Constructors

Here's class `Chapter`, with three fields, the chapter number, chapter title, and the previous chapter. Here is an instance of that class, created using an assignment `c= new Chapter();` .

The only way to initialize the fields of the new instance is through calls on the setter methods, for example:

```
c.setNumber(1);  c.setTitle("Intro");  c.setPrev(null);
```

If we had more fields —and many classes will have 5 or 10 or even more— it would become more laborious and awkward to initialize the fields. We need a simpler way to initialize fields of an new object.

Java lets us indicate the initialization in the new-expression itself. Instead of having to create the instance and then initialize the fields, we will be able to do it like this —the 1 is the value for field `number` and the string is the value for field `title`.

```
c= new Chapter(1, "Intro");
```

We won't have to initialize field `prev` because that is initialized to a default value of **`null`**, anyway.

But this doesn't work yet!. Before it will work, we have to write a new kind of method, called a *constructor*. Here's how we do it. First, write a specification:

```
/** Constructor: an instance with chapter number n,
                 title t, and previous chapter null */
```

Then, we write the constructor: it will be **`public`**. Since it is a constructor, we do *not* put a type or **`void`**. Since it is a constructor, its name is the name of the class. And we put parameter declarations for the parameters that will contain the chapter number and the title. Let's compile to make sure that the syntax is correct.

Now, write the constructor body: assign parameter `n` to field `number` and parameter `t` to `title`. We don't have to assign anything to field `prev` because its initial value is **`null`**, anyway. We compile.

This constructor now appears in each instance of the class, just like the other methods.

Let's see whether it works. Create an instance, giving argument values for fields `number` and `title`, and store the instance in variable `c`. Then, access all three fields. It works!

```
c= new Chapter(1, "Intro");
c.getNumber()
c.getTitle()
c.getPrev()
```

About constructors

Now let's give the rules for constructors. First, the constructor is a new kind of method. It has a purpose, which you should memorize:

> **Purpose of a constructor**: to initialize some or all of the fields of the newly created object during evaluation of a new-expression.

Next, the constructor definition includes:

1. An access modifier —usually **`public`**, so that the constructor can be called from anywhere.
2. The name of the constructor , which is the same as the name of the class! Notice that neither a type nor keyword **`void`** appears in a constructor definition.
3. One parameter declaration for each field for which a value is to be given in the new-expression.
4. A method body, which assigns each parameter to the appropriate field. Here, **`int`** parameter `n` is stored in field `number` and `String` parameter `t` is stored in field `title`.

The constructor, then, is the third kind of method. It looks different from a function or procedure because it does not have a return type or keyword void and because its name is the name of the class. Now, let's see how to call it.

Constructors

A new look at the new-expression

We can now give a complete explanation of the new-expression, in terms of class `Chapter` and this call:

```
new Chapter(1, "Intro")
```

You can read the general discussion in Gries/Gries, Section 3.2.2, p. 116.

We describe the three steps in evaluating a new-expression —this is something you should memorize and be able to perform on your own.

Step 1. *Create a new instance of class `Chapter`, with default values for the fields*.

The default values for fields are 0 for integral types, 0.0 for the floating point types, **`false`** for **`boolean`**, and **`null`** for all class-types.

Here is the new object, with all methods except for the constructor *not* shown.

The beginning of the new-expression helps you remember step 1, for it says “new Chapter”!

Step 2. *Execute the constructor call of the new-expression* —in this case, `Chapter(1, "Intro")`.

We know that this call will assign 1 to `n` and `"Intro"` to `title` and then execute the two assignment statements, making `number` 1 and `title` `"Intro"`.

Also in this step, the form of the new-expression tells you what to do. There is a constructor call; execute it.

Step 3. *Yield as value of the new-expression the name (on the tab) of the new object*. In this case, the value of the new-expression is `a0`.

Conclusion

And that is how a new-expression is evaluated. You will fully understand what is going on only when you have memorized the three steps and can carry them out yourself, as we have done here. The computer doesn’t have to evaluate a new-expression. *YOU* can do it too.

This is evaluation is so important for your understanding that we ask you to take a quiz on it.