The Unreferenced Leaves
Garbage Collect At My Feet

Jonathan Lung – CSC207H: Software Design

October 16, 2015
Following program flow
The Java Memory Model
Using the Java Memory model

Going With the Flow

_____ : To follow the course or trail of.

- It is often useful to _____ program flow.
- The first step to fixing a bug (the Right Way) is to identify and understand the relevant code – what led to the undesired state?
Going With the Flow

- There are three common approaches to ______ the course of execution:
  - Inserting print statements,
  - Use a debugger, and
  - ______ by hand.
- Professional programmers rarely debug using print statements.
The Java Memory Model

- Every interface, class, object, and running method has its own region of memory to track and related information.
- Instructions in a (single-threaded) program execute in a order. Instructions change these regions of memory.
The Java Memory Model

- We represent these regions using *memory boxes*.

<table>
<thead>
<tr>
<th>Name</th>
<th>super</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Contents</td>
<td></td>
</tr>
</tbody>
</table>

- Static information (interfaces, classes) is stored in the *static space*.
- Method information is stored in the ____________.
- We do not draw boxes for ____________ we do not use and did not declare ourselves (e.g., `Object`).
- If we use a class we did not write, we should make one up to the best of our ability.
The Java Memory Model

Note: Memory box for $00005712$ omitted due to lack of space.
Static Space

- Draw a memory box for each class and interface in the program.
- In these boxes, record:
  - the class/interface name (upper-left corner),
  - for a class, the superclass it extends (upper-right corner),
  - the list of interfaces (if any) implemented/extended (upper-right corner),
  - the name(s) and value(s) of static variables (in the main area of the box); this includes interface constants, and
  - for a class, the signatures of any static methods (in the main area of the box).
Object Space

- Whenever a new object is created, draw a new memory box for it in the object space.
- Divide the box horizontally into sub-boxes, one for each ancestor in the class hierarchy, containing instance variables defined for the class and its ancestors with the top-most for class closest to `Object` and bottom-most for the class’ instance variables.
- In the main memory box for the object, record:
  - a memory address not currently in use (upper-left box),
  - the class to which the object belongs (upper-right box).
Object Space

- Each variable starts with the default value for its type, unless it has an initializer. Record the values.
- Execute the constructor as if it was an instance method called on the object through a reference of the same type as the object.
- The constructor starts by executing the code in the superclass default constructor, unless the first statement specifically specifies how to start:
  - `this(...)` selects another constructor of the class via the usual overloading mechanism and
  - `super(...)` selects a superclass constructor via the usual overloading mechanisms.
- When the constructor is done, the value of the `new` expression is the address of the new object.
Resolving Variables and Methods

- Code can refer to local variables and parameters inside a method, instance variables/methods, and static variables/methods. If the code is executing in a class $C$, it can directly refer to the static method of $C$ and $C$’s ancestors, and the interfaces implemented by them.
- If a name is not followed by a dot, look for it first as a local variable or parameter. Failing that . . .
- Look “up” through instance variables and methods until you find it.
Method Calls

- Evaluate them in order – inside out, left to right.
  - Put their values on top of the stack in that order.

- Executing the method
  - Find the target of the method.
  - Draw a frame for the method on the stack.
  - Include information about where the method was found in the upper-right corner.
  - Execute the body of the method line by line.

- Returning values: `return expr;`
  - Evaluate `expr` and replace the current method frame with the value.
  - It disappears if not used immediately (either as an argument to another method or assigned to a variable).