

# COMP 504: Graduate Object-Oriented Programming and Design

## Lecture 4: Template Design Pattern and Property Changes

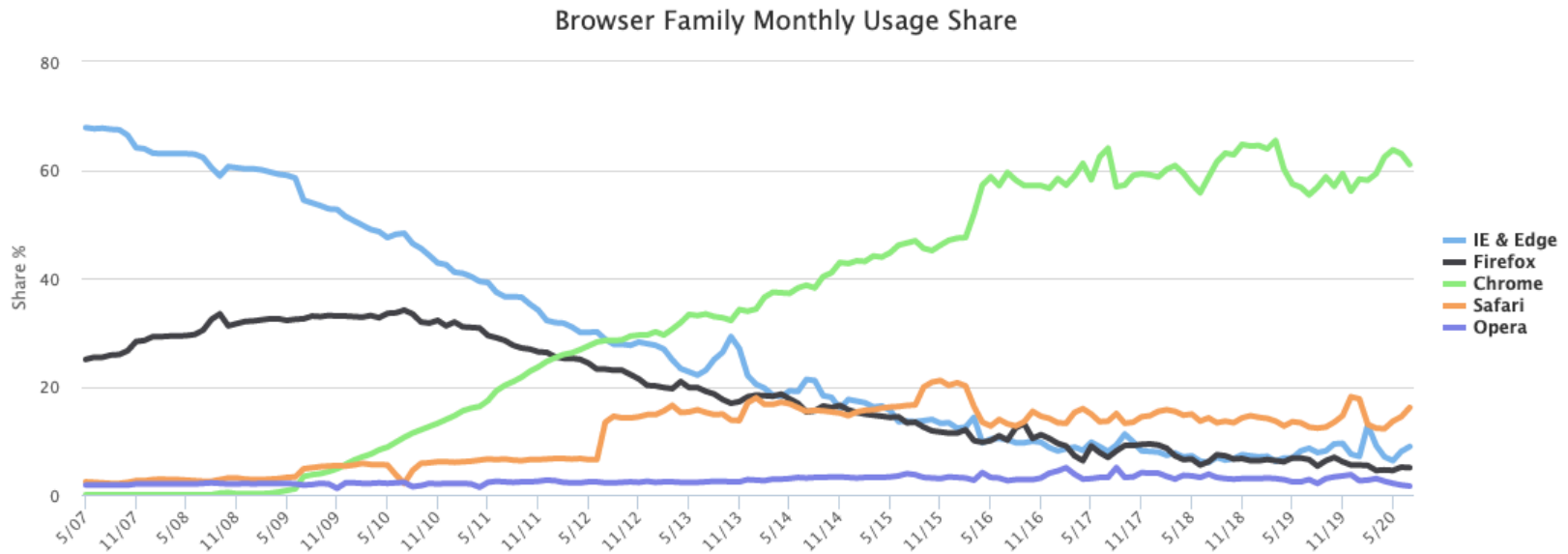
Mack Joyner ([mjoyner@rice.edu](mailto:mjoyner@rice.edu))

