CS1114
Intro to Software Design
Michael Irwin - Fall 2019

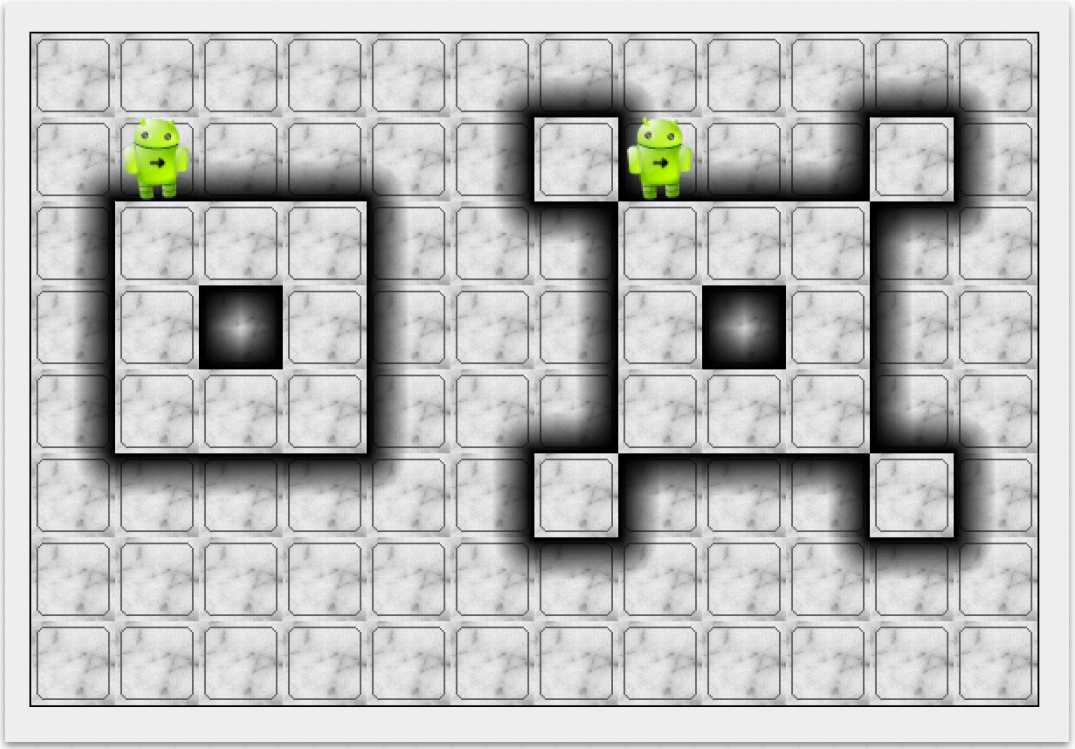
Events/Reminders

HW #1 due tonight at 11:59pm
Lab 2 this week
Reading Quiz 2 due Sunday at 11:59pm
Program 1 due Sept 10
Gobblerfest on Friday from 4-7pm

VIRGINIA TECH

Program #1

Our scenario for the day...



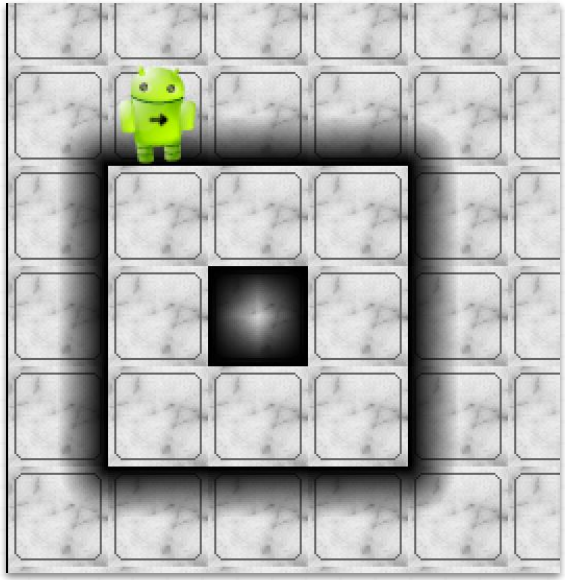
VIRGINIA TECH™

A modified PatrolBot

- walkOneWall will now use a new method to turn around the corner
- Today's scenario starts with this refactor already completed

```
public void walkOneWall() {
    this.move();
    this.move();
    this.turnCorner();
}

public void turnCorner() {
    this.move();
    this.turnRight();
    this.move();
}
```



Let's make a new bot!

