Announcements

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 @10 every day for any updates.
- Also search existing questions!
- 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. If one of you submits before forming the group, the course staff will have to do extra work, and you'll receive a small penalty of 4 points.
- Reminder: groups must complete the assignment working together.

Big ideas so far

- Java variables have types (L1)
  - A type is a set of values and operations on them
    (int: +, -, *, /, %, etc.)
- Classes define new types (L2)
  - Methods are the operations on objects of that class.
  - Fields allow objects to store data (L3)
- A software engineering principle: give user access to functionality, not the implementation details...

Today's topics

- Class Object
- Extends, is-a
- Method toString(), object names, overriding
- Keyword this, shadowing
- Static components

Review: Method specs should not mention fields

public class Time {
  private int hr; // in 0..23
  private int min; // in 0..59

  /** return hour of day */
  public int getHour() {
    return hr;
  }

  /** return hour of day */
  public int getHour() {
    return min / 60;
  }
}

Running example: Class W (for Worker)

/** 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 getName() {
    return n;
  }

  /** = last 4 SSN digits */
  public String getSan() {
    return s % 10000;
  }

  /** = worker's boss (null if none) */
  public W getBoss() {
    return b;
  }

  /** Set boss to b */
  public void setBoss(W b) {
    this.boss = b;
  }

  @Override
  public String toString() {
    return name + "\n" + ssn + "\n" + boss;
  }
}

Contains other methods
Class Object

Java: Every class that does not extend another extends class Object. That is, public class W {...} is equivalent to public class W extends Object {...}

We often omit this partition to reduce clutter; we know that it is always there.

Extends: “Is A”

- Extension should reflect semantic data model: meaning in real world
- A should extend B if and only if A “is a” B
  - An elephant is an animal, so Elephant extends Animal
  - A car is a vehicle, so Car extends Vehicle
  - An instance of any class is an object, so AnyClass extends java.lang.Object

Extends: “Is A”

- Which of the following seem like reasonable designs?
  1. Triangle extends Shape { ... }
  2. PhDTester extends PhD { ... }
  3. BankAccount extends CheckingAccount { ... }

  A. Yes! A triangle is a kind of shape.
  B. No! A PhDTester “tests a” PhD, but itself is not a PhD.
  C. No! A checking account is a kind of bank account; we likely would prefer: CheckingAccount extends BankAccount { ... }

Investigate: JFrame

1. How many levels deep is JFrame in the class hierarchy?
   - (Object is JFrame’s super-super-superclass. How many supers are there?)

2. In which class is JFrame’s getHeight() method defined?
   - [hint: it’s not JFrame!]

What’s in a name?

The name of the object below is PhD@aa11bb24
The name is <class> @ <address in memory>.

Variable e, declared as PhD e, contains not the object but the name of the object (i.e., it is a reference to the object).

<table>
<thead>
<tr>
<th>PhD@aa11bb24</th>
<th>PhD</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>&quot;Gries&quot;</td>
</tr>
<tr>
<td>ad1</td>
<td>null</td>
</tr>
<tr>
<td>ad2</td>
<td>null</td>
</tr>
<tr>
<td>advises</td>
<td>17</td>
</tr>
</tbody>
</table>
Method toString()

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

```java
public class W {
    public String toString() {
        return "Worker " + lname + " has SSN ??-??-?? + " + getSsn() + (boss == null ? " and boss " + boss.lname);
    }
}
```

### Another example of toString()

```java
/** An instance represents a point (x, y) in the plane */
public class Point {
    private int x; // x-coordinate
    private int y; // y-coordinate

    public String toString() {
        return "(" + x + ", " + y + ");"
    }
}
```

### this: the object’s own name

- **this** keyword: this evaluates to the name of the object in which it occurs
- Makes it possible for an object to access its own name
- Example: a shadowed class field

```java
public class Point {
    public int x= 0;
    public int y= 0;
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

### Static components

- **this** = “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

```java
public boolean isBoss(W c) {
    return this == c.boss;
}
```

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

  Body doesn’t refer to any field or method in the object.
  Why put method in object?

```java
public boolean isBoss(W b, W c) {
    return b == c.getBoss();
}
```
**Static components**

```java
/** = b is c’s boss. */
public static boolean isBoss(W b, W c) {
    return b == c.getBoss();
}
```

**A use for static fields (aka class variables): Maintain info about created objects**

```java
public class W {
    private static int numObs; // number of W objects created
    public W(...) {
        numObs = numObs + 1;
    }
}
```

**Class java.awt.Color uses static fields**

```java
public static final Color black = ...;
public static final Color blue = ...;
public static final Color cyan = new Color(0, 255, 255);
public static final Color darkGray = ...;
public static final Color gray = ...;
```

**Java application**

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

**Uses of static fields: Implement the Singleton pattern**

```java
public class Singleton {
    private static final Singleton instance = new Singleton();
    private Singleton() {
        // ... constructor
    }
    public static Singleton getInstance() {
        return instance;
    }
    // ... methods
}
```
Looking ahead: Recitation 3

- No prework! Concentrate on A1 this weekend
- TA teaches testing; you test a class using Junit
- You can work in groups of up to 3; form a CMS group before submitting
- You will find faults in the class (fun!) and fix them
- Upload to CMS when done
  - Hopefully during recitation
  - If not, on/by Friday