Software Design

- Concepts
  - Abstraction & encapsulation (Feb 1)
  - Design patterns
    - Observer (Feb 3)
    - Strategy (Feb 5)
    - **State (Feb 10)**
  - Dependency inversion
  - Dependency injection

Reminder:
Mid-term is two weeks from today (reading break in-between). It will cover material from start of term to end of Feb 12.

Learning Objectives for Today

- Given a design/code and an appropriate feature request:
    - apply the State design pattern to implement the feature
    - explain the benefits and limitations of the use of the State design pattern for the design/code and feature request
- Explain the difference between the Strategy and State design patterns

Plan for Today

Design terminology

How to apply a design pattern in general

Using State design pattern to vary behaviour based on object state

A simple example to demonstrate State

An example of State in Typescript

Applying State for a new feature in Mario

State pattern highlights

Design Terminology

Some design terms you should be comfortable with:

- Coupling
- Cohesion
- Aggregation (Composition) and Association
- Inheritance
- Delegation
- Interface (what they are and why we use them)

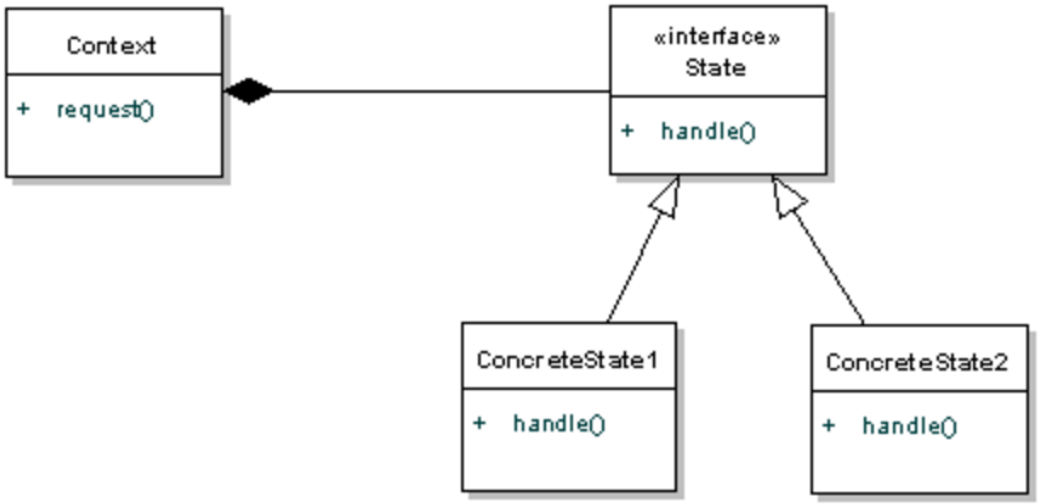
Applying a Design Pattern

- What are the critical roles in the Design Pattern?
- Two decisions are inter-related:
  - How are the roles going to map to the classes/ interfaces in the program (e.g., Mario5TS)?
  - What new classes or interfaces need to be added to the program (e.g., Mario5TS)
- How will the Design Pattern be initialized/triggered?

State Design Pattern

"Allows an object to alter its behaviour when its internal state changes."

