Asking For Help

- Start homeworks As Soon As Possible
- You may post homework questions to Canvas: Please do not cut and paste entire lines of code, but rather ask conceptual questions.
- You may also ask me questions via email.
- I am often here early, or have time for a question after class.

HINTS

- Carefully read the error messages in the console
- Look for the line number, and then at that line
- Fix anything that is easy to fix, and re-compile or re-run.
- Repeat
- When in doubt, carefully read the instructions.

Reminder: Class v. Object

Classes

- Describe a type of thing
- Have default values or attributes which must be initialized
- Generic student
- Only one class definition is needed

Objects

- Describe a particular instance of a thing
- "instantiate" a class by defining non-default values
- The student "Ada"
- Have many objects of the same class

Make Your Own Robot Commands

- So far we've done simple stuff
- We're (eventually) going to be doing more complicated stuff
- We need ways to make the program seem more simple, so we don't get (as) lost.

- Today: Creating a new type of robot!

Appendix F.1, Chapter 2.1, 2.2
Extending a Class

- Extension (B extends A)
- Extending the Robot Class
- Superclass and Subclass
- Constructor
- Adding a Service
  - turnAround();
  - turnRight();
- The This Keyword (Implicit Parameter)
- Putting It All Together

Constructor

```
import becker.robots.*;


public class Example extends Object
{
   public static void main(String[] args)
   {
      City bothell = new City();

      Robot Kelsey = new Robot(bothell, 3, 3, Direction.NORTH, 0);
```

Here, when we create a new **instance** (an **object**) of the **Robot** class, a default **constructor** works in the background to make sure that *Kelsey* **inherits** all the **attributes** and **methods** available to Robots, including its **placement** on a particular **Street** and **Avenue** and **Direction** in a particular **City**, and that it can use all of the actions (**methods**) available to the **Robot** class (including **move()**, **pickThing()**, **turnLeft()**, **putThing()**, **frontIsClear()**, etc.)

http://www.learningwithrobots.com/doc/

```
import becker.robots.*;

public class Example extends Object
{
   public static void main(String[] args)
   {
      City bothell = new City();

      Robot Kelsey = new Robot(bothell, 3, 3, Direction.NORTH, 0);

      new Thing(bothell, 2, 2);
      new Wall(bothell, 3, 3, Direction.EAST);
      new Wall(bothell, 3, 3, Direction.NORTH);
      new Wall(bothell, 3, 3, Direction.SOUTH);

      Kelsey.turnLeft();
      Kelsey.move();
      Kelsey.turnLeft();
      Kelsey.turnLeft();
      Kelsey.turnLeft();
      Kelsey.move();
      Kelsey.pickThing();
      Kelsey.turnLeft();
      Kelsey.turnLeft();
      Kelsey.turnLeft();
      Kelsey.move();
      Kelsey.move();
      Kelsey.turnLeft();
      Kelsey.turnLeft();
      Kelsey.turnLeft();
      Kelsey.move();
      Kelsey.putThing();
      Kelsey.turnLeft();
      Kelsey.move();
   }
}
```

These methods are all Robot methods, inherited from the Robot class when the new instance of the Robot class called "Kelsey" was made.

Now … what if there is an action that you might want **Kelsey** to do that isn't found in the **Robot** class?

For instance, instead of invoking the the **turnLeft()** method three times, you could just call up a **turnRight()** ?

```
Kelsey.turnLeft();
Kelsey.turnLeft();
Kelsey.turnLeft();
```

The problem is, the Robot class does **not** have a **turnRight()** command (**method**). The Robot class has been **finalized**. You cannot **add** to it.

The good news is, you can create a **new method** like **turnRight()** that will do what you want the robot to do!

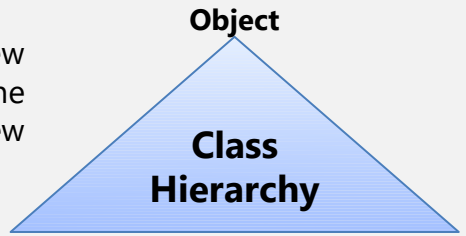
But in order to make this happen, you need to **extend** the **Robot** class …

Extending a Class: Where Class B extends Class A

In Plain Ol' English: Where Class B "inherits" the attributes and actions of Class A … then adds new functionality to them.

When we're not interested in extending a class because we're happy with the methods that come with that class just the way they are, then we declare our class the 'normal' default way: **public class Example extends Object**

Object is the top class of all class hierarchies. When a new instance of ***anything*** is made in Java, then it inherits **all** the *attributes* and *actions* of the **Object** class. You can't get a new object without **Object**.



