Java Generic's Wildcards

By Jakob Jenkov

Table of Contents

Java Generic's wildcards is a mechanism in Java Generics aimed at making it possible to cast a collection of a certain class, e.g A, to a collection of a subclass or superclass of A. This text explains how.

Here is a list of the topics covered:

The Basic Generic Collection Assignment Problem

Imagine you have the following class hierarchy:

```
public class A { }
public class B extends A { }
public class C extends A { }
```

The classes B and C both inherit from A.

Then look at these two `List` variables:

```
List<A> listA = new ArrayList<A>();
List<B> listB = new ArrayList<B>();
```

Can you set `listA` to point to `listB` ? or set `listB` to point to `listA`? In other words, are these assignments valid:

```
listA = listB;

listB = listA;
```

The answer is no in both cases. Here is why:

In `listA` you can insert objects that are either instances of A, or subclasses of A (B and C). If you could do this:

```
List<B> listB = listA;
```

then you could risk that `listA` contains non-B objects. When you then try to take objects out of `listB` you could risk to get non-B objects out (e.g. an A or a C). That breaks the contract of the `listB` variable declaration.

Assigning `listB` to `listA` also poses a problem. This assignment, more specifically:

```
listA = listB;
```

If you could make this assignment, it would be possible to insert A and C instances into the `List<B>` pointed to by `listB`. You could do that via the `listA` reference, which is declared to be of `List<A>`. Thus you could insert non-B objects into a list declared to hold B (or B subclass) instances.

When are Such Assignments Needed?

The need for making assignments of the type shown earlier in this text arises when creating reusable methods that operate on collections of a specific type.

Imagine you have a method that processes the elements of a `List`, e.g. print out all elements in the `List`. Here is how such a method could look: