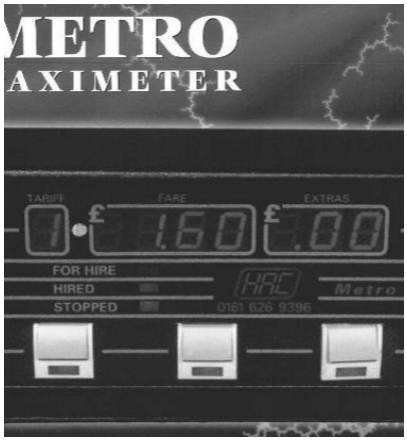
TaxiBot

New attributes
Variables
Math!

TaxiBot



- TaxiBot operates in the city BUT it charges you for its actions
- TaxiBot extends RobotSE
- TaxiBot
  - displays how much is owed

TaxiBot charges

- Moving 1 space $0.50
- Turning left $0.50
- Turning right $1.25
- Turning around $0.75
- Picking something up $1
- Dropping off something $1.25

- Oh yes, just for using a TaxiBot there is a $2.50 charge

Open JCreator

- Create TaxiBot
  - TaxiBot `extends` RobotSE
- Add constructor & main method

```
import becker.robots.*;
public class TaxiBot extends RobotSE {

}
```

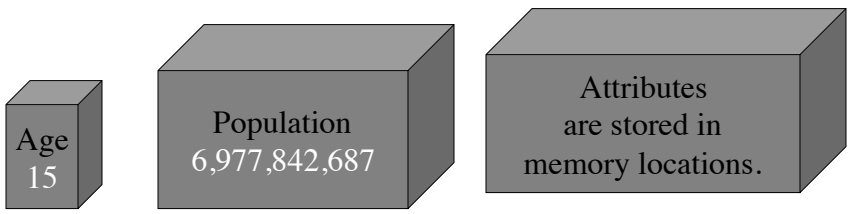
Don't forget this brace!

What are attributes?

How to add them correctly

Attributes in Java

- Attributes contain information about the object
- Attributes must have a **type**
  - This lets Java know how much memory space to set aside to contain the information



Types in Java

- There are many different types of attributes in Java
- The constructor in a Robot has three types in it
  - What are they?
- City, `int`, Direction

Types...

- City, Direction –types created by Becker
- `int` – a type created by Java's developers
  - It is built-in to Java ("intrinsic" or "primitive")
  - It does not allow decimals to be used
- Other built-in types are available

Other types

- `double` – allows numbers that make use of decimals
- `boolean` – true or false
- Etc. There is a chart in textbook...Chapter 7...page 301

Numeric types

Figure 7-3: Integer types and their ranges.

Type	Smallest Value	Largest Value
byte	-128	127
short	-32,768	32,767
int	-2,147,483,648	2,147,483,647
long	-9,223,372,036,854,775,808	9,223,372,036,854,775,807

Figure 7-4: The ranges and precisions of the various floating point types.

	float	double
Largest value	±3.40282347E+38	±1.79769313486231570E+308
Smallest value	±1.40239846E-45	±4.94065645841246544E-324
Precision	About 7 significant digits.	About 16 significant digits.

Adding the attribute

```
public class TaxiBot extends RobotSE {
    // attributes
    // keep track of the charge on the metre
    // keep it private -- other classes can't change it
    // without permission from this class
    private double metre ;
```

- TaxiBot needs to keep track of the charge
- Add an attribute to do this
  - Since it may have decimals, what type will it be?
- Note that we add the word `private` too

What is data hiding?

- Data hiding is a term used to indicate that a class's internal state is hidden from its users.
- It does the job, but we don't know how it does it
- Items that are `private` are hidden from outside view

public vs. private Attributes

- Attributes are often called fields in Java
- The keywords `public` and `private` are referred to as access modifiers.
  - Use `private` keyword to hide field from users
    - Can't be accessed directly without permission by an outside class (e.g. a tester class)
  - Use `public` keyword to allow clients access to field
    - Can be accessed by outside classes

Methods

3 basic types and how to set them up correctly

Methods

- Methods define the behavior of the class.
- Methods can do a job or answer a question

Changing or getting values

- Following the concept of data hiding
  - If appropriate
  - Programmer creates methods that are:
  - Public for others to use
    - Others can use to get some values
    - Others can use to change some values
  - Private for use of this class only
    - No outsider can use

Type 1: Void methods

- Methods that just do a job:
  - Use the keyword `void` if a method does a job and doesn't answer a question
  - StairClimbingRobot method:

