Review

• We’ve covered a lot of topics in the last couple of lectures
  – Elements of a class
  – Abstract data types
  – Access specifiers: private, public, default
  – instance and class methods
  – instance and class variables
  – constructors
  – toString() method

Let’s put it all together

• Write a class Clock that represents a 24 hour clock
  – Midnight = 0
• This class should have a constructor that takes
  three parameters: hour, minute, and second
• Include the following instance methods:
  – add( Clock c ) add c to a Clock instance
    • For example, add 1:00:00 to 9:30:00
  – toString() - format c as "HH:MM:SS"

Let’s put it all together

• Access
  – instance variables should be accessible only
    from the class
  – instance methods should be accessible from
    everywhere

Class Constants

• It is generally a bad idea for instance or class
  variables to be public
  – It is better to provide accessor and mutator methods
  – This allows us to guarantee certain conditions about
    the data
• However, there is one type of class data that is
  commonly made public: Constants
  – Immutable variables with special values
Class Constants and Final

- **Examples**
  - Math.PI
  - Integer.MAXINT
- Constants generally written in UPPER CASE
- Defined with the keyword `final`

  ```java
  public static final double PI = 3.14159265;
  ```

- Any attempt to modify the value of a constant will result in an error.

Why Use Constants?

- Constants are only defined once
  - Some numbers, such as pi, are frequently used
- Constants allow us to name a value
  - This makes the code more clear
  - Which is easier to understand?
    - 12 or MONTHSPERYEAR

Clock constants

- Let's add some class constants to the Clock class
  - SECONDSPERMINUTE
  - MINUTESPERHOUR
  - HOURSPERDAY

Calling Methods

- There are three ways to call a method, depending on
  - whether or not the method is in the same class
  - whether the method is an instance method or a class method
Calling Methods in the Same Class

• If a method is in the same class as the calling method, just use the method name
  – a = foo(b);
  – Instance Instance
  – Instance Class
  – Class Class
  – Class cannot call Instance
  • Why not?

Calling Methods in Another Class

• From another class:
  – To call an instance method, you must use the name of an instance of the class
    String s = "abc";
    int a = s.length();
  – To call a class method, you must use the name of the class
    double r = Math.random();

Calling Methods

• Let's add a method tick() to our clock class
  – Advance the time 1 second
  – Call another method in Clock to implement this method.

Accessing Another Object's Private Variables

• An instance method operating on one object can access another object’s private fields
  – Remember, private limits variable access to methods in the class
  – The add() method of Clock does this

Clock when = new Clock(15, 30, 0);
Clock howLong = new Clock(1, 0, 0);
when.add(howLong);

– Implementation details are hidden from other classes, but not from objects in the same class
Passing Objects to Methods

- As with Arrays, objects are references
  - So, when we pass an object to a method, we can change its value

```java
class PassingReferences {
    public static void main(String[] args) {
        StringBuffer sbuf = new StringBuffer("testing");
        System.out.println("sbuf is now " + sbuf);
        modify(sbuf);
        System.out.println("sbuf is now " + sbuf);
    }
    static void modify(StringBuffer sb) {
        sb.append("123");
    }
}
```

Passing Objects to Methods

- But, you can't change the value of the reference

```java
class ModifyParameters {
    public static void main(String[] args) {
        StringBuffer sbuf = new StringBuffer("testing");
        System.out.println("sbuf is now " + sbuf);
        modify(sbuf);
        System.out.println("sbuf is now " + sbuf);
    }
    static void modify(StringBuffer sb) {
        sb = new StringBuffer("doesn't work");
    }
}
```