Java 9 @SafeVarargs Annotation

It is an annotation which applies on a method or constructor that takes **varargs parameters**. It is used to ensure that the method does not perform unsafe operations on its varargs parameters.

It was included in Java7 and can only be applied on

- Final methods
- Static methods
- Constructors

From Java 9, it can also be used with **private instance methods**.

Note: The @SafeVarargs annotation can be applied only to methods that cannot be overridden. Applying to the other methods will throw a compile time error.

Let's see some examples, in first example, we are not using @SafeVarargs annotation and compiling code . See, what happens?

Java 9 @SafeVarargs Annotation Example

```
import java.util.ArrayList;
import java.util.List;
public class SafeVar{
   private void display(List<String>... products) { // Not using @SaveVarargs
      for (List<String> product : products) {
         System.out.println(product);
      }
   }
   public static void main(String[] args) {
      SafeVar p = new SafeVar();
      List<String> list = new ArrayList<String>();
      list.add("Laptop");
      list.add("Tablet");
      p.display(list);
   }
}
```

It produces **warning messages** at compile time, but compiles without errors.

Output:

```
At compile time:
Note: SafeVar.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
At runtime:
[Laptop, Tablet]
```

This is a compiler generated warning regarding unsafe varargs type.

To avoid it, we should use @SaveVarargs notation to the method, as we did in the following example.

Java 9 @SafeVarargs Annotation Example

```
import java.util.ArrayList;
import java.util.List;
public class SafeVar{
   // Applying @SaveVarargs annotation
   @SafeVarargs
   private void display(List<String>... products) { // Not using @SaveVarargs
      for (List<String> product : products) {
         System.out.println(product);
      }
   }
   public static void main(String[] args) {
      SafeVar p = new SafeVar();
      List<String> list = new ArrayList<String>();
      list.add("Laptop");
      list.add("Tablet");
      p.display(list);
   }
}
```

Now, compiler does not produce warning message, code compiles and runs successfully.

Output:

```
[Laptop, Tablet]
```

Note: To apply @SaveVarargs annotation on private instance methods, compile code using Java 9 or higher versions only.

What happens? If we compile the following code by using older versions of Java.

Java @SafeVarargs Annotation Example

```
import java.util.ArrayList;
import java.util.List;
public class SafeVar{
   @SafeVarargs
   private void display(List<String>... products) {
      for (List<String> product : products) {
         System.out.println(product);
      }
   }
   public static void main(String[] args) {
      SafeVar v = new SafeVar();
      List<String> list = new ArrayList<String>();
      list.add("Laptop");
      list.add("Tablet");
      v.display(list);
   }
}
```

Output:

```
SafeVar.java:6: error: Invalid SafeVarargs annotation. Instance method display(List<String>...) is not final.
        private void display(List<String>... products) {
                     ^
Note: SafeVar.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
1 error
```