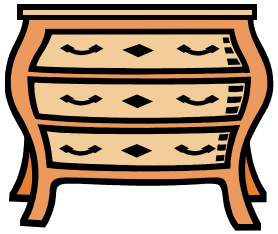
The University Of Sheffield.

COM1008: Web and Internet Technology

Lecture 13: JavaScript ‘classes’, constructors and prototypes



Dr. Steve Maddock
s.maddock@sheffield.ac.uk

1. Introduction: what is an object?

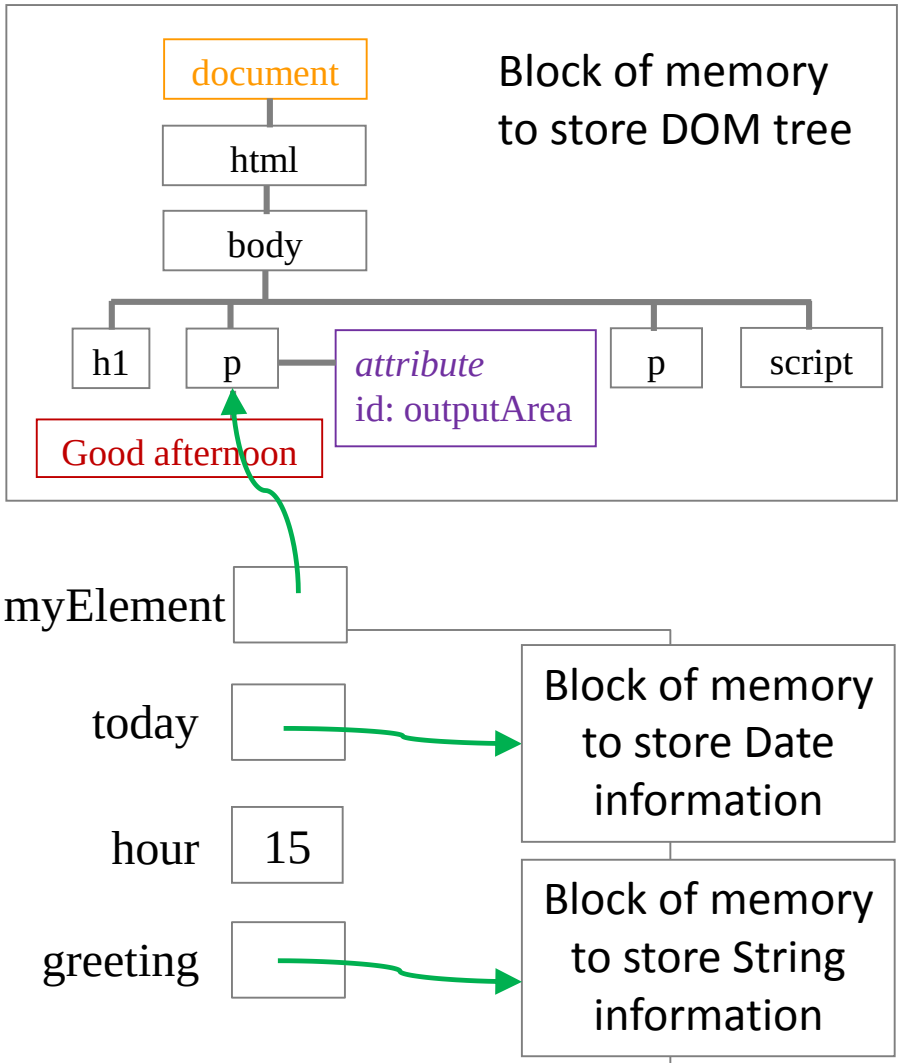
- In computer programming, an **object** can be used to represent a physical thing
- *Example*: A car
- An object is a collection of **properties** and **methods**
- Properties are name-value pairs
  - **Key-value pairs**
  - Data fields
- Methods can be used to set, query and change the properties
  - Ways to access the data fields
- We create **instances** of the car object

***Object type*: car**
Properties *make*: Ford *currentSpeed*: 30mph *colour*: blue *fuel*: petrol **Methods** setSpeed() getSpeed() changeSpeed()

***Object type*: car**
Properties *make*: VW *currentSpeed*: 50mph *colour*: silver *fuel*: diesel **Methods** setSpeed() getSpeed() changeSpeed()

