1.00 Lecture 8

Classes, continued

Reading for next time: Big Java: sections 7.9

Using An Existing Class, cont.

- **From last time:**
- **BusRoute is a Java class used by the BusTransfer class**
- **BusTransfer uses BusRoute objects:**
  - **First construct the objects and specify their initial state**
    - **Constructors are special methods to construct and initialize objects**
    - **They may take arguments (parameters)**
  - **Then apply methods to the objects**
    - **This is the same as "sending messages" to them to invoke their behaviors**

Constructor for BusRoute Object

- To construct a new BusRoute object, two things are required:
  - Create the object (using its constructor)

```
new BusRoute(1, 300, 80.0);  // Use original example
// 'new' allocates memory and calls constructor
```

  - Give the object a name or identity:

```
BusRoute bus1;
// Object name (bus1) is a reference to the object
// BusRoute is the data type of bus1
```

  - Combine these two things into a single step:

```
BusRoute bus1= new BusRoute(1, 300, 80.0);
```

  - We now have a BusRoute object containing the values:
    - Route number 1
    - Number of passengers 300
    - Percent transferring 80.0
  - We can now apply methods to it.

Using Methods

- Methods are invoked using the dot (.) operator
  - Method always ends with parentheses

```
BusRoute bus1= new BusRoute(1, 300, 80.0);
BusRoute bus2= new BusRoute(47, 400, 30.0);
int r1= bus1.getRteNumber();       // Dot operator
int p2= bus2.getPassengers();      // Dot operator
bus1.setPassengers(p2+100);        // Dot operator
```

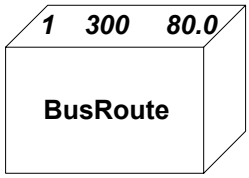
  - Methods are usually public and can be invoked anywhere
- Data fields are also invoked with the dot (.) operator
  - No parentheses after field name

```
int j= bus1.rteNumber;       // Syntax ok
                             // but won't compile
```

  - Private data fields can't be accessed outside their class
    - None of the data fields in our bus example can be accessed this way because they're all private

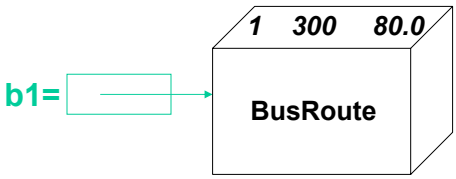
Objects and Names

```
new BusRoute(1, 300, 80.0);
```



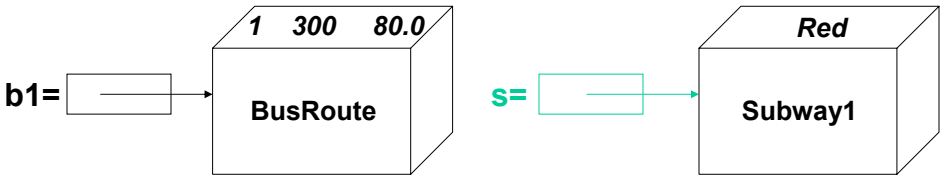
Objects and Names

```
BusRoute b1= new BusRoute(1, 300, 80.0);
```



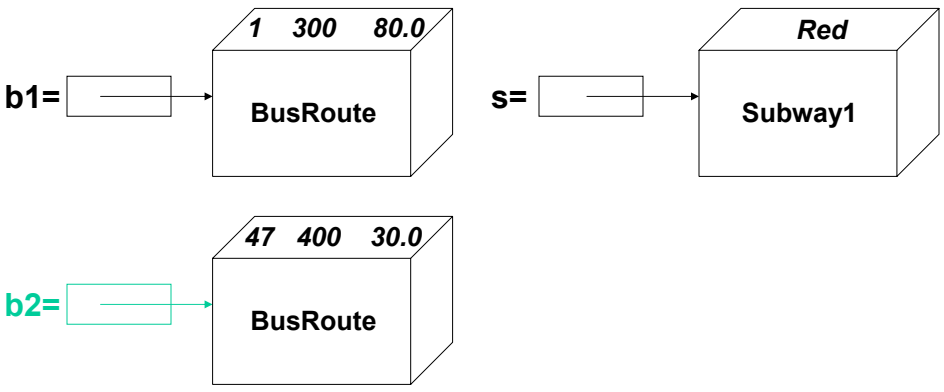
Objects and Names

```
BusRoute b1= new BusRoute(1, 300, 80.0);
Subway1 s= new Subway1("Red");
```