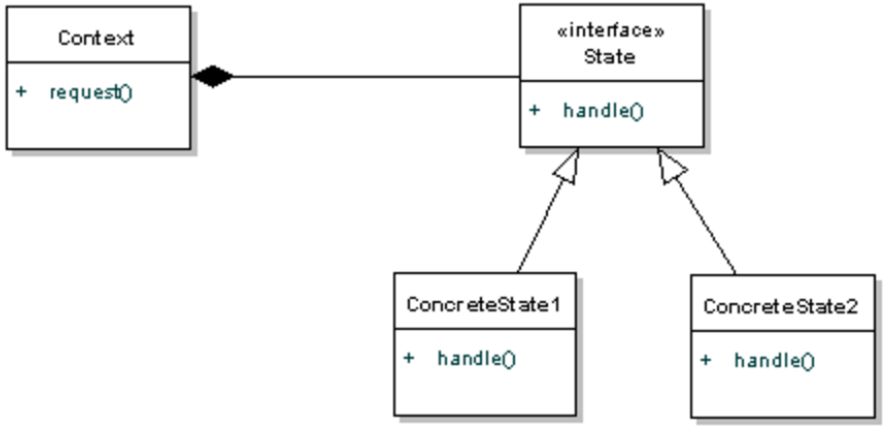
— Design Patterns book



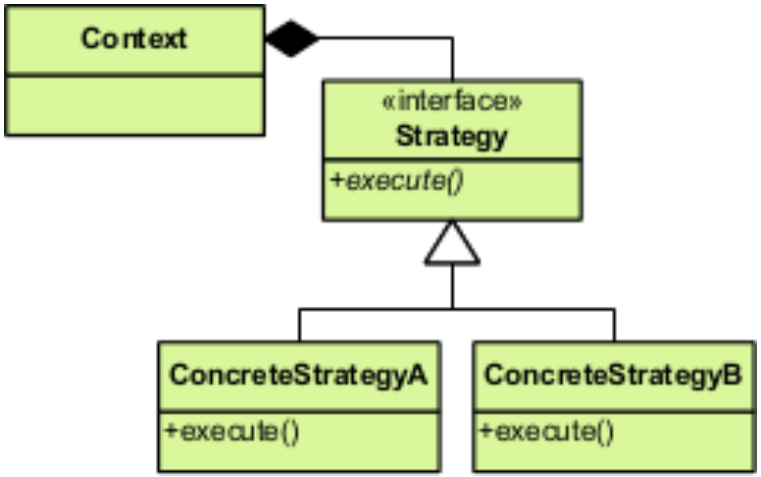
A decent short overview of State is available at: https://dzone.com/articles/design-patterns-state

But isn't that the same as Strategy?

State design pattern



Strategy design pattern



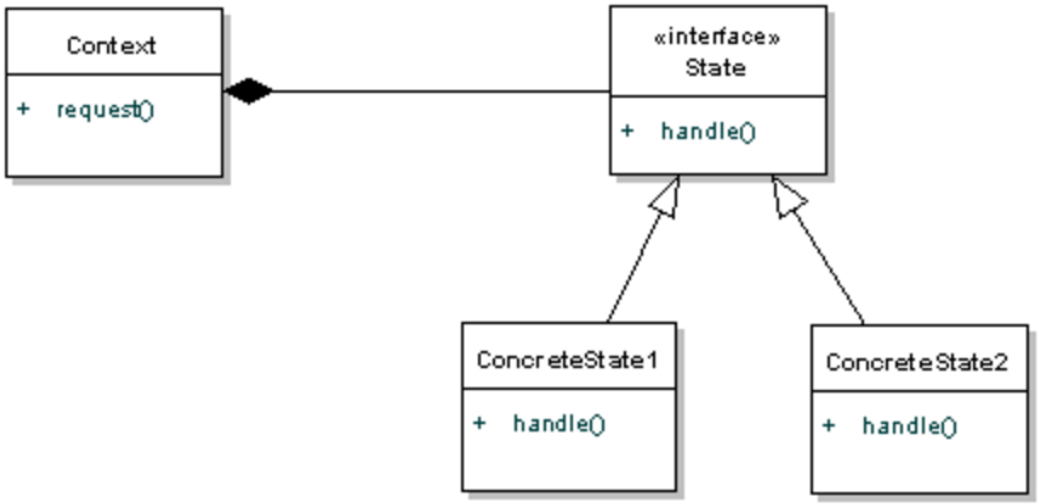
The intent of each pattern is different even though structure is similar.

We'll come back to this after we look at State…

State Design Pattern

"Allows an object to alter its behaviour when its internal state changes."

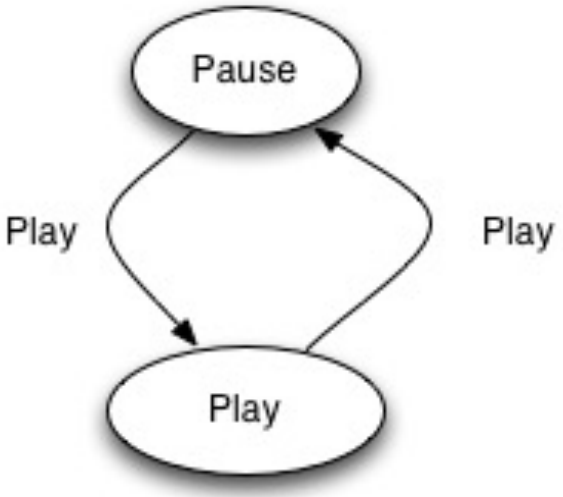
— Design Patterns book

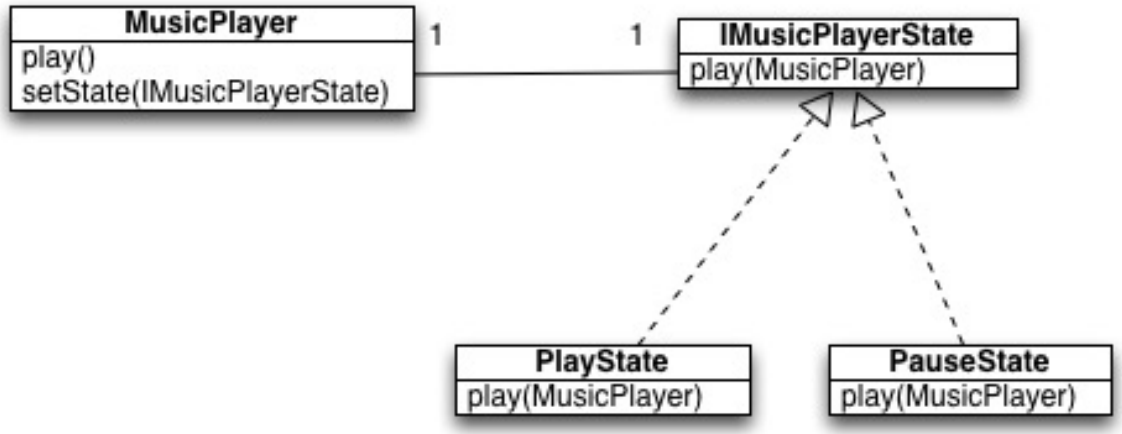


A decent short overview of State is available at: https://dzone.com/articles/design-patterns-state

A Simple Example

Imagine you are designing the software for a simple music player. When you hit the “Play” button, music plays. When you hit the “Play” button again, the music pauses.





