Classes And Objects II
Recall the LightSwitch Class

class LightSwitch {

    boolean on = true;

    boolean isOn() {
        return on;
    }

    void switch() {
        on = !on;
    }
}

A Different LightSwitch Class

class LightSwitch {

    int on = 1;

    boolean isOn() {
        return on == 1;
    }

    void switch() {
        on = 1 - on;
    }
}

Abstraction

- Both LightSwitch classes behave the same.

- We treat LightSwitch as an **abstraction**: we do not care about the internal code of LightSwitch, only the external behavior.

- Internal code = *implementation*

- External behavior = *interface*
Why is Abstraction Important?

- We can continue to refine and improve the implementation of a class so long as the interface remains the same.

- All we need is the interface to an Object in order to use it, we do not need to know anything about how it performs its prescribed behavior.
A user of LightSwitch that relied on the boolean field would break if we changed to an integer field

```java
class AbstractionBreaker {
    public static void main(String[] args) {
        LightSwitch ls = new LightSwitch();

        if (ls.on) // now broken!
            System.out.println("light is on");
        else
            System.out.println("light is off");
    }
}
```
Public versus Private

- Label fields and methods **private** to ensure other classes can't access them.

- Label fields and methods **public** to ensure other classes can access them.

- If they are not labeled public or private, for now consider them public.
A Better LightSwitch

class LightSwitch {

    private boolean on = true;

    public boolean isOn() {
        return on;
    }

    public void switch() {
        on = !on;
    }

}
Enforcing the Abstraction Barrier

- By labeling the `on` field private ...

```java
class LightSwitch {
    private boolean on = true;
    // ...
}
```

- Now AbstractionBreaker's attempt to access the `on` field would not have compiled to begin with.

```java
if (ls.on)  // would never have compiled
```
Equality Quiz 1

- **Is** \((a == b)\) ?

  ```java
  int a = 7;
  int b = 7;
  ```

  **Answer: Yes**

- **Is** \((g == h)\) ?

  ```java
  Person g = new Person("Jamal", 26);
  Person h = new Person("Jamal", 26);
  ```

  **Answer: No**
Primitives vs Objects

- Two datatypes in Java: *primitives* and *objects*

- Primitives: byte, short, int, long, double, float, boolean, char
  
  `==` tests if two primitives have the same value

- Objects: defined in Java classes
  
  `==` tests if two objects are the same object
The **new** keyword always constructs a new unique instance of a class.

When an instance is assigned to a variable, that variable is said to *hold a reference or point* to that object.

```java
Person g = new Person("Jamal", 26);
Person h = new Person("Jamal", 26);
```

`g` and `h` hold references to two different objects that happen to have identical state.
Reference Inequality

- \( g \neq h \) because \( g \) and \( h \) hold references to different objects

```java
Person g = new Person("Jamal", 26);
Person h = new Person("Jamal", 26);
```

```
"Jamal"  
26
```

```
"Jamal"  
26
```
Reference Equality

- `greg1 == greg2` because `greg1` and `greg2` hold references to the same object

```java
Person greg1 = new Person("Greg", 23);
Person greg2 = greg1;
```
Equality Quiz 2

true or false?

Person g = new Person("Jamal", 26);
Person h = new Person("Jamal", 26);
Person greg1 = new Person("Greg", 23);
Person greg2 = greg1;

a) g == h  
   false

b) g.getAge() == h.getAge()  
   true

c) greg1 == greg2  
   true

d) greg1.getAge() == greg2.getAge();  
   true
Java API

- You can get information on all in-built Java classes/methods by browsing the Java Application Programming Interface (API)

- This documentation is essential to building any substantial Java application

- Available on your CD's