1. Introduction: use of existing objects

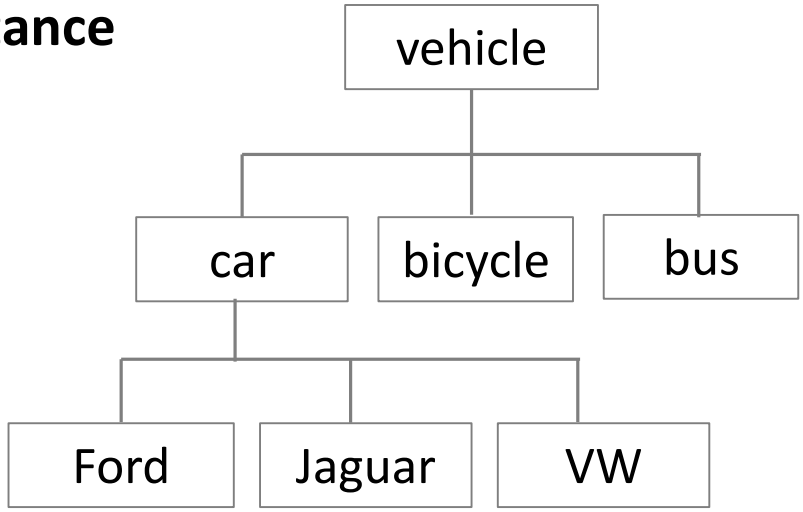
- **new** keyword to create an instance of an object
- **dot notation** is used to refer to methods and properties




```
var today = new Date();
var hour = today.getHours();
var greeting = "";
if (hour < 12) {
  greeting = "Good morning";
}
else if (hour < 18) {
  greeting = "Good afternoon";
}
else {
  greeting = "Good evening";
}
var myElement = document.getElementById('outputArea');
myElement.textContent = greeting;
```

1. Introduction: writing your own objects

- JavaScript is an ‘object oriented programming language’
- JavaScript uses **prototyping**
  - Java uses **classes**
  - ECMAScript 6 (2015) introduces classes, supported in some recent browsers using strict mode
- We will only briefly touch on **inheritance**
  - A detailed consideration is beyond the scope of this module



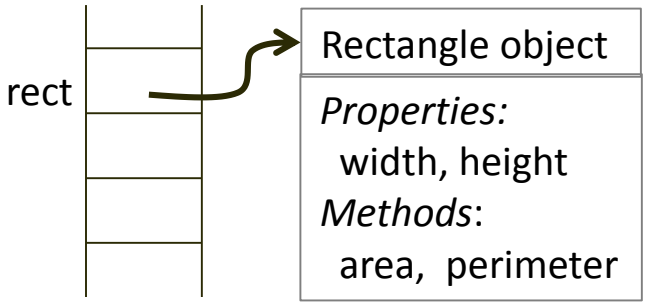
2. Objects in JavaScript

There are several ways to create objects in JavaScript:

- Object literal
  - Easiest, but suited to single objects
- Using the Object() constructor function
- Using object constructors
  - Creating multiple objects using a template
- Using prototyping
  - More advanced approach supporting inheritance
- *Example*: a rectangle object

Rectangle object
Properties: width, height *Methods*: area perimeter

3. Object literal (Object initializer)



- This is the easiest way to create an object
- But, tedious to create multiple objects

```
var rect = {
  width: 3,
  height: 4,
  area: function() {
    return this.width * this.height;
  },
  perimeter: function() {
    return 2 * (this.width + this.height);
  }
};
```

Use a colon between a key and its value.
Use a comma between each property and method

The keyword **this** refers to 'this instance of the object'

```
console.log("Width= " + rect.width);
console.log("Area= " + rect.area());
```

demo

3. Object literal

- Can also access the properties (but not methods) using **square bracket syntax**

```
var rect = {
  width: 3,
  height: 4,
  area: function() {
    return this.width * this.height;
  },
  perimeter: function() {
    return 2 * (this.width + this.height);
  }
};
```

```
console.log("Width= " + rect["width"]);
console.log("Area= " + rect.area());
```

demo

4. Using the Object() constructor function

- Create a blank object using the **new** keyword and the **Object()** constructor function
- Then add properties and methods using dot notation

```
var rect = new Object();
rect.width = 3;
rect.height = 4;
rect.area = function() {
  return this.width * this.height;
};
rect.perimeter = function() {
  return 2 * (this.width + this.height);
};
```

```
console.log("Width= " + rect.width);
console.log("Area= " + rect.area());
```

demo

5. Creating many objects

```
function Rectangle(width, height) {
  this.width = width;
  this.height = height;
  this.area = function() {
    return this.width * this.height;
  };
  this.perimeter = function() {
    return 2 * (this.width + this.height);
  };
}
```

