

Behavioral Patterns

JS Patterns and Anti Patterns

Malte Brockmann, Jun Heui Cho

Outline

- Behavior pattern in general
- Command
- Memento
- Chain of responsibility
- Observer

Behavior Pattern in general

- Mainly concerned with the communication between objects.
- Describe a process or a flow
- encapsulating behavior and delegating of requests
- increases flexibility

Command

- Encapsulate a request as an object
- Request without knowing anything about the operation being requested. - “Black box execute()”
- Uses: GUI buttons, Networking, Multi-level undo, Progress bar

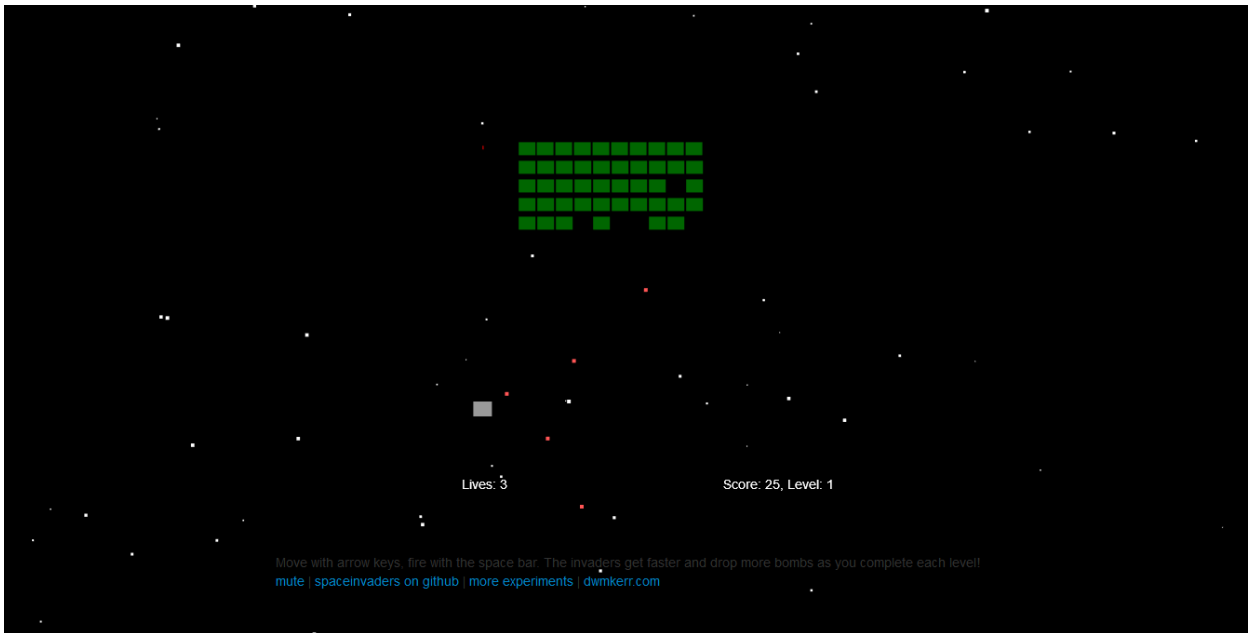
Command - Participants

- **Client:** decides which command at which point
- **Receiver:** knows how to carry out the operation
- **Command:** execute()
- **Invoker:** knows how to execute



Spaceinvader

- Retro Game: shooting Spaceinvader
- Level bases
- State bases (Welcome-, GameOver-, PlayState, ect.)



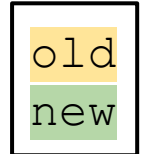
Command - Spaceinvader 1/4

before:

```
if(game.pressedKeys[37]) {  
    this.ship.x -= this.shipSpeed * dt; //dt = Delta time  
}                                     //    = 1/fps  
  
if(game.pressedKeys[39]) {  
    this.ship.x += this.shipSpeed * dt;  
}  
if(game.pressedKeys[32]) {  
    this.fireRocket();  
}  
[...]  
bomb.y += dt * bomb.velocity;  
[...]  
rocket.y -= dt * rocket.velocity;
```

Command - Spaceinvader 2/4

after: (Commands)



```
var goLeft = {  
  execute : function(obj, speed) {  
    obj.x -= speed * dt;  
  }  
}  
  
var goRight = {  
  execute : function(obj, speed) {  
    obj.x += speed * dt;  
  }  
}  
  
var shoot = {  
  execute : function(obj) {  
    obj.fireRocket();  
  }  
}
```