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

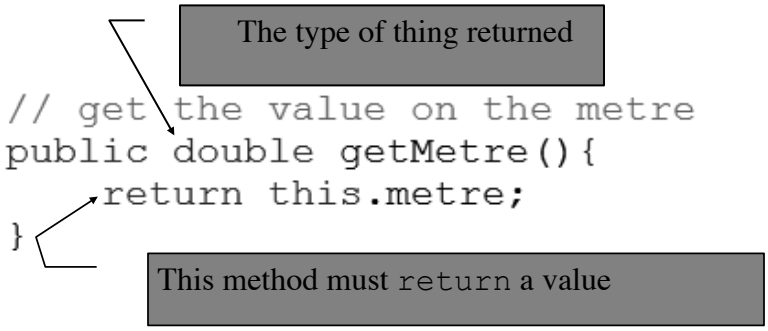
Type 2: Methods returning info

- Methods that answer a question:
  - These methods `return` a value
  - We must know the type of thing returned
  - Return type can be an intrinsic type or an object.

boolean	**canPickThing**() Determine whether this robot is on the same intersection as a thing it can pick up.
int	**countThingsInBackpack**() How many things are in this robot's backpack?

Methods returning info

- Methods that answers the question:
- "What is the value on the meter?"



Also called: Accessor Methods

- Accessor methods return information
  - Typically named to indicate that a value is being returned:
    - getAge()
    - getAccountBalance()

```
// get the value on the metre
public double getMetre(){
    return this.metre;
}
```

Type 3: Passing info to a method (change something)

- Some methods need additional info to work:
  - A parameter is a required item you pass to the method when you call it (use it)

4 parameters for constructor to work

```
IfBot iBot = new IfBot(c, 1,1, Direction.NORTH);
```

1 parameter for setColor to work

```
this.setColor(Color.blue);
```

Passing info to a method

- Methods can accept parameters:
  - A parameter is an item you pass in to the method when you call it (use it)

Parameters for constructor to work

```
// constructor
public StairClimbingRobot ( City theCity, int avenue,
                            int street,
                            Direction direction ) {
    super ( theCity, avenue, street, direction );
}
```

Mutate /Changing values

- Mutator methods modify the state (value) of the object
  - Typically named to indicate that a value is being changed:
    - `setAge()`
    - `addDeposit()`

Mutator method 1: Modifying the metre

- This method modifies the charge
- It needs one parameter to do its job:
  It has to know how much to charge
  += is Java's way to saying add the charge to metre

```
// a mutator method: changes the value of the metre
    private void modifyMetre( double charge ) {
        this.metre += ( charge );
    }
```

Mutator Method 2: Show the meter on the robot

- Robots can say something by using the `setLabel()` method (check the documentation)
- Requires one parameter of type `String`
  - Problem: the metre is of type `double` → convert it!

```
// gets the total charge and displays
// in on the robot
private void showMetre() {
        this.setLabel( Double.toString( this.getMetre()) );
}
```

TaxiBot constructor

- Use the super constructor
- Modify the metre by the initial $2.50 charge
- Have the bot show the metre

```
/ constructor
ublic TaxiBot ( City theCity, int street, int avenue
                        Direction direction ) {
   super ( theCity, street, avenue, direction );
   this.modifyMetre(2.50);
   this.showMetre();
```

TaxiBot charges

- Moving 1 space $0.50
- Turning left $0.50
- Turning right $1.25
- Turning around $0.75
- Picking something up $1
- Dropping off something $1.25

- Oh yes, just for using a TaxiBot there is a $2.50 charge

Overriding move()

```
// taxi move method $0.50 per space!
public void move() {
    super.move();                // as parent knows
    this.modifyMetre ( 0.50 ); // update charge
    this.showMetre();            // show new amount
}
```

- Move as parent does
- Change the metre by 50 cents
- Show the new metre value

TaxiBot charges

- Moving 1 space $0.50
- Turning left $0.50
- Turning right $1.25
- Turning around $0.75
- Picking something up $1
- Dropping off something $1.25

- Oh yes, just for using a TaxiBot there is a $2.50 charge

Refine taxibot

Formatting output

- Better to have 2 decimals for metre
- Need to add 3 changes to TaxiBot
- Use a formatting trick to make the metre appear to have 2 decimals
  - (note that the actual number remains the same)

Step 1

```
import java.text.DecimalFormat;
```

- Import the ability to do the formatting
- (add this at the top)

Step 2: Create an object that formats

```
private double metre ;
private DecimalFormat df = new DecimalFormat("0.00");
```

- In the attributes section
- Declare a DecimalFormat object

  "0.00" means at least one number before decimal point and only 2 numbers after decimal

Step 3: Use the formatting object

```
private void showMetre() {
        //this.setLabel( Double.toString( this.getMetre()));
        this.setLabel(df.format(this.getMetre()));
}
```

- Update `showMetre()` method
- Format converts a number to string and can be used in the `setLabel()` method