However, if we want to add new functionality (**methods**) to the **Robot** class (like **turnRight**) then we need to **extend** the **Robot** class (which is *itself* an extension of **Object**)

public class MrRoboto extends Robot

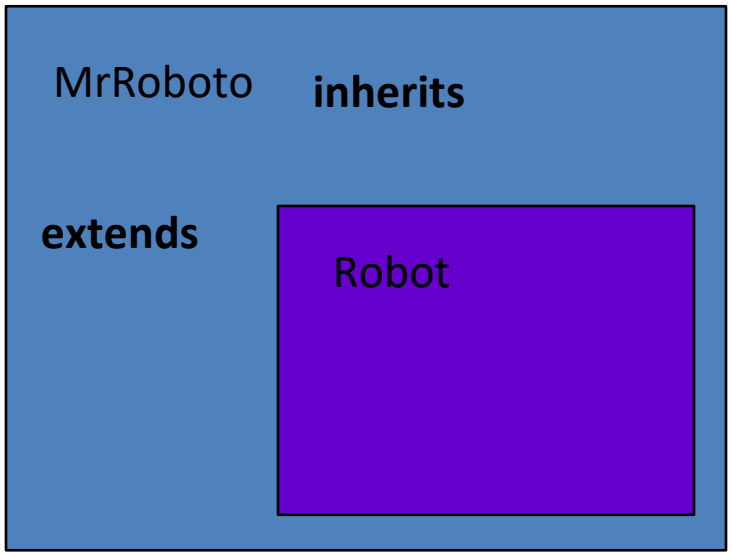
Extending a Class: Where ClassB extends ClassA

Instance vs. Extension?

Instance creates a new ***object*** from a **class**, but **extension** extends a new *class* from a class through *inheritance*, allowing for an improved class that might offer additional attributes and services (**methods**) not available in the original class …

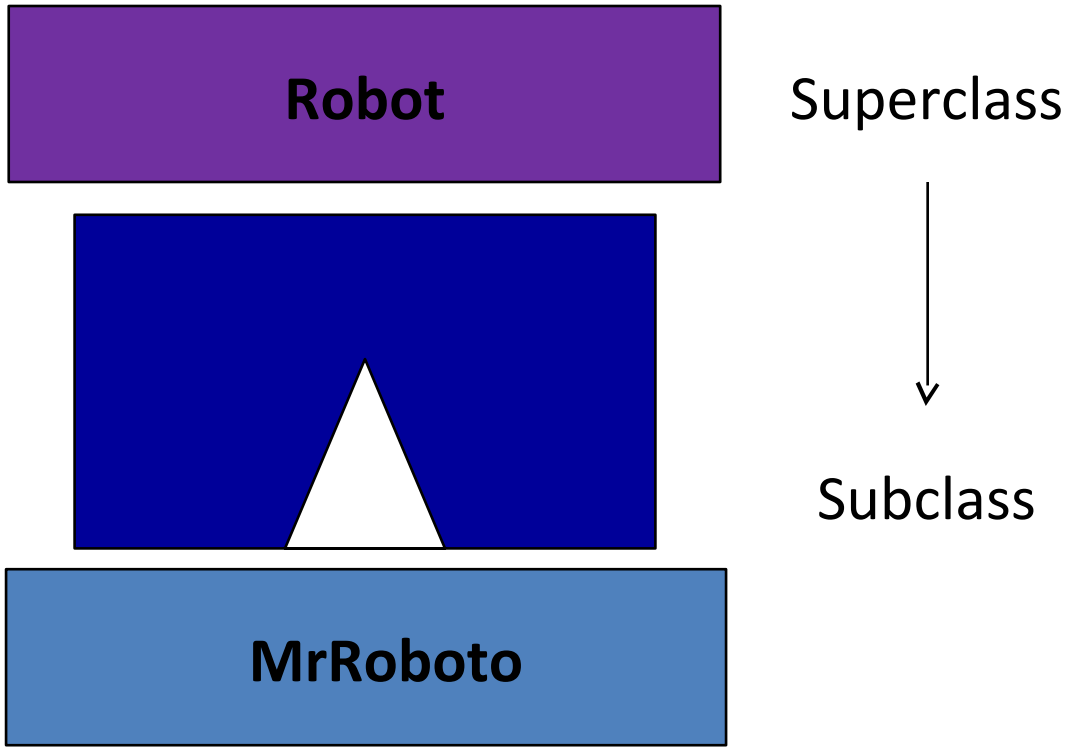
Extending the Robot Class

```
public class MrRoboto extends Robot
```



MrRoboto "inherits" all of the **Robot** attributes and services and then can have additional attributes and services of its own (i.e., those not shared by **Robot**).

