Calling constructors from constructors

Calling another constructor of in the same instance

Class `Shape` contains the coordinates of the top-left position of the bounding box of some shape. `Shape` meant to be subclassed in different ways to describe different shapes.

This class has *two* constructors. The first allows the user to give the position of the bounding box. The second uses the default bounding-box position (100, 100). One often writes several constructors, to provide flexibility for the user.

The constructor with no parameters directly assigns values to the fields. Instead, it should be possible to call the other constructor, say using

```
Shape(100, 100);
```

However, as you can see, Java does not allow this call. The Java convention to call another constructor in this class is to use keyword **`this`** instead of the class name:

```
this(100, 100);
```

The use of a constructor call in this instance does not save much, since there are only two fields to initialize. But in other cases, where there are more fields or whether a great deal of calculation has to go on in to initialize the fields, the use of a constructor call can save time and effort. Using a constructor call here follows a principle that has been mentioned several times elsewhere:

> **Principle**: Always use previously written methods, in order to save time and effort — both in writing a program and in testing and debugging it.

A constructor call must be first

The call of another constructor from within a constructor must be the *first* statement of the constructor. We can illustrate this by inserting an assignment as the first statement of the constructor, before the constructor call. And compile. There, see the error message that is produced?

Calling a constructor in the super class

Now let's write class `Circle`. We have declared the field, stubbed in the constructor, and written the getter function. Notice the specification of the getter function. It doesn't say "= field radius". The user does not know what the field is named. The spec is written in terms of the meaning of the class and not its implementation. It says simply, hey, get me the radius of the circle.

The constructor has to initialize fields `x`, `y`, and `radius`. It makes sense to follow the principle that:

> **Principle**: When initializing fields of an object during execution of a constructor, the inherited fields should be initialized first.

Since the inherited fields are private, and since there are not setter procedures for them, the inherited fields have to be initialized using a call on a constructor in the superclass. We can't use

```
Shape(x, y);
```

We can see that the program with this statement in it won't compile. Instead of the name `Shape`, use keyword super:

```
super(x, y);
```

And that's how you call a superclass constructor.

Java almost enforces the principal that inherited fields should be initialized first:

Principal: Initalize inherited fields before initializing newly declare fields

You can see this by putting the superclass call after the assignment to field `radius` and compiling. See? There is an error. *A call on a constructor cannot appear anywhere but as the first statement of the constructor body.*