Lecture 5: Class & interface hierarchy

- Classes vs. interfaces
- Abstract classes
- Inheritance, upside-down, inside out
- Exit
**Interfaces**

- is a specification
  - list of methods (not an implementation)
  - Javadoc comments
    - preconditions
    - postconditions
    - conceptual idea
    - efficiency constraints

**Classes**

- a specification + an implementation
  - list of fields
  - class variants
  - code to implement methods

- can extend (one) other class.
- can extend (many) other classes.

```java
interface I1
{/* no begged */
  int f();}

interface I2 extends I1{
  /* return a positive int */
  int f2();}
```

- can add more methods
- can add restrictions on existing methods
- can't remove methods or restrictions.
  (actually: can contain "final" fields: constants)

```java
class TimeImpl {...}

class SecondsTimeImpl extends TimeImpl {
  ...
  inherits all the details of parent class
  "subclass" extends "superclass".
```
interface Time
interface SecondsTime extends Time

every SecondsTime is a Time.

Time t;
SecondsTime st;
t = st; // makes sense, because every ST is a T.
t . minutes();
st = t; // all I know is that t is a Time.

error!
st . seconds();

class diagram

<table>
<thead>
<tr>
<th>hours()</th>
</tr>
</thead>
<tbody>
<tr>
<td>minutes()</td>
</tr>
</tbody>
</table>

extends

<table>
<thead>
<tr>
<th>ST Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>seconds()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ST Mutable Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>set Hours (int)</td>
</tr>
<tr>
<td>set Minutes (int)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ST Mutable SecondsTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>set Seconds (int)</td>
</tr>
</tbody>
</table>

interface MutableST
extends ST, MT
**Keyword**

Super: means use Superclass implementation.

class Sub extends Superclass {
    int f() {
        super.f();
        return 3;
    }
}

Sub() x = new Sub();

**Java Code**

```java
class C1 {
    int f() {
        return 1;
    }
}

class C2 {
    int f() {
        return 2;
    }
}

class C3 {
    int f() {
        return 3;
    }
}

class Car {
    // Other class
}

interface Airplane {
    boolean canFly();
    int getSpeed();
}

class AirplaneCoy {
    boolean canFly() {
        return true;
    }
}

// Method f returns >0 and <0.

A: For dunno

A: Too many

B: cannot have multiple superclasses.

Subclass can override a method of its superclass (i.e., can have a new implementation w/ same type & args)

```