Convention:

- **first character of class name is uppercase**
- first character of property/method name is lowercase

```
var rect1 = new Rectangle(3,4);
console.log("Width= " + rect1.width);
console.log("Area= " + rect1.area());
var rect2 = new Rectangle(5,6);
console.log("Width= " + rect2.width);
console.log("Area= " + rect2.area());
```

- Multiple objects can now easily be created using the **new** keyword and the **Rectangle** object constructor function
- But wasteful on memory

demo

6. Encapsulation

```
function Rectangle(width, height) {
  this.width = width;
  this.height = height;
  this.area = function() {
    return this.width * this.height;
  };
  this.perimeter = function() {
    return 2 * (this.width + this.height);
  };
}
```

- Bundling of data and methods
- Restricting access to some of the object's components – not satisfied in JavaScript

- In JavaScript, properties can be accessed directly, e.g. rect.width;

```
var rect1 = new Rectangle(3,4);
console.log("Width= " + rect1.width);
console.log("Area= " + rect1.area());
var rect2 = new Rectangle(5,6);
console.log("Width= " + rect2.width);
console.log("Area= " + rect2.area());
```

demo

6. Encapsulation

```
function Rectangle(width, height) {
  this.width = width;
  this.height = height;
  this.getWidth = function() {
    return this.width;
  };
  this.setWidth = function(width) {
    this.width = width;
  };
  this.area = function() {
    return this.width * this.height;
  };
}
```

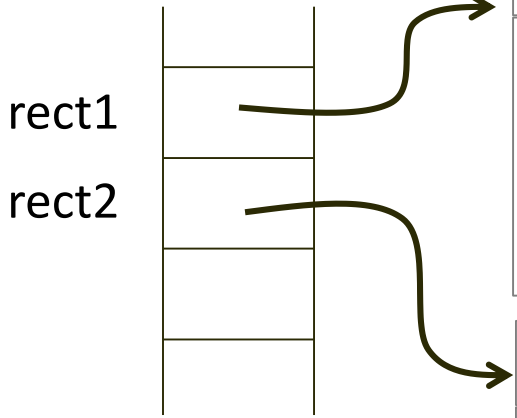
- We can encourage better encapsulation in JavaScript by writing methods to access the properties

```
var rect1 = new Rectangle(3,4);
console.log("Width= " + rect1.getWidth());
console.log("Area= " + rect1.area());
```

demo

7. Adding extra properties and methods

- Extra properties can be added to an object during program execution
- Properties can also be removed using the keyword **delete**



