1.00 Lecture 8

Classes, continued

Reading for next time: Big Java: section 8.9

Building Classes, cont.

- From last time:
- Tank is a Java class used by the TankTest class
- TankTest uses Tank objects:
  - First construct the objects and specify their initial state
    - Constructors are special methods to construct and initialize objects
    - They may take arguments
  - Then apply methods to the objects
    - This is the same as sending messages to them to invoke their behaviors
    - The messages respond with their return values

Constructor for Tank Object

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

```
new Tank(0.5, 4.0, 0.04);    // Use original example
// 'new' allocates memory and calls constructor
```

  - **Give the object a name or identity:**

```
Tank tank0;
// Object name (tank0) is a reference to the object
// Tank is the data type of tank0
```

  - **Combine these two things into a single step:**

```
Tank tank0= new Tank(0.5, 4.0, 0.04);
```

  - **We now have a Tank object containing the values:**
    - **Radius 0.5 meters**
    - **Length 4.0 meters**
    - **Thickness 0.04 meters**
  - **We can now apply methods to it.**

Using Methods

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

```
Tank tank0= new Tank(0.5, 4.0, 0.04);
Tank tank1= new Tank(1.0, 1.0, 0.04);
double v= tank0.getVolume();      // Dot operator
double w= tank1.getWeldLength();  // 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**

```
double r= tank0.radius;
double t= tank0.thickness;
```

  - **Private data fields can t be accessed outside their class**
    - **The data fields in our Tank example cannot be accessed this way in TankTest because they're all private to Tank**
    - **If they were public in Tank, they could be seen from TankTest**
    - **Private fields can be accessed this way within class Tank**

Get() and Set() Methods

- We've seen get() methods
  - They ask an object to compute or return a fact about itself

```
public double getVolume() {
  return Math.PI*radius*radius*length;
}
public double getRadius() {return radius;}
```

- Set() methods tell an object to change one if its data members

```
public void setRadius(double r) {
     radius=  r;
}
```

Get() and Set() Methods 2

- We've seen get() methods
  - They ask an object to compute or return a fact about itself

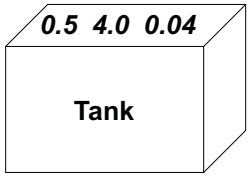
```
public double getVolume() {
  return Math.PI*radius*radius*length;
}
public double getRadius() {return radius;}
```

- Set() methods tell an object to change one if its data members

```
public void setRadius(double radius) {
     this.radius=  radius;
}  // 'this' is keyword for current object
```

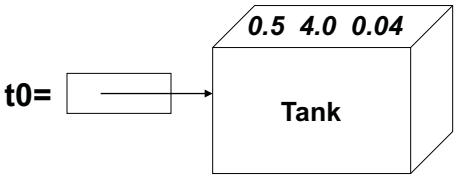
Objects and Names

```
new Tank(0.5, 4.0, 0.04);
```



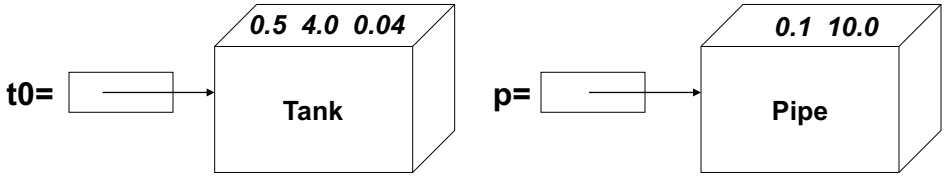
Objects and Names

```
Tank t0= new Tank(0.5, 4.0, 0.04);
```



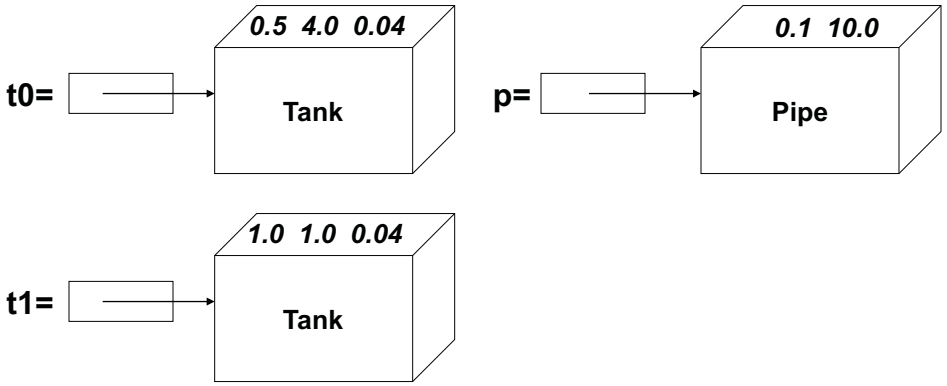
Objects and Names

