Conditional Statements

Friday, September 28, 2007

CS111 Computer Programming

Department of Computer Science
Wellesley College

Spider sense

Actuators change the world:

```
forward();
left();
brushDown();
dropBagel();
```

Sensors detect the world:

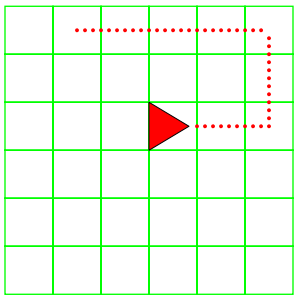
```
isFacingWall();
getColor();
isOverBagel();
```

Control takes input from sensors to do something through actuators

WallHuggerWorld

Straight-line code

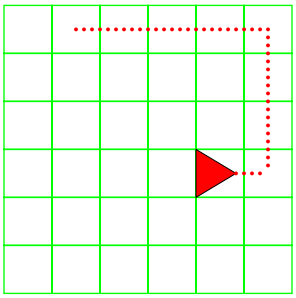
```
forward();
forward();
  left();
forward();
```



Cruelty to Buggles

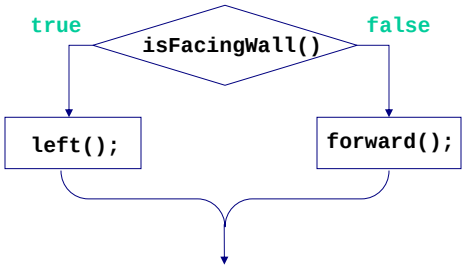
Straight-line code

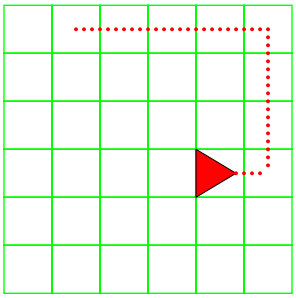
```
forward();
forward();
  left();
forward();
```



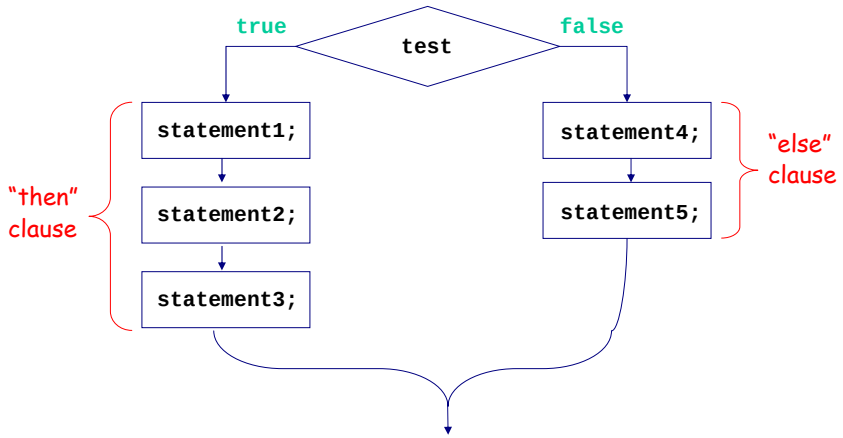
Decisions

Branching code



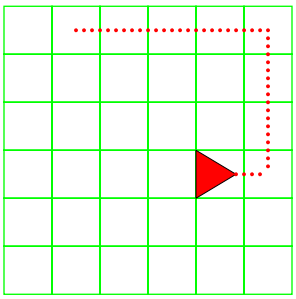


Control diagrams



Conditional statements*

```
class WallHugger extends Buggle {

     // Move forward one step
     // unless facing a wall,
     // in which case turn left.
     public void followWall() {

          if (isFacingWall()) {
               left();
          } else {
               forward();
          }
     }
}
```