Command - Spaceinvader 3/4

after: (Commands)

```
var goUp = {  
  execute : function(obj, speed) {  
    obj.y -= speed * dt;  
  }  
}
```

```
var goDown = {  
  execute : function(obj, speed) {  
    obj.y += speed * dt;  
  }  
}
```

Command - Spaceinvader 4/4

after:

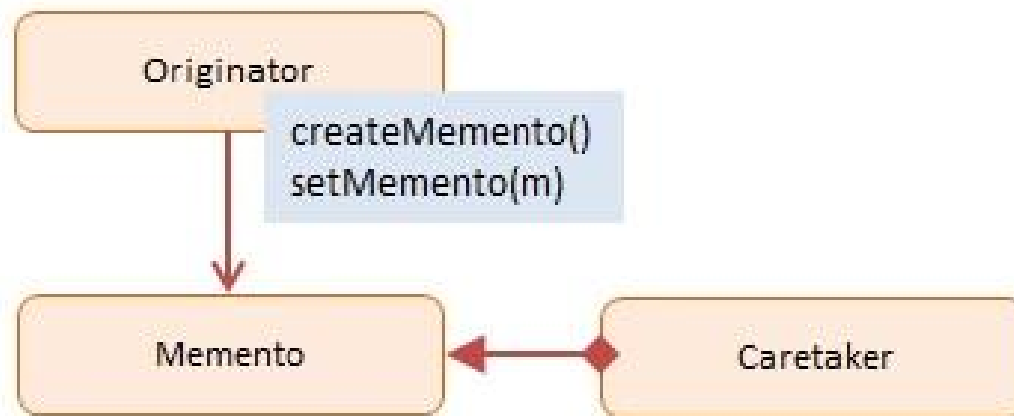
```
//The Client is the main loop and the invoker is the user
if (game.pressedKeys[37]) { //<-receiver
    goLeft.execute(this.ship, this.shipSpeed);
}
if (game.pressedKeys[39]) {
    goRight.execute(this.ship, this.shipSpeed);
}
if (game.pressedKeys[32]) {
    shoot.execute(this);
}
[...]
goDown.execute(bomb, bomb.velocity);
[...]
goUp.execute(rocket, rocket.velocity);
```

Memento

- Capturing and externalizing an object's internal state to be restored later.
- Database of “save point”
- Use: used to avoid disclosure of implementation details

Memento - Participants

- **Originator**: interface to create and restore mementos
- **Memento**: ordinator object
- **Caretaker**: stores mementos



Memento - Spaceinvader 1/4

before:

```
WelcomeState.prototype.keyDown = function(game, keyCode) {  
  [...]  
    game.moveToState(new LevelIntroState(game.level));  
};  
[...]
```

```
GameOverState.prototype.keyDown = function(game, keyCode)  
[...]  
  game.moveToState(new LevelIntroState(1));  
}
```

Memento - Spaceinvader 2/4

after:

```
function Memento(state) {  
    this.state = state;  
    this.getSavedState = function() {  
        return this.state;  
    };  
};  
  
function Caretaker() {  
    var saveState = [];  
    this.addMemento = function(memento) {  
        saveState.push(memento);  
    };  
    this.getMemento = function(index) {  
        return saveState[index];  
    };  
};
```

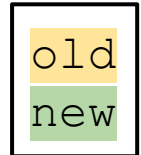
Memento - Spaceinvader 3/4

after:

