Java Memory and Intro to OOP

CSCI-UA 0101
Instructor: Randy Shepherd
What is Java?

• Programming language

• Standard libraries
  • ex. Math.random()

• Compiler & runtime (aka the JVM)
What is the JVM

• stands for is a ‘Java Virtual Machine’

• is the ‘runtime’ i.e. the thing that runs your program once it is compiled

• sits between your program and the operating system and *abstracts* away complexity

• including managing the *memory* of your program
Java Code (.java) -> JAVAC compiler -> Byte Code (.class) -> JVM -> Windows, Linux, Mac
JVM Memory Management

• When the JVM is started it asks for a specific amount of memory from the operating system.

• You can control this yourself, but it defaults to some number of megabytes, probably something like 512 (depends on the system).

• If your program exceeds that amount, the JVM will handle asking for more from the operating system.
Types of Memory in Java

• When you declare a variable in a program, Java allocates space for that variable from one of several memory regions.

• One region of memory is reserved for variables that are never created or destroyed as the program runs, such as constants. This information is called static data.

• Whenever you create a new object (using the new keyword), Java allocates space from a pool of memory called the heap.

• Each time you call a method, Java allocates a new block of memory called a stack frame to hold its variables. These stack frames come from a region of memory called the stack.
Types of Memory in Java

```java
public class TypesOfMemory() {

    public static final int STATIC_ALLOCATED = 0;

    public static void main(String[] args) {
        String referenceToObjectOnHeap = new String("On the heap");

        int stackAllocatedArgument = 0;
        method(stackAllocatedArgument);
    }

    public void method(int stackAllocatedParameter) {
        int stackAllocatedVariable = stackAllocatedParameter + 1;
        System.out.println(stackAllocatedVariable);
    }
}
```
With me so far?

• JVM manages memory for your variables

• Depending on the characteristics of the variable it stores the data in different ‘regions’

• The regions are:
  • Static
  • Heap
  • Stack

• We are not going to cover static in detail today. Much easier after we understand classes.
The Stack
What is a Stack?

• The ‘stack’ we have been talking about so far is a memory region. However, it gets its name from the data structure it uses.

• By data structure, we mean something that is meant to hold data and provides certain operations on that data.

• An array is an example of simple data structure. Its ‘operations’ are using offsets with brackets.
  
  • ex. someArray[2];
What is a Stack?

• A stack is a bit more sophisticated than an array, but we don’t need to understand it deeply for this class.

• The two important things..
  • Its “first-in, last-out”
  • Its operations are “push” and “pop”

• Its easy to understand with an analogy....
What is a Stack?

• Imagine a stack of plates…
  
  • You can “push” a new plate onto the stack of plates by sticking a plate on top
  
  • You can “pop” a plate off the stack by taking the top one
Call Stack

• The stack memory region works like the stack data structure

• What gets pushed and popped from it are “stack frames”
Stack Frame

• Every time a method is called a “stack frame” gets created.

• You can think of the “stack frame” as the set of all the variables needed for that method.
Pushing and Popping Stack Frames

• Every time a method is invoked a “stack frame” for that method is *pushed* onto the stack memory region.

• When the method returns, the “stack frame” gets *popped*, and all those variables are deleted!

• Once a stack frame is popped, that region of memory becomes available for other stack variables.
Trace the Call Stack

```java
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println(
        "The maximum between " + i + " and " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
```

i is declared and initialized

i: 5

The main method is invoked.
public static void main(String[] args) { 
    int i = 5; 
    int j = 2; 
    int k = max(i, j); 
    System.out.println(
        "The maximum between " + i + 
        " and " + j + " is " + k); 
} 

public static int max(int num1, int num2) { 
    int result; 
    if (num1 > num2) 
        result = num1; 
    else 
        result = num2; 
    return result; 
}
Trace the Call Stack

public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between "+i+
    " and "+j+" is "+k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
Trace the Call Stack

public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println(  
        "The maximum between " + i +  
        " and " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
Trace the Call Stack

public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}

pass the values of i and j to num1 and num2

Space required for the main method

k:
j: 2
i: 5

The max method is invoked.
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println(
        "The maximum between " + i + 
        " and " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2) {
        result = num1;
    } else {
        result = num2;
    }
    return result;
}
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between " + i + ", " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}

(num1 > num2) is true

Space required for the main method
result: num2: 2 num1: 5
k: j: 2 i: 5
The max method is invoked.

Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.
Trace the Call Stack

```java
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println("The maximum between " + i + "+ j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
```

Assign num1 to result

Space required for the max method
- `result: 5`
- `num2: 2`
- `num1: 5`

Space required for the main method
- `k: j: 2`
- `i: 5`

The max method is invoked.

Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.
public static void main(String[] args) {  
    int i = 5;  
    int j = 2;  
    int k = max(i, j);  
    System.out.println(  
        "The maximum between " + i +  
        " and " + j + " is " + k);  
}  

public static int max(int num1, int num2) {  
    int result;  
    if (num1 > num2)  
        result = num1;  
    else  
        result = num2;  
    return result;  
}
Trace the Call Stack

```java
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between "+i+" and "+j+" is "+k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```

Execute print statement

Space required for the main method
- k: 5
- j: 2
- i: 5

The main method is invoked.
Trace the Call Stack

```java
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
```

Complete the method
With me so far?

- For every method call (even the main method!) a stack frame is created.
- All variables declared in the method are in the stack frame.
- When the method exits, those variables go bye-bye!
Heap
‘Reference’ Types

• In Java we have two types of variables, *primitives* and *references*

• Reference types are *classes*

• The class can be a class that’s provided as part of the Java API class library..
  
  • ex String, Scanner, arrays, etc.

• ..or a class that you write yourself. (We’ll be doing that soon).
Variables on the Heap

• When you create a variable of a reference type, Java allocates the amount of memory the object requires to store the object on the heap.

• (You know this happened when you see `new`)

• Then, the variable is actually assigned a reference to the object, not the object itself.

```java
String referenceToObjectOnHeap = new String("On the heap");

^^^^ reference (aka pointer)          ^^^ creates variable on heap
```
Key Idea

- Again, when working with reference types, a variable of a particular type *doesn’t actually contain an object of that type.*

- Instead, it contains a *reference to an object* of that type that is resident on the heap!

- This is the *key intuition* and you must understand this in order to understand why object oriented constructs behave the way they do. (when we get there)
We’ve seen this before

• Yet again a variable of a reference type doesn’t actually contain an object of that type.

• An important side effect is that two variables can refer to the same object.

• Remember this?

```java
char[] a1 = new char[];

// Doing this does not copy the array!
// a1 and a2 are now literally the same array.
char[] a2 = a1;
```
Why?

• Consider this code...
• We call a method on line 1.
• What happens?

```java
public class WhyHeap {
    public static void main(String[] args) {
        int[] array = createArray();
        System.out.println(array);
    }

    public static int[] createArray() {
        int[] array = new int[2];
        int[0] = 1;
        int[1] = 2;
        return array;
    }
}
```
Why?

- We create an array on line 9.
- What happens?
Why?

- We return the array at line 12.
- What happens?

```java
public class WhyHeap {

    public static int[] createArray() {
        int[] array = new int[2];
        int[0] = 1;
        int[1] = 2;
        return array;
    }

    public static void main(String[] args) {
        int[] array = createArray();
        System.out.println(array);
    }
}
```
Why?

- If it were not for the heap this code would not work as we expect or be horrifically inefficient.
Intro to Object Oriented Programming
What is OOP

- The most popular programming languages developed in the last 30 years are, for the most part, Object-Oriented languages, but this wasn't always the way.

- The prevailing paradigm before that was “Procedural Programming”

- It had functions, but the source files for large programs would be many thousands of lines long!

- This led to software that was extremely difficult to understand and change.
What is OOP

- OOP was a direct response to that condition
- In an OOP language, this one large program will instead be split apart into self contained objects, almost like having several mini-programs.
- Each object represents a different part of the application. Now each object contains its own data and its own logic, and they communicate and collaborate to execute your program.
What is OOP

• Furthermore, OOP is…
  
  • a programming method that is used to help organize your code sanely when you have complex programs that require a lot of code.
  
  • a set of language constructs that help you organize your code according to this method.
  
  • *That's it.*
Nouns & Verbs

• So how do we make these mini-programs?

• Identify the entities in the program. Our nouns.

• Identify the actions in the program. Our verbs.

• For example, let’s say we had wanted to write the software for an online shopping application.

  • What types of entities would be in the system? ex ‘Shopping Cart’

  • What kinds of actions would be performed by the actors? ex. ‘checkout’
Classes

• A class represents a ‘noun’ in our program

• It has two attributes..
  • State - the variables on the class.
  • Methods - the ways in which the object can interact with its data, the ‘verbs’.
Classes

- If we think of a real-world object, such as a television, it will have several features and properties:
  - We do not have to open the case to use it.
  - We have some controls to use it (buttons on the box, or a remote control).
  - We understand the concept of a television without necessarily understanding how it is built and functions.
  - It is complete when we purchase it, with any external requirements well documented.
  - Very much how you might describe a class!
Classes

• Similarly, a class should...
  
  • Provide a well-defined purpose
  
  • Represent a clear concept
  
  • Be complete and well-documented
  
  • Do one thing and do it well!
Objects

• An object is an instance of a class. What we get when we call `new`

• You could think of a class as the description of a concept, and an object as the realization of this description
Classes & Object Example

```java
public class Rectangle {

    final int length;
    final int width;

    // Constructor.. special kind of function.
    public Rectangle(int w, int l) {
        width = w;
        length = l;
    }

    public int getArea() {
        return width * length;
    }
}
```

```java
public class SomeOtherClass {

    public static void main(String[] args) {
        Rectangle r = new Rectangle(100, 50);
        System.out.println(r.getArea());
    }
}
```