CS 11114

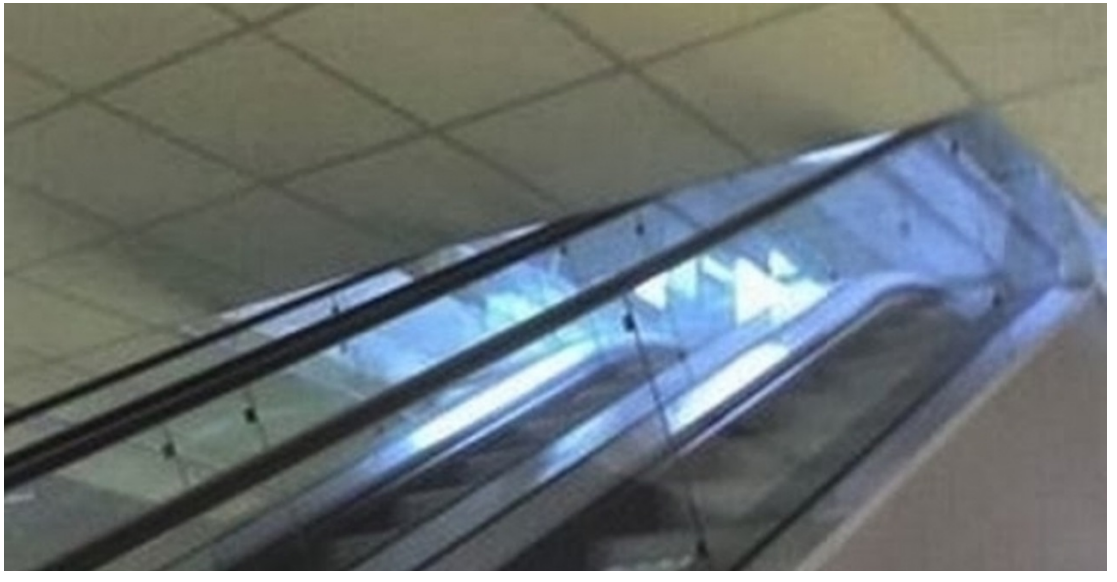
Introduction to Software Design

Spring 2017 - Michael Irwin

Force Add Procedure

- https://www.cs.vt.edu/S17Force-Adds (https://www.cs.vt.edu/S17Force-Adds)
- Go to our lecture - Irwin MW (1:25)
- Password: 1114msi!
- Open during only first and second lectures

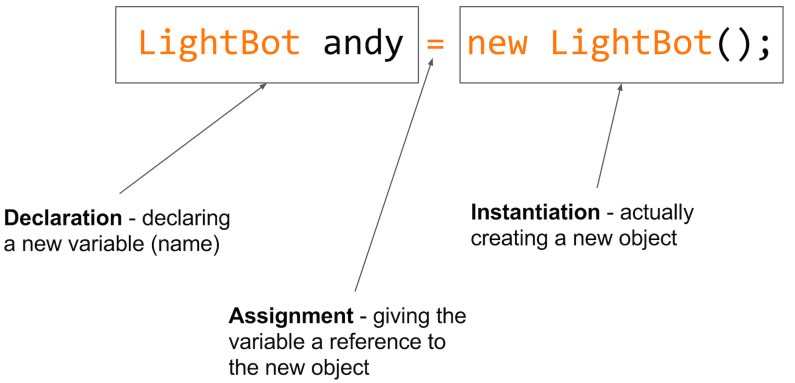
Who needs headroom?



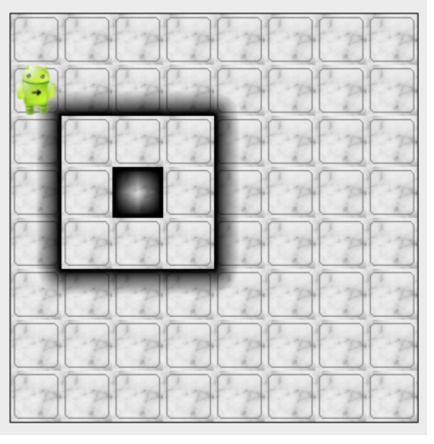
Who needs privacy?