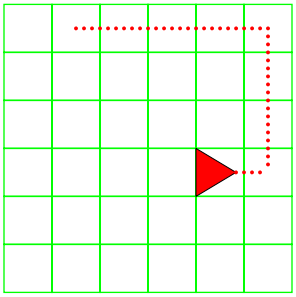


*In general, write:

```
if ( <test expression> ) {
   <statements in "then" clause>
} else {
   <statements in "else" clause>
}
```


Repeating ourselves

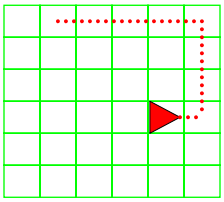
```
class WallHugger extends Buggle {
     public void followWall() {
          if (isFacingWall()) {
               left();
          } else {
               forward();
          }
     }
}
```



```
public class WallHuggerWorld extends BuggleWorld {
     public void run () {
         WallHugger peterParker = new WallHugger();
         peterParker.setPosition(new Location(5,3));
         peterParker.followWall();
         peterParker.followWall();
         …
     }
}
```

Repeating ourselves more succinctly

```
class WallHugger extends Buggle {
     public void followWall() {
          … // same as before
     }

     public void followWall2() {
          followWall();
          followWall();
     }

     public void followWall4() {
          followWall2();
          followWall2();
     }
     public void followWall8() {
          followWall4();
          followWall4();
     }
}
```



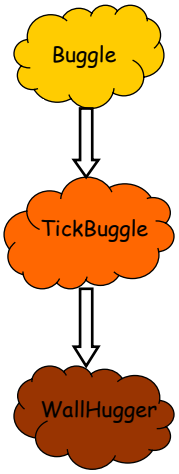
```
public class WallHuggerWorld
                  extends BuggleWorld {

   public void run () {
        WallHugger peterParker =
                  new WallHugger();
        peterParker.setPosition(
                  new Location(5, 3));
        peterParker.followWall8();
   }
}
```


Capturing the pattern

```
// Allows instances of subclasses to
// perform some action on every clock
// tick (a predetermined number of times).
public class TickBuggle extends Buggle {

     public void tick() {
          // Default is to do nothing
     }

     public void tick2() {
          tick();
          tick();
     }

     public void tick4() {
          tick2();
          tick2();
     }
     ...

     public void tick1024() {
          tick512();
          tick512();
     }

} // class TickBuggle
```

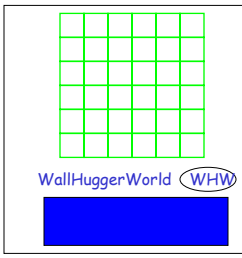
Using the pattern

```
class WallHugger extends TickBuggle {

     public void followWall() {
          …
     }

     public void tick() {
          // insert code here that we
          // want executed repeatedly
          followWall();
     }
}

public class WallHuggerWorld extends BuggleWorld {
     public void run() {
          WallHugger peterParker = new WallHugger();
          peterParker.setPosition(new Location(5,3));
          peterParker.tick128();
     }
}
```

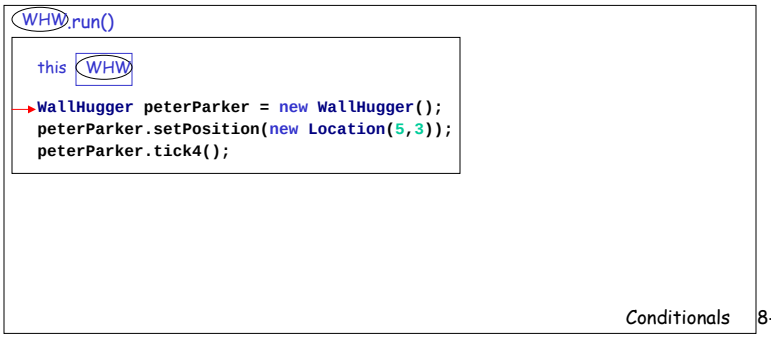


JEM traces Spiderman

Object Land

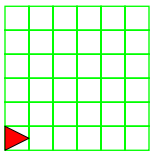


Execution Land



A spider bites Peter

Object Land



WallHuggerWorld WHW

WallHugger WH1

position	(1,1)
heading	EAST
color	red
brushDown	true

Execution Land

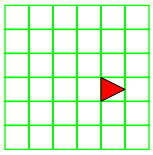
WHW.run()

this WHW peterParker WH1

```
WallHugger peterParker =  WH1
peterParker.setPosition(new Location(5,3));
peterParker.tick4();
```