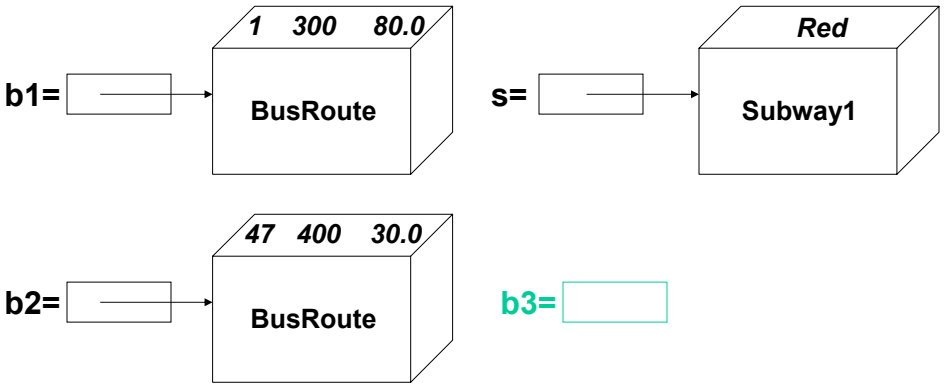
Objects and Names

```
BusRoute b1= new BusRoute(1, 300, 80.0);
Subway1 s= new Subway1("Red");
BusRoute b2= new BusRoute(47, 400, 30.0);
```



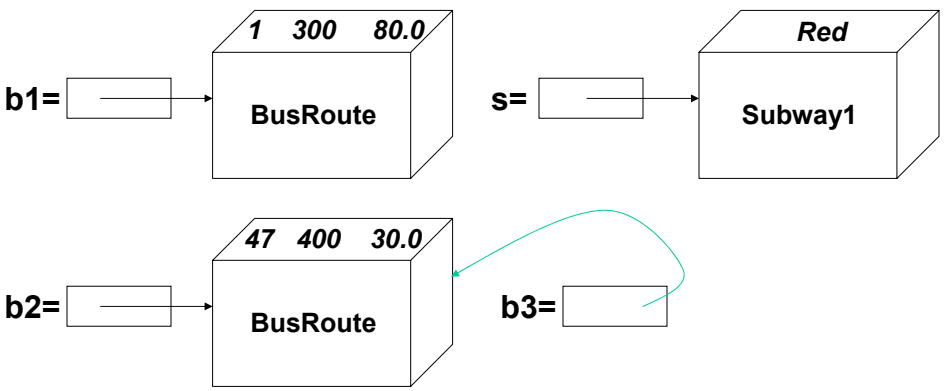
Objects and Names

```
BusRoute b1= new BusRoute(1, 300, 80.0);
Subway1 s= new Subway1("Red");
BusRoute b2= new BusRoute(47, 400, 30.0);
BusRoute b3;
```



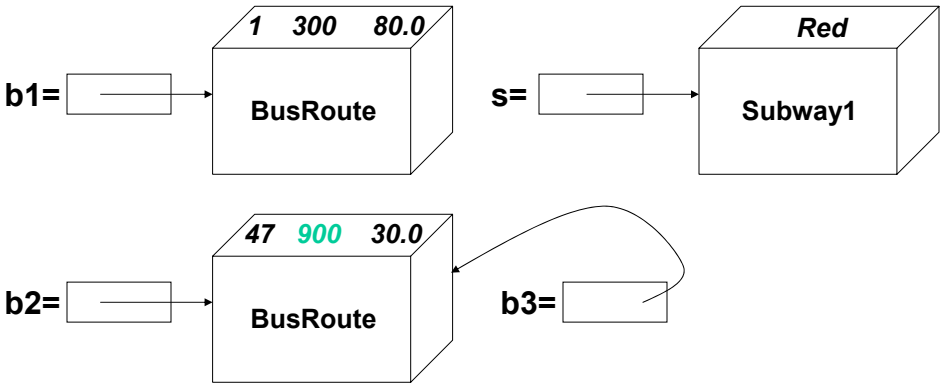
Objects and Names

```
BusRoute b1= new BusRoute(1, 300, 80.0);
Subway1 s= new Subway1("Red");
BusRoute b2= new BusRoute(47, 400, 30.0);
BusRoute b3;
b3= b2;
```



