Java SE Documentation

Type Inference for Generic Instance Creation

You can replace the type arguments required to invoke the constructor of a generic class with an empty set of type parameters (`<>`) as long as the compiler can infer the type arguments from the context. This pair of angle brackets is informally called the *diamond*.

For example, consider the following variable declaration:

```
Map<String, List<String>> myMap = new HashMap<String, List<String>>();
```

In Java SE 7, you can substitute the parameterized type of the constructor with an empty set of type parameters (`<>`):

```
Map<String, List<String>> myMap = new HashMap<>();
```

Note that to take advantage of automatic type inference during generic class instantiation, you must specify the diamond. In the following example, the compiler generates an unchecked conversion warning because the `HashMap()` constructor refers to the `HashMap` raw type, not the `Map<String, List<String>>` type:

```
Map<String, List<String>> myMap = new HashMap(); // unchecked conversion warning
```

Java SE 7 supports limited type inference for generic instance creation; you can only use type inference if the parameterized type of the constructor is obvious from the context. For example, the following example does not compile:

```
List<String> list = new ArrayList<>();
list.add("A");

  // The following statement should fail since addAll expects
  // Collection<? extends String>

list.addAll(new ArrayList<>());
```

Note that the diamond often works in method calls; however, it is suggested that you use the diamond primarily for variable declarations.

In comparison, the following example compiles:

```
// The following statements compile:

List<? extends String> list2 = new ArrayList<>();
list.addAll(list2);
```

Type Inference and Generic Constructors of Generic and Non-Generic Classes

Note that constructors can be generic (in other words, declare their own formal type parameters) in both generic and non-generic classes. Consider the following example:

```
class MyClass<X> {
  <T> MyClass(T t) {
    // ...
  }
}
```

Consider the following instantiation of the class `MyClass`, which is valid in Java SE 7 and prior releases:

```
new MyClass<Integer>("")
```

This statement creates an instance of the parameterized type `MyClass<Integer>`; the statement explicitly specifies the type `Integer` for the formal type parameter, `X`, of the generic class `MyClass<X>`. Note that the constructor for this generic class contains a formal type parameter, `T`. The compiler infers the type `String` for the formal type parameter, `T`, of the constructor of this generic class (because the actual parameter of this constructor is a `String` object).

Compilers from releases prior to Java SE 7 are able to infer the actual type parameters of generic constructors, similar to generic methods. However, the compiler in Java SE 7 can infer the actual type parameters *of the generic class being instantiated* if you use the diamond (`<>`). Consider the following example which is valid for Java SE 7 and later:

```
MyClass<Integer> myObject = new MyClass<>("");
```

In this example, the compiler infers the type `Integer` for the formal type parameter, `X`, of the generic class `MyClass<X>`. It infers the type `String` for the formal type parameter, `T`, of the constructor of this generic class.