

Java Control: Functions

**CPSC 233: Introduction to Computer Science for Computer Science
Majors II
Winter 2022**

Jonathan Hudson, Ph.D.
Instructor
Department of Computer Science
University of Calgary

Thursday, 4 November 2021

Copyright © 2021



Functions

- A function is a block of code that accomplishes a single purpose, such as calculating an average from a list of values, or printing a message to console.
 - A function (optionally) takes some inputs (also named arguments), performs some operations, and (optionally) gives back a result.
- Proper design and use of functions can save time and make a program easy to read, maintain, organized, and efficient.



**You've
already
been using
functions**

Built-in functions are provided by the language

Library functions are similar but provided by another programmer

User-defined functions are created by programmers

All functions operate the same

Build your own



Adventure?

User-Defined Functions

- Programmers can define their own functions.
- A function consists of the following components:
 - (Required) **Function name, return type of void, primitive, class name, (), {}**
 - (Semi-Optional) **public/static** (needed if you want their behaviour)
 - (Optional) **Function parameters**
 - (Optional) **Function body** (code) : The function's body must contain at least {}.
- The following is a minimal function that does nothing:

```
public static void foo(){  
}
```

User-Defined Functions

- Programmers can define their own functions.
- A function consists of the following components:
 - (Required) **Function name, return type of void, primitive, class name, (), {}**
 - (Semi-Optional) **public/static** (needed if you want their behaviour)
 - (Optional) **Function parameters**
 - (Optional) **Function body** (code) : The function's body must contain at least {}.
- The following is a minimal function that does nothing:

Java

```
public static void foo(){  
}
```

Python

```
def foo():  
    pass
```

User-Defined Functions – Java Versus Python

Python functions always returned None

Instead Java has two types of functions

1. those that return **void** (syntactically cannot be assigned to a variable)
2. those that return a **type** (can return object/null [like Python None] or type like int)

Return void

```
public static void foo(){  
}
```

Return type

```
public static int bar(){  
    return 1;  
}
```

User-Defined Functions – Java Versus Python

I can call either foo/bar without storing result
(like Python)

But only bar() can be stored

`int x = foo();` is a syntax error

```
public static void main(String[] args) {  
    foo();  
    bar();  
    int x = foo();  
    int y = bar();  
}
```

Return void

```
public static void foo(){  
}
```

Return type

```
public static int bar(){  
    return 1;  
}
```


User-Defined Static Functions (callable from main)

- You can define your own function using the following syntax:

For now all our functions will be public static

Function name

optional one or more **TYPED** parameters (separated by commas)

```
public static double foobar(int x, double y, String s){  
    double z = x * y;  
    return z;  
}
```

{ } bracketing

optional if non-void then one **return must exist** in function, can only return at most one thing

Overloading functions

What makes a function unique?

Function Overloading

- The whole Java file is read before main() starts (so all 4 functions and main will have been stored in memory)
- **Java functions are unique by name, parameter type/order.**
 - (Ex. foo(double,int) is unique from foo(int,double)
- If a function name is used more than once we call this function (**over-loaded**).
- All four of these functions can be called in the same program. In any order. As many times as we want.
- They all exist at the same time. (In Python only one function with any name existed at a time)

```
public static void main(String[] args) {
    foobar();
    foobar(1);
    foobar(2.0);
    foobar(1, 2.0);
    foobar(2.0);
    foobar(1);
    foobar();
}

public static double foobar() {
    return 1.0;
}

public static double foobar(int x) {
    return 2.0;
}

public static double foobar(double y) {
    return 3.0;
}

public static double foobar(int x, double y) {
    return 4.0;
}
```

Function Overloading

- The whole Java file is read before main() starts (so all 4 functions and main will have been stored in memory)
- Java functions are unique by name, parameter type/order.
- If a function name is used more than once we call this function (over-loaded).
- All four of these functions can be called in the same program. In any order. As many times as we want.
- They all exist at the same time. (In Python only one function with any name existed at a time)
- **There is no foobar(double,int) here, so that would be a syntax error if we tried to use it**

```
public static void main(String[] args) {  
    foobar();  
    foobar(1);  
    foobar(2.0);  
    foobar(1, 2.0);  
    foobar(2.0);  
    foobar(1);  
    foobar();  
}  
  
public static double foobar() {  
    return 1.0;  
}  
  
public static double foobar(int x) {  
    return 2.0;  
}  
  
public static double foobar(double y) {  
    return 3.0;  
}  
  
public static double foobar(int x, double y) {  
    return 4.0;  
}
```

