

The Java™ Tutorials

Search

Hide TOC

Classes and Objects

Classes

Declaring Classes

Declaring Member

Variables

Defining Methods

Providing Constructors

for Your Classes

Passing Information to a

Method or a

Constructor

Objects

Creating Objects

Using Objects

More on Classes

Returning a Value from

a Method

Using the this Keyword

Controlling Access to

Members of a Class

Understanding Class

Members

Initializing Fields

Summary of Creating

and Using Classes and

Objects

Questions and Exercises

Questions and Exercises

Nested Classes

Inner Class Example

Local Classes

Anonymous Classes

Lambda Expressions

Method References

When to Use Nested

Classes, Local

Classes, Anonymous

Classes, and Lambda

Expressions

Questions and Exercises

Enum Types

Questions and Exercises

« Previous • Trail • Next »

Home Page > Learning the Java Language > Classes and Objects

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases and might use technology no longer available.

See [Java Language Changes](#) for a summary of updated language features in Java SE 9 and subsequent releases.

See [JDK Release Notes](#) for information about new features, enhancements, and removed or deprecated options for all JDK releases.

Declaring Classes

You've seen classes defined in the following way:

```
class MyClass {
    // field, constructor, and
    // method declarations
}
```

This is a *class declaration*. The *class body* (the area between the braces) contains all the code that provides for the life cycle of the objects created from the class: constructors for initializing new objects, declarations for the fields that provide the state of the class and its objects, and methods to implement the behavior of the class and its objects.

The preceding class declaration is a minimal one. It contains only those components of a class declaration that are required. You can provide more information about the class, such as the name of its superclass, whether it implements any interfaces, and so on, at the start of the class declaration. For example,

```
class MyClass extends MySuperClass implements YourInterface {
    // field, constructor, and
    // method declarations
}
```

means that `MyClass` is a subclass of `MySuperClass` and that it implements the `YourInterface` interface.

You can also add modifiers like *public* or *private* at the very beginning—so you can see that the opening line of a class declaration can become quite complicated. The modifiers *public* and *private*, which determine what other classes can access `MyClass`, are discussed later in this lesson. The lesson on interfaces and inheritance will explain how and why you would use the *extends* and *implements* keywords in a class declaration. For the moment you do not need to worry about these extra complications.

In general, class declarations can include these components, in order:

1. Modifiers such as *public*, *private*, and a number of others that you will encounter later. (However, note that the *private* modifier can only be applied to [Nested Classes](#).)
2. The class name, with the initial letter capitalized by convention.
3. The name of the class's parent (superclass), if any, preceded by the keyword *extends*. A class can only *extend* (subclass) one parent.
4. A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword *implements*. A class can *implement* more than one interface.
5. The class body, surrounded by braces, `{}`.

« Previous • Trail • Next »