Announcements

- **A0** has been graded
  - Everyone who submitted gets a grade of 1 (the max)
  - We're **not checking** submissions! We wanted you to learn how to make sure that assert statements are executed.

- We're pleased with how many people are already working on **A1**, as evidenced by Piazza activity
  - Please be sure to look at [Piazza note @68](#) every day for frequently asked questions and answers
  - It's due Friday night
  - **Groups:** Forming a group of two? Do it **well before** you submit — at least one day before. **Both members must act:** one invites, the other accepts. Thereafter, only **one** member has to submit the files.

- **A2:** Practice with strings
  - Now available on course website + CMS
- A bit about testing and test cases
- Class Object, superest class of them all.  
  Text: C.23 slide 30
- Function toString() C.24 slide 31-33
- Overriding a method C15–C16 slide 31-32
- Static components (methods and fields) B.27 slide 21, 45
- Java application: a program with a class that declares a method with this signature:
  
  ```java
  public static void main(String[])```
  

Homework

1. Read the text, Appendix A.1–A.3
2. Read the text, about the if-statement: A.38–A.40
3. Visit course website, click on Resources and then on Code Style Guidelines. Study
   - 2. Format Conventions
   - 4.5 About then-part and else-part of if-statement
Test case: Set of input values, together with the expected output.

Develop test cases for a method from its specification --- even before you write the methods body.

/** = number of vowels in word w. 
Precondition: w contains at least one letter and nothing but letters */ 

public int numberOfVowels(String w) {
    ...
}

Developing test cases first, in “critique” mode, can prevent wasted work and errors

How many vowels in each of these words? 
creek
syzygy
/** Constructor: worker with last name n, SSN s, boss b (null if none).
   Prec: n not null, s in 0..999999999 with no leading zeros.*/
public W(String n, int s, W b)
/** = worker's last name */
public String getLname()
/** = last 4 SSN digits */
public String getSsn()
/** = worker's boss (null if none) */
public W getBoss()
/** Set boss to b */
public void setBoss(W b)

Contains other methods!
**Java**: Every class that does not extend another extends class Object. That is,

```java
public class W {...}
```

is equivalent to

```java
public class W extends Object {...}
```

We often leave off this to reduce clutter; we know that it is effectively always there.

We draw object like this

```plaintext
W@af
```

```
| toString() |
| equals(Object) |
| hashCode() |
```

```
| Iname |
| ssn |
| boss |
| W(…) |
| getLname() |
| getSsn(), getBoss() |
| setBoss(W) |
```

```
W
```

```
Object
```

```
“Obama”
```

```
123456789
```

```
nul
```

```
null
```
A note on design

- Don’t use `extends` just to get access to hidden members!

```java
public class PhD {
    protected String name;
    ...
}

public class PhDTester extends PhD {
    ...
    if (student.name == ...) ...
}
```
A note on design

- Don’t use `extends` just to get access to hidden members!
A note on design

- Don’t use \texttt{extends} just to get access to hidden members!
- A should extend B if and only if A “is a” B
  - An elephant is an animal, so \texttt{Elephant extends Animal}
  - A car is a vehicle, so \texttt{Car extends Vehicle}
  - An instance of any class is an object, so \texttt{AnyClass extends java.lang.Object}
  - A PhDTester is not a PhD student!
- The inheritance hierarchy should reflect \textit{modeling semantics}, not implementational shortcuts
What is “the name of” the object?

The name of the object below is

\texttt{PhD@aa11bb24}

It contains a pointer to the object –i.e. its address in memory, and you can call it a pointer if you wish. But it contains more than that.

Variable \texttt{e}, declared as

\texttt{PhD e;}

contains not the object but the name of the object (or a pointer to the object).

\texttt{e} \texttt{PhD@aa11bb24}
**Method toString**

`toString()` in **Object** returns the name of the object: `W@af`

**Java Convention**: Define `toString()` in any class to return a representation of an object, giving info about the values in its fields.

New definitions of `toString()` **override** the definition in **Object.toString()**

In appropriate places, the expression `c` automatically does `c.toString()`

`c.toString()` calls this method
Method toString

toString() in Object returns the name of the object:  W@af

public class W {

    ... 

    /** Return a representation of this object */
    public String toString() {
        return “Worker ” + lname + “. “ + “ S oc sec: “ + getSSn() + “.” + 
        (boss == null ? “” : “Boss ” + boss.lname + “.”);
    }

    c.toString() calls this method
Another example of toString()

/** An instance represents a point \((x, y)\) in the plane */

```java
public class Point {
    private int x;  // x-coordinate
    private int y;  // y-coordinate

    ...

    /** = repr. of this point in form \(\langle x, y \rangle\) */
    public String toString() {
        return "\(\langle \)" + x + ", " + y + "\rangle\";
    }
}
```

Function toString should give the values in the fields in a format that makes sense for the class.
What about **this**

- **this** keyword: **this** evaluates to the name of the object in which it occurs
- Let’s an object instance access its own object reference
- Example: Referencing a shadowed class field

```java
public class Point {
    public int x = 0;
    public int y = 0;

    //constructor
    public Point(int x, int y) {
        x = x;
        y = y;
    }
}
```

```java
public class Point {
    public int x = 0;
    public int y = 0;

    //constructor
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```
Intro to static components

/** = “this object is c’s boss”. Pre: c is not null. */
public boolean isBoss(W c) {
    return this == c.boss;
}

Spec: return the value of that true-false sentence. True if this object is c’s boss, false otherwise

keyword this evaluates to the name of the object in which it appears

x.isBoss(y) is false
y.isBoss(x) is true
/** = “b is c’s boss”.  
  Pre: b and c are not null. */
public boolean isBoss(W b, W c) {
  return b == c.getBoss();
}

/** = “this object is c’s boss”.  
  Pre: c is not null. */
public boolean isBoss(W c) {
  return this == c.boss;
}
/** = “b is c’s boss”.
Pre: b and c are not null. */

public static boolean isBoss(W b, W c) {
    return b == c.getBoss();
}

static: there is only one copy of the method. It is not in each object

Preferred:
W.isBoss(x, y)

Box for W (objects, static components)
Good example of static methods

- `java.lang.Math`
- [http://docs.oracle.com/javase/7/docs/api/java/lang/Math.html](http://docs.oracle.com/javase/7/docs/api/java/lang/Math.html)
Java application

Java application: bunch of classes with at least one class that has this procedure:

```java
public static void main(String[] args) {
    ...
}
```

Type `String[]`: array of elements of type `String`. We will discuss later

Running the application effectively calls the method `main`

Command line arguments can be entered with `args`
Uses of static variables:
Maintaining info about created objects

```java
public class W {
    private static int numObjects;
    ...

    /** Constructor: */
    public W(...) {
        ...
        numObjects = numObjects + 1;
    }
}
```

To have `numObjects` contain the number of Objects of class `W` that have been created, simply increment it in constructors.

Box for `W`
public class Singleton {
    private static final Singleton INSTANCE = new Singleton();

    private Singleton() { } // ... constructor

    public static Singleton getInstance() {
        return INSTANCE;
    }

    // ... methods
}

Uses of static variables:
Implementing the Singleton pattern

Only one Singleton can ever exist.