- Rather than updating the `PatrolBot`, let's make a specialized bot that can work with the turrets
- Since we still want the behavior of the `PatrolBot`, it'll be the parent
  - Our new bot will inherit the `turnCorner()` and `patrolCastle()` methods

```
public class TurretBot extends PatrolBot {
    // New stuff will go here
}
```

Overriding methods

- Overriding methods allows a subclass to change the behavior of a method defined in a parent class
- To override a method, the method must:
  - Have the same return type (`void` for our methods so far)
  - Have the same name
  - Have the same arguments (none for our methods so far)

```
public class TurretBot extends PatrolBot {
    public void turnCorner() {
      // New behavior goes here
    }
}
```

The updated `turnCorner`

- Putting this into the `TurretBot`, it should now work

```
public void turnCorner() {
    this.turnLeft();
    this.move();
    this.turnRight();
    this.move();
    this.move();
    this.turnRight();
    this.move();
    this.move();
    this.turnRight();
    this.move();
    this.turnLeft();
}
```

VIRGINIA TECH

Any smells here?

We have some repetition!

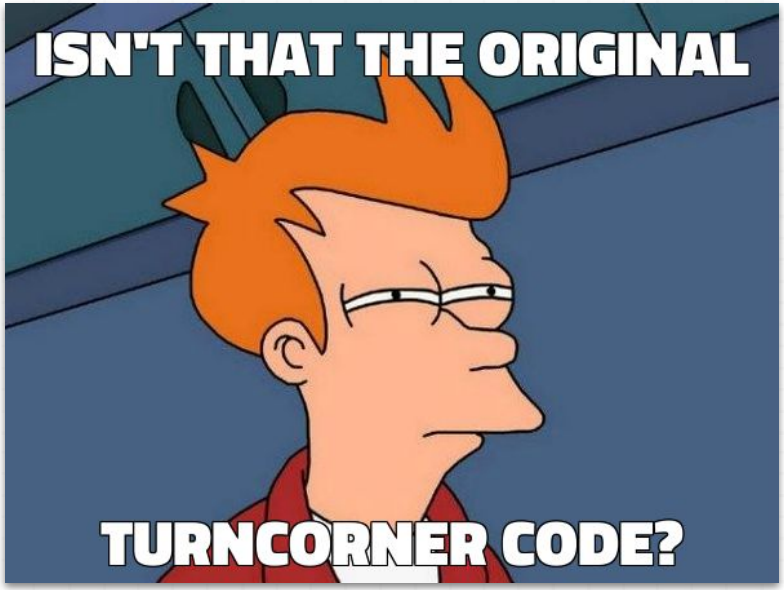
```
public void turnCorner() {
    this.turnLeft();

    this.move();
    this.turnRight();
    this.move();

    this.move();
    this.turnRight();
    this.move();

    this.move();
    this.turnRight();
    this.move();

    this.turnLeft();
}
```



VIRGINIA TECH

Introducing super

- The super keyword allows us to invoke methods on a parent class
- Used quite frequently when overriding methods

```
public void turnCorner() {
    this.turnLeft();

    this.move();
    this.turnRight();
    this.move();

    this.move();
    this.turnRight();
    this.move();

    this.move();
    this.turnRight();
    this.move();

    this.turnLeft();
}
```

```
public void turnCorner() {
    this.turnLeft();

    super.turnCorner();
    super.turnCorner();
    super.turnCorner();

    this.turnLeft();
}
```

VIRGINIA TECH

Polymorphism

> Polymorphism allows the expression of some sort of contract, with potentially many types implementing that contract (whether through class inheritance or not) in different ways, each according to their own purpose. Code using that contract should not have to care about which implementation is involved, only that the contract will be obeyed."
>
> - http://stackoverflow.com/a/409982/502139

```
// What's going to happen here?
PatrolBot bot = new TurretBot();
bot.patrolCastle();
```

WHAAA?!?

VIRGINIA TECH™

Why did it work that way?

- The first line creates a variable named `bot` with a type of `PatrolBot`
  - Anything that is assigned to bot MUST be a `PatrolBot`
  - Statement is valid because all `TurretBot`s are `PatrolBot`s
- When we invoke a method on `bot`, it's calls the code on the assigned object
  - In this case, it's invoking the `patrolCastle` method on the `TurretBot`

```
// What's going to happen here?
PatrolBot bot = new TurretBot();
bot.patrolCastle();
```