State Pattern: A Typescript Example

You have been asked to design a shopping cart that can be used on an e-commerce site.

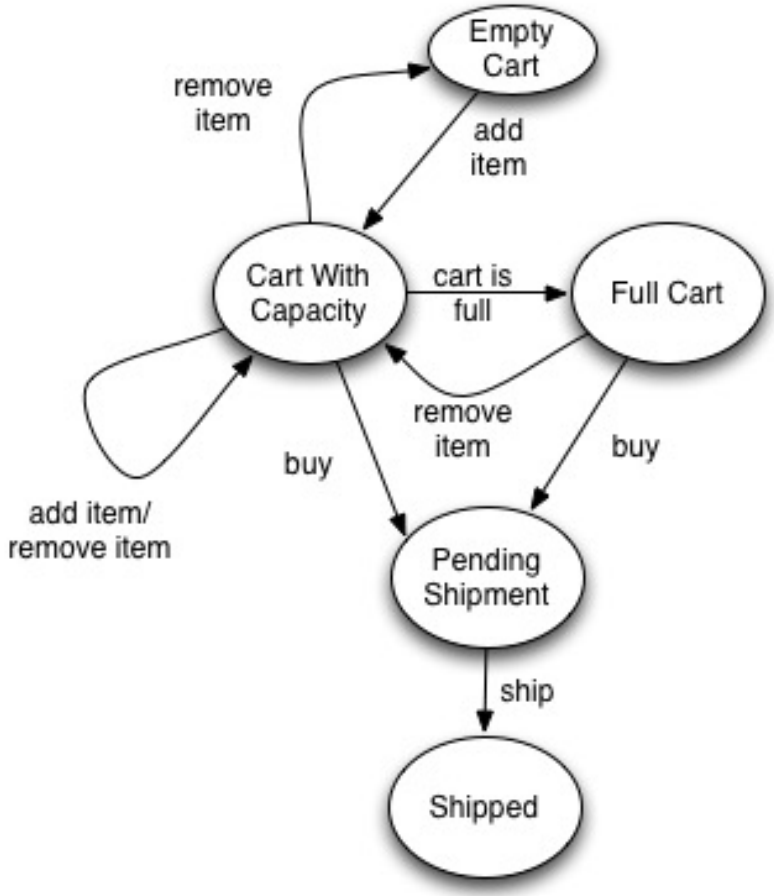
What might be the states a shopping cart object can be in?

How does the shopping cart move from state to state?

The StateExample code linked from the “Software Design Unit” post on piazza has a solution.

State Pattern: A Typescript Example

One possible finite state machine describing the shopping cart…



State Pattern: A Typescript Example

Try applying the State pattern…

What will be the classes representing the States?

What will be in the interface for those classes?

What methods will the ShoppingCart class have and what will they do?

The StateExample code linked from the “Software Design Unit” post on piazza has a solution. Try extracting a class diagram from this code.

The State Pattern in Mario

You have been asked to change the Gumpa so that when the “z” key is pressed down, the Gumpa enters a supercharged state and the visual appearance of the Gumpa changes. When the “z” key is released, the Gumpa goes back to normal.

Apply the State Pattern to enable the Gumpa to either be in the normal state or the supercharged state.

The Mario code is linked from the “Software Design Unit” post on piazza. Search for “STATE” to find how the solution was implemented.

Strategy and State: What's the difference?

Intent

Strategy encapsulates related algorithms and enables choice of algorithm at run-time. State enables different behaviours based on the state of an object.

Strategy is more about "how" something is done and State is more about "what is done.

Coupling

State much more coupled to its "Context" class role than Strategy.

Who Does What

Client changes Strategy that is used. Changes in state may be by "Context" role or State itself.

See: http://javarevisited.blogspot.ca/2014/04/difference-between-state-and-strategy-design-pattern-java.html

Highlights of State Design Pattern

- Benefits:
  - improves maintainability by avoiding large conditional statements that would otherwise be needed in the “Context” class
  - makes state transitions explicit
  - can help to protect “Context” object from inconsistent internal states (that might otherwise be represented by values of multiple fields)
- Limitations:
  - can cause the creation of many classes if many states must be represented

On-line Resources for State Design Pattern

- http://gameprogrammingpatterns.com/state.html
- http://www.codeproject.com/Articles/38962/State-Design-Pattern
- You can find a lot of examples, etc. on the web