Rectangle object, rect1

Properties:
~~**width**~~, height
Methods:
area, perimeter

Rectangle object, rect2

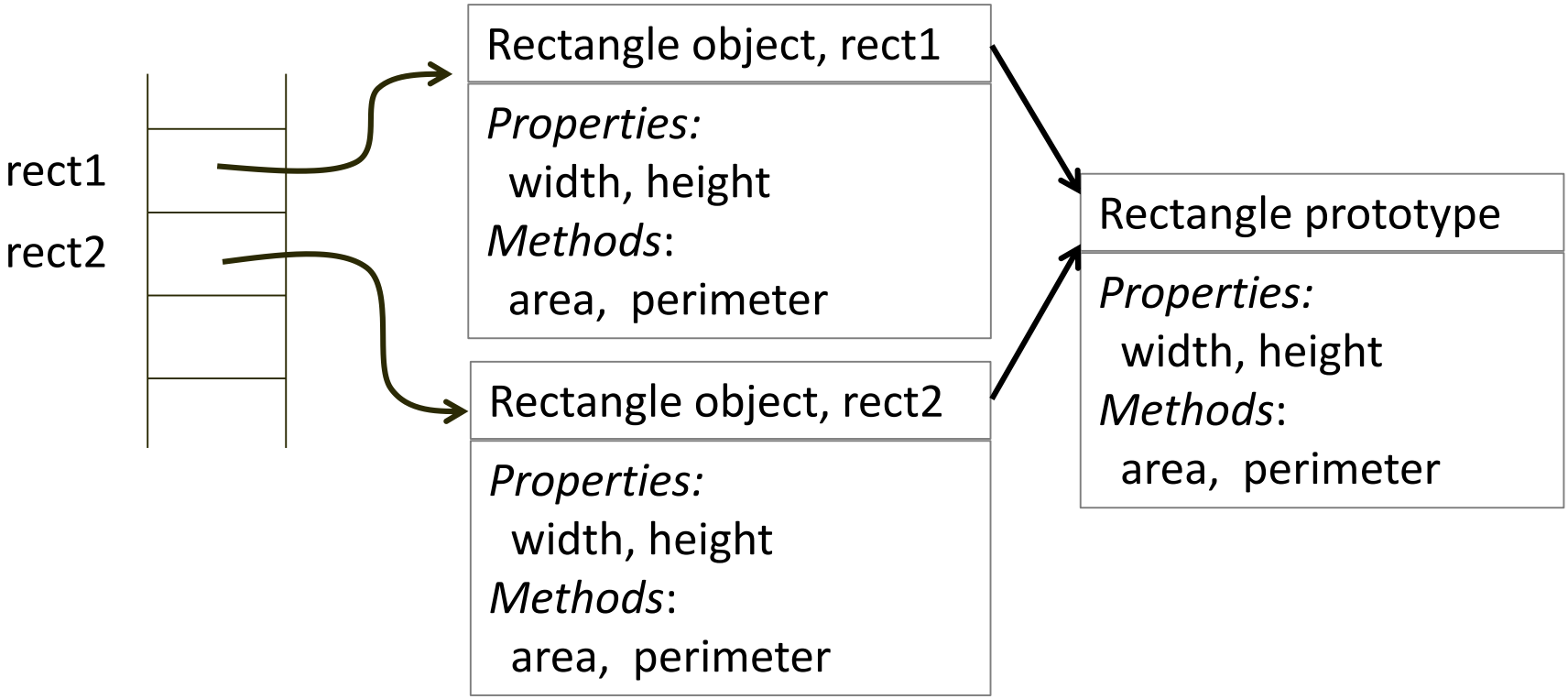
Properties:
width, height, **name**
Methods:
area, perimeter

```
var rect1 = new Rectangle(3,4);
var rect2 = new Rectangle(5,6);
rect2.name = "Kitchen floor";
console.log(rect1);
console.log(rect2);
delete rect1.width;
console.log(rect1);
console.log(rect2);
```

demo

8. Prototypes

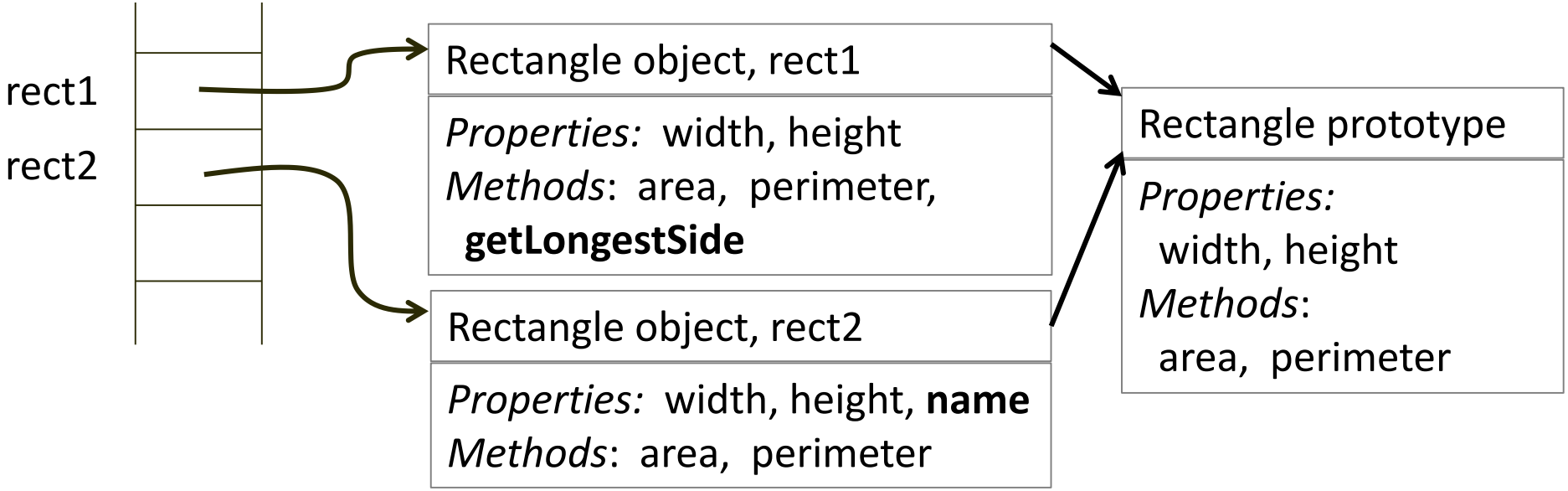
- Every JavaScript object has a prototype, from which it inherits its properties and methods
- Can be used to implement inheritance



