Kinds of Inheritance
- examples in **Java** and JavaScript

- Object-oriented language based on ***prototypes:***
  - No distinction between Classes and object instances
  - JavaScript - is now officially called ECMAScript:
    - ECMA stands for **E**uropean **C**omputer **M**anufacturers **A**ssoctn
- **Prototypical object** - an object used as a template for the initial properties of a new object.
- Any object (instance) can be used as the **prototype** for another object
  - Allows the second object to 'share' the first object's **properties** and **<u>values</u>**.
- Ref: http://developer.netscape.com/docs/manuals/communicator/jsobj/index.htm

Matthew Morgenstern 1 CS211 Accel/Proj - Sept. 13 & 14, 2000

Instance / Value Inheritance (not Java)

- Object **inherits *value dynamically*** from prototype/parent ***instance***
  - Specify **prototype chain** for **inheritance** of property *value* and property *definition* - Note: **".prototype"** :
    - **Manager.prototype = Employee**
      - Specifies that Manager inherits from Employee
    - **Employee.prototype.WorkAddress = value**
      propagates updated value of Property **WorkAddress** to Manager

Unless:

- ***Local*** **property value at Creation Time of object:**
  - **Constructor** creates local property (default or ***inherit current*** **value**)**.**
  - **Manager.WorkAddress = value** - creates or updates local copy.

Matthew Morgenstern 2 CS211 Accel/Proj - Sept. 13 & 14, 2000

Instance / Value Inheritance (not Java) - Example

- **Employee1.WorkAddress = “Ithaca”**
  Employee1.Dept = “CS”
  - sets values of properties for Employee1 object instance.
- **Manager.prototype = Employee1**
  - designate Employee1 object as **prototype** for Manager
  - establishes inheritance path: *prototype chain*
  - When you create a new Manager, it ‘inherits’ the WorkAddress and dept properties *and values* from that Employee1 object.
  - so **Manager.WorkAddress** *has-value* **”Ithaca”**
- **Employee.prototype.WorkAddress = “Cornell”**
  - dynamic value inheritance from Employee prototype to Manager:
  - so **Manager.WorkAddress** *has-value* **”Cornell”**

Dynamic Type Specification & Propagation (not Java)

- **Dynamic Type Specification:**
  (can exist with or without instance inheritance)
  - An object can specify and add property **definitions**
    - and can do so dynamically **even at runtime**.
      – **Employee1.prototype.bldg = “Upson”**
  - Object (instance) can be created *without any prior definitions*:
    objectName = { property1:value1, property2:value2,...,
    property*N*:value*N* }
- **Dynamic change propagation** of **property definitions** and **values:** (can exist with or without instance inheritance)
  - If you add a property to an object that is used as the **prototype** for a set of objects, the objects for which it is the prototype also get the new property and value:
  - **Employee1.prototype.bldg = “Upson”**
    causes **Manager.bldg** *to have value* **“Upson”**

Other Forms of Inheritance (not Java)

- **Selective Inheritance**:
  - Once a **prototype property inheritance chain** is established,
    B.prototype = A
    the inheritance of properties and values from **A** to **B** is (somewhat) **selective** in JavaScript:
    - only for those properties *not* defined *locally* by **B**.
- **Selective Inheritance** in general would allow:
  (artificial syntax):
  - Selective property **definition** inheritance:
    Manager.inherits = [Emp.Dept, Emp.Bldg]
  - Selective property **value** inheritance:
    Manager.inherits = [emp1.Dept, emp1.Bldg]

Comparison of class-based (Java) and prototype-based (JavaScript) object systems

Class-based (Java)	**Prototype-based (JavaScript)**
Class and instance are distinct entities.	All objects are instances.
Define a class with a class definition; instantiate a class with constructor methods.	Define and create a set of objects with constructor functions.
Create a single object with the `new` operator.	Same.
Construct an object hierarchy by using class definitions to define subclasses of existing classes.	Construct an object hierarchy by assigning an object as the prototype associated with a constructor function.
Inherit properties by following the class chain.	Inherit properties by following the prototype chain.
Class definition specifies *all* properties of all instances of a class. No way to add properties dynamically at runtime.	Constructor function or prototype specifies an *initial set* of properties. Can add or remove properties dynamically to individual objects or to the entire set of objects.

Dimensions for different kinds of Inheritance

- Time/when:
  - – Program Construction Time - hard-coded , typical
  - – Class Declaration / Creation time - parameterized Class defn
  - – Instance Creation time - properties defined at creation time
  - – Instance Access time - properties updated at access time
- Inheritance of:
  - Property Definition vs
  - Value/instance inheritance
- Change Propagation
  - Can changes be made and at what stages
  - When do changes propagate.

A Challenge:

- Come up with one or more ways of accomplishing these capabilities, *or similar*, in Java:
  - IF no way to accomplish, explain why.
  - state the limitations / compromises (expected)
  - what objective are achieved / supported
- **Multiple Inheritance:** inherit from 2 or more large predefined classes
  - Are templates enough? Simplifying the "**re**implementation" w/in each class to satisfy the template.
- Optional, may be submitted - next week - extra credit
- General discussion, next week or following week
  - You are encouraged to discuss your ideas