Java 9 Diamond Operator Improvements

pramodbablad September 25, 2021 0

Diamond operator is used to denote the enclosing type of a class. For example, List<String> denotes list of strings, Set<Integer> denotes set of integers etc… Empty diamond operator <> is introduced from Java 7 to implement automatic type inference feature in the code. Empty diamond operator removes the redundant code by leaving the generic type in the right side of the declaration statement and thus removing the verbosity in the code. Till Java 9, empty diamond operator is allowed to use with normal classes only. It is not allowed to use with anonymous inner classes. But, from Java 9, empty diamond operator can also be used with anonymous inner classes. In this post, we will see the Java 9 improvements regarding anonymous inner classes and diamond operator.

Diamond Operator : Before Java 7

According to oracle docs, type inference is the compiler's ability to check each method invocation and corresponding declaration statements to determine the type of the arguments. In simple terms, Java compiler checks the type on the left side of the declaration statement to determine the type on the right side of the statement. Before Java 7, you need to explicitly mention type on both side of the declaration statement.

For example, in the below code snippet, type is mentioned on both the side using <>.

```
1  List<String> list = new
2  Set<Integer> set = new H
3  Map<Integer, String> map
```

and the same rule applies to anonymous inner classes also.

```
abstract class Addition
{
    abstract void add(
}

public class Java6Diamo
{
    public static void
    {
        //Before Java 7

        Addition<Intege
            @Override
            void add(In
            {
                System.
            }
        };
    }
}
```

Diamond Operator : After Java 7

With the introduction of empty diamond operator <> from Java 7, you need not mention type on the right side of the declaration statement. You can leave empty inside the diamond operator. Java compiler automatically determines the type on the right side of the declaration statement.

For example, above declaration statements can be written as below from Java 7.

```
List<String> list = new
Set<Integer> set = new H
Map<Integer, String> map
```

But, that rule doesn't apply to anonymous inner classes. Empty diamond operator can not be used with anonymous inner classes.

For example, the below code gives compile time error if you run it in Java 7 environment.

```
1  abstract class Additior
2  {
3      abstract void add(T
4  }
5
6  public class Java7Diamc
7  {
8      public static void
9      {
10         //Compile time
11         //'<>' cannot b
12
13         Addition<Intege
14             @Override
15             void add(In
16             {
17                 System.
18             }
19         };
20     }
21 }
```

This issue has been resolved from Java 9.

Diamond Operator : From Java 9

From Java 9, you can use empty diamond operator <> for anonymous inner classes also. The above code doesn't show any compile time errors if you execute it in Java 9 environment.

```
1  abstract class Additior
2  {
3      abstract void add(T
4  }
5
```

```
6  public class Java9Diamc
7  {
8      public static void
9      {
10         //No error, fro
11
12         Addition<Intege
13             @Override
14             void add(Ir
15             {
16                 System.
17             }
18         };
19     }
20 }
```

Below table summarizes how to use diamond operator before Java 7, after Java 7 and after Java 9.

Diamond Operator <>		
Before Java 7	**After Java 7**	**After Java 9**
//Need to mention type on both sides List<String> list = new ArrayList<String>(); Set<Integer> set = new HashSet<Integer>(); Map<Integer, String> map = new HashMap<Integer, String>();	//No need to mention type on right side List<String> list = new ArrayList< >(); Set<Integer> set = new HashSet< >(); Map<Integer, String> map = new HashMap< >();	//No need to mention type on right side List<String> list = new ArrayList< >(); Set<Integer> set = new HashSet< >(); Map<Integer, String> map = new HashMap< >();

//Need to mention type on both sides for anonymous classes also
Addition<Integer> integerAddition = new Addition<Integer>() {
@Override
void add(Integer t1, Integer t2)
{
System.out.println(t1+t2);
}
};
	//But, need to mention type on both sides for anonymous classes
	Addition<Integer> integerAddition = new Addition<Integer>() {
	@Override
	void add(Integer t1, Integer t2)
	{
	System.out.println(t1+t2);
	}
	};
		//No need to mention type on right side for anonymous classes also
		Addition<Integer> integerAddition = new Addition< >() {
		@Override
		void add(Integer t1, Integer t2)
		{
		System.out.println(t1+t2);
		}
		};

Also Read :