

Potpourri: Memory Leaks and Random

Lecture 29

Memory Management

- Java (generally) manages memory for you
- Every call to “new” creates a new instance
 - Memory allocated to hold instance
- When is this memory released?
 - Answer: when there are no references to this instance
 - eg End of scope
 - ```
void someMethod() {
 someClass x = new someClass();
 . . .
} //x goes out of scope
```
  - (Beware of aliases of course)

## Example “Memory Leak”

```
public class Stack {
 private Object[] elements;
 private int size = 0;

 public Stack (int initialCapacity) {
 elements = new Object[initialCapacity];
 }

 public void push (Object e) {
 ensureCapacity();
 elements[size++] = e;
 }

 public Object pop () {
 if (size == 0)
 throw new EmptyStackException();
 return elements[--size];
 }
}
```

## Example Continued

```
//class Stack continued...

private void ensureCapacity() {
 if (elements.length == size) {
 Object[] oldElements = elements;
 elements = new Object[2*elements.length + 1];
 System.arraycopy(oldElements, 0,
 elements, 0, size);
 }
}
```

## Example Repaired

```
public Object pop() {
 if (size == 0)
 throw new EmptyStackException();
 Object result = elements[--size];
 elements[size] = null;
 return result;
}
```

## Memory Leak: Problem and Solution

- Problem: Keeping obsolete references
  - Stack has array of reference that will *never* be dereferenced
- Solution: explicitly null-out reference
  - `someReference = null;`
- But, do *not* do this needlessly
  - Clumsy and complicates code
- When is it needed?
  - Classes that manage their own memory
  - Classes that keep caches
    - WeakHashMap discards entries when key no longer accessible

## Know The Libraries: Random

Computer Science and Engineering @ The Ohio State University

### □ Generating uniform random [0..bound)

```
import java.util.Random;
Random rnd = new Random(); //time seed
int x = rnd.nextInt(bound);
```

### □ Do *not* scale using 0-argument version

```
int x = Math.abs(rnd.nextInt()) % bound;
```

#### ■ Problems

- No abs for Integer.MIN\_VALUE
- Short repetition period for bounds small power of 2
- Uneven distribution for some bounds

## To Ponder

Computer Science and Engineering @ The Ohio State University

```
static Random rnd = new Random();

static int random(int n) {
 return Math.abs(rnd.nextInt()) % n;
}

public static void main(String args[]) {
 int b = 2 * (Integer.MAX_VALUE / 3);
 int low = 0;
 for (int i=0; i < 1000000; i++)
 if (random(b) < b/2)
 low++;

 System.out.println(low); //prints ~666,666
}
```

## Summary

Computer Science and Engineering @ The Ohio State University

### □ Singleton

- Instantiated at most once
- Private constructor ensures no default constructor
- Static factory returns existing reference
- Lazy initialization defers instantiation

### □ Memory Leaks

- Problem: indefinitely retaining obsolete reference
- Solution: explicit null-out (only when necessary!)

### □ Random

- Use 1-argument (bounded) nextInt method