8.1 Adding extra properties and methods

- Extra properties can be added to an object during program execution
- Would be better to add them to the prototype

```
var rect1 = new Rectangle(3,4);
var rect2 = new Rectangle(5,6);
rect2.name = "Kitchen floor";
rect1.getLongestSide = function { // function body };
```



8.2 The constructor function

- Instead of this version

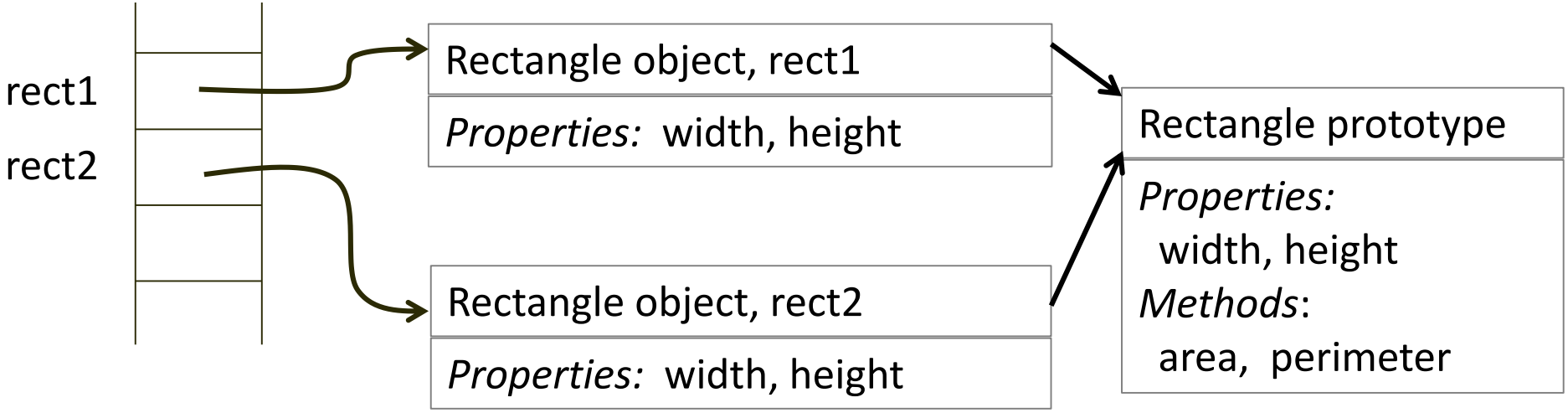
```
function Rectangle(width, height) {
  this.width = width;
  this.height = height;
  this.area = function() {
    return this.width * this.height;
  };
}
```

- Use this version

```
function Rectangle(w,h) {
  this.width = w;
  this.height = h;
}

Rectangle.prototype.area = function() {
  return this.width * this.height;
}
```