Peter jumps

Object Land



WallHuggerWorld WHW

WallHugger WH1

position	(5,3)
heading	EAST
color	red
brushDown	true

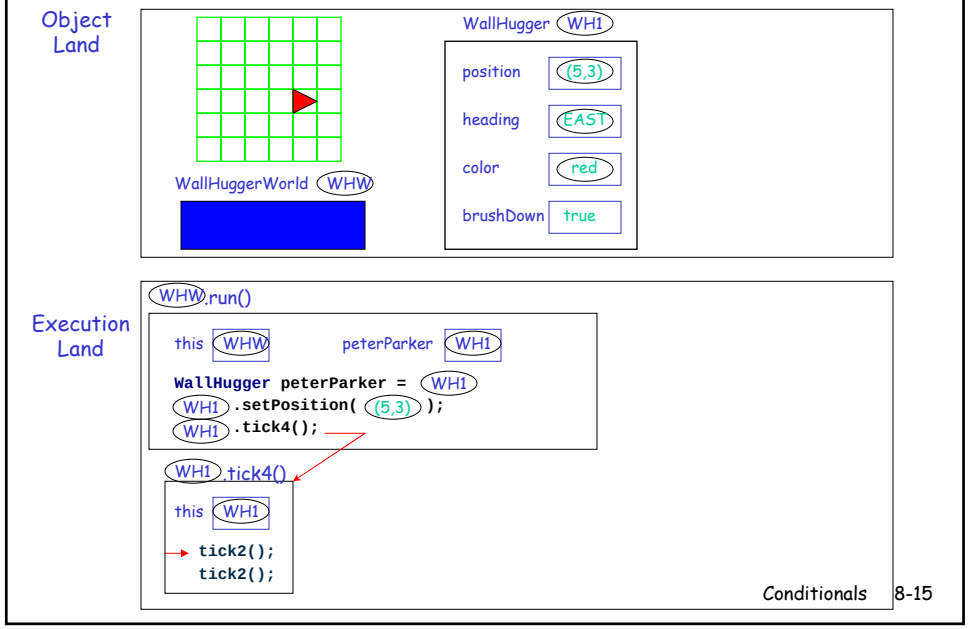
Execution Land

WHW.run()

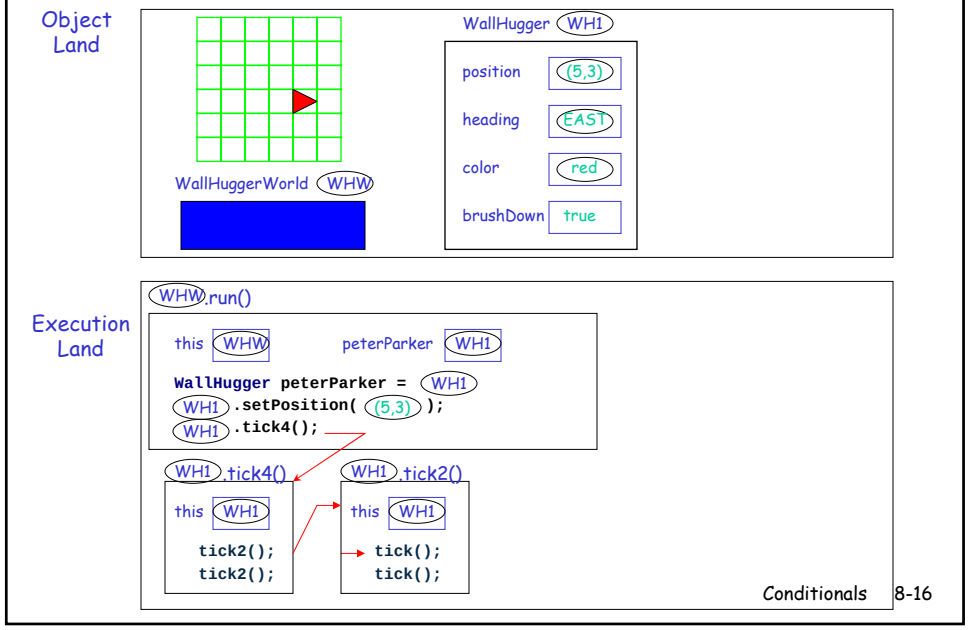
this WHW peterParker WH1

```
WallHugger peterParker =  WH1
WH1 .setPosition( (5,3) );
peterParker.tick4();
```

