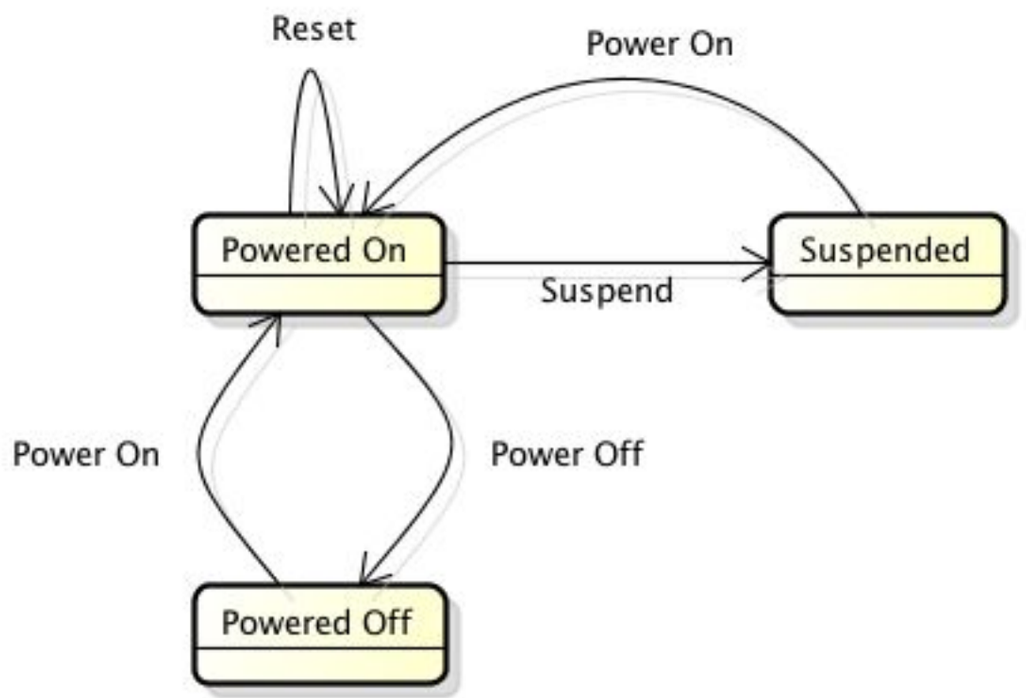
Ties object circumstances to its behavior, allowing the object to behave in different ways based upon its internal state.

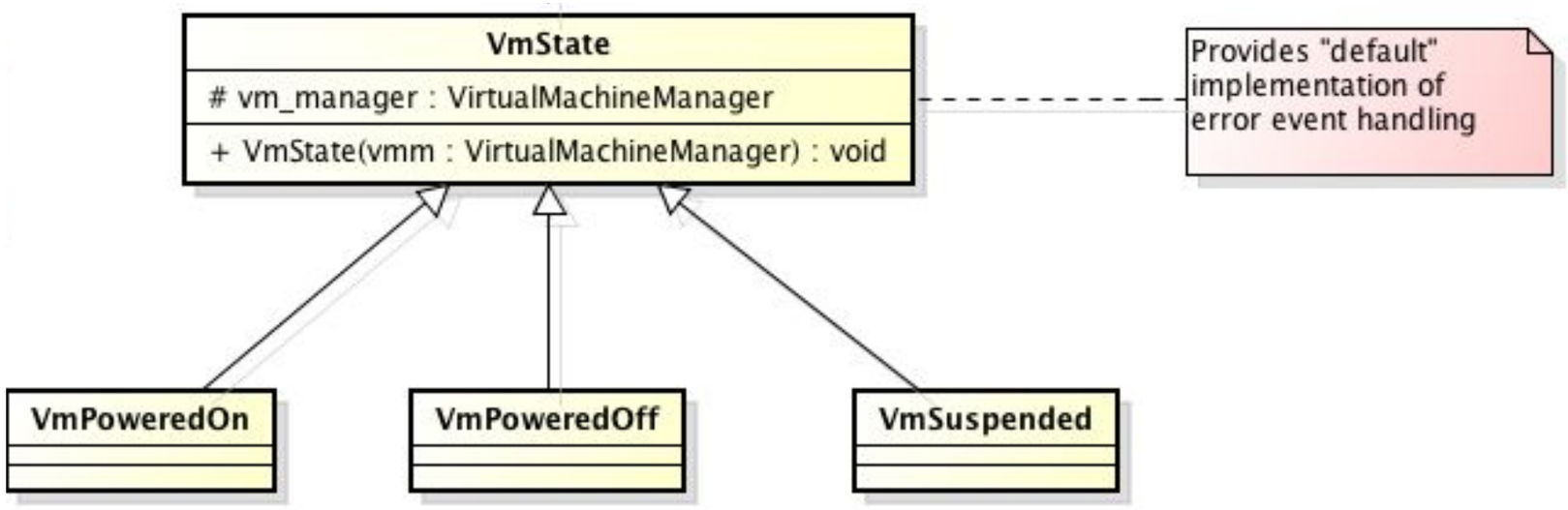
Use When

- The behavior of an object should be influenced by its state.
- Complex conditions tie object behavior to its state.
- Transitions between states need to be explicit.

