

## The Java™ Tutorials

Hide TOC Search

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** 

iriitializirig i lelas

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.

## **Providing Constructors for Your Classes**

A class contains constructors that are invoked to create objects from the class blueprint. Constructor declarations look like method declarations—except that they use the name of the class and have no return type. For example, Bicycle has one constructor:

```
public Bicycle(int startCadence, int startSpeed, int startGear) {
    gear = startGear;
    cadence = startCadence;
    speed = startSpeed;
}
```

To create a new Bicycle object called myBike, a constructor is called by the new operator:

```
Bicycle myBike = new Bicycle(30, 0, 8);
```

new Bicycle(30, 0, 8) creates space in memory for the object and initializes its fields.

Although Bicycle only has one constructor, it could have others, including a no-argument constructor:

```
public Bicycle() {
    gear = 1;
    cadence = 10;
    speed = 0;
}
```

Bicycle yourBike = new Bicycle(); invokes the no-argument constructor to create a new Bicycle object called yourBike.

Both constructors could have been declared in Bicycle because they have different argument lists. As with methods, the Java platform differentiates constructors on the basis of the number of arguments in the list and their types. You cannot write two constructors that have the same number and type of arguments for the same class, because the platform would not be able to tell them apart. Doing so causes a compile-time error.

You don't have to provide any constructors for your class, but you must be careful when doing this. The compiler automatically provides a no-argument, default constructor for any class without constructors. This default constructor will call the no-argument constructor of the superclass. In this situation, the compiler will complain if the superclass doesn't have a no-argument constructor so you must verify that it does. If your class has no explicit superclass, then it has an implicit superclass of Object, which *does* have a no-argument constructor.

You can use a superclass constructor yourself. The MountainBike class at the beginning of this lesson did just that. This will be discussed later, in the lesson on interfaces and inheritance.

You can use access modifiers in a constructor's declaration to control which other classes can call the constructor.

Note: If another class cannot call a MyClass constructor, it cannot directly create MyClass objects.

```
« Previous • Trail • Next »
```

About Oracle I Contact Us I Legal Notices I Terms of Use I Your Privacy Rights

Convright @ 1995\_2022 Oracle and/or its affiliates. All rights reserved