SafeVarargs Annotation Enhancements

This SafeVarargs Annotation was introduced in Java 7.

Prior to Java 9, we can use this annotation for final methods, static methods and constructors.

But from Java 9 onwards we can use for private methods also.

To understand the importance of this annotation, first we should aware var-arg methods and heap pollution problem.

What is var-arg method?

Until 1.4 version, we can't declared a method with variable number of arguments. If there is a change in no of arguments compulsory we have to define a new method. This approach increases length of the code and reduces readability.

But from 1.5 version onwards, we can declare a method with variable number of arguments, such type of methods are called var-arg methods.

**class** suku{

**public** **void** sum(**int** ... a)

{

**int** sum=0;

**for**(**int** i=0;i<a.length;i++) {

sum=sum+a[i];

}

System.***out***.println("Sum of Elements:"+sum);

}

**public** **static** **void** main(String args[])

{

suku s1=**new** suku();

s1.sum(1);

s1.sum(1,2);

s1.sum(1,2,3);

}

}

#### Output

Sum of Elements:1

Sum of Elements:3

Sum of Elements:6

Internally var-arg parameter will be converted into array.

# Var-arg method with Generic Type:

If we use var-arg methods with Generic Type then there may be a chance of Heap Pollution.

At runtime if one type variable trying to point to another type value, then there may be a chance of ClasssCastException. This problem is called Heap Pollution.

In our code, if there is any chance of heap pollution then compiler will generate warnings.

Example:

import java.util.\*;

class suku{

public static void sum(List<Integer> ...a)

{

Object []o=a;

String s=(String)o[0];

}

public static void main(String args[])

{

suku s1=new suku();

ArrayList<Integer>a1=new ArrayList();

a1.add(10);

a1.add(20);

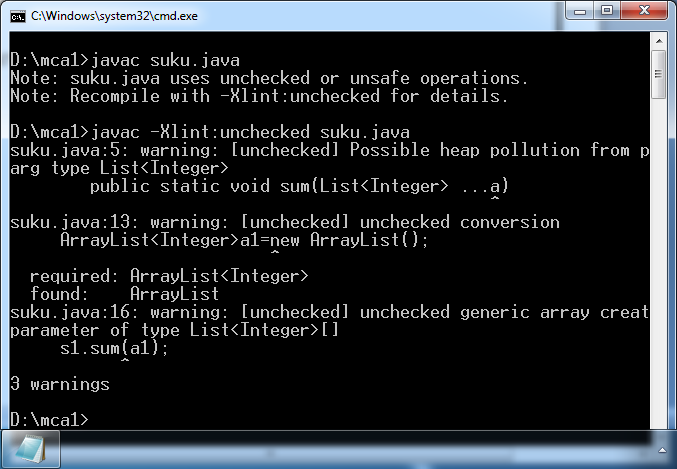
s1.sum(a1);

}

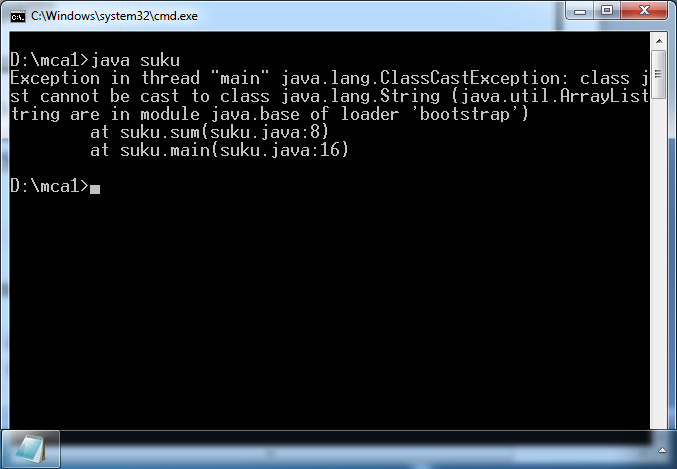
}

Compilation:

----------------



### Execution:



In the above program at runtime, String type variable name is trying to point to Integer type,which causes Heap Pollution and results ClassCastException.

String name = (String)o[0];

# Need of @SafeVarargs Annotation:

Very few Var-arg Methods cause Heap Pollution, not all the var-arg methods. If we know that our method won't cause Heap Pollution, then we can suppress compiler warnings with @SafeVarargs annotation.

import java.util.\*;

class suku{

@SafeVarargs

public static void sum(List<Integer> ...a)

{

Object []o=a;

String s=(String)o[0];

}

public static void main(String args[])

{

suku s1=new suku();

ArrayList<Integer>a1=new ArrayList();

a1.add(10);

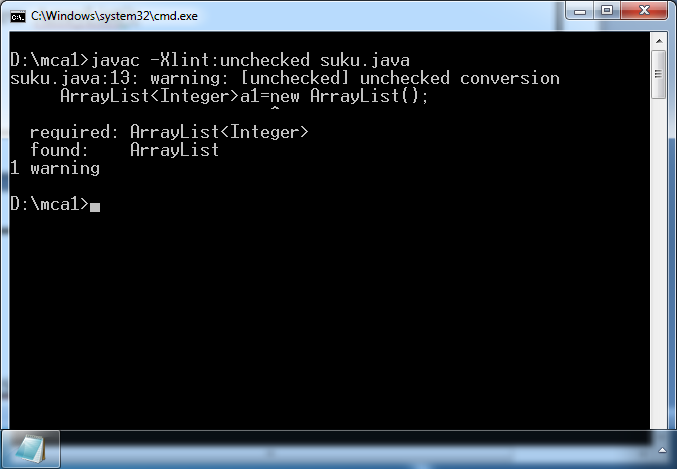
a1.add(20);

s1.sum(a1);

}

}

Compilation:-



# Java 9 Enhancements to @SafeVarargs Annotation:

@SafeVarargs Annotation introduced in Java 7.

Unitl Java 8, this annotation is applicable only for static methods, final methods and constructors. But from Java 9 onwards, we can also use for private instance methods also.

|  |  |  |
| --- | --- | --- |
| 1) import java.util.\*; | | |
| 2) public class Test | | |
| 3) | { |  |
| 4) @SafeVarargs //valid | | |
| 5) public Test(List<String>... l) | | |
| 6) |  | { |
| 7) |  | } |
| 8) @SafeVarargs //valid | | |
| 9) public static void m1(List<String>... l) | | |
| 10) { | | |
| 11) } | | |
| 12) @SafeVarargs //valid | | |
| 13) public final void m2(List<String>... l) | | |
| 14) { | | |
| 15) } | | |
| 16) @SafeVarargs //valid in Java 9 but not in Java 8 | | |
| 17) private void m3(List<String>... l) { | | |
| 18) } | | |
| 19) } | | |

javac -source 1.8 Test.java

error: Invalid SafeVarargs annotation. Instance method m3(List<String>...) is not final. private void m3(List<String>... l)

^

javac -source 1.9 Test.java

We won't get any compile time error.

## FAQs:

Q1. For which purpose we can use @SafeVarargs annotation? Q2. What is Heap Pollution?