

# 1.00 Lecture 7

## Strings Constructors this Overloading

### Strings

```
public class StringExample {
    public static void main(String[] args) {
        String first= "George ";
        String middle= "H.W. ";
        String last= "Bush";
        String full= first + middle + last;
        System.out.println(full);

        // Testing for equality in strings (objects in general)
        String full2= "George H.W. Bush";
        if (full.equals(full2)) // Right way
            System.out.println("Strings equal");
        if (full == full2) // wrong way
            System.out.println("A miracle!");
        if (first == "George ") // wrong way, but works
            System.out.println("Not a miracle!");

        // Modifying strings must be done indirectly-constant
        middle= middle.substring(0, 2) + " ";
        full= first + middle + last;
        System.out.println(full);    }    }
```

## Springs

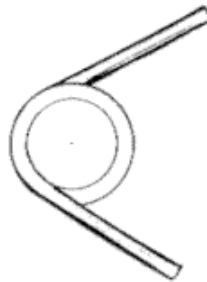
Compression



Tension



Torsion



$$F = k dx$$

## Spring Class Constructors

```
class Spring {
    private String material= "steel";           // Initialized
    private double length;                     // Compress only
    private double maxDeflect;
    private double k;

    public Spring(String m, double len, double md, double k) {
        material= m;
        length= len;
        maxDeflect= md;
        this.k= k;                             // "this"
    }
    public Spring(double len, double k) {
        this("steel", len, 0.25*len, k);       // "this"
    }
    public Spring(double len) {
        this(len, 0.5*len);
    }
    public Spring() {
        this(10.0);    }
}
```

## Spring Class Methods

```
public String getMaterial() { return material; }
public double getLength() { return length; }
public double getMaxDef() { return maxDeflect; }
public double getK() { return k; } // Don't need 'this'

public void setMaterial(String m) {material= m; }
public void setLength(double len) {length= Math.abs(len);}
public void setMaxDef(double m) {maxDeflect= Math.abs(m);}
public void setK(double k) {this.k= k; } // this

public double getForce(double deflection) {
    if (deflect > maxDeflect)
        deflect= maxDeflect;
    return k*deflect;
}
public double getForce() { // overloaded methods
    return k*maxDeflect;
}
}
```

## Spring Main

```
public class SpringExample {
    public static void main(String[] args) {
        Spring one= new Spring("aluminum", 2.0, 1.0, 5.0);
        Spring two= new Spring(5.0, 3.0);
        Spring three= new Spring(); // 3 diff constructors

        double f1= one.getForce(0.5);
        double f2= two.getForce(1.5);
        double f3= three.getForce(0.1);
        System.out.println("f1: " + f1 + "\nf2: " + f2 +
            "\nf3: " + f3);

        double f4= one.getForce(); // overloaded methods
        double f5= two.getForce();
        double f6= three.getForce();
        System.out.println("f4: " + f4 + "\nf5: " + f5 +
            "\nf6: " + f6);
        System.exit(0);
    }
}
```

## Spring Class Design

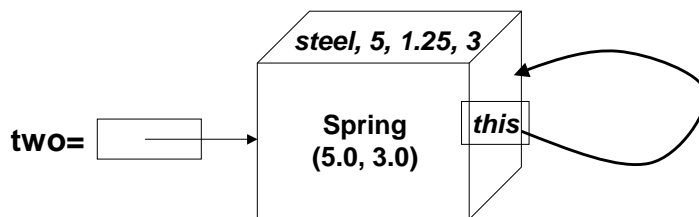
- **All the Spring methods are public**
  - Any method of any class can call these methods
  - Private methods can be used, as helpers or for tricky things that only the class itself should do
- **Data fields in Spring are private**
  - Only Spring methods can access them
  - Public data fields are almost never used
- **Constructor name must be same as class name**
  - Constructors are called with new only
  - Constructors cannot have a return value

