Overview

- Memory Maps
- Parameter Passing
Java Program Memory

Stack

Heap

Static Area (for static variables)
Static Area

- A static field is shared among all instances of the class
- **Example**: Banana.java
- Let’s draw a diagram for the above code
- Notice that count is shared and each increase method call modifies the same count variable
Call Stack

- Stack → Abstract data type that allows insertions/removals only from one end
  - Example: stack of clean plates in a dinner
  - Operations:
    - Push → add an element to the stack
    - Pop → remove an element from the stack
    - Follows a LIFO (Last-In-First-Out) policy
- Java call stack
  - Each time a method is called an entry in this stack is added
  - The entry is called a frame
  - The frame has local variables, parameters and other data
  - Also referred to as the Activation Record Stack
public static void A(int x) {
    int y = x + 2;
    B(y);
}

public static void B(int w) {
    int k = 3;
    System.out.println(k + w);
}

public static void main(String[] args) {
    int k = 10;
    A(k);
}
Memory Maps

- Memory maps will be used to represent the state of the call stack and the heap
- You need to know how to draw memory maps 😊

```
<table>
<thead>
<tr>
<th>Call Stack</th>
<th>Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>a 30</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
</tr>
<tr>
<td>d 754</td>
<td></td>
</tr>
<tr>
<td>q</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td></td>
</tr>
<tr>
<td>args null</td>
<td></td>
</tr>
</tbody>
</table>
```

![Diagram](image)
Call Stack

• The call stack makes it possible for:
  • Local variables to be created/destroyed
    • Variables are created when frame is created and placed in the stack
    • Variables are destroyed when frame is removed from the stack
  • The order in which methods are called is respected
    • If A calls B, and B calls C, then C’s frame must be removed before B’s, and B’s before A’s
  • Us to use the same name for local variables in different methods
    • Each frame “hides” the variable
  • In our examples, the frame will include local variables and parameters
Passing Parameters

• In many languages there are different ways to pass parameters
  • By value
  • By reference

• In Java ALL parameters are passed by value
  • A copy of the argument is used to initialized the parameter
  • If you change the value assigned to a parameter that will not change the value of the argument

• IMPORTANT: It is the nature of what you pass what makes a difference
  • If you pass a primitive type value, nothing can change the argument
  • If you pass a reference then we may change the object associated with the reference in the method that has been called.
    • Remember that a reference variable does not store the object
      • It stores an address
Passing Parameters

• **Primitive Types**
  • Let’s see passing parameters when dealing with primitive types
  • Let’s draw a memory map for the code in Swapping.java

• **Reference Types**
  • Let’s see passing parameters when dealing with reference types
  • Let’s draw a memory map for the code in Increasing.java
  • You MUST DOCUMENT if you method is going to change a value passed via a parameter
    • Some call it a destructive method
    • How would we prevent a parameter from being modified?
      • Copy constructor