Variable Argument Lists

Adding more variability

Definition

- Also called Variadic function
- Methods containing parameter(s) that can accept different number of arguments
- Support for variadic functions differs among different programming languages
  - **C** - C++
  - **C#** - **Java**
  - LISP - **Visual Basic**
  - **PHP** **- Ruby**

C Programming Language

- Most common variable argument functions are
  - void printf(const char* fmt, …);
  - void scanf(const char* fmt, …);
- Parameters include
  - One named argument usually a format string
  - Ellipsis indicating the function may take any number of arguments and their types

Creating a Variadic Function

- Use standard header stdarg.h
- 4 macros (not prototypes) needed from this header file
  - va_list
  - va_start
  - va_args
  - va_end

Variable arguments in C

- va_list
  - Stores the list of arguments (argument pointer)
  - Declared like any other variable
  - Ex. va_list ap
- va_start
  - initializes the list, point to the first argument
  - Accepts two arguments
    - va_list
    - Name of the variable that directly precedes the ellipsis
  - Ex. va_start(ap, fmt)

Variable arguments in C

- va_args
  - returns the next argument of whatever type it is told
  - moves down to the next argument in the list
  - Accepts two arguments
    - va_list
    - Variable argument type
  - Ex. va_args(ap, int) //returns next int value in the argument
- va_end
  - Cleans up the variable argument list
  - Ex. va_end(ap)

C Sample Code

```
#include <stdio.h>
#include <stdarg.h>

double average ( int num, ... )
{
	va_list ap;
	double sum = 0;
	va_start ( ap, num ); //initializes, store all values
	int x;
    for ( x = 0; x < num; x++ )
		sum += va_arg ( ap, double );
	va_end ( ap );
	return sum/num;
}
int main()
{
	printf( "%f\n", average ( 3, 12.2, 22.3, 4.5 ));
	printf("%f\n", average ( 5, 3.3, 2.2, 1.1, 5.5, 3.3 ));
	return 0;
}
```

Java

- Basic syntax
  - *type … variableName*
- Argument passed to a method is converted into an array of the same-typed values
  - sum (10,20) ⇔sum(new int[] {10,20})

Sample Code

```
public static void main(String[] args)
{
    System.out.println(“The sum is ” + sum(10,20,30));
}
public int sum (double … numbers)
{
    int total = 0;
    for (int i = 0; i < numbers.length; i++)
       total += numbers[i];
    return total;
}
```

Yuck, lets rewrite

```
public int sum (int … numbers)
{
        return sumHelper(0, numbers);
}

private int sumHelper (int starter, int … numbers)
{
        for (int i = 0; i < numbers.length; i++)
          starter += numbers[i];
        return starter;
}
```

Lets rewrite again

```
public int sum (int … numbers)
{
        return sumHelper(0, numbers);
}

private int sumHelper (int starter, int … numbers)
{
        for (int i : numbers)
          starter += i;
        return starter;
}
```

Formatting in Java

- Employs variable arguments
- printf() and format() method of PrintStream and String
  - System.out.printf(“%5d %6.2f”, 23, 45.6);
  - String s = String.format(“%.2f”, 1234.567)

Varargs – common usage

- The following constructor

```
public Person (String name, String details… )
```

Can be called with many different invocations:

```
new Person (“Alexander ”);
new Person (“Alexander ”, “Bell ”);
new Person (“Alexander ”,”Graham”, “Bell ”);
```

Varargs

- **Before:**

```
print( new String[] { "1", "2", "3" } );
...
private void print( String[] array ) {
    for ( int j = 0; j < array.length; j++ ) {
        System.out.println( array [ j ] );
    }
}
```

- **After:**

```
print( "1", "2", "3", "4" ); // put as many as you need
...
private void print( String ... array ) {
    for ( String s : array ) {
        System.out.println( s ); // same as array[ j ]
    }
}
```