



CSN-103: Fundamentals of Object Oriented Programming

Instructor: Dr. Rahul Thakur

Assistant Professor, Computer Science and Engineering, IIT Roorkee



Introducing final

- If a variable is declared as **final**
 - Its content **can't be modified**
 - **Must initialize** a **final** variable (Assignment done only **once**)

- Example:

```
final int FILE_NEW = 1;
```

or

```
final int FILE_NEW;
```

```
FILE_NEW=1;
```

```
FILE_NEW=2;
```

Convention to use all
UPPERCASE identifiers
for **final** variables

Error: error: variable FILE_NEW might already have been assigned

Introducing final

- Final variables are essentially **constants**
- Keyword **final** can also be applied to:
 - **Methods:** Different meaning
 - Related to **inheritance**
 - **Classes and objects:**
 - A class that is declared final cannot be **subclassed**
 - A final object's instance variables **can be** modified, but the reference can't

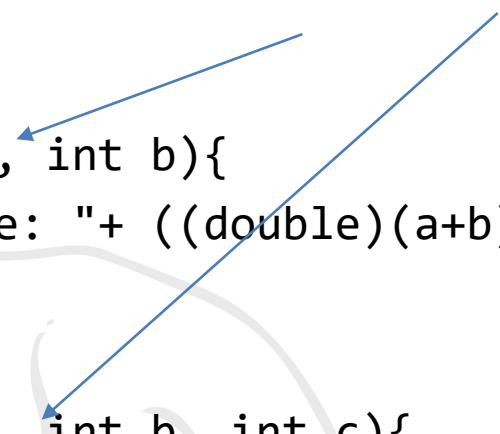
Variable-Length Arguments

- Example:
 - Marks of a number of students
 - Wants to calculate the average marks
- You need to know the marks of individual student
- You need to know the **number** of students
- **Methods** that need to take a variable number of arguments
 - This feature is called *variable-length arguments* (*varargs*)
 - A method that takes a variable number of arguments is called *varargs method*

Variable-Length Arguments

```
class CalculateAvg {  
    public static void main(String args[]) {  
        calculateAverage(2,3);  
        calculateAverage(2,3,4);  
    }  
  
    static void calculateAverage(int a, int b){  
        System.out.println("Average: "+ ((double)(a+b)/2));  
    }  
  
    static void calculateAverage(int a, int b, int c){  
        System.out.println("Average: "+ ((double)(a+b+c)/3));  
    }  
}
```

Overloaded
Parameterized Methods

Two blue arrows originate from the text "Overloaded Parameterized Methods". One arrow points to the parameter list "int a, int b" in the second method signature. The other arrow points to the parameter list "int a, int b, int c" in the third method signature.

Good enough approach: If max number
of arguments are is small

Variable-Length Arguments

```
class CalculateAvg2 {  
    public static void main(String args[]) {  
        int m1[] = {2,3};  
        int m2[] = {2,3,4,5,6};  
        calculateAverage(m1);  
        calculateAverage(m2);  
    }  
  
    static void calculateAverage(int a[]){  
        int sum=0;  
        for(int i=0;i<a.length;i++)  
            sum+=a[i];  
        System.out.println("Average: "+ (double)sum/a.length));  
    }  
}
```

Arguments are put into an Array
Array is passed to the method

Drawback: Declare and initialize
array with arguments

Variable-Length Arguments

- In Java, you can specify variable length argument by **three periods (...)**

- Example:

```
static void calculateAverage (int ... argt)
```

calculateAverage() can be called with **zero** or more arguments

- ***argt*** is implicitly declared **as an array** of type *int[]*
 - ***argt*** is accessed using the **normal array syntax**

Variable-Length Arguments

```
class CalculateAvg3 {  
    public static void main(String args[]) {  
        calculateAverage(2,3);  
        calculateAverage(2,3,4,5,6);  
    }  
  
    static void calculateAverage(int ... a){  
        int sum=0;  
        for(int i=0;i<a.length;i++)  
            sum+=a[i];  
        System.out.println("Average: "+ ((double)sum/a.length));  
    }  
}
```

← No array required

← No overloaded methods needed

Variable-Length Arguments

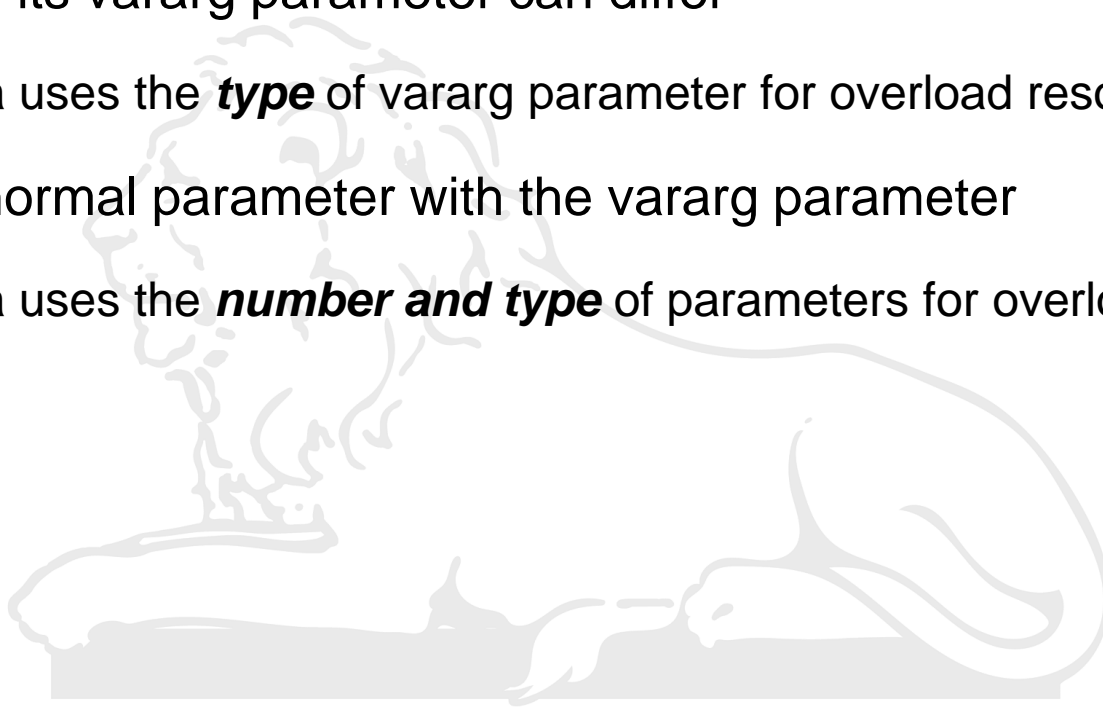
- Normal parameters can also be specified with a variable-length parameter
- Restrictions
 - Variable-length parameter **must be the last** parameter
 - There can be **only one** variable-length parameter
- Examples:
 - VALID:

```
void calculateAverage (int a, char b, double c, int ... vals){
```
 - INVALID:

```
void calculateAverage(int a, char b, int ... vals, double c){  
void calculateAverage(int a, char b, int ... vals, float ... fls){
```

Overloading Vararg Method

- Vararg method can be overloaded in two ways
 - **Type** of its vararg parameter can differ
 - Java uses the **type** of vararg parameter for overload resolution
 - Add a normal parameter with the vararg parameter
 - Java uses the **number and type** of parameters for overload resolution



Vararg and Ambiguity

- It is possible to create an ambiguous call to an overloaded vararg method because of:
 - Empty vararg parameter
 - A normal parameter along with vararg parameter of the same type

