Today …

• Go over Quiz 6
• Finish up Exercise 10, Parts 1 & 2
• More on the Object class (hashCode)
• Exercise Part 3
• Exercise Part 4 (if time)
Quiz 6

Example of using assert to test Account

```java
public class Account implements Cloneable {
    private int balance;
    public Account(int balance) {
        this.balance = balance;
    }
    public int getBalance() {
        return balance;
    }
    public Account clone() throws CloneNotSupportedException {
        return (Account)super.clone();
    }
    public static void main(String [] args) throws CloneNotSupportedException {
        Account a1 = new Account(100);
        Account a2 = new Account(100);
        Account a3 = a1.clone();
        assert a1 != a2 : ("" + a1 + " != "+ a2 + " FAILED");
        assert a1 != a3 : ("" + a1 + " != "+ a3 + " FAILED");
        assert a2 != a3 : ("" + a2 + " != "+ a3 + " FAILED");
    }
}
```

This would also work (in this case)
return new Account(balance);
Be sure you understand why!

The equals() Object method

Hints for implementing equals()

1. Check if (this == obj) ... remember operator=
2. Use instanceof to check for correct type
   - If you do not use instanceof, and use getClass() instead, you also need to check that (obj != null)
3. Cast the argument to the correct type
4. For each "significant" field, check if the argument's field value is equal to current object's field value
   - For primitive fields, use ==
   - For reference fields, use equals() (can be complicated)
5. Double check that it is satisfies the equals contract
Exercise 10

- Do parts 1 and 2
- Try using the `assert` keyword for your tests
- Remember to use the `java -ea` flag

Exercise 10, Parts 1 & 2

Note: I put `Point` and `ColorPoint` in different files
- Each contains a main method for testing
- You could have also put these into a single file
- But only one class in the file can be public
The equals() Object method

equals() must also be

– “Consistent”:
  • For any non-null reference values x and y
  • Multiple calls to x.equals(y) must return same value

– “Non-nullity”:
  • For any non-null reference values x
  • x.equals(null) must return false

– And there is one more constraint ...

A common mistake

What is wrong with this equals() method for Point?

```java
public boolean equals(Point obj) {
    return obj.x == x && obj.y == y;
}
```

• This doesn’t work because
  – We are overloading equals … not overriding it!
  – Good practice to use the @Override annotation here
public int hashCode()

• What does it do?
  – Returns an integer representation of an object

• When should it be used?
  – You’ll rarely use this (if ever)
  – It is used by the default implementation of toString()
  – It is also used in various collections
  – E.g., HashMap, HashSet, and Hashtable
  – We’ll talk more about these later

• What is the default implementation (in Object)?
  – Derived from the object’s memory address
  – (This is typical, but not required by Java)

• Should we override hashCode()?  
  *If we override equals() then we must override hashCode()*!
  – It is required by the contract for equals()
The `hashCode()` Object method

How should we override `hashCode()`?

Another contract

- **Consistent**
  - Return same value for the same object (during program execution)

- **Equality**
  - If two objects are equal according to the `equals()` method
  - Then return the same integer for both objects

- **Inequality**
  - It is OK to return the same value for non-equal objects
  - May lead to poor hashtable performance though! (Chaining)

• Ensuring equal objects have the same hash code values causes most problems
  - Otherwise, we could just use the default implementation
  - But, this won’t work since we can have 2 different objects that are equal (i.e., 2 different memory locations)

• Trivial to write a `hashCode()` satisfying the contract:

  ```java
  public int hashCode() { return 41; }
  ```
  - But this degrades hash tables to linked lists!
The hashCode() Object method

• The hashCode() method in String

  public int hashCode() {
    int hash = 0;
    for (int i = 0; i < length(); i++)
      hash = 31 * hash + charAt(i);
  }

• In general, you will want to do something similar

  – Assign each field a hash code, and give it a “weight”
  – Goal is to evenly distribute the values (doesn’t need to be a perfect hash function)

Exercise 10

• Do part 3

• Use the assert keyword for your tests

• Remember to use the java -ea flag

• Note:

  – Compute hash code from the parts of points, color points
  – For primitive values (e.g., int), convert to wrapper types (e.g., Integer) to obtain a hash code