```
Tank t0= new Tank(0.5, 4.0, 0.04);
Pipe p= new Pipe(0.1, 10.0);     // radius, length
```



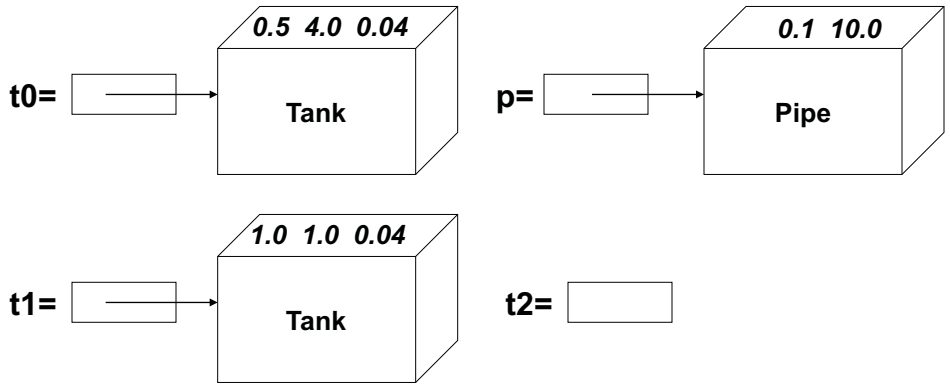
Objects and Names

```
Tank t0= new Tank(0.5, 4.0, 0.04);
Pipe p= new Pipe(0.1, 10.0);     // radius, length
Tank t1= new Tank(1.0, 1.0, 0.04);
```



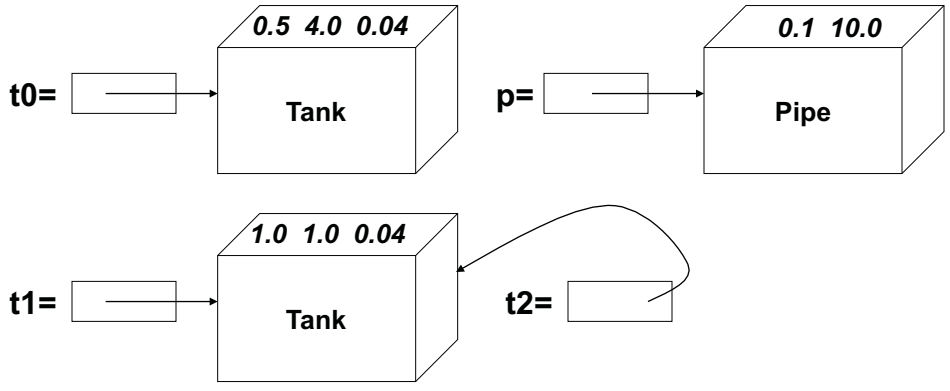
Objects and Names

```
Tank t0= new Tank(0.5, 4.0, 0.04);
Pipe p= new Pipe(0.1, 10.0);     // radius, length
Tank t1= new Tank(1.0, 1.0, 0.04);
Tank t2;
```



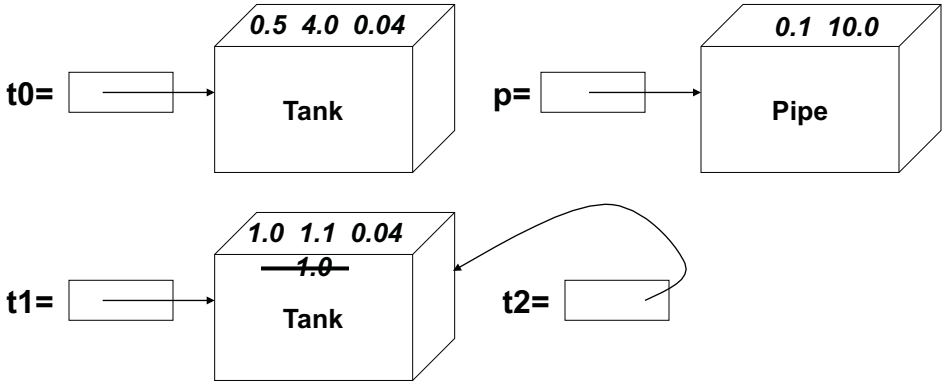
Objects and Names

```
Tank t0= new Tank(0.5, 4.0, 0.04);
Pipe p= new Pipe(0.1, 10.0);     // radius, length
Tank t1= new Tank(1.0, 1.0, 0.04);
Tank t2;
t2= t1;
```



Objects and Names

```
Tank t0= new Tank(0.5, 4.0, 0.04);
Pipe p= new Pipe(0.1, 10.0);     // radius, length
Tank t1= new Tank(1.0, 1.0, 0.04);
Tank t2;
t2= t1;
t2.setRadius(1.1);  // Easy to do accidentally
```



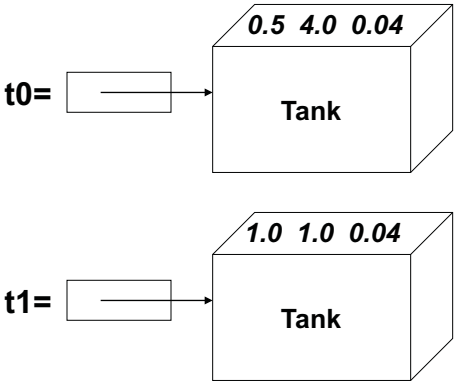
Pipe, Pipe2 class