Storing functions

Think of what “import java.util.Math” did

Accessing functions in other libraries

Fun.java

```
public class Fun{
    public static void fun(){
        System.out.println("Hello, world!");
    }
}
```

Foo.java

```
public class Foo{
    public static void foo(){
        System.out.println("Goodbye, world!");
    }
}
```

Main.java

```
public class Main {
    public static void main(String[] args) {
        Foo.foo();
        Fun.fun();
    }
}
```

No import needed if folders in same directory
Compile all 3 and run.
Functions protected by Class name they are stored
in (which matches filename)

Non-public?

Fun.java

```
public class Fun{
    _static void fun(){
        System.out.println("Hello, world!");
    }
}
```

Foo.java

```
public class Foo{
    private static void foo(){
        System.out.println("Goodbye, world!");
    }
}
```

Main.java

```
public class Main {
    public static void main(String[] args) {
        Foo.foo();
        Fun.fun();
    }
}
```

Other modifiers can prevent Main class from accessing the code in the other classes

More on this later in course!

Non-public?

Modifiers won't affect function access in same class file

```
public class Main {  
    public static void main(String[] args) {  
        foo();  
        fun();  
    }  
    private static void foo(){  
        System.out.println("Goodbye, world!");  
    }  
    _static void fun(){  
        System.out.println("Hello, world!");  
    }  
}
```


Parameters

**There are no optional
parameters**

Function Overloading – Optional?

- We can simulate Python style optional parameters with an overloaded function in which we replace the parameter that wasn't provided with a default value.
- We cannot call parameters by name
 - `foobar(x=10);` is not valid syntax

```
public static void main(String[] args) {
    foobar();
    foobar(1);
    foobar(2.0);
    foobar(1, 2.0);
    foobar(2.0);
    foobar(1);
    foobar();
}

public static double foobar() {
    return foobar(1, 2.0);
}

public static double foobar(int x) {
    return foobar(x, 2.0)
}

public static double foobar(double y) {
    return foobar(1, y);
}

public static double foobar(int x, double y) {
    return 4.0;
}
```

Variable Length Arguments

Variable Argument Functions

- int...
- String...
- double...
- Produces an iterable (foreach can loop) list of parameters

- We can only have one of these at end
- We can add our non flexible parameters in front of this variable length portion

```
public static void main(String[] args) {  
    System.out.println(sum(1,2,3,4,5));  
}  
  
public static int sum(int... numbers) {  
    int sum = 0;  
    for (int num : numbers) {  
        sum = sum + num;  
    }  
    return sum;  
}
```

null?

Functions return null to mean Nothing

Non-void functions always return something

- That something can also be **null** if what is being returned is an **Object reference**
- **null** is a special keyword
- (We often use **null** in other places in our code to show nothing has been stored in an **Object** variable yet)

```
public static Integer Nothing_Integer() {  
    return null;  
}  
  
public static int Nothing_int() {  
    return null; //SYNTAX ERROR  
}
```

Examples

Some simple functions

```
public static void main(String[] args) {  
    System.out.println(circle_area(10));  
    System.out.println(sum_to(10));  
    System.out.println(is_even(202));  
    System.out.println(is_odd(501));  
}
```

```
public static double circle_area(double radius) {  
    return Math.PI * Math.pow(radius, 2);  
}
```

```
public static double sum_to(int n) {  
    return (n * (n + 1)) / 2;  
}
```

```
public static boolean is_even(int number) {  
    return number % 2 == 0;  
}
```

```
public static boolean is_odd(int number) {  
    return !is_even(number);  
}
```

Design

User-Defined Functions - Commenting

- A good function always contains explicit comments that describe the purpose of the function, the parameters, and returned values. – Javadoc commenting

```
/**
 * Calculate area of circle from double radius
 * @param radius Double radius of circle
 * @return Area of the circle using pi * (radius ^ 2)
 */
public static double circle_area(double radius) {
    return Math.PI * Math.pow(radius, 2);
}
```

