Announcements

• Lecture examples on CVS
• Quiz Thursday
  – OOP concepts (last Monday through this Wednesday)
  – Study questions up tonight
• Lab tomorrow
  – Grading policy + “challenge question”
• P3 on Wednesday
• Midterm 1
Midterm 1

- Handed back tomorrow in lab
- 1 week for written re-grade requests

\[
\begin{align*}
\leq C & \quad \approx B & \quad \approx A
\end{align*}
\]
Static fields and instance fields

Static fields

everyOnesData 8

Instance fields

obj1
this. onesData 3

obj2
this. onesData 4

obj3
this. onesData 2

obj4
this. onesData 0

obj5
this. onesData 8
Static/instance usage

//Static usage
M yclassname.staticField
e.g. Integer.MAX_VALUE

//Instance usage
myObject.instanceField
e.g. myInt.intValue

Something previously constructed with the new keyword
Static methods and instance methods

Static methods
...obj.onesData...

Instance methods

obj1
this.
onesData

obj2
this.
onesData

obj3
this.
onesData

obj4
this.
onesData

obj5
this.
onesData
Static methods and instance methods

Static methods

...obj.onesData...

Instance methods

obj1
this.onesData
3

obj2
this.onesData
4

obj3
this.onesData
2

obj4
this.onesData
0

obj5
this.onesData
8
Static methods and instance methods

//instance method
public void doSomething() {
    //access this.instanceField
}

//static method
public void doSomething(MyClass myObject) {
    //access myObject.instanceField
}
//Typical constructor
public MyClass(int valueToAssign) {
    this.value = valueToAssign;
}

//Rewritten as a static method
public static MyClass makeNew(int val) {
    MyClass mc = new MyClass(); //default constructor
    mc.value = val;
    return mc;
}
Access modifiers - APIs

Application Programming Interface

- A set of methods available to programmers
- “Encapsulation”: The inner workings behind the method calls are treated like a black box
  - Java APIs
    - String
    - Math
    - Etc.
  - What are the trade-offs of encapsulation?
Access modifiers – member level

• “Members” of a class: fields and methods
• Modifying access to members:
  – public keyword: Accessible to any object of any class
  – private keyword: only accessible to other methods within the same class
  – More to come:
    • package-private (Wednesday)
    • protected (In a few weeks)
• “Top-level” modifiers: modify access to classes as a whole (more on Wednesday)
Memory Model

- **Stack**
  - Contains the local variables used in each method call:
    - primitive data values
    - reference variables (memory addresses)

- **Heap**
  - Contains all of the object instance data
    - Where the reference variables “point to”

- **Permanent generation**
  - Stores class definitions (things like the instructions for each method)
Stack

Stack frame example:

method1

x

3
Stack

Stack frame example:

- method1
- method2
- x
- @
Stack

Stack frame example:

method1

method2

method3

x

7
Stack

Stack frame example:

method1

method2

X @
Stack

Stack frame example:

method1

x 3
Stack vs Heap

This is how you should draw the stack, heap, variables, and objects on exams.
Aliasing

Stack
- a
- b
- c
- d
- e

Heap
- One object
  - Data: 1

- Another object
  - Data: 1
Deeper referencing

Stack
a 3
b @
c 7

Heap
one object
data 1
ref @

another
data 8
ref @

another
data 8
ref ?
The `null` literal

- Used to indicate an empty reference variable (Doesn’t point to a location in memory)
- E.g.
  ```java
  MyClass myObject = null;
  ```
- Not a keyword
Deeper referencing

Stack
a 3
b @
c 7

Heap
one object
  data 1
  ref @

another
  data 8
  ref @

another
  data 8
  ref null
Deeper aliasing

Stack

| a | 3 |
| b | @ |
| c | 7 |
| d | @ |

Heap

one object
- data: 1
- ref: @

another
- data: 8
- ref: @

another
- data: 8
- ref: null
.equals method

• == compares memory addresses
• .equals can compare instance data
• Should contain a null-check:

```java
class MyObject {
  public boolean equals(MyObject other) {
    if (other != null) {
      return this.data == other.data;
    } else {
      return false;
    }
    // Rewrite using a single line?
  }
}
```
Hard coding data structure

//Data for student 01
String student_01_Name;
int student_01_Age;
int student_01_Year;
String student_01_Major;

//Data for student 02
String student_02_Name;
...

//Data for student 03
String student_03_Name;
...
(Partially) soft-coding data structure

//Data for student 01
Student student_01 = new Student("Bob", 17, 2, "CMSC");

//Data for student 02
Student student_02 = new Student("Bill", 20, 4, "CMSC");
...

//Data for student 03
Student student_02 = new Student(...);
...
Fully soft-coding data structure

• Arrays (more later)
  – Contiguous block of memory addresses

• Linked list
  – Non-contiguous: each entry saves the memory address of the subsequent entry

• Trade-offs?