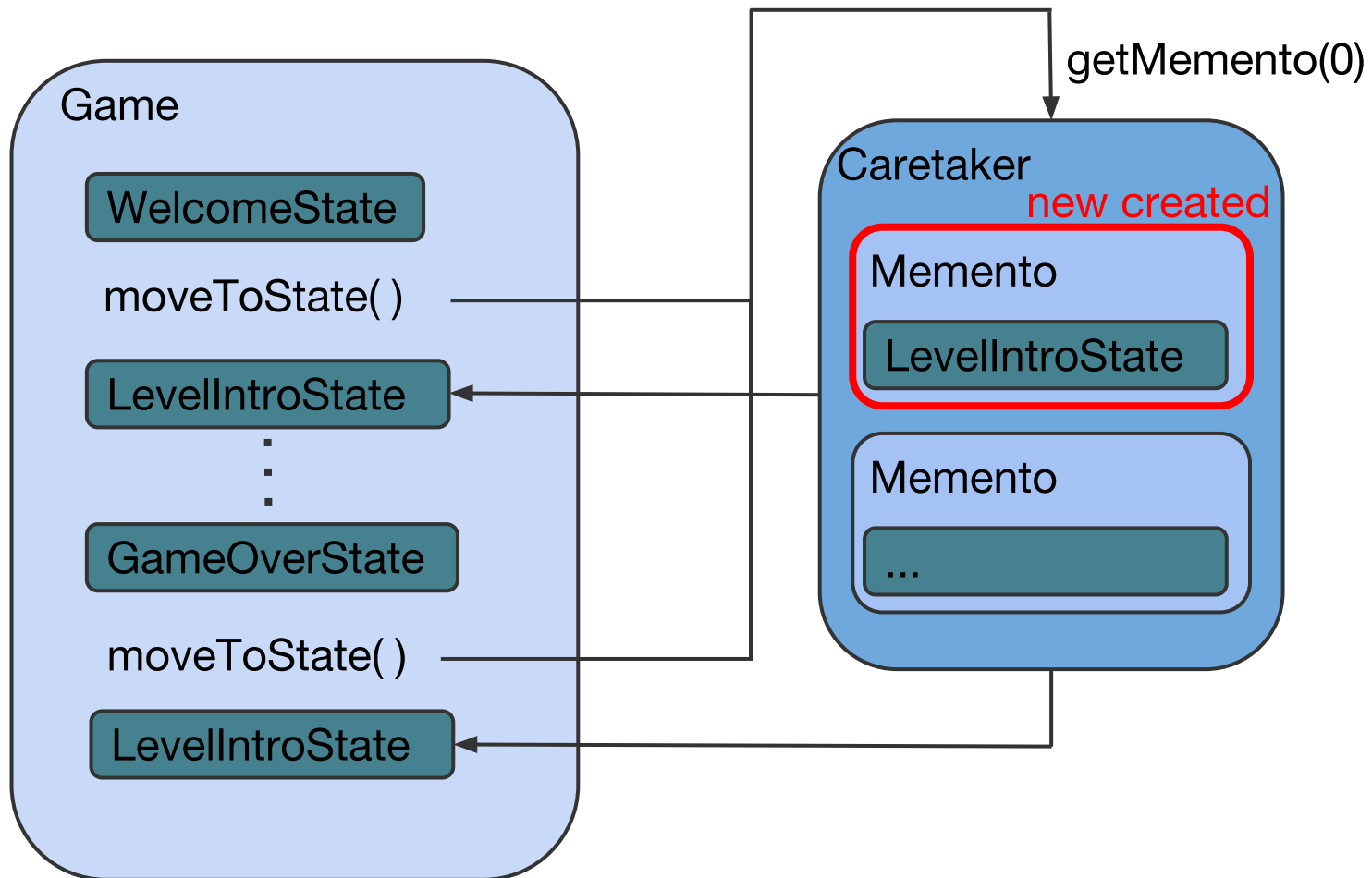
```
//In this case an Originator is for example a LevelIntroState  
caretaker = new Caretaker();  
[...]
```

```
WelcomeState.prototype.keyDown = function(game, keyCode) {  
[...]  
caretaker.addMemento(new Memento(new LevelIntroState(game.level)));  
game.moveToState((caretaker.getMemento(0)).getSavedState());  
}  
};  
[...]
```

```
GameOverState.prototype.keyDown = function(game, keyCode) {  
[...]  
game.moveToState((caretaker.getMemento(0)).getSavedState());  
}  
};
```



Memento - Spaceinvader 4/4

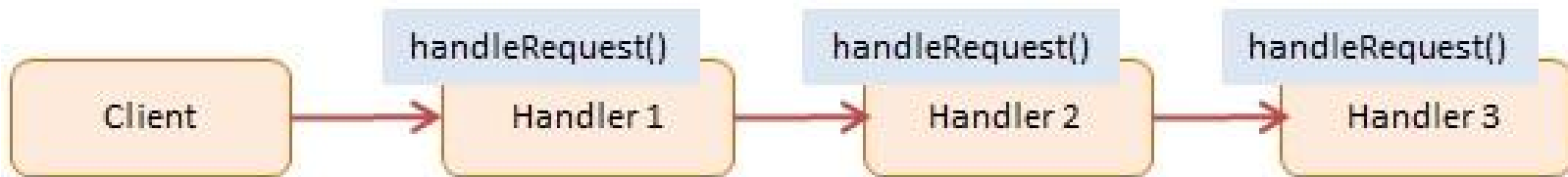


Chain of responsibility

- Avoid coupling between the sender and the receiver of a request.
- More than one object have the chance to handle the request.
- linear search for a handler

