Class examples

// 1. Data available directly. Discouraged!

public class Employee {
    public String name;
    public double htInch;
}

// Using the Employee class
Employee e1 = new Employee();
e1.name="Joe";
e1.htInch=72.5;
System.out.println(e1.name+" is "+e1.htInch+" inches tall.");

// 2. Data available through reader/writer methods only. Encouraged!

public class Employee {
    private String name;
    private double htInch;

    // Readers (getters)
    public String getName(){return name;}
    public double getHtInch() {return htInch;}

    // Writers (setters)
    public void setName(String n){name=n;}
    public void setHtInch(double h) {htInch=h;}
}

// Using the Employee class
Employee e1 = new Employee();
e1.setName("Joe");
e1.setHtInch(72.5);
System.out.println(e1.getName()+" is "+e1.getHtInch()+" inches tall.");

// 3. Add centimeter height. Data available directly. DANGER!

public class Employee {
    public String name;
    public double htInch;
    public double htCm;
}

// Using the Employee class
Employee e1 = new Employee();
e1.name="Joe";
e1.htInch=72.5;
e1.htCm=e1.htInch * 2.54;
System.out.println(e1.name+" is "+e1.htInch+" inches tall, and "+
e1.htCm+" cm tall.");

// Allows inconsistency!
e1.htInch=72.5;
e1.htCm=50;
// 4. Avoid inconsistency!

public class Employee
{
    private String name;
    private double htInch;
    private double htCm;

    // Readers
    public String getName()  {return name;}
    public double getHtInch(){return htInch;}
    public double getHtCm()  {return htCm;}

    // Writers (assures consistency)
    public void setName(String n)  {name=n;}
    public void setHtInch(double h){htInch=h; htCm=h*2.54;}
    public void setHtCm(double h)  {htCm=h; htInch=h/2.54;}
}

// Using the Employee class.  No inconsistency possible.
Employee e1 = new Employee();
e1.setName("Joe");
e1.setHtInch(72.5);
System.out.println(e1.getName()+" is "+e1.getHtInch()+
    " inches tall, and "+e1.getHtCm()+" cm tall.");

// 5. Alternate version. How is it different? What are the tradeoffs?
// Would someone using this class or the previous class notice any
// difference?

public class Employee
{
    private String name;
    private double htInch;

    // Readers
    public String getName()  {return name;}
    public double getHtInch(){return htInch;}
    public double getHtCm()  {return htInch*2.54;}

    // Writers
    public void setName(String n)  {name=n;}
    public void setHtInch(double h){htInch=h;}
    public void setHtCm(double h)  {htInch=h/2.54;}
}

// Using the Employee class.  No inconsistency possible.
Employee e1 = new Employee();
e1.setName("Joe");
e1.setHtInch(72.5);
System.out.println(e1.getName()+" is "+e1.getHtInch()+
    " inches tall, and "+e1.getHtCm()+" cm tall.");
6. Adding a constructor.

```java
public class Employee {
    private String name;
    private double htInch;

    // Constructor
    public Employee(String n, double h) {
        name = n;
        htInch = h;
    }

    // Readers
    public String getName() { return name; }
    public double getHtInch() { return htInch; }
    public double getHtCm() { return htInch * 2.54; }

    // Writers
    public void setName(String n) { name = n; }
    public void setHtInch(double h) { htInch = h; }
    public void setHtCm(double h) { htInch = h / 2.54; }
}

// Using the Employee class. No inconsistency possible.
Employee e1 = new Employee("Joe", 72.5);
System.out.println(e1.getName() + " is " + e1.getHtInch() + " inches tall, and "+e1.getHtCm()+" cm tall.");
```

7. Adding a second (default) constructor.

```java
public class Employee {
    private String name;
    private double htInch;

    // Constructors
    public Employee() // Default
    { name = "No name"; htInch = 0.0; }

    public Employee(String n, double h) {
        name = n; htInch = h;
    }

    // Readers
    public String getName() { return name; }
    public double getHtInch() { return htInch; }
    public double getHtCm() { return htInch * 2.54; }

    // Writers
    public void setName(String n) { name = n; }
    public void setHtInch(double h) { htInch = h; }
    public void setHtCm(double h) { htInch = h / 2.54; }
}

// Using the Employee class. No inconsistency possible.
Employee e1 = new Employee("Joe", 72.5);
Employee e2 = new Employee();
System.out.println(e1.getName() + " is " + e1.getHtInch() + " inches tall, and "+e1.getHtCm()+" cm tall."); System.out.println(e2.getName() + " is " + e2.getHtInch() + " inches tall, and "+e2.getHtCm()+" cm tall.");
```
// 8. Creating an array of Employee objects.