<https://www.clear.rice.edu/comp504>



# Web Browser Usage Trends

<https://www.w3counter.com/trends>



# Representational State Transfer (REST)

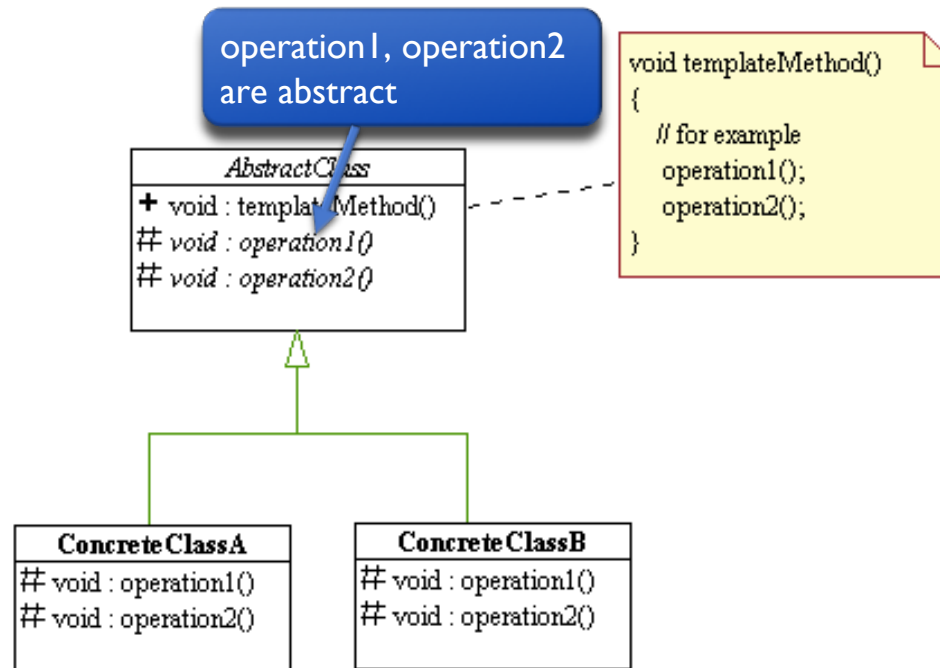
---

- Client-Server separation of concerns to simplify component implementation, reduction of complexity, increase scalability
- Standard HTTP method verbs
- Typically JSON (JavaScript Object Notation) messages
- REST is stateless and cacheable



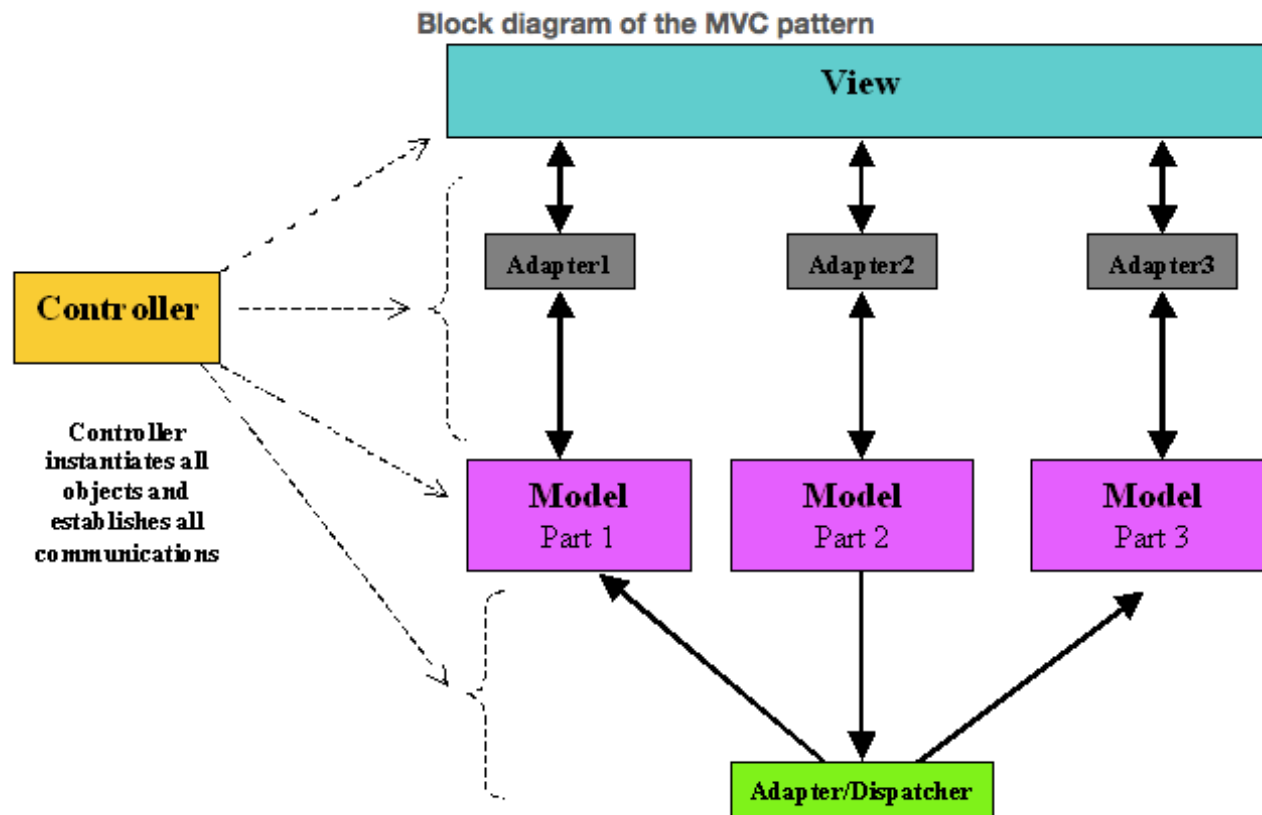
# Template Design Pattern

- Abstract class sets up **invariant** skeleton of the algorithm
- Concrete subclasses implement specific **variant** algorithmic details