- See:

<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Math.html>

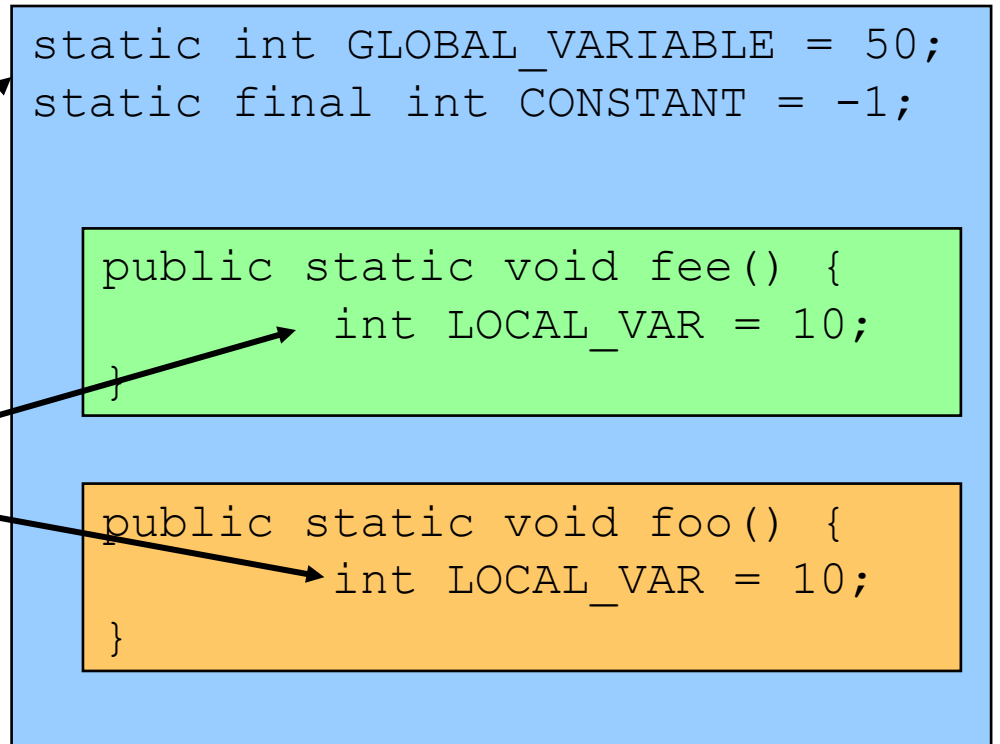
Scope

Scope of Variables

- Variables are memory locations that are used for the temporary storage of data
- The scope of a variable is the section of code in which it is accessible

The global scope:
Accessible by both functions

Local scopes:
Two different memory spaces,
Accessible only within their
functions



Scope of Variables

- Produces?

```
static int GLOBAL_VARIABLE = 50;  
static final int CONSTANT = -1;
```

```
public static void fee() {  
    int LOCAL_VAR = 10;  
    GLOBAL_VARIABLE = 0;  
}
```

```
public static void foo() {  
    int LOCAL_VAR = 20;  
    GLOBAL_VARIABLE = 99;  
}
```

```
public static void main(String[] args) {  
    System.out.println(GLOBAL_VARIABLE + " " + CONSTANT);  
    fee();  
    System.out.println(GLOBAL_VARIABLE + " " + CONSTANT);  
    foo();  
    System.out.println(GLOBAL_VARIABLE + " " + CONSTANT);  
}
```

Scope of Variables

- 50 -1
- 0 -1
- 99 -1

```
static int GLOBAL_VARIABLE = 50;  
static final int CONSTANT = -1;
```

```
public static void fee() {  
    int LOCAL_VAR = 10;  
    GLOBAL_VARIABLE = 0;  
}
```

```
public static void foo() {  
    int LOCAL_VAR = 20;  
    GLOBAL_VARIABLE = 99;  
}
```

```
public static void main(String[] args) {  
    System.out.println(GLOBAL_VARIABLE + " " + CONSTANT);  
    fee();  
    System.out.println(GLOBAL_VARIABLE + " " + CONSTANT);  
    foo();  
    System.out.println(GLOBAL_VARIABLE + " " + CONSTANT);  
}
```