8.3 Preferred version

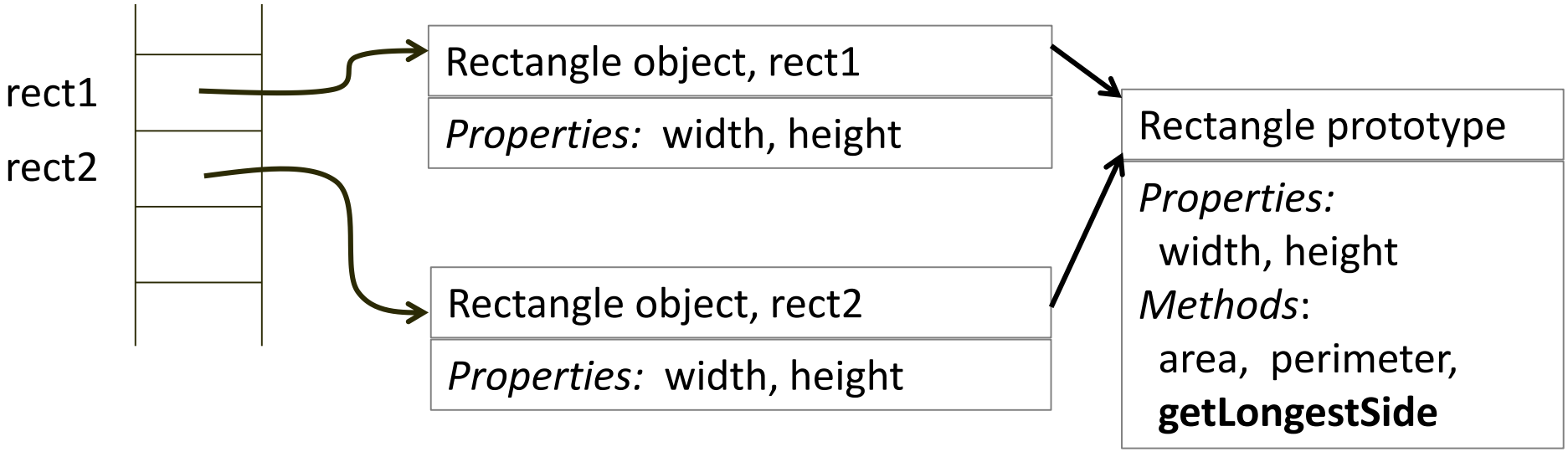
```
function Rectangle(w,h) {
  this.width = w;
  this.height = h;
}

Rectangle.prototype.area = function() {
  return this.width * this.height;
}
```



demo

8.3 Preferred version

```
function Rectangle(w,h) { this.width = w; this.height = h; }

Rectangle.prototype.area = function() {
  return this.width * this.height;
}
Rectangle.prototype.getLongestSide = function() {
  // function body
}
```



8.4 Encapsulation

- We could define methods to retrieve and set the properties:

```
function Rectangle(w,h) {
  this.width = w;
  this.height = h;
}

Rectangle.prototype.getWidth = function() {
  return this.width;
}

Rectangle.prototype.setWidth = function(w) {
  this.width = w;
}

Rectangle.prototype.area = function() {
  return this.width * this.height;
}
```

8.4 Accessing the properties

- The JavaScript default is public access for the properties
- Thus can still access properties directly:

```
var r = new Rectangle(3,4);
var w1 = r.getWidth();
var w2 = r.width;
```

You'll see both versions in use. Some people prefer the use of getWidth()

```
function Rectangle(w,h) {
  this.width = w;
  this.height = h;
}

Rectangle.prototype.getWidth = function() {
  return this.width;
}
```