## Spring Class Methods

- **Why have all these methods?**
  - Get methods are only access from other classes to Spring data
    - Allow us to reimplement Spring if we need
  - Set methods should do error checking
    - Our method should make sure that length>0, max deflection < length, etc.
- **Overloaded methods must have different signatures (arguments)**
  - Different return types cannot distinguish two otherwise identical methods
    - `int getForce(int a)`      **and**
    - `double getForce(int b)`    **would not compile**

## This

```
Spring two= new Spring(5.0, 3.0);
```

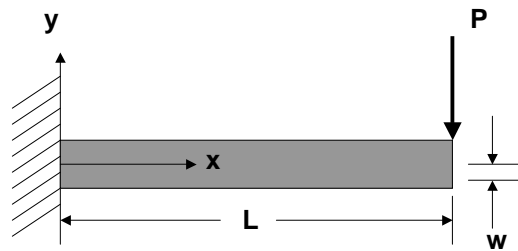


**“this” is also used as shorthand to refer to other constructors for the class**

## Object Destruction

- **Java reclaims object memory automatically using ‘garbage collection’ when there are no active references to the object**
  - C++ requires the programmer to do this manually. You use ‘new’ sparingly in C++ because you have to use ‘delete’ when done, to avoid ‘memory leaks’
  - In C++, only use ‘new’ in constructors, never in `main()` or other methods. Very unlike Java.
- **Java has finalizers to clean up other resources (files, devices locked, etc.) when an object is destroyed**
  - Informal advice: never use finalizers
  - They can invoke any object, so garbage collector is wildly inefficient

## Beam Exercise



Write a Java class to model this beam and compute its maximum deflection:

$$w = - P L^3 / 3 E I$$

where

P= load at end (1200 N)

L= length of beam (20 m)

E= elastic modulus (30000 N/ m<sup>2</sup>)

I= moment of inertia (1000 m<sup>4</sup>)

w= deflection (m)

## Beam Exercise, p.2

- **Data fields:**
  - Which are beam characteristics?
  - Which are external?
- **Write two constructors:**
  - One with all fields as arguments
    - Use same names for arguments and fields
  - One with only length argument
    - Other fields default as on previous slide
    - Use 'this' to invoke other constructor or rely on initialization
  - Use initialization in your class: assume you're dealing with many beams like the example beam
- **Write two methods to return the deflection w**
  - Use same method name for both (overloading)
  - One takes load as argument, 2<sup>nd</sup> takes load and units (ft or m) as a String; convert result using 1 m= 3.3 ft
- **Don't write any get or set methods**
- **Write a class with a main() to create a beam, compute its deflection and print the result**

## Beam Exercise, p.3

- **Optional, advanced:**
  - **Add dimensional analysis:**
    - **Store the units for each variable in the class**
      - Decide how you'll code them ( exponents, etc.)
    - **Modify constructors to take unit arguments**
    - **Convert units if needed (N – lbf, m – ft)**
      - 1 lbf= 4.4 N, 1 ft= 0.3 m
    - **Make sure units match in the calculation**
    - **Output the units with the method result**

## Beam Class

```
class Beam {
    private double L;
    private double E= 30000.0;           // Initialization
    private double I= 1000.0;
    public Beam(double L, double E, double I) {
        this.L= L;           // this to access object fields
        this.E= E;
        this.I= I;
    }
    public Beam(double L) {
        this.L= L;           // Rely on initialization for others
    }
    // Could use this(L, 30000., 1000.);
    public double getDeflection(double P) {
        return -P*L*L*L/(3.0*E*I);
    }
    public double getDeflection(double P, String u) {
        if (u.equals("ft"))           // not ==
            return -3.3*P*L*L*L/(3.0*E*I);
        else
            return -P*L*L*L/(3.0*E*I);
    }
}
```

## Beam Main()

```
public class BeamExample {  
    public static void main(String[] args) {  
        Beam one= new Beam(20.0);  
        double w1= one.getDeflection(1200.0);  
        System.out.println("w1: " + w1);  
        double w2= one.getDeflection(1200.0, "ft");  
        System.out.println("w2: " + w2);  
        System.exit(0);  
    }  
}
```