Scope of Variables

- What about?

```
static int GLOBAL_VARIABLE = 50;  
static final int CONSTANT = -1;
```

```
public static void fee() {  
    int LOCAL_VAR = 10;  
    GLOBAL_VARIABLE = 0;  
}
```

```
public static void foo() {  
    int LOCAL_VAR = 20;  
    GLOBAL_VARIABLE = 99;  
}
```

```
public static void main(String[] args) {  
    CONSTANT = 10000;  
    System.out.println(GLOBAL_VARIABLE + " " + CONSTANT);  
    fee();  
    System.out.println(GLOBAL_VARIABLE + " " + CONSTANT);  
    foo();  
    System.out.println(GLOBAL_VARIABLE + " " + CONSTANT);  
}
```


Scope of Variables

- Syntax error as constants can't be changed

```
static int GLOBAL_VARIABLE = 50;  
static final int CONSTANT = -1;
```

```
public static void fee() {  
    int LOCAL_VAR = 10;  
    GLOBAL_VARIABLE = 0;  
}
```

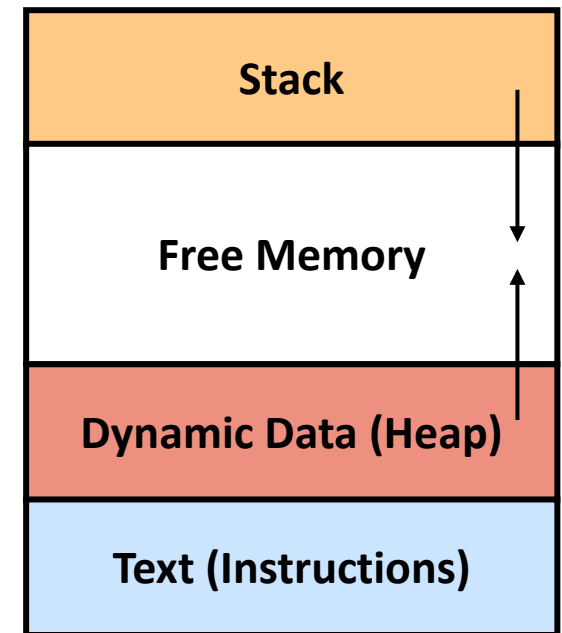
```
public static void foo() {  
    int LOCAL_VAR = 20;  
    GLOBAL_VARIABLE = 99;  
}
```

```
public static void main(String[] args) {  
    CONSTANT = 10000;  
    System.out.println(GLOBAL_VARIABLE + " " + CONSTANT);  
    fee();  
    System.out.println(GLOBAL_VARIABLE + " " + CONSTANT);  
    foo();  
    System.out.println(GLOBAL_VARIABLE + " " + CONSTANT);  
}
```

Memory

Memory Organization

- The memory for a program is organized into three regions
 - **Text (Instructions)**: holds program instructions. Contains your **bytecode**
 - **Dynamic Data (Heap)**: objects (referenced from stack)
 - **Stack**: functions, local primitive variables and data, local reference object variables in functions (data in heap)
- *In python we had no primitives so we only had reference object variables and all of the referenced data were objects in heap*



Abusing Memory

- Running out of stack space in Java
 - Recursion forever
 - Exception in thread "main"
`java.lang.StackOverflowError`
- Running out of heap space in Java
 - Make objects forever
 - Exception in thread "main"
`java.lang.OutOfMemoryError: Java heap space`

```
public static void kill_stack() {  
    Long LOCAL_VAR = 10L;  
    kill_stack();  
}
```

```
public static void kill_heap() {  
    ArrayList list = new ArrayList();  
    Long LOCAL_VAR = 10L;  
    while (true){  
        list.add(LOCAL_VAR++);  
    }  
}
```

Onward to ... Data Structures.

Jonathan Hudson
jwhudson@ucalgary.ca
<https://pages.cpsc.ucalgary.ca/~jwhudson/>



UNIVERSITY OF
CALGARY