Objects and Names

```
BusRoute b1= new BusRoute(1, 300, 80.0);
Subway1 s= new Subway1("Red");
BusRoute b2= new BusRoute(47, 400, 30.0);
BusRoute b3;
b3= b2;
b3.setPassengers(900);  // Easy to do accidentally
```



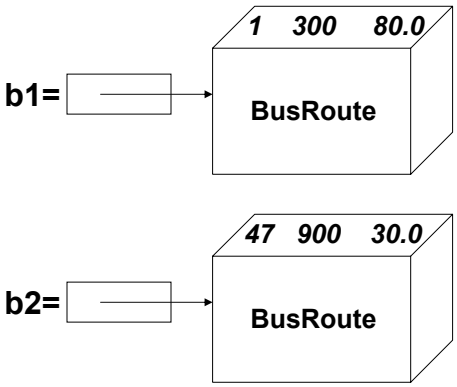
Subway class

```
public class Subway {
   private String name;
   private BusRoute2 bus1;     // Connecting bus route
   private BusRoute2 bus2;     // Connecting bus route
   public Subway(String n, BusRoute2 b1, BusRoute2 b2) {
        name= n;
        bus1= b1;
        bus2= b2;
   }
   public double getTransferPassengers(){
        return bus1.getConnectionPassengers() +
        bus2.getConnectionPassengers();
}}
```

```
public class SubwayTransfer {
   public static void main(String[] args) {
        BusRoute2 bus1 = new BusRoute2(1, 300, 80.0, 5);
        BusRoute2 bus2 = new BusRoute2(47, 400, 30.0, 10);
        Subway s= new Subway("Red", bus1, bus2);
        double transfer= s.getTransferPassengers();
        System.out.println("Subway psgrs: "+transfer);
}}
```

Draw the picture

```
// Assume b1 and b2 exist
Subway s= new Subway("Red", b1, b2);
```



Summary-classes

- Classes are a pattern for creating objects
- Objects have:
  - A name (reference, which is actually a memory location)
  - A data type (their class)
    - We generalize this later; objects can have many types
  - A block of memory to hold their member data, allocated by the 'new' keyword
  - Member data, usually private, whose values are set by their constructor, called when 'new' is used
    - Member data can be built-in data types or objects of any kind
  - Methods, usually public, to get and set all values
  - Methods to do some computation

Summary- constructors

- A constructor is a special method
  - Same name as the class
  - A class can have many of them, though each must have different arguments
  - Has no return value (never 'responds')
  - Generally sets all data members to their initial values
  - Implements the 'existence' behavior
  - Is called once when the object is first created with 'new' in a program that wants to use it
- Example

```
public class Flagpole {
        private double height;
        public Flagpole() {height= 100.0;}
        public Flagpole(double h) { height= h;}
        …
 }
```

Building Classes

- A window company has 3 plants
  - A makes wood frames
    - Produces 200 frames/day, unit cost $25/frame
  - B makes glass
    - Produces 200 panes/day, unit cost $5/pane
  - C, adjacent to B, assembles windows
    - Assembles 200 windows/day, unit assembly cost $12
- We'll write the classes for this problem
  - There are many alternatives; we guide you to use a straightforward one
  - This will not be a general solution. It will work only for one product, taking one frame and one pane of glass. It may seem awkward in places, but it's a typical starting point.
  - Use the 'spiral model' to make your solution more general in a second or third pass.

Plant Class

- **Write the class `Plant` for plants producing frames or glass. Ignore window assembly.**
- **Data fields:**

  _____
  _____
  _____

- **Constructor:**

  _____
  _____
  _____
  _____
  _____

Plant Class Methods

- **Don't write any "set" methods. The plant data is set by the constructor and we won't change it after that in this problem.**
- **"Get" methods, for each private field:**

  _____
  _____
  _____

- **Computational method**

  _____

Assembly Plant Class

- We assemble one product from two parts. Write class `Assembly`. Eclipse: New->Class Assembly
- Data fields:

  _____
  _____
  _____
  _____
- Constructor

  _____
  _____
  _____
  _____
  _____

Assembly Class Methods

- Don't write any "set" or "get" methods.
- Computational methods (cost, production)

  _____
  _____
  _____
  _____
  _____
  _____
  _____
  _____
  _____
  _____

Main()

- **Write a main() to create the three plants and to output the cost per product and the production rate.**