This is called an instance method

Aside: Java

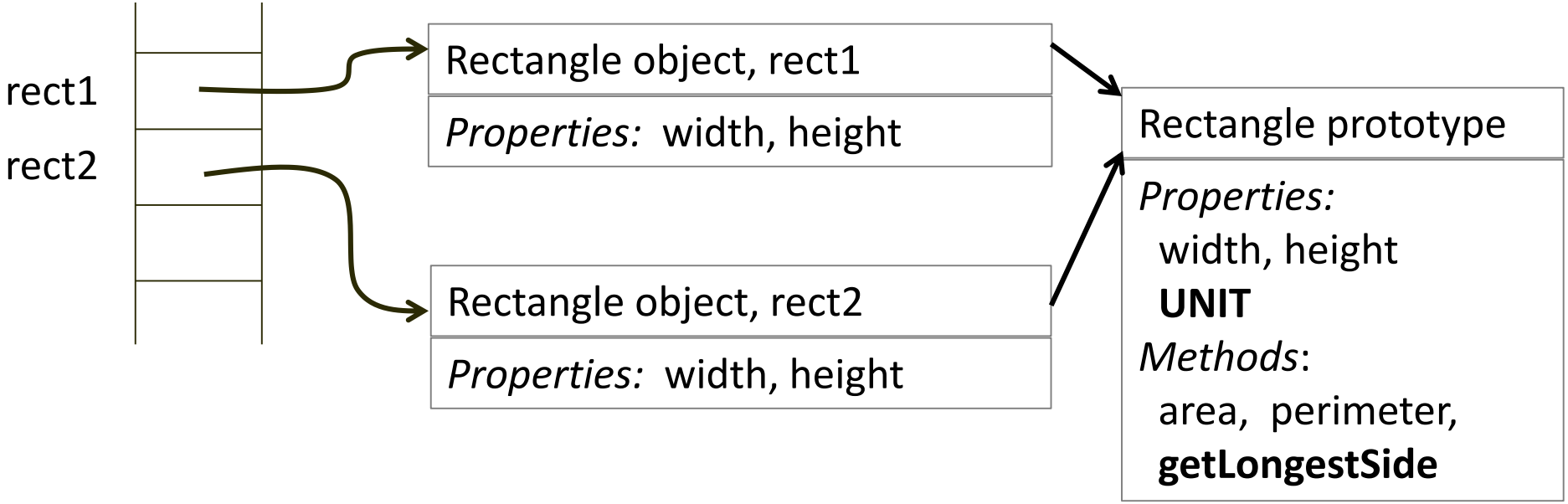
- **Java** is an object-oriented language
- Uses classes and objects
- For *public* methods of the object, use objectName.methodName
- For *public* attributes of the object, use objectName.attributeName
- However, attributes (i.e. fields) are usually *private*

```
Meal curry = new Meal();          // curry is an instance of the class Meal
curry.setPrice(3.99);             // setPrice is a method of Meal
int cost = curry.getPrice();      // Get the price of the Meal curry and
                                  // store it in the variable subtotal

Person p = new Person();          // p is an instance of the class Person
p.setAge(18);                     // setAge is a method of Person
int age = p.getAge();             // Get the age of the Person p and store it
                                  // in the variable age
```

8.5 'Class' properties

- A property can be added to the prototype to create a 'class' property

```
var rect1 = new Rectangle(3,4);
var rect2 = new Rectangle(5,6);
Rectangle.UNIT = new Rectangle(1,1);
var a = rect1.area();
console.log(a);
var b = Rectangle.UNIT.area();
console.log(b);
```



8.6 'Class' methods

- A method can be added to the prototype to create a 'class' method

```
function Rectangle(w,h) {
  this.width = w;
  this.height = h;
}

Rectangle.max = function(a,b) {
  if (a.area() > b.area()) return a;
  else return b;
}
```

```
var rect1 = new Rectangle(2,3);
var rect2 = new Rectangle(1,8);
var bigger = Rectangle.max(rect1,rect2);
```

8.7 toString()

- Every object has a toString() method (inherited from the global Object)
- It should be overridden in the creation of a new ‘class’

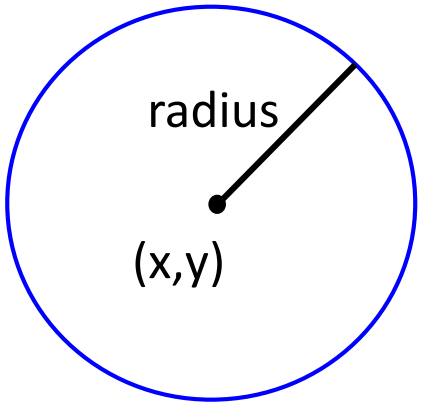
```
function Rectangle(w,h) {
  this.width = w;
  this.height = h;
}

Rectangle.prototype.toString = function() {
  return ("(" + this.width + "," + this.height + ")");
}
```

```
var rect1 = new Rectangle(2,3);
console.log(rect1);
```

demo

9. A ‘class’ for a Ball object

- We wish to create a picture containing lots of moving balls
- Start with an object that represents a ball, which will have an (x,y) position for its centre, a radius and a colour
- Properties: x, y, r, colour
- Methods: setX, getX,… draw,…