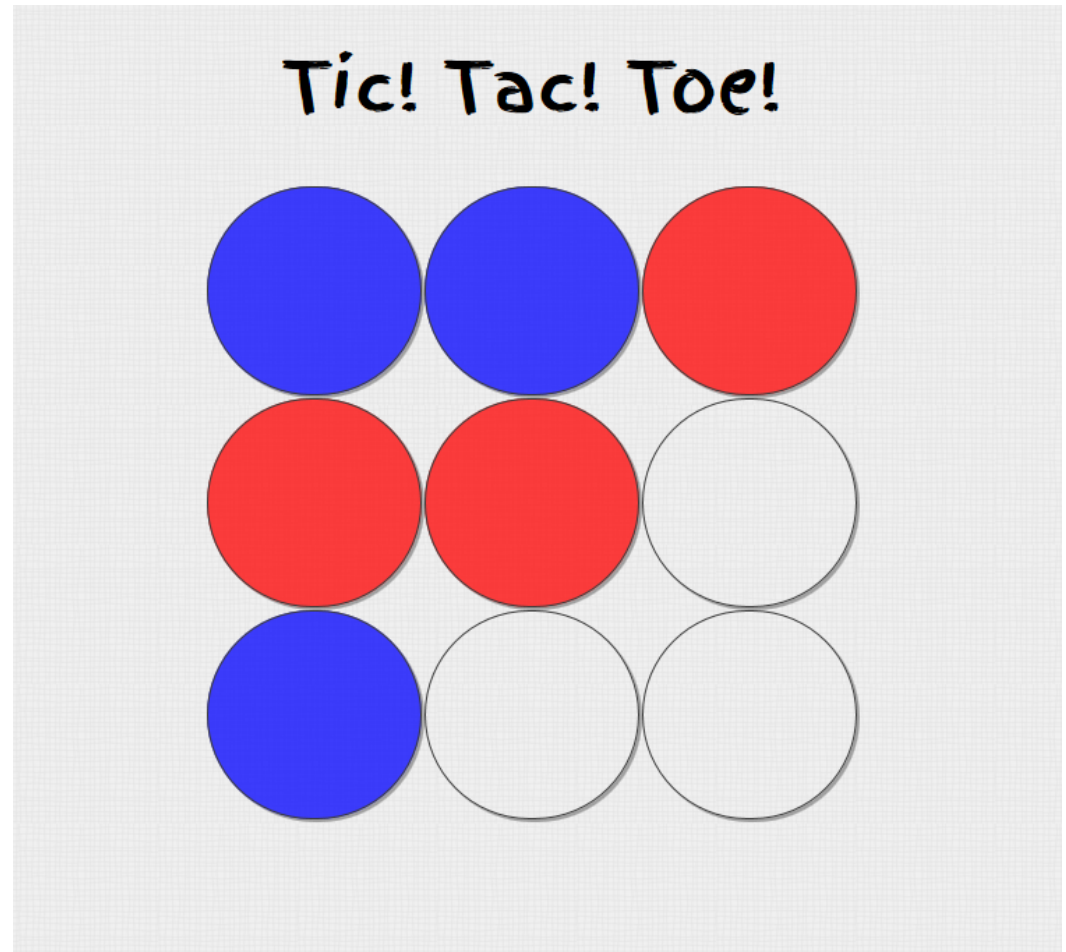
Chain of responsibility - Participants

- **Client:** initiator of the request
- **Handler:** has an interface for handling the request



Tic Tac Toe

- retro Game
- 2 player
- checks winner or tie after each turn
- restarts



Chain of responsibility - Tic tac toe 1/7

before:

```
function checkWinner() {  
    if (checkRows() === true || checkCols() === true ||  
checkDiag() === true) {  
        winningPlayer = turn.currentPlayerColor();  
        // Alert winner  
        endGame("Player " + winningPlayer + ", you win!");  
    }  
    else if (checkTie() === true) {  
        endGame("It's a tie...");  
    }  
    else {  
        turn.changeTurn();  
    }  
}
```

Chain of responsibility - Tic tac toe 2/7

before:

```
function checkRows() {  
    for (i = 0; i < board.length; i++) {  
        var same = true;  
        for (j = 0; j < board[i].length; j++) {  
            if (board[i][j] === 0 || board[i][j] !== board  
[i][0]) {  
                same = false;  
            }  
        }  
        if (same) {  
            return same;  
        }  
    }  
}
```