// Create the array of 10 references.
Employee[] eList = new Employee[10];

// Create the 10 Employee objects; default values.
for (int i=0; i<eList.length; i++)
    eList[i] = new Employee();

// Set two of the Employee objects.
eList[0].setName("Joe");
eList[0].setHtInch(72.5);
eList[1].setName("Sue");
eList[1].setHtCm(125);

// 9. Adding static data and method to keep track of the number
//    of Employee objects created.

public class Employee
{
    private String name;
    private double htInch;

    // Constructors. Note each increases employee count.
    public Employee()
    {empCount++; name="No name"; htInch=0.0;}

    public Employee(String n, double h)
    {empCount++; name=n; htInch=h;}

    private static int empCount=0;
    public  static int getEmpCount() {return empCount;}

    // Readers
    public String getName()  {return name;}
    public double getHtInch(){return htInch;}
    public double getHtCm()  {return htInch*2.54;}

    // Writers
    public void setName(String n)  {name=n;}
    public void setHtInch(double h){htInch=h;}
    public void setHtCm(double h)  {htInch=h/2.54;}
}

// Using the static information. This works no matter how many
// Employee objects have been created – even zero.
System.out.println("There have been "+Employee.getEmpCount()+
    " employees created so far. ");

Employee e1 = new Employee();
System.out.println("There have been "+Employee.getEmpCount()+
    " employees created so far. ");
// 10. Adding a toString() method.

class Employee
{
    private String name;
    private double htInch;

    // Constructors
    public Employee()
    {empCount++; name="No name"; htInch=0.0;}

    public Employee(String n, double h)
    {empCount++; name=n; htInch=h;}

    private static int empCount=0;
    public static int getEmpCount();{return empCount;}

    // Readers
    public String getName()  {return name;}
    public double getHtInch(){return htInch;}
    public double getHtCm()  {return htInch*2.54;}

    // Writers
    public void setName(String n)  {name=n;}
    public void setHtInch(double h){htInch=h;}
    public void setHtCm(double h)  {htInch=h/2.54;}

    // toString
    public String toString()
    { return name+"\’s height is "+htInch+" inches or "+getHtCm()+" cm. ";}
}

// Using toString().
Employee e1 = new Employee("Joe", 72.5);
System.out.println(e1);  // Automatic call to e1.toString().

Employee e1 = new Employee("Joe", 72.5);
e1 = new Employee("Sue", 65.5);

// Joe’s object is now garbage; there are no references pointing at
// it. But not to worry. Java’s automatic garbage collector will
// periodically reclaim garbage.
// 12. How would you make a new Employee object that is a copy of
// an existing one?

Employee e2 = new Employee(e1.getName(), e1.getHtInch());

This works, but gets very tedious if the object has a lot of properties
(data elements). Alternatives include creating a new constructor that takes
an Employee object as its parameter, or creating a clone method in the
Employee class that returns a copy of itself. Below are the constructor and
clone methods. One of these could be included in the Employee class
definition.

// Constructor (goes in Employee class)
public Employee(Employee e)
{
    name = e.getName();
    htInch = e.getHtInch();
}

// Using constructor to make a copy.
Employee e1 = new Employee("Joe", 72.5);
Employee e2 = new Employee(e1);  // Make e2 a copy of e1.

Employee e3 = e1;  // What does this do? How is it different from
                   // the previous line?

// Clone method (goes in Employee class)
public Employee clone()
{ return new Employee(name, htInch); }

// Using the clone method to make a copy.
Employee e1 = new Employee("Joe", 72.5);
Employee e2 = e1.clone();  // Make e2 a copy of e1.