```
public class Pipe {          // Simple Pipe class
   private double radius;
   private double length;
   public Pipe(double r, double len) {
        radius = r;
        length = len;
} }
```

```
public class Pipe2 {         // Pipe attached to two Tanks
   private double length;
   private double radius;
   private Tank tank1;       // Same Tank class as lecture 7
   private Tank tank2;
   public Pipe2(double len, double r, Tank t1, Tank t2) {
        length = len;
        radius = r;
        tank1 = t1;
        tank2 = t2;
   }
   public double getSystemVolume() {
        return tank1.getVolume() + tank2.getVolume() +
               length*Math.PI*radius*radius;
} }
```

Draw the picture

```
// Assume t0 and t1 exist
Pipe2 p2= new Pipe2(0.1, 10.0, t0, t1);
```



Class TankTest

```
public class TankTest {
   public static void main(String[] args) {
        Tank t0= new Tank(0.5, 4.0, 0.04);
        Tank t1= new Tank(1.0, 1.0, 0.04);
        System.out.println(t0.getVolume()+" "+t1.getVolume());

        Tank t2;
        t2= t1;
        t2.setRadius(1.1);     // Easy to do accidentally
        System.out.println(t0.getVolume()+" "+t1.getVolume());
        // Note that t1's volume changed

        Pipe2 p= new Pipe2(0.1, 10.0, t0, t1);
        double volume= p.getSystemVolume();
        System.out.println("System volume: " + volume);
   }
}       // Same Tank class as lecture 7. See download.
```

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 be 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
    - Member data is initialized to 0, 0.0 or false for primitive types
    - Member data is initialized to null (a keyword) for objects
  - Methods, usually public, to get and set member data
  - Methods, usually public, to do computation, using the member data

Summary- constructors

- A constructor is a special method
  - Same name as the class
  - 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
  - A class can have many constructors, though each must have different arguments. For example:

```
public class Tank {
     private double radius;
     private double height;
     public Tank() { height= 1.0; radius= 2.0;}
     public Tank(double h) { height= h; radius= 2.0}
     public Tank(double h, double r) {
         height= h; radius= r; }
     …
}
```

Building Classes

- A window company has 3 plants
  - Parts plant A makes wood frames
    - Unit cost $25/frame
  - Parts plant B makes glass
    - Unit cost $5/pane
  - Assembly plant C, adjacent to plant B, assembles windows
    - Unit assembly cost $12
  - How many classes? How many objects?
- 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 too restrictive, but it s a typical starting point.
  - Use the spiral model to make your solution more general in a second or third pass.

PartsPlant Class

- Write the class `PartsPlant` for plants producing one item: frames or glass. Ignore window assembly for now. This is a recipe for making a plant

```
public class PartsPlant {
// Data fields:
  _____
  _____
// Constructor:
  _____
  _____
  _____
  _____

 // This is all outside the main() method. PartsPlant doesn't
 // have and doesn't need main(). Delete it if you have one.
```

PartsPlant Class Methods

- Don't write any "set" methods. The parts plant data is set by the constructor and we won't change it after that in this problem.

  // Get methods, for each private field:

  _____
  _____
  _____
  _____

AssemblyPlant Class

- We assemble one product from parts produced by two `Plants`. Write class `AssemblyPlant`:
  Eclipse: New->Class: AssemblyPlant
- This is a recipe for making an AssemblyPlant

```
public class AssemblyPlant {
```

// Data fields: what is made, what parts plants are used, cost

_____
_____
_____
_____

// Constructor

_____
_____
_____

AssemblyPlant Class Methods

- Don't write any "set" or "get" methods other than the ones below.

  // Computational method (cost)

  _____
  _____
  _____
  _____
  _____
  _____
  _____
  _____

  // Get method to return name of product being assembled

  _____
  _____

main()

- In a new class, write a main() method to:
  - Create two parts plants. This uses their recipes to make objects
  - Create assembly plant. This uses its recipe to make an object
  - Find the cost of windows. Ask the AssemblyPlant object
  - Output window cost. Use System.out.println()
  - Output the name of the assembled product and its components. Ask the objects; then use System.out.println()

```
public class GlassTest {
  public static void main(String[] args) {
_____
_____
_____
_____
_____
_____
  }
}
```

MIT OpenCourseWare
http://ocw.mit.edu

1.00 / 1.001 / 1.002 Introduction to Computers and Engineering Problem Solving
Spring 2012

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.