The default constructor

The default constructor

If a constructor is not declared in a class `C`, then Java provides a default constructor. This constructor simply calls the superclass constructor that has no parameters.

```
public C() {
    super();
}
```

But if a constructor *is* declared in a class, then the default constructor is not included, unless you declare it yourself.

We verify this using the class `C` that appears in DrJava. `C` does not explicitly declare a constructor, but we can still create an instance using **`new`** `C()`. However, if we declare a constructor, say to store something in field `x`, and compile, and try to evaluate the same expression **`new`** `C()`, we get a "No Such Method Exception". The method does not exist.

A second point. If a constructor body does not begin with a constructor call, Java inserts the call

super();

Let's illustrate this point. The body of the constructor in class Circle does nothing —it's wrong. Let's create a new instance of Circle. It works. Now let's turn to class Shape and, for the moment, remove the constructor with no parameters. And compile.

Look, now it says that on line 15 of class Circle, which is the beginning of the body of Circle's constructor, a symbol cannot be found: constructor Shape(). So this indicates that on line 15, in the beginning of the constructor body, the call super(); is illegal because a constructor without parameters does not exist in class Shape!

Let's put the constructor back in and recompile. Now, it compiles.

The first time see an error because there is no constructor with 0 parameters, you will be confused. I suggest that you watch this complete lecture again, so that perhaps you will remember about the default constructor — what it looks like and when it is inserted automatically and when it isn't.