Chain of responsibility - Tic tac toe 3/7

before:

```
function checkTie() {  
    var flattenedBoard = Array.prototype.concat.apply([],  
board);  
    for(i = 0; i < flattenedBoard.length; i++){  
        if(flattenedBoard[i] === 0){  
            console.log(i);  
            return false;  
        }  
    }  
    return true;  
}
```

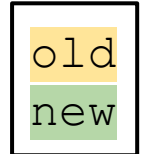
Chain of responsibility - Tic tac toe 4/7

after:

```
function checkWinner() {  
    checkRows();  
  
}
```

Chain of responsibility - Tic tac toe 5/7

after:



```
function checkRows() {  
    for (i = 0; i < board.length; i++) {  
        var same = true;  
        for (j = 0; j < board[i].length; j++) {  
            if (board[i][j] === 0 || board[i][j] !== board[i][0])  
            {  
                same = false;  
            }  
        }  
        if (same) {  
            winningPlayer = turn.currentPlayerColor();  
            // Alert winner  
            endGame("Player " + winningPlayer + ", you win!");  
        }  
    }  
    checkCols(); };
```

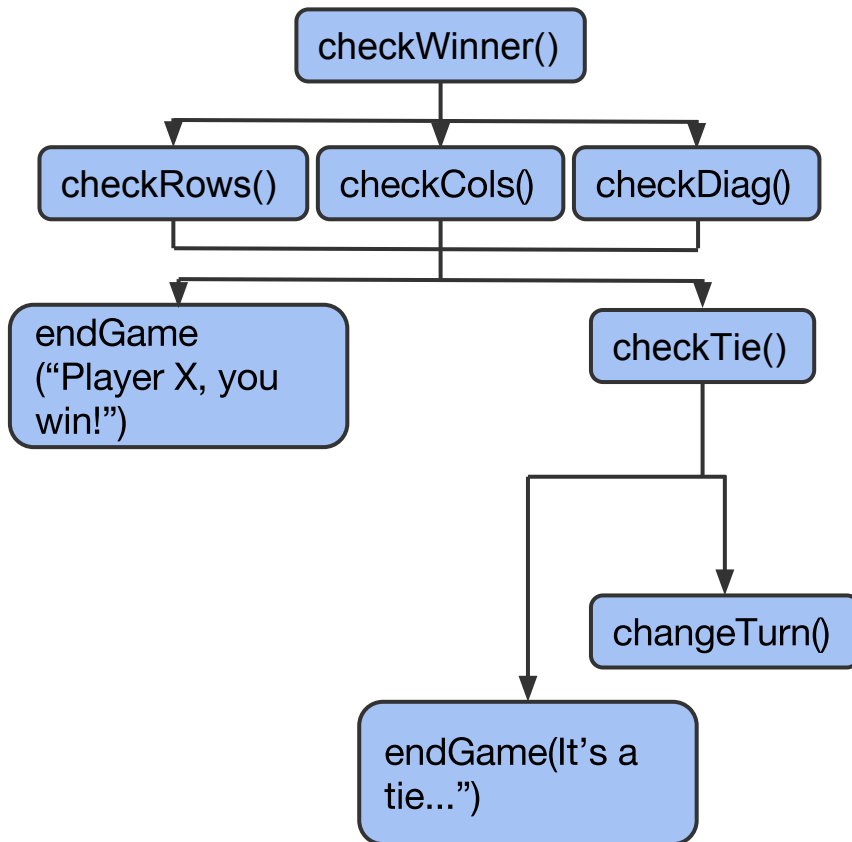

Chain of responsibility - Tic tac toe 6/7

after:

