CSE 11

final review

and the last discussion
Recursion

• You have not been tested on this before, so be ready

• Example (from practice final):

```java
public void recurse(int x) {
    if (x < 1)
        System.out.println("complete");
    else {
        System.out.println(x);
        recurse(x - 1);
    }
}
```
Using stack frames

- Recursive functions require you to keep track of previous results
- Draw stack frames to help you keep track of where each recursive call is waiting. Procedure:
  1. Write down all the local variables on the top of the stack
  2. Execute the function like normal
  3. If you hit a recursive call, remember where you are currently executing and go to step 1
  4. If the function hits the end without any more recursive calls, cross out what’s on top of the stack and move down. Resume where that stack frame left off.
- Example: `recurse(0)` using the example function

```java
public static void recurse(int x){
    if (x >= 8)
        System.out.println("complete");
    else {
        System.out.println(x);
        recurse(x + 2);
        recurse(x + 2);
        System.out.println(x);
    }
}
```

Stack frame: Output:

```
  0
  x=0
```
Example

public static void recurse(int x){
    if (x >= 8)
        System.out.println("complete");
    else {
        System.out.println(x);
        recurse(x + 2);
        System.out.println(x);
    }
}
Example

```java
public static void recurse(int x){
    if (x >= 8)
        System.out.println("complete");
    else {
        System.out.println(x);
        recurse(x + 2);
        System.out.println(x);
    }
}
```

Stack frame:

<table>
<thead>
<tr>
<th>x=2</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=0</td>
</tr>
</tbody>
</table>

Output:

0
2
Example

```
public static void recurse(int x){
    if (x >= 8)
        System.out.println("complete");
    else {
        System.out.println(x);
        recurse(x + 2);
        System.out.println(x);
    }
}
```

Stack frame:

<table>
<thead>
<tr>
<th>x=0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=2</td>
</tr>
<tr>
<td>x=4</td>
</tr>
</tbody>
</table>

Output:

0
2
4
Example

```java
public static void recurse(int x){
    if (x >= 8)
        System.out.println("complete");
    else {
        System.out.println(x);
        recurse(x + 2);
        System.out.println(x);
    }
}
```

Stack frame:  

<table>
<thead>
<tr>
<th></th>
<th>x=6</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x=4</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>x=2</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>x=0</td>
<td>6</td>
</tr>
</tbody>
</table>

Output:
Example

```java
public static void recurse(int x){
    if (x >= 8)
        System.out.println("complete");
    else {
        System.out.println(x);
        recurse(x + 2);
        System.out.println(x);
    }
}
```

<table>
<thead>
<tr>
<th>Stack frame:</th>
<th>Output:</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=8</td>
<td>0</td>
</tr>
<tr>
<td>x=6</td>
<td>2</td>
</tr>
<tr>
<td>x=4</td>
<td>4</td>
</tr>
<tr>
<td>x=2</td>
<td>6</td>
</tr>
<tr>
<td>x=0</td>
<td>complete</td>
</tr>
</tbody>
</table>
Example

```
public static void recurse(int x){
    if (x >= 8)
        System.out.println("complete");
    else {
        System.out.println(x);
        recurse(x + 2);
        System.out.println(x);
    }
}
```

Stack frame:

<table>
<thead>
<tr>
<th>x = 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 4</td>
</tr>
<tr>
<td>x = 2</td>
</tr>
<tr>
<td>x = 0</td>
</tr>
</tbody>
</table>

Output:

0
2
4
6
complete
Example

```
public static void recurse(int x){
    if (x >= 8)
        System.out.println("complete");
    else {
        System.out.println(x);
        recurse(x + 2);
        System.out.println(x);
    }
}
```

Stack frame:

<table>
<thead>
<tr>
<th>x=6</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=4</td>
</tr>
<tr>
<td>x=2</td>
</tr>
<tr>
<td>x=0</td>
</tr>
</tbody>
</table>

Output:

0
2
4
6
complete
6
Example

```java
public static void recurse(int x)
{
    if (x >= 8)
        System.out.println("complete");
    else {
        System.out.println(x);
        recurse(x + 2);
        System.out.println(x);
    }
}
```

<table>
<thead>
<tr>
<th>Stack frame:</th>
<th>Output:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td>x=4</td>
<td>4</td>
</tr>
<tr>
<td>x=2</td>
<td>6</td>
</tr>
<tr>
<td>x=0</td>
<td>complete</td>
</tr>
<tr>
<td></td>
<td>6</td>
</tr>
</tbody>
</table>
Example

```java
public static void recurse(int x){
    if (x >= 8)
        System.out.println("complete");
    else {
        System.out.println(x);
        recurse(x + 2);
        System.out.println(x);
    }
}
```

<table>
<thead>
<tr>
<th>Stack frame:</th>
<th>Output:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td>x=2</td>
<td>6</td>
</tr>
<tr>
<td>x=4</td>
<td>complete</td>
</tr>
<tr>
<td>x=0</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>
Example

```java
public static void recurse(int x)
{
    if (x >= 8)
        System.out.println("complete");
    else {
        System.out.println(x);
        recurse(x + 2);
        System.out.println(x);
    }
}
```

Stack frame:

<table>
<thead>
<tr>
<th>x=2</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=0</td>
</tr>
</tbody>
</table>

Output:

0
2
4
6
6
complete
6
4
Example

```java
public static void recurse(int x) {
    if (x >= 8)
        System.out.println("complete");
    else {
        System.out.println(x);
        recurse(x + 2);
        System.out.println(x);
    }
}
```

