# Generic Types

A generic type is a generic class or interface that is parameterized over types. The following Box class will be modified to demonstrate the concept.

## A Simple Box Class

Begin by examining a non-generic Box class that operates on objects of any type. It needs only to provide two methods: set, which adds an object to the box, and get, which retrieves it:

public class Box {

private Object object;

public void set(Object object) { this.object = object; }

public Object get() { return object; }

}

Since its methods accept or return an Object, you are free to pass in whatever you want, provided that it is not one of the primitive types. There is no way to verify, at compile time, how the class is used. One part of the code may place an Integer in the box and expect to get Integers out of it, while another part of the code may mistakenly pass in a String, resulting in a runtime error.

## A Generic Version of the Box Class

A generic class is defined with the following format:

class name<T1, T2, ..., Tn> { /\* ... \*/ }

The type parameter section, delimited by angle brackets (<>), follows the class name. It specifies the type parameters (also called type variables) T1, T2, ..., and Tn.

To update the Box class to use generics, you create a generic type declaration by changing the code "public class Box" to "public class Box<T>". This introduces the type variable, T, that can be used anywhere inside the class.

With this change, the Box class becomes:

/\*\*

\* Generic version of the Box class.

\* @param <T> the type of the value being boxed

\*/

public class Box<T> {

// T stands for "Type"

private T t;

public void set(T t) { this.t = t; }

public T get() { return t; }

}

As you can see, all occurrences of Object are replaced by T. A type variable can be any **non-primitive** type you specify: any class type, any interface type, any array type, or even another type variable.

This same technique can be applied to create generic interfaces.

## Type Parameter Naming Conventions

By convention, type parameter names are single, uppercase letters. This stands in sharp contrast to the variable [naming](http://docs.oracle.com/javase/tutorial/java/nutsandbolts/variables.html#naming) conventions that you already know about, and with good reason: Without this convention, it would be difficult to tell the difference between a type variable and an ordinary class or interface name.

The most commonly used type parameter names are:

* E - Element (used extensively by the Java Collections Framework)
* K - Key
* N - Number
* T - Type
* V - Value
* S,U,V etc. - 2nd, 3rd, 4th types

You'll see these names used throughout the Java SE API and the rest of this lesson.

## Invoking and Instantiating a Generic Type

To reference the generic Box class from within your code, you must perform a generic type invocation, which replaces T with some concrete value, such as Integer:

Box<Integer> integerBox;

You can think of a generic type invocation as being similar to an ordinary method invocation, but instead of passing an argument to a method, you are passing a type argument — Integer in this case — to the Box class itself.

**Type Parameter and Type Argument Terminology:** Many developers use the terms "type parameter" and "type argument" interchangeably, but these terms are not the same. When coding, one provides type arguments in order to create a parameterized type. Therefore, the T in Foo<T> is a type parameter and the String in Foo<String> f is a type argument. This lesson observes this definition when using these terms.

Like any other variable declaration, this code does not actually create a new Box object. It simply declares that integerBox will hold a reference to a "Box of Integer", which is how Box<Integer> is read.

An invocation of a generic type is generally known as a parameterized type.

To instantiate this class, use the new keyword, as usual, but place <Integer> between the class name and the parenthesis:

Box<Integer> integerBox = new Box<Integer>();

You can also use <> to instantiate the class (applies from Java 7):

Box<Integer> integerBox = new Box<>();