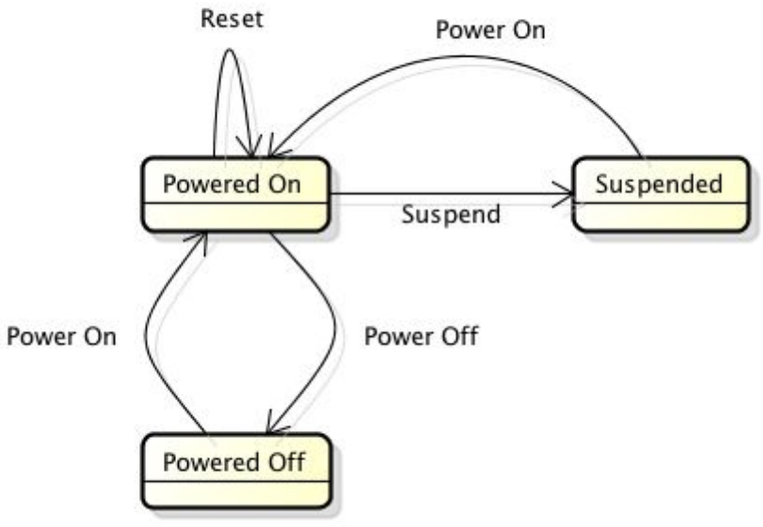

state.handleEvent()
Context
+ request(event : int) : void
~ setNext(next : State) : void
- context
*
<<interface>>
State
+ handleEvent() : void
+ setContext(stateMachine : Context) : void
ConcreteStateA
ConcreteStateB
Additional Concrete States...
context.setNext(nextState)

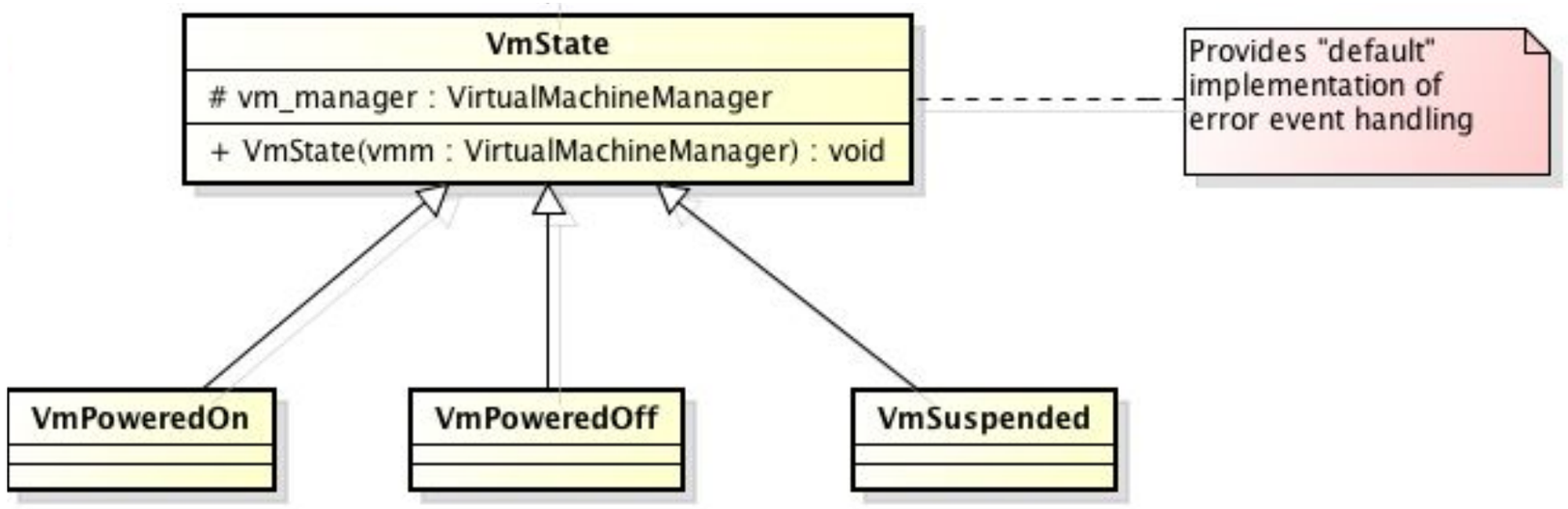

VirtualMachineManager
- currentState : VirtualMachineState
+ powerOn() : void
+ powerOff() : void
+ suspend() : void
+ reset() : void
~ setState(nextState : VmStates) : void
+ showState() : void
<<interface>>
VirtualMachineState
+ doPowerOn() : void
+ doPowerOff() : void
+ doSuspend() : void
+ doReset() : void
*
VmStates (Enum):
ON
OFF
SUSPENDED
VmState
vm_manager : VirtualMachineManager
+ VmState(vmm : VirtualMachineManager) : void
Provides "default" implementation of error event handling
VmPoweredOn
VmPoweredOff
VmSuspended