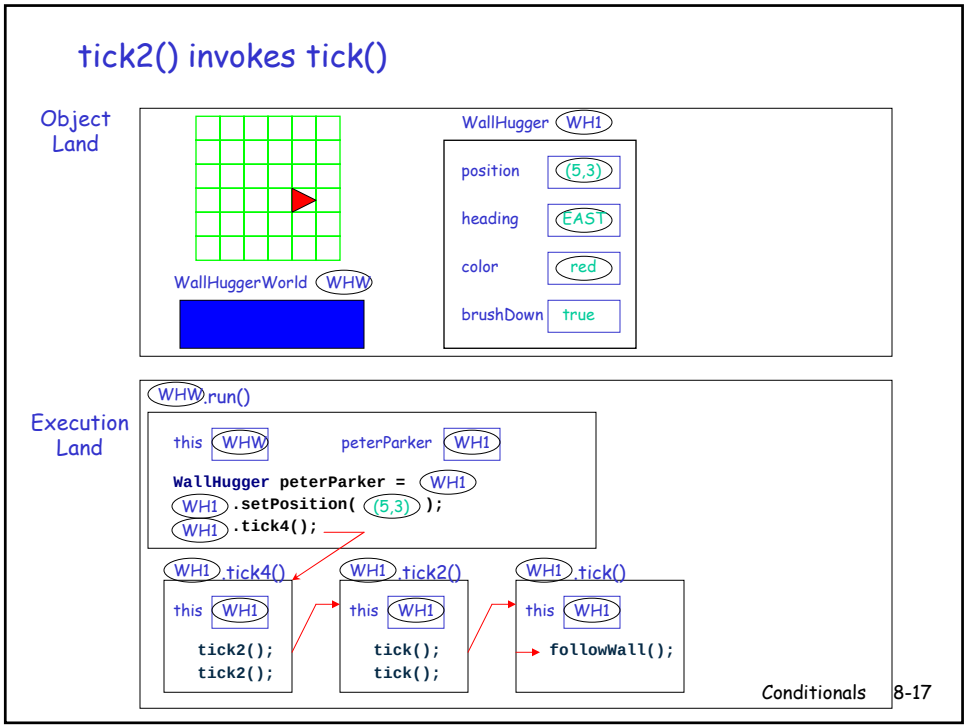
And the clock starts ticking...