# MVC Design Pattern

Controller may use an adapter to communicate with model and view



# Ball World

---

- Multiple moving balls collide with canvas boundary
  - Canvas boundaries are walls
- BallWorld should update all balls
  - view determines when periodic update occurs
- PropertyChangeListener and PropertyChangeSupport framework
  - PropertyChangeListeners (balls) listen for property changes
  - PropertyChangeSupport notifies balls when it's time to update



# Property Change Support

- Setup support to tell objects a change occurred
  - Allowed to name the property
  - Register objects to listen for specific property change
- Objects listening for change need to implement *PropertyChangeListener*
  - Provide *propertyChange* function
  - *PropertyChangeEvent* argument

## Adapter

```
15     private PropertyChangeSupport pcs;
16     private Point dims;
17
18     /**
19      * Constructor
20      */
21     public DispatchAdapter() {
22         pcs = new PropertyChangeSupport(this);
23     }
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50     /**
51      * Add a ball that will observe a property change (i.e. time elapsed)
52      * @param pcl The ball
53      */
54     private void addListener(PropertyChangeListener pcl) {
55         pcs.addPropertyChangeListener("theClock", pcl);
56     }
```



# Property Change Listener

---

Objects listening for change need to implement *PropertyChangeListener*

- Provide *propertyChange* function
- *PropertyChangeEvent* argument

```
/**
 * Represents a ball in the ball world.
 */
public class Ball implements PropertyChangeListener {

    /**
     * The ball is listening for a property change. This is the property change event handler.
     * @param evt The property change event
     */
    @Override
    public void propertyChange(PropertyChangeEvent evt) {

    }
}
```





# Property Change Event

- Fire property change event
  - Get property name
  - Event fired if *oldValue* not equal to *newValue*
  - Objects listening to specific property change

## Adapter

```
33     /**
34      * Call the update method on all the ball observers to update their position .
35      */
36     public PropertyChangeListener[] updateBallWorld() {
37         pcs.firePropertyChange("theClock", false, true);
38         return null;
39     }
```

- All objects (PCL) listening for property has *propertyChange* method called
- Return property change listeners with (PCS)  
*getPropertyChangeListeners*



# Null Object Listener

---

- A null object listener is useful for testing or avoiding null pointer exceptions
  - Has no behavior (e.g. never moves when update is called)
- Recommend way of identifying null object listener
  - Specific color unique to null object observer (e.g. always black)
- Useful when using select instead of buttons
  - Not ideal to have a button for each option



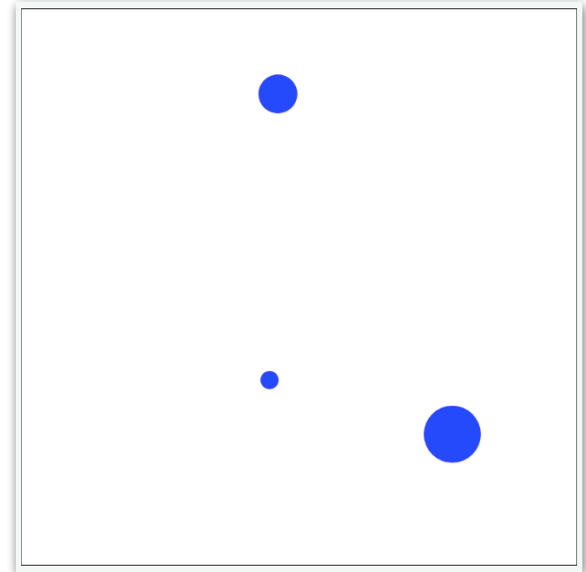
# Worksheet #3: Template Design Pattern

---

Assume a Ball World where balls have a color and a ball type. The balls move in some pattern based on their ball type. When balls collide with a wall, they start traveling in opposite direction.

Which of the following ball attributes and behaviors could be invariant and which are variant?

collision detection, ball color, ball movement, ball type, new direction after wall collision