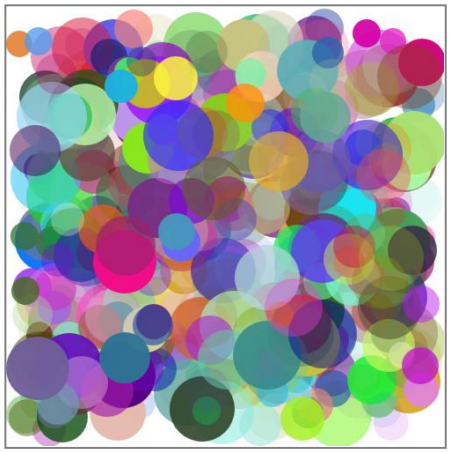


9. A ‘class’ for a Ball object

- This follows a similar pattern to the Rectangle object
- A constructor function is defined, followed by a series of prototype methods

```
function Ball(x,y,r,c) {
  this.x = x;
  this.y = y;
  this.r = r;
  this.c = c;
}

Ball.prototype.setX = function(x) {
  this.x = x;
}
```

10. A collection of Balls



- We need to maintain a list of balls.
- Solution: A BallList ‘class’
- Properties: numBalls, array of Balls
- Methods: getNumBalls, etc…

```
BallList.MAX_NUMBALLS = 500;

function BallList() {
  this.balls = new Array(BallList.MAX_NUMBALLS);
  this.numBalls = 0;
}

BallList.prototype.getNumBalls = function() {
  return this.numBalls;
}
```

10.1 Using the collection

- The following loop creates a collection of Balls

```
var balls = new BallList();

for (var i=0; i < NUM_BALLS; ++i) {
  var r = ???  // some radius
  var sx = ??? // some width
  var sy = ??? // some height
  var c = ???  // some random colour

  var ball = new Ball(sx,sy,r,c);
  balls.add(ball);
}
```

10.2 Adding and getting individual Balls

- We need methods to add a Ball to the list and to get a specific Ball

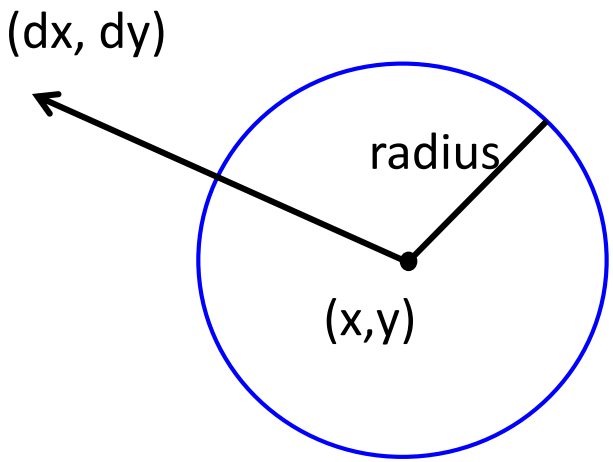
```
BallList.prototype.add = function(b) {
  if (this.numBalls >= BallList.MAX_NUMBALLS) {
    console.log("too many balls");
    return;
  }
  this.balls[this.numBalls] = b;
  this.numBalls++;
}

BallList.prototype.get = function(i) {
  if (i >= this.numBalls) return 0;
  return this.balls[i];
}
```

Basic error checking

Assume i is positive

10.3 Further properties and methods

- Properties: direction and speed of travel
  - Per frame of animation: x+=dx; y+=dy;
  - Rebound off walls
- Methods: draw a Ball, draw a list of Balls
- See next week's lab sheet



11. Summary

- JavaScript is an object-oriented language
- In JavaScript, we use a constructor function and prototype to create 'classes' and objects
  - ECMAScript 6 (2015) introduces classes; "use strict";
- A detailed consideration of inheritance is beyond the scope of this module
- *More details:*
  - Flanagan, D. "JavaScript: The Definitive Guide", O'Reilly Books, 2011
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects
- *Next lecture:* Graphics on the Canvas

```
"use strict";

class MyRectangle {
  constructor(width, height) {
    this.width = width;
    this.height = height;
  }

  getWidth() {
    return this.width;
  }

  area() {
    return this.width * this.height;
  }

  toString() {
    return '(' + this.width + ', ' + this.height + ')';
  }
}

// main program body
var rect1 = new MyRectangle(3,4);
var rect2 = new MyRectangle(5,6);
console.log("Area = " + rect1.area());
```