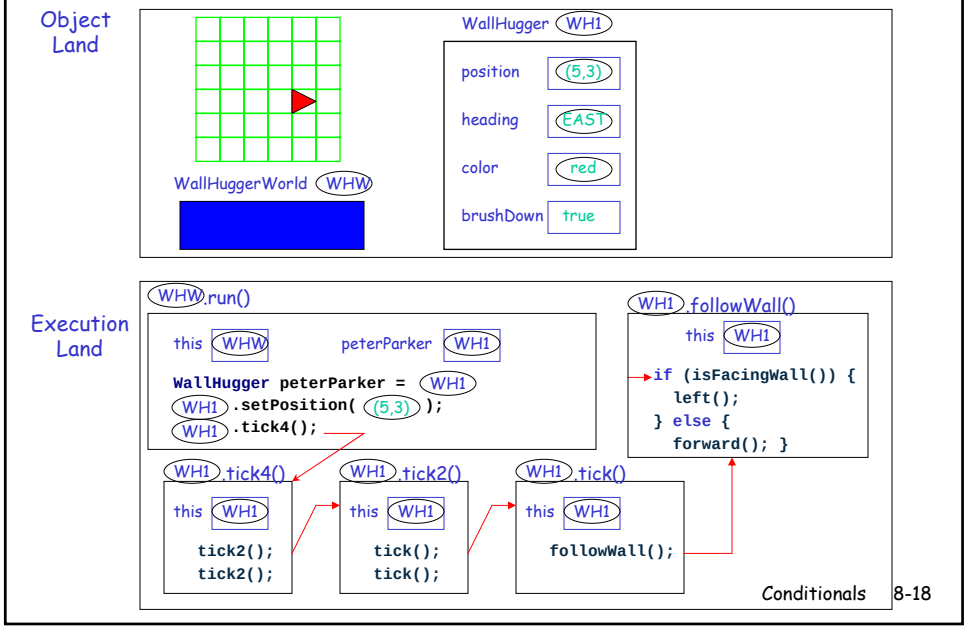
tick4() invokes tick2()



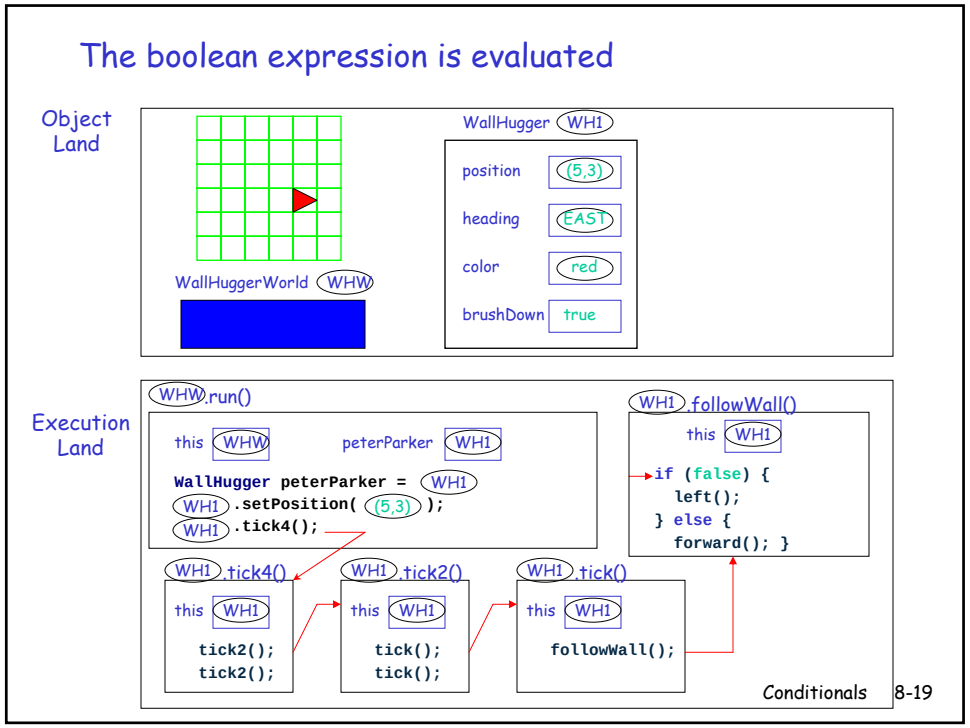
tick2() invokes tick()