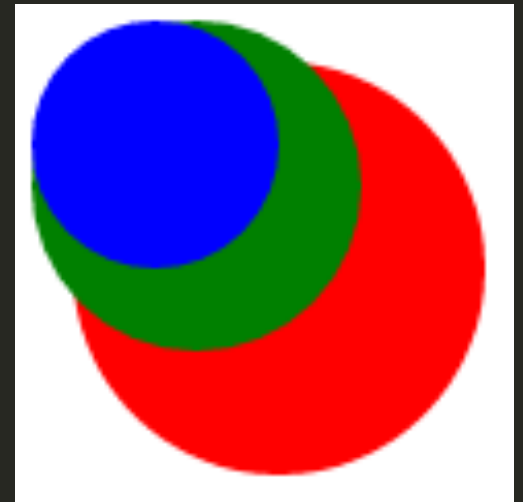


# Canvas

```
<canvas></canvas>
<script>
var c = document.querySelector("canvas").getContext("2d");

function fillCircle(x, y, r, color) {
  c.fillStyle = color;
  c.beginPath()
  c.arc(x, y, r, 0, 2 * Math.PI, false)
  c.closePath()
  c.fill()
}

fillCircle(60, 60, 50, 'red')
fillCircle(40, 40, 40, 'green')
fillCircle(30, 30, 30, 'blue')
</script>
```



# JavaScript Functions

```
var namedFunction = function aName() {  
    // This is a comment  
    return 9;  
}  
  
var unnamedFunction = function () {  
    return 7;  
}  
  
function globalFunction() {  
    return 33;  
}
```

```
> namedFunction  
< function aName()  
  
> namedFunction.name  
< "aName"  
  
> unnamedFunction  
< function unnamedFunction()  
  
> unnamedFunction.name  
< ""  
  
> globalFunction  
< function globalFunction()  
  
> globalFunction.name  
< "globalFunction"
```



# JavaScript Function: createApp

- *createApp* is a global function that returns an object
- Objects have key-value pairs
- The object **key** is a member of
- The object **value** is the actual value of the key
  - value can be anything (i.e function, object, array, primitive type)

```
3 //app to draw polymorphic shapes on canvas
4 var app;
5
6 function createApp(canvas) {
7     var c = canvas.getContext("2d");
8
9     let drawCircle = function(x, y, radius, color) {
10         c.fillStyle = color;
11         c.beginPath();
12         c.arc(x, y, radius, 0, 2 * Math.PI, false);
13         c.closePath();
14         c.fill();
15     };
16
17
18     let clear = function() {
19         c.clearRect(0,0, canvas.width, canvas.height);
20     };
21
22     return {
23         drawCircle: drawCircle,
24         clear: clear
25     }
26 }
```



# JavaScript Data Types

---

- boolean (true, false)
- null
- undefined
- number
- String
- Object
  - Array
  - Function



# JavaScript Objects

- No need to create new `Object()`
- Outside of primitives, everything is an object
- View expects controller to always return an JavaScript **object**

```
> var a = { foo: "bar" }
< undefined
> a
< {foo: "bar"} i
  foo: "bar"
  __proto__: Object
> a.foo
< "bar"
> a["foo"]
< "bar"
> a.baz = "boo"
< "boo"
> a
< {foo: "bar", baz: "boo"}
```

