Variables and Classes

In which we over-ride the super class, work with multiple classes, and move towards non-robot based programming

Classes

```
Class <NewClassName> extends <SuperClass> {

    Class attributes:
    <datatype> <instanceVariableName>
    Static <datatype> <class Variable Name>

    Class services:
    <return datatype> <methodName> () {
        <datatype> <localVariableName>
        // code block defining service
        // behavior.
    }
```

Reminders:

- Classes are generic, instances (or objects) are specific versions.
- Classes contain data (attributes) and methods (services) associated with a coherent idea.

Variable Types

Variable Type	How it is defined	Scope (how long it lasts)	Uses
Local Variable	<type> Name = value; In function or method code block.	Until the closing bracket for the code block it is in.	Temporary use in a method, or loop.
Parameters	<type> Name in method declaration. Value is passed when method is called	For the duration of the Method. Copies value inside of variable.	Passing information to a Method.
Instance Variables (Non-static Fields)	<type> Name during class definition	For the lifetime of an object (an instance of the class)	Values that help define an instance of the class and persist through multiple service calls
Class Variables (Static Fields)	Static <type> Name during class definition	For the life time of the class (can be accessed through class definition)	Values that are the same for EVERY instance of the class

Over-riding inherited methods

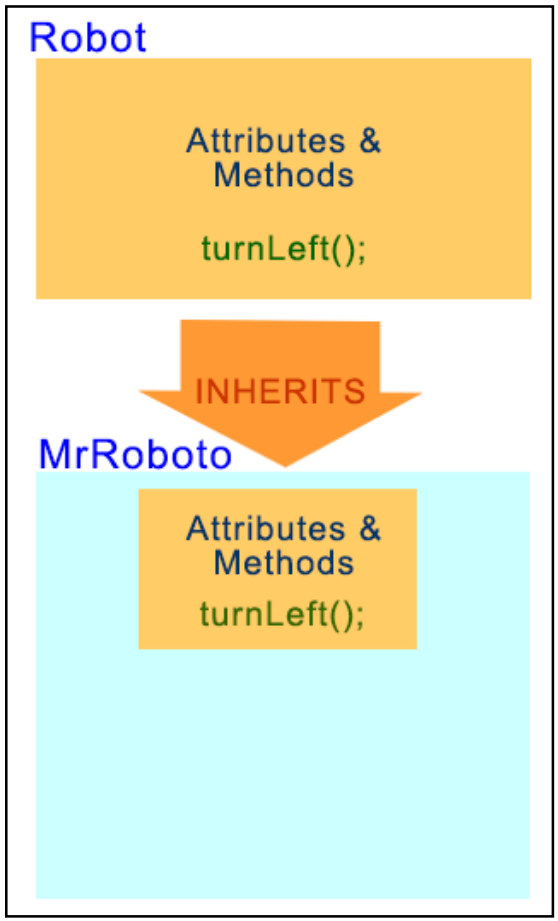
```
import becker.robots.*;

public class MrRoboto extends Robot
{ // Construct a new MrRoboto
  public MrRoboto(City theCity, int street, int avenue, Direction aDirection)
  {
    super(theCity, street, avenue, aDirection);
  }

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

  public void move3()
  {
    this.move();
    this.move();
    this.move();
  }

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



*All public methods are inherited by a subclass

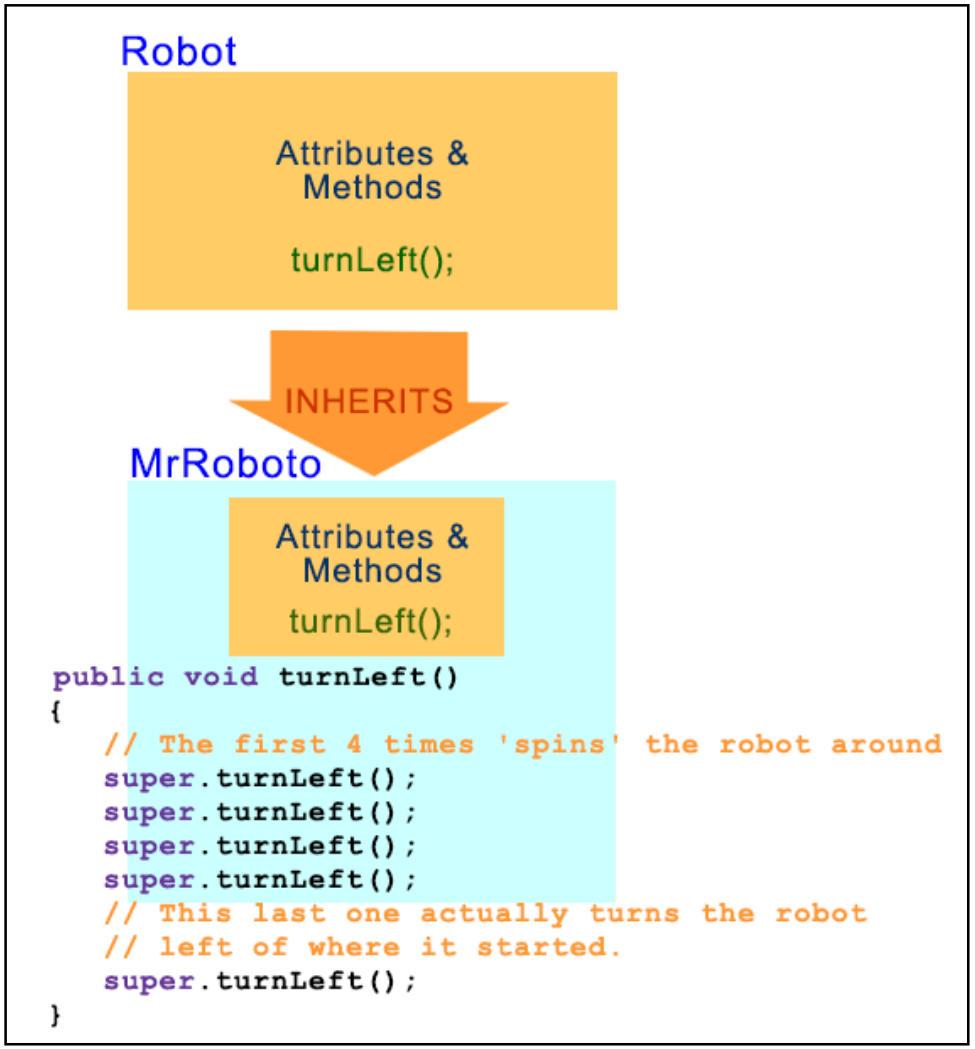
Over-riding

Java allows us to **override** methods **inherited** from the **superclass** using the **super.** ('super dot') keyword in the **method**.

Both the **original** method and the **overriding** method must:

- have the same **name**
- declare the same **data type**
- accept same number of **parameters**
- return the same **data type** *(we'll be going over* return *in the next lecture)*

Overriding allows classes polymorphism: subclasses support all behavior of superclasses, but may implement it in ways specific to the subclass.