Superclass and Subclass



Constructor

```
import becker.robots.*;


public class Example extends Object
{
   public static void main(String[] args)
   {
      City bothell = new City();

      Robot Kelsey = new Robot(bothell, 3, 3, Direction.NORTH, 0);
```

Remember, here the constructor gives our new object, Kelsey, all the attributes and services associated with the Robot class. We are passing the constructor parameters such as the city, and where Kelsey is standing.

http://www.learningwithrobots.com/doc/

Constructor

```
public class MrRoboto extends Robot
{
   public MrRoboto(City theCity, int street, int avenue, Direction aDirection)
     {
      super(theCity, street, avenue, aDirection);
     }
}
```

Here, when we define a new class, MrRoboto, that extends the existing class Robot. The new class has access to all the parameters and services of the super class.
We need to define a new constructor for the new class, and we pass through the parameters to the constructor that sets up the super class. We explicitly do this by making the first command in the new constructor a call to the 'super' constructor for the super class.

http://www.learningwithrobots.com/doc/

Adding New Services

```
public void turnAround()
{  this.turnLeft();
   this.turnLeft();
}

public void move3()
{  this.move();
   this.move();
   this.move();
}
.
public void turnRight()
{  this.turnAround();
   this.turnLeft();
}
```

The **this** *keyword*

The new Java feature in the new services we created is the use of the **this** keyword.

The keyword **this** is useful when you need to refer to an instance of the class from its method, but without having to refer to it by a specific name. Why? Because when you create the new method, you don't know the name of the particular robot that is going to use it, so 'this' is a kind of placeholder name.

The **this** keyword helps us to avoid name conflicts, and also creates a shortcut to having to invent a unique name for each field in the different methods.

```
public void turnAround()
  {  this.turnLeft();
     this.turnLeft();
  }
```

Putting It All Together

Two Ways of Doing the Same Thing, however:

THE CLASS THAT CONTAINS MAIN HAS TO BE THE SAME NAME AS THE FILE

```
1 import becker.robots.*;
2
3 public class MrRoboto extends Robot
4 {
5    // Construct a new MrRoboto
6    public MrRoboto(City theCity, int avenue, int street, Direction aDirection)
7    {  super(theCity, avenue, street, aDirection);
8    }
9
10    public void turnAround()
11   {  this.turnLeft();
12      this.turnLeft();
13   }
14
15   public void move3()
16   {  this.move();
17      this.move();
18      this.move();
19   }
20
21   public void turnRight()
22   {  this.turnAround();
23      this.turnLeft();
24   }
25
26   public static void main(String[] args)
27   {  City bothell = new City();
28      MrRoboto lisa = new MrRoboto(bothell, 3, 2, Direction.SOUTH);
29
30      lisa.move3();
31      lisa.turnRight();
32      lisa.move3();
33      lisa.turnAround();
34      lisa.move3();
35      lisa.turnLeft();
36      lisa.move3();
37      lisa.turnAround();
38   }
39 }
```

Version 1: One Class
MrRoboto.java

```
1 import becker.robots.*;
2
3 public class MrRoboto2 extends Robot
4 {
5    // Construct a new MrRoboto2
6    public MrRoboto2(City theCity, int avenue, int street, Direction aDirection)
7    {  super(theCity, avenue, street, aDirection);
8    }
9
10    public void turnAround()
11   {  this.turnLeft();
12      this.turnLeft();
13   }
14
15   public void move3()
16   {  this.move();
17      this.move();
18      this.move();
19   }
20
21   public void turnRight()
22   {  this.turnAround();
23      this.turnLeft();
24   }
25 }
26
27
28
29
```

```
1 import becker.robots.*;
2
3 public class MrRobotoTest2
4 {
5    public static void main(String[] args)
6    {  City bothell = new City();
7       MrRoboto2 lisa = new MrRoboto2(bothell, 3, 2, Direction.SOUTH);
8
9       lisa.move3();
10      lisa.turnRight();
11      lisa.move3();
12      lisa.turnAround();
13      lisa.move3();
14      lisa.turnLeft();
15      lisa.move3();
16      lisa.turnAround();
17   }
18 }
19
```

Version 2: Two Classes
MrRobotoTest2.java