As you arrive...play around with the hashCode function in the Double class. The person who can find the number between 0 and 1 that has the largest hashCode() and emails it to me before 10:07am gets a prize.

```
Double d = new Double(.000000003);
System.out.println(d.hashCode());
```
Recitation 4: compareTo, equals, and hashCode()

In which all objects are not created equal, but some are not unequal in the way we want
Solution to the prep problem

```java
public int compareTo(ThreeInts other) {
    int sum = myOne + myTwo + myThree;

    // I could also make a sum() function so I wouldn't
    // have to use other's fields directly
    int otherSum = other.myOne + other.myTwo + other.myThree;

    return sum - otherSum;
}
```
Code Writing Time!

1. Create a new class ComplexNumber using ThreeInts as your guide

2. ComplexNumber objects should have two double fields myR and myI

3. Write a compareTo for complex numbers. Complex numbers should be compared using their magnitudes. The magnitude of a complex number is $\sqrt{r^2 + i^2}$. So for two complex numbers a and b, $a > b$ if

$$\sqrt{r_a^2 + i_a^2} > \sqrt{r_b^2 + i_b^2}$$

4. Once you’re done, grab the main code linked off the syllabus page and test to make sure your implementation works (just the main function in the ComplexNumber.java).
Any questions about compareTo?
Bearing in mind you will have to write a compareTo function as part of the upcoming Markov assignment.
An Annoying Problem

ComplexNumber a = new ComplexNumber(1,7);
ComplexNumber b = new ComplexNumber(1,7);

if(a.equals(b))
{
    System.out.println("The complex numbers are equal");
} else {
    System.out.println("The complex numbers are not equal");
}
hashCode()...an odd function

- “cat” hashes to 98262
- “bat” hashes to 97301
- “act” hashes to 96402
- [4] hashes to 35
- [4, 6] hashes to 1091
- [4, 6] hashes to 1091
The rules of the `hashCode()` function

- Every object should have a `hashCode`
- An object’s `hashCode` should not change unless its value changes
- Any two objects which are `equals()` to each other should have the same `hashCode`. So if you change the way `equals()` works for your object, you must change the way `hashCode` works.
- If two objects are not equal to each other, they should have different `hashCodes`... and they should be close to random (if not, you get slowness)
Hints for Making HashCodes

1. Don’t write your hashCode() function from scratch...use some of Java’s existing hashcode functions in creative ways

2. Computing a hashcode is slow (often) so save the value so you don’t have to recompute it again
ThreeInts Example
Add equals() and hashCode() to the ComplexNumber class

• Two complex numbers are equal if their real and imaginary parts are equal
• Remember that hashCode *must* be equal if the real and imaginary parts are equal (you’ll get very funky bugs otherwise)
• Also try and ensure that the hash isn’t equal when they are not equal. In particular…try and ensure that order matters (e.g. the hashcode of 7 + 3i does not equal the hashcode of 3 + 7i)
• Once again you can test your code with the othermain from the Syllabus page