```
public void turnLeft()
{
   // The first 4 times 'spins' the robot around
   super.turnLeft();
   super.turnLeft();
   super.turnLeft();
   super.turnLeft();
   // This last one actually turns the robot
   // left of where it started.
   super.turnLeft();
}
```

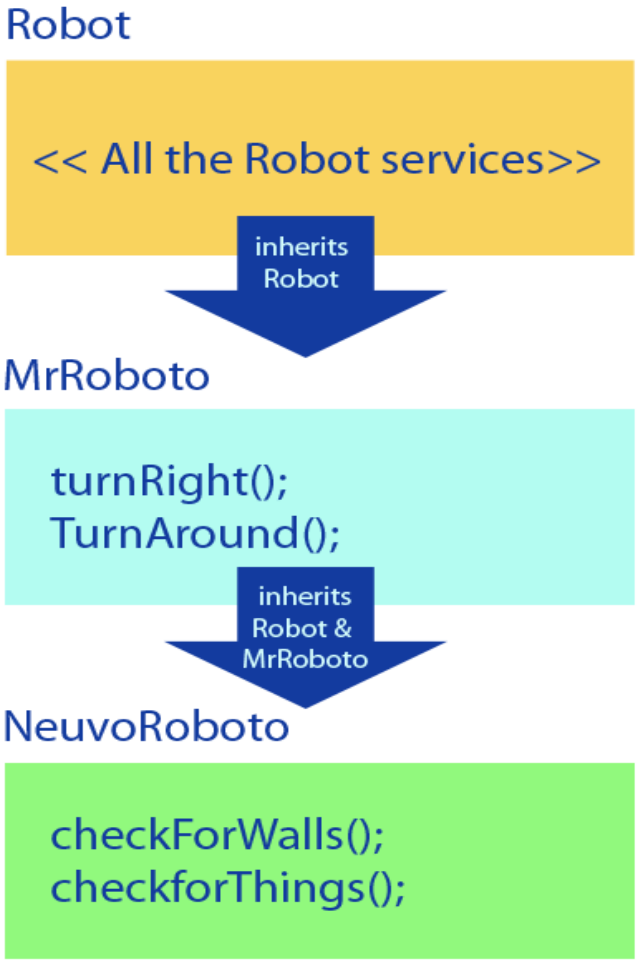
Overriding: Use Super

