STATE PATTERN

Jonathan Ozik

CSPP 51023, Winter 2010

** Venetian (i.e., (city-)State) Slide Theme*

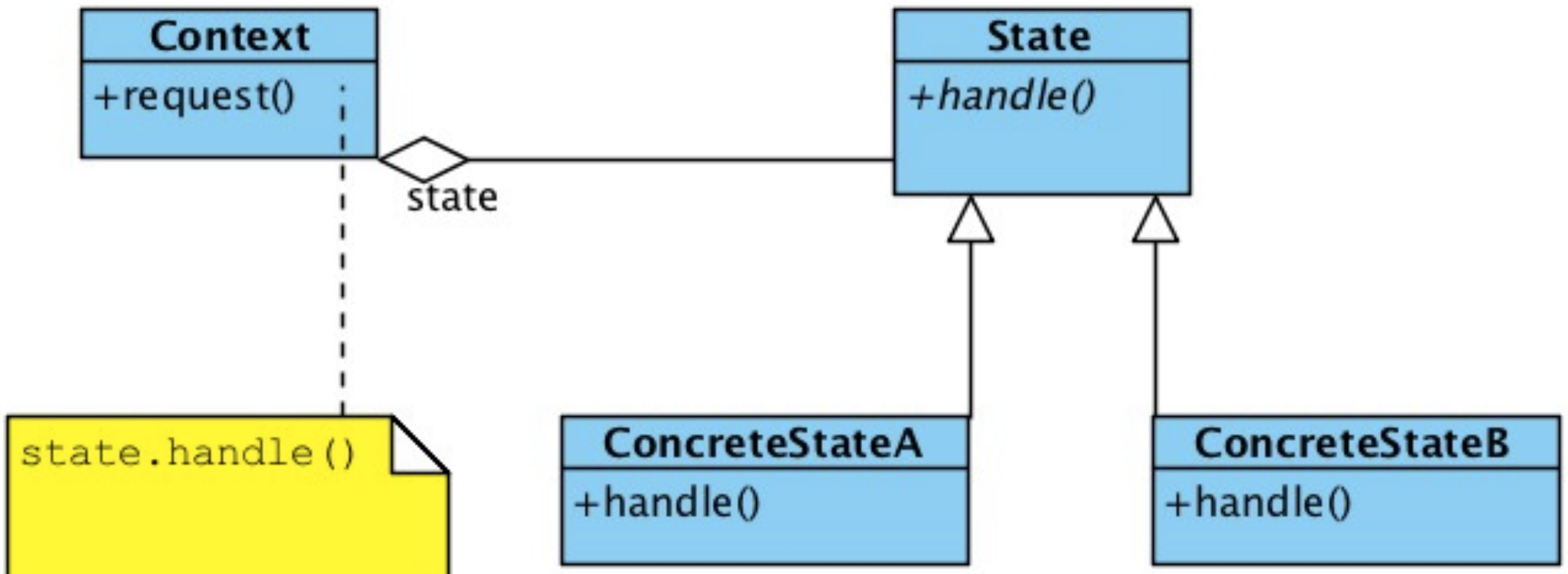
PATTERN PURPOSE

- Intent (GoF*):
  - Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.

- Localizes and partitions state based behaviors.
  - Avoid unwieldy multipart conditional statements.
  - State transitions are explicit.
  - Structure clarifies intent.

- Change behavior at runtime.
  - Similar to Strategy pattern but different purpose.

* *Design Patterns*, Gamma, et. al., Addison Wesley, 1995.



PATTERN STRUCTURE

* Adapted from *Design Patterns*, Gamma, et. al., Addison Wesley, 1995, pg. 306.

PARTICIPANTS

- **Context**
  - Contains an instance of State.
  - Clients call Context after (optionally) initializing it with a particular ConcreteState.

- **State**
  - Abstract class defining behaviors interface.

- **ConcreteState subclasses**
  - Classes implementing state based behavior.

ADDITIONAL KEY IDEAS

- State transitions can be handled either by the Context (centralized) or the ConcreteState subclasses (decentralized).
- If the State objects have no instance variables, they can be shared (Flyweight).
- State objects are usually Singletons.
- Creation
  - Create and destroy as necessary (unknown states, change infrequently)
  - Create ahead of time and store (rapid changes).

ADDITIONAL POINT REGARDING DYNAMIC LANGUAGES

- Certain OO languages allow runtime class modifications.
  - Groovy: via MOP, create behaviors "on the fly"

```
String.metaClass.hello = { ->
    println "Hello there!"
}
"A plain old String".hello()
```

```
Hello there!
```

- These languages can support the State pattern in different ways.

DEMOS

- Simple Example

- Less Simple Example