Something doesn't line up

Creating Objects



Our scenario for the day...



The simple way...

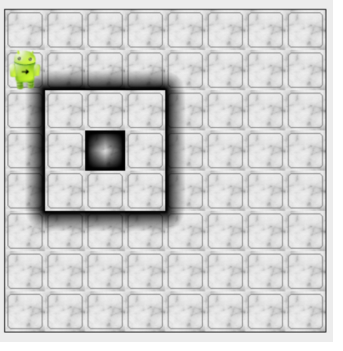
- We could simply tell the bot to move four times, turn right, move four times, turn right...

```
bot.move();
bot.move();
bot.move();
bot.move();
bot.turnRight();


bot.move();
bot.move();
bot.move();
bot.move();
bot.turnRight();
```

```
bot.move();
bot.move();
bot.move();
bot.move();
bot.turnRight();


bot.move();
bot.move();
bot.move();
bot.move();
bot.turnRight();
```



Introducing class inheritance

- Could we make our bot smarter, so he could patrol on his own?
- Two problems with doing that…
  - Not all LightBots will patrol (cohesion)
  - We don't have the source for `LightBot` anyways :(

```
public class <NewClass> extends <ParentClass> {
  // New stuff goes here
}
```

- The `NewClass` will inherit all methods from `ParentClass`

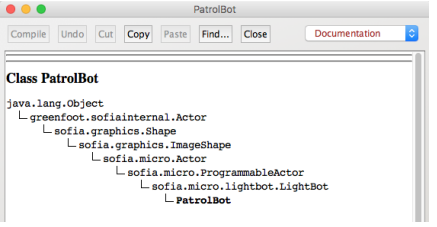
Making our PatrolBot

```
public class PatrolBot extends LightBot {
  // New stuff goes here
}
```

- `PatrolBot` inherits all methods of `LightBot`, but can have its own methods
  - Example... it has methods named `move()`, `turnRight()`, `turnLeft()`
  - We can add our own `patrolCastle()` method
- `PatrolBot` is the *child* or *subclass* of `LightBot`
- `LightBot` is the *parent* or *superclass* of `PatrolBot`

Parent vs Child

- You can view the JavaDoc for a class by switching the editor into **Documentation** mode.
- Parents are listed at top, with children branching underneath
- Every class in Java extends from `Object` (don't need to explicitly extend it)



Creating new methods

```
public <ReturnType> <methodName>() {
  // Method body here
}
```

```
public void patrolCastle() {
  // Method body here
}
```

- The `public` access modifier indicates anyone can call the method
  - We'll talk about other access modifiers later in the semester
- The `<ReturnType>` indicates what will be returned
  - If nothing will be returned, use `void`
- The method's name should reflect what it will do
- Don't forget to document your new methods

Updating PatrolBot

- Add the `patrolCastle()` method to the `PatrolBot` and use code we wrote earlier
- Only change... it's no longer `bot.move()`, but simply `move()`. Why??

```
public void patrolCastle() {
  move();
  move();
  move();
  move();
  turnRight();

  move();
  move();
  move();
  move();
  turnRight();

  // Walk the wall two more times...
}
```

Making it cleaner

- Created a helper method named `walkOneWall`. It's also `public`, so can be invoked from outside the `PatrolBot`.

```
public class PatrolBot extends LightBot {

  /**
   * Perform a single patrol around the castle
   */
  public void patrolCastle() {
    walkOneWall();
    walkOneWall();
    walkOneWall();
    walkOneWall();
  }

  public void walkOneWall() {
    move();
    move();
    move();
    move();
    turnRight();
  }

}
```