```
public class SpinningRobot extends Robot
{ // Construct a new SpinningRobot
  public SpinningRobot(City theCity, int street, int avenue, Direction aDirection)
  {
    super(theCity, street, avenue, aDirection);
  }

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

  public turnLeft()  // override the turnLeft so the robot spins!
  {
    super.turnLeft();  // can't use “this.turnLeft();” because it would refer
    super.turnLeft();  // back to the SpinningRobot version and cause
        super.turnLeft();  // an infinite loop.
        super.turnLeft();
        super.turnLeft();
  }
```

Again: Classes are Building blocks!



- New classes can inherit from subclasses, and the same rules apply.
- Most code is made of many, many classes.
  - (Robot class, City class, Scanner class, Random class, etc.)

Multiple Files

Separating main from new *classes* and new class *methods* to create *more manageable code*

- a **main** file
- one or more **class/method** files
- an optional .txt **text** file for configuration, additional input, or output (like creating a log file)

Things to remember when dealing with multiple .java files:

- **class** name must match **file** name
- **All files** must be grouped **together** in the **same area** (folder, directory, desktop, drive, etc.)
- jGrasp *automatically* saves any changes made in the **class** files when compiling **main**

Using only one file

```
import becker.robots.*;

  public class MrRoboto extends Robot
  {
     // Construct a new MrRoboto
     public MrRoboto(City c, int s, int a, Direction d)
     {  super(c, s, a, d);
     }

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

     public void move3()
     {  this.move();
        this.move();
        this.move();
     }

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

     public static void main(String[] args)
     {  City lfp = new City();
        MrRoboto lisa = new MrRoboto(lfp, 3, 2, Direction.SOUTH);

        lisa.move3();
        lisa.turnRight();
        lisa.move3();
        lisa.turnAround();
        lisa.move3();
        lisa.turnLeft();
        lisa.move3();
        lisa.turnAround();
     }
  }
```

```
import becker.robots.*;

  class MrRoboto extends Robot
  {
     public MrRoboto(City c, int s, int a, Direction d)
     {  super(c, s, a, d);
     }

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

     public void move3()
     {  this.move();
        this.move();
        this.move();
     }

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

  public class MrRobotoMain extends Object
  {
     public static void main(String[] args)
     {
        City lfp= new City();
        MrRoboto lisa = new MrRoboto(lfp, 3, 2, Direction.SOUTH);

        lisa.move3();
        lisa.turnRight();
        lisa.move3();
        lisa.turnAround();
        lisa.move3();
        lisa.turnLeft();
        lisa.move3();
        lisa.turnAround();
     }
  }
```

Using Two Files!

```
import becker.robots.*;              Needs its own 'import' statement

  class MrRoboto extends Robot
  {
     // Construct a new MrRoboto
     public MrRoboto(City theCity, int avenue, int street, Direction aDirection)
     {  super(theCity, avenue, street, aDirection);
     }

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

     public void move3()
     {  this.move();
        this.move();
        this.move();
     }

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

In this case, since there are <u>two</u> files, then the **class names** must match the **files names**, and <u>both</u> files must be in the same folder/directory. Each file needs to include the line **import becker.robots.*;** as well.

```
import becker.robots.*;            Needs its own 'import' statement

  public class MrRobotoMain extends Object
  {
     public static void main(String[] args)
     {
        City bothell = new City();
        MrRoboto lisa = new MrRoboto(bothell, 3, 2, Direction.SOUTH);

        lisa.move3();
        lisa.turnRight();
        lisa.move3();
        lisa.turnAround();
     }
  }
```

Always **compile** the file that contains **main** when working with multiple files, since you cannot compile a file that does <u>not</u> contain main

Debugging using println

Using System.out for simple debugging

In a method:

```
System.out.println(this);
```

In an object (for example, a Robot object named **rigby**):

```
System.out.println(rigby);
```

```
[street=1, avenue=4, direction=EAST, isBroken=false, numThingsInBackpack=3]
```

In an object (for example, a Thing object named **t1**):

```
System.out.println(t1);
```

```
[street=1, avenue=4]
```

Packages

- Two java files in the same folder are compiled into the same package, and thus can reference each other.
- Java provides a method for creating packages, which can then be imported elsewhere (like becker.robots.*)
  - Each file must contain the package definition in the first line: `package newPackageName;`
  - The files must reside in a directory tree that matches the name: `package BIT115;` has files that live in the BIT115 folder.
  - You must import the classes that are a part of a package to use them.