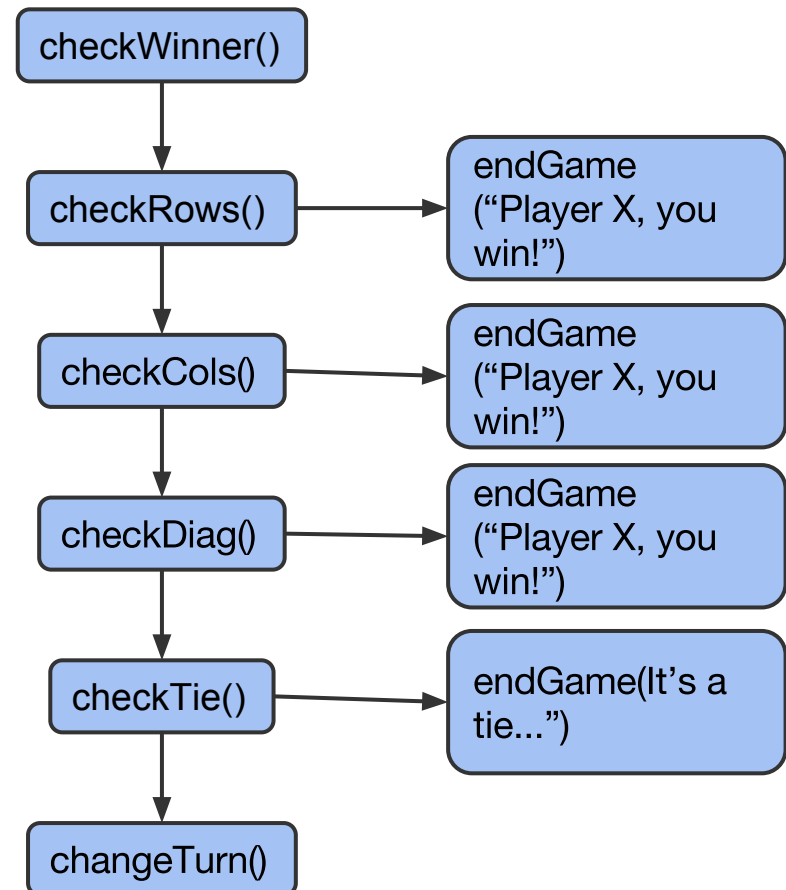
```
function checkTie() {  
    var flattenedBoard = Array.prototype.concat.apply([], board);  
    for(i = 0; i < flattenedBoard.length; i++){  
        if(flattenedBoard[i] === 0){  
            console.log(i);  
            turn.changeTurn();  
            return;  
        }  
    }  
    endGame("It's a tie...");  
}
```

Chain of responsibility - Tic tac toe 7/7

before:



after:

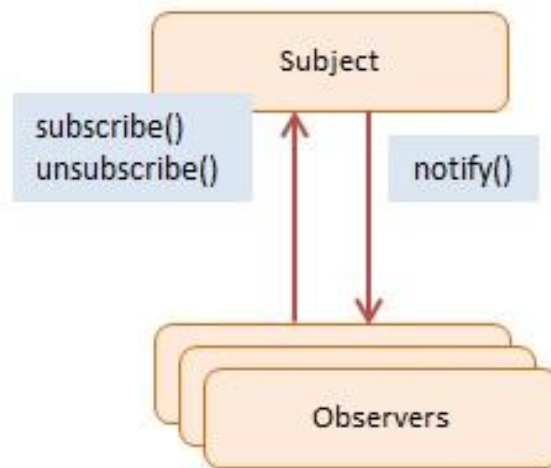


Observer

- Define a one-to-many dependency between objects
- When one object (Observable) changes its state, all dependent objects (Observers) are notified (usually with a message)
- Notified objects handle their own update

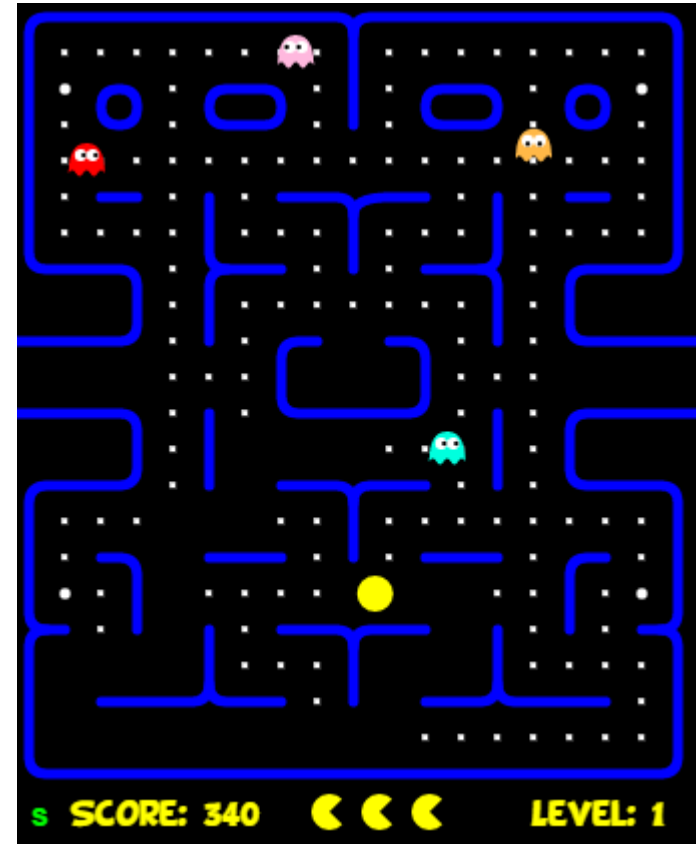
Observer - Participants

- **Subject / Observable:** maintains a list of observers, lets them subscribe/unsubscribe, and notifies them about changes
- **Observers:** has a function that can be invoked when notified