Stack frame:

```
0
2
4
6
x=2
x=0
```

Output:

```
0
2
4
6
complete
6
4
2
```
Example

```java
public static void recurse(int x)
{
    if (x >= 8)
        System.out.println("complete");
    else {
        System.out.println(x);
        recurse(x + 2);
        System.out.println(x);
    }
}
```

Stack frame:

```
   x=0
```

Output:

```
0
2
4
6
6
4
2
complete
```
Example

```java
public static void recurse(int x){
    if (x >= 8)
        System.out.println("complete");
    else {
        System.out.println(x);
        recurse(x + 2);
        System.out.println(x);
    }
}
```

Stack frame:

Output:

```
0
2
4
6
complete
6
4
2
0
```
Example

```java
public static void recurse(int x) {
    if (x >= 8)
        System.out.println("complete");
    else {
        System.out.println(x);
        recurse(x + 2);
        System.out.println(x);
    }
}
```

- Remember: take things off the stack after the function finishes

Stack frame: Output:

```
  
  
  
  0
  2
  4
  6
  complete
  6
  4
  2
  0
(finish)
```
A trick when looking at recursive functions

- Some recursive functions are simple loops in disguise
- No need to use stack frames!
- This trick only works when nothing executes after the recursive call
- (for the curious: this is known as tail recursion)

```java
public void recurse(int x) {
    if (x < 1)
        System.out.println("complete");
    else {
        System.out.println(x);
        recurse(x - 1);
    }
}
```

Just restart from the beginning with the new value of x.

The recursive call ends the function. No need to remember where we left off.
Threads

• When you want to run multiple things at the same time.

• In general, threads are super unpredictable

```java
new Thread(){
    public void run(){
        // am i running...?
    }
}.start();

new Thread(){
    public void run(){
        // ... or am i running...?
    }
}.start();

new Thread(){
    public void run(){
        // ... or are we all running at the exact same time?
    }
}.start();
```

• The underlying thread scheduler has a mind of its own. It will run your threads whenever it wants to.
Abstract classes

- Abstract classes can have abstract methods
- Abstract classes can also have real code that use the abstract methods
- Abstract method: “I don’t know what this function does yet, but I’ll let someone else deal with it.”

```java
public abstract class Politician {
    abstract String getEnemy();
    public String action() {
        return getEnemy() + " must be dealt with";
    }
}

class BernieSanders extends Politician{
    String getEnemy() {
        return "Wall Street";
    }
}

class DonaldTrump extends Politician{
    String getEnemy() {
        return "Islam";
    }
}
Interfaces

• Similar to abstract classes, but all methods are abstract

```java
public interface Counter {
    void count(int i);
}

class CountUp implements Counter{
    int count;
    public void count(int i) {
        count += i;
    }
}

class CountDown implements Counter{
    int count;
    public void count(int i) {
        count -= i;
    }
}

Counter[] counters = {new CountUp(), new CountDown()};
for(Counter c : counters){
    //count up, count down, doesn't matter
    //with a common interface they look the same
    c.count(4);
}

• objects can always be cast back to their interface
Extending stuff

• **super**: only takes effect where it’s defined, not how it’s used.

```java
public class Dog {
    public void speak(){
        System.out.println("woof");
    }
}

public class SuaveDog extends Dog {
    public void speak(){
        super.speak();
        System.out.println("Excuse me good sir, may I interest you in a game of fetch?"};
    }
}

Dog jimmy = new SuaveDog();
Dog dog = (Dog)jimmy;
```

//On the outside, we may have typecasted jimmy to an average dog
//but on the inside, he's still a cool dog, and speaks the way he wants
//there's some profound life lesson going on here
dog.speak();

• Jimmy does what he wants. Jimmy doesn’t conform to your typecasting. Be like Jimmy.
Big-O

• Nothing too tricky here since you weren’t exposed to the full theory behind big-O.

• Constant time, O(1):
  ```java
  long x = 0;
  for (int i = 0; i < 10000; i++) {
    for (int j = 0; j < 10000; j++) {
      for (int k = 0; k < 10000; k++) {
        // slow, but still constant time
        x += i * j * k;
      }
    }
  }
  }
  ```

• Linear time, O(n):
  ```java
  for (int i = n; i > 0; i--) {
    for (int j = 100; j > 20; j--) {
      System.out.println(i + j);
    }
  }
  ```

• Quadratic time, O(n^2). Usually when there’s a nested loop.
  ```java
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j += 10) {
      System.out.println(j * i);
    }
  }
  ```
Big-O

• Not quadratic, the inner loop is $O(1)$:
  ```java
  for (int i = 0; i < n; i++) {
      for (int j = n; j < n + 10; j++) {
          System.out.println(i * j);
      }
  }
  ```

• Log time, $O(\log n)$
  ```java
  for (int i = n; i != 0; i /= 3) {
      System.out.println(i * 3);
  }
  ```

• $O(\log n)$ when the problem size gets divided at each step

• Here’s a big giveaway for log time: you can multiply the input size by a constant, but you’ll just get a constant amount of time added to the old runtime.

• Replace $n$ with $3n$. Does it go 3 times slower?

• Nope, it just runs (previous number of iterations) + 1
Other stuff you should know:

- Unit testing with JUnit
- Data structures (Map, Set, ArrayList, etc.)
- try/catch/finally
- short-circuiting operators ‘||’ and ‘&&’
  
  - true  ||  (don’t care) is true, false  &&  (don’t care) is false

- Extending classes. Especially constructors (like the 1st question of quiz2)
- Method overriding/overloading
- UML
- Basic Linux commands and shell operators
- Stack and heap
  
  - Stack: local variables and primitives
  
  - Heap: pretty much everything that you create with ‘new’
Stay hungry
Stay foolish