Reset
Power On
Powered On
Suspended
Suspend
Power On
Power Off
Powered Off




```
public class VmState implements VirtualMachineState {

    VirtualMachineManager vm_manager;

    public VmState(VirtualMachineManager vmm) {
        vm_manager = vmm ;
    }

    public void doPowerOn() {
        System.out.println( "Power On is not valid in " + this.getClass().getName() + " state." );
    }

    public void doPowerOff() {
        System.out.println( "Power Off is not valid in " + this.getClass().getName() + " state." );
    }

    public void doSuspend() {
        System.out.println( "Suspend is not valid in " + this.getClass().getName() + " state." );
    }

    public void doReset() {
        System.out.println( "Reset is not valid in " + this.getClass().getName() + " state." );
    }

}
```

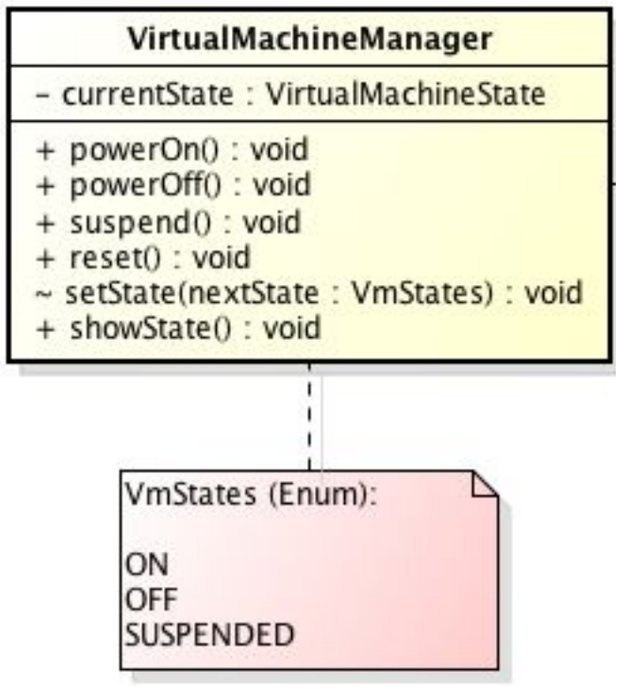





```java
public class VmPoweredOn extends VmState {

    public VmPoweredOn( VirtualMachineManager vmm )
    {
        super( vmm ) ;
    }

    @Override
    public void doPowerOff() {
        vm_manager.setState( VmStates.OFF );
    }

    @Override
    public void doSuspend() {
        vm_manager.setState( VmStates.SUSPENDED );
    }

    @Override
    public void doReset() {
        vm_manager.setState( VmStates.ON );
    }

}
```

```java
public class VmPoweredOff extends VmState {

    public VmPoweredOff( VirtualMachineManager vmm )
    {
        super( vmm ) ;
    }

    @Override
    public void doPowerOn() {
        vm_manager.setState( VmStates.ON );
    }

}
```

```java
public class VmSuspended extends VmState {

    public VmSuspended( VirtualMachineManager vmm )
    {
        super( vmm ) ;
    }

    @Override
    public void doPowerOn() {
        vm_manager.setState( VmStates.ON );
    }

}
```




```java
public class VirtualMachineManager {

    VirtualMachineState poweredOnState ;
    VirtualMachineState poweredOffState ;
    VirtualMachineState suspendedState ;
    VirtualMachineState currentState ;

    public VirtualMachineManager()
    {
        poweredOnState = new VmPoweredOn(this);
        poweredOffState = new VmPoweredOff(this);
        suspendedState = new VmSuspended(this);
        currentState = poweredOffState ;
    }

    public void powerOn() {
        System.out.println( "powering on...");
        currentState.doPowerOn();
    }

    public void powerOff() {
        System.out.println( "powering off...");
        currentState.doPowerOff();
    }

    public void suspend() {
        System.out.println( "suspending...");
        currentState.doSuspend();
    }

    public void reset() {
        System.out.println( "reset vm...");
        currentState.doReset();
    }

    void setState(VmStates nextState) {
        switch( nextState ) {
            case OFF :          currentState = poweredOffState ; break ;
            case ON :           currentState = poweredOnState ; break ;
            case SUSPENDED:     currentState = suspendedState ; break ;
        }

    }

    public void showState()
    {
        System.out.println( "Current State: " + currentState.getClass().getName());
    }

}
```