Pac Man

- retro game (classic pacman)
- 3 lives
- avoid getting eaten by ghosts
- can eat and “jail” the ghosts for a short time after eating “beans”
- eat all the blocks to a level



Observer – Pac Man 1/5

before:

```
function startLevel() {  
    user.resetPosition();  
    for (var i = 0; i < ghosts.length; i += 1) {  
        ghosts[i].reset();  
    }  
    audio.play("start");  
    timerStart = tick;  
    setState(COUNTDOWN);  
}
```

Observer – Pac Man 2/5

before:

```
function eatenPill() {  
    audio.play("eatpill");  
    timerStart = tick;  
    eatenCount = 0;  
    for (i = 0; i < ghosts.length; i += 1) {  
        ghosts[i].makeEatable(ctx);  
    }  
};
```

Observer – Pac Man 3/5

after:

```
function startLevel() {  
    user.resetPosition();  
    notifyObservers("levelstarted");  
    timerStart = tick;  
    setState(COUNTDOWN);  
}
```

[...]

```
function eatenPill() {  
    timerStart = tick;  
    eatenCount = 0;  
    notifyObservers("pilleaten");  
};
```


Observer – Pac Man 4/5

after:

//REFACTOR: adding observable functionalities

```
function subscribe(o) {  
    observers.push(o);  
};
```

```
function unsubscribe(o) {  
    observers = observers.filter(  
        function(item) {  
            if (item !== o) { return item; } }  
    ); };
```

```
function notifyObservers(message) {  
    for (var i = observers.length - 1; i >= 0; i--) {  
        observers[i].notify(message);  
    }; };
```

Observer – Pac Man 5/5

after:

```
//REFACTOR: adding observer functionalities for Ghost  
function notify(message) {  
    switch(message) {  
        case "levelstarted":  
            reset();  
            break;  
        case "pilleaten":  
            makeEatable();  
            break;  
        default:  
            break;  
    }  
};
```

(Analog for Audio)

Sources

<http://www.dofactory.com/javascript/design-patterns>

https://sourcemaking.com/design_patterns

<http://www.blackwasp.co.uk/DesignPatternsArticles.aspx>

https://en.wikipedia.org/wiki/Command_pattern

https://de.wikipedia.org/wiki/Memento_%28Entwurfsmuster%29

https://en.wikipedia.org/wiki/Chain-of-responsibility_pattern

Projects

Spaceinvader: <https://github.com/dwmkerr/spaceinvaders>

Tic Tac Toe: <https://github.com/negomi/tic-tac-toe>

Pacman: <https://github.com/daleharvey/pacman>

Iterator

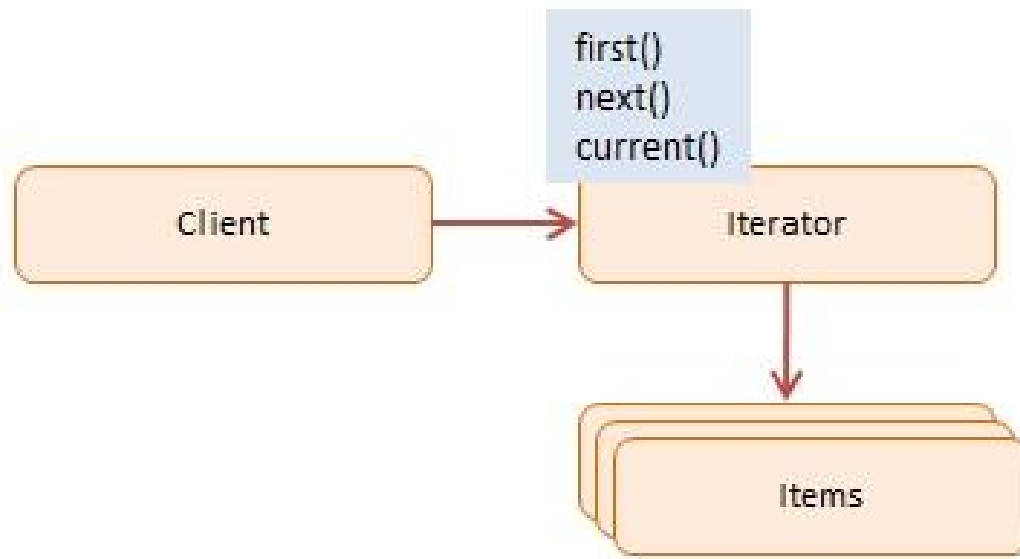
- access elements without knowing the underlying structure of the object
- effectively loop over a object collection
- object store as list, trees or more complex structures
- many language have build in iterator, but not JavaScript
- Iterator is the “secretary”