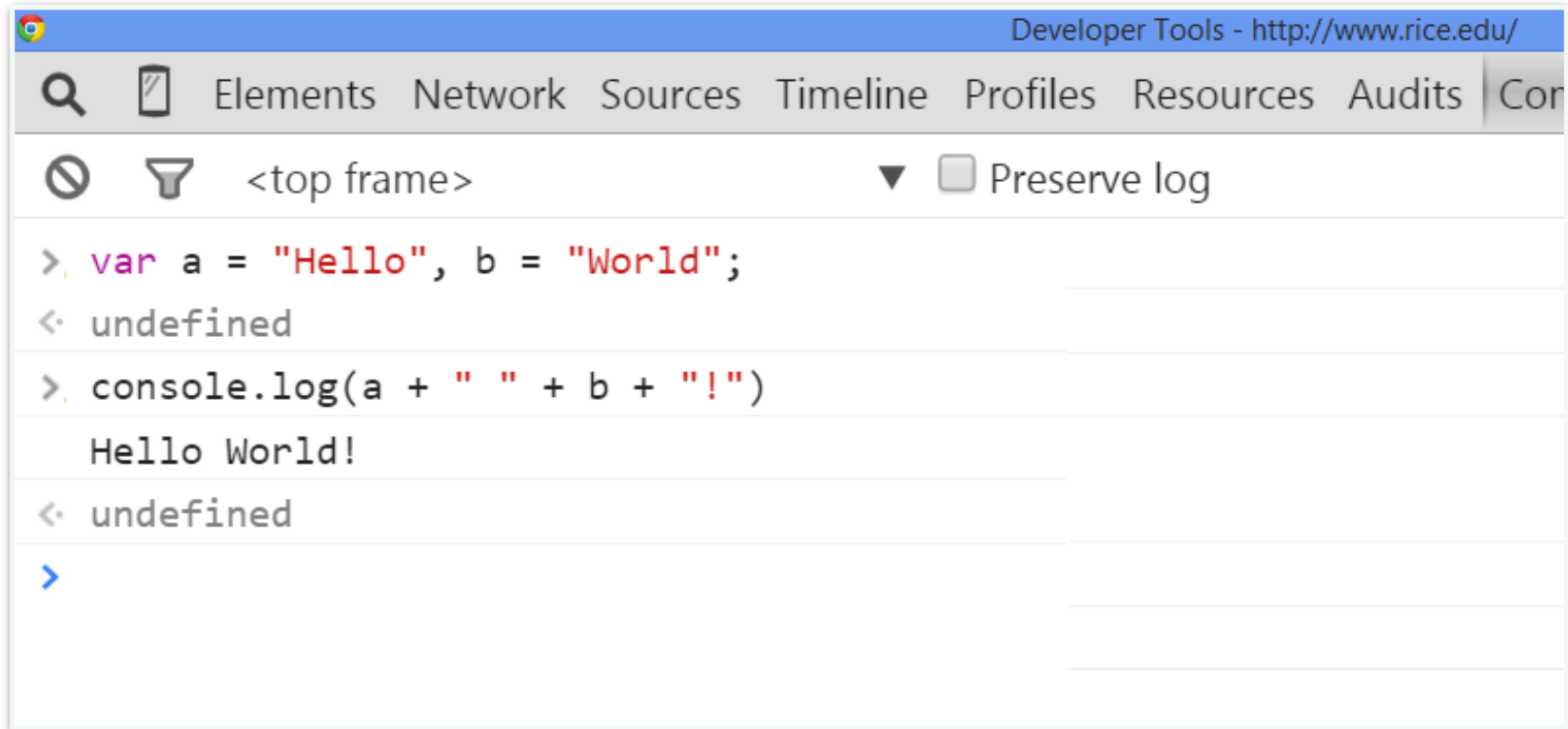
[createCircle in view.js](#)

```
36  /**
37   * Create a circle at a location on the canvas
38   */
39  function createCircle() {
40      $.get("/shape/circle", function (data) {
41          console.log("data is " + data);
42          // TODO: expect something like this for circle
43          app.drawCircle(100, 100, 25, "red");
44      }, "json");
45  }
--
```





# Chrome Developer Tools Console



# Interval

Used to create periodic executions (time between interval executions is not guaranteed)

```
> var writeDom = function() {  
    document.writeln('<tr><td>Another row</td><tr>')  
}  
< undefined  
> document.writeln('<table>')  
< undefined  
> setInterval(writeDom, 1500)  
< 1
```

Another row  
Another row  
Another row  
Another row  
Another row  
Another row



# JavaScript Arrays

- No need to create new Array()
- Array traversal
  - for-in provides index values
  - forEach provides the values themselves

```
> var a = [ 24, "bar", 42 ]  
< undefined  
  
> var sum = 0  
< undefined  
  
> for (var i in a) { sum += a[i]; }  
< "24bar42"
```




# Array forEach

`array.forEach(function(element) { .... })`

```
> var a = [1, 2, 3, 4, 5];
```

element will represent  
each value in array



```
> a.forEach(function(element) { console.log(element + element); });  
2  
4  
6  
8  
10
```

```
> a.forEach(function(element) { console.log(element * element); });  
1  
4  
9  
16  
25
```



# Array *forEach*

---

- Controller returns array of balls to draw
- Array (data) *forEach* accesses each element

localhost:4567/update: [{"loc":{"x":465,"y":122},"color":"blue",...}, {"loc":{"x":501,"y":534},"color":"blue",...}]

localhost:4567/update: [{"loc":{"x":466,"y":122},"color":"blue",...}, {"loc":{"x":502,"y":534},"color":"blue",...}]

localhost:4567/update: [{"loc":{"x":467,"y":122},"color":"blue",...}, {"loc":{"x":503,"y":534},"color":"blue",...}]

localhost:4567/update: [{"loc":{"x":468,"y":122},"color":"blue",...}, {"loc":{"x":504,"y":534},"color":"blue",...}]



# Updating Object Collection

---

- May add/remove objects listening for property change
- View's *setInterval* function can determine how frequent updates will occur
- Template design pattern is useful to separate variant and invariant listener *update* behavior



# Announcements & Reminders

---

- HW #1 is due **Fri, Sep 4th by 11:59pm**
- Use Piazza (public or private posts, as appropriate) for all communications re. COMP 504
  - Do not include code in a public post (**could be considered an honor code violation**)
- See course web site for syllabus, work assignments, due dates, office hours schedule.