Iterator - Participants

Client: uses the iterator

Iterator: interface with methods like `first()`, `next()`, `hasNext()`

Items: individual objects



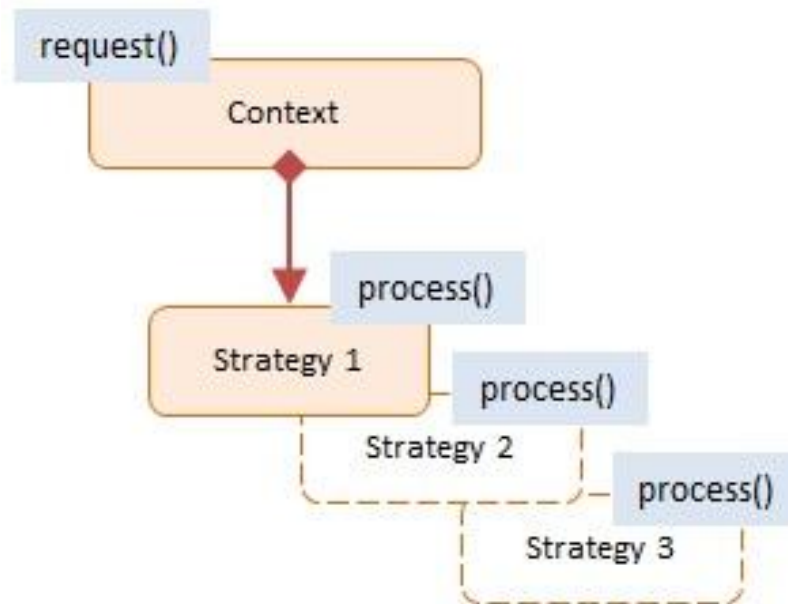
Strategie

- Interchangeable set of algorithms
- Swapped out at runtime
- Minimizing coupling
- Option to hide implementation

Strategie - Participants

Context: reference to the current Strategy, the option to change it and to calculate the “cost” of each strategy

Strategy: implementation of different option for a task



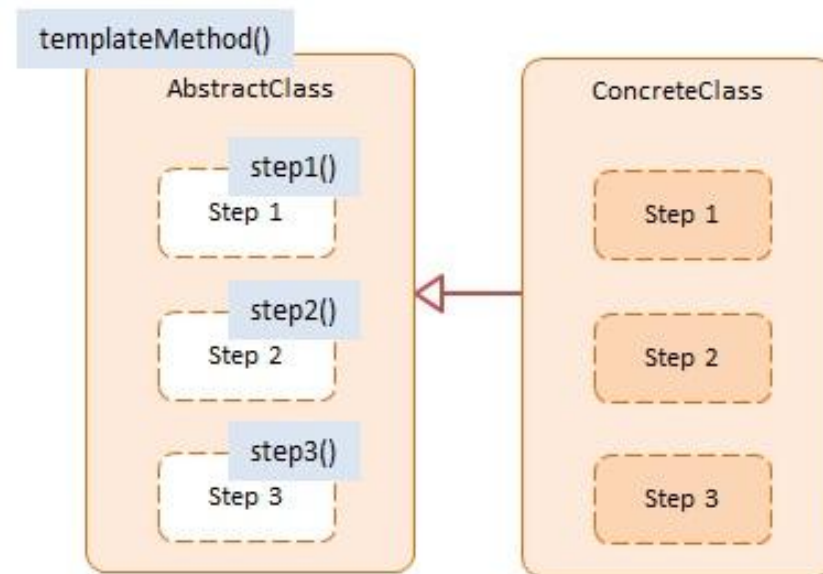
Template method

- Outline of a series of steps for an algorithm
- Subclasses can redefine certain steps of an algorithm without changing the algorithms structure
- Offers extensibility to the client developer

Template method - Participants

AbstractClass: template method defining the primitive steps for an algorithms

ConcreteClass: implements the primitive steps as defined



State

- A object can alter its behaviour when its internal state changes
- Object appears to have changed its class
- E.g. state machines

State - Participants

Context: maintains a reference to a object, defines its current state, and allows it to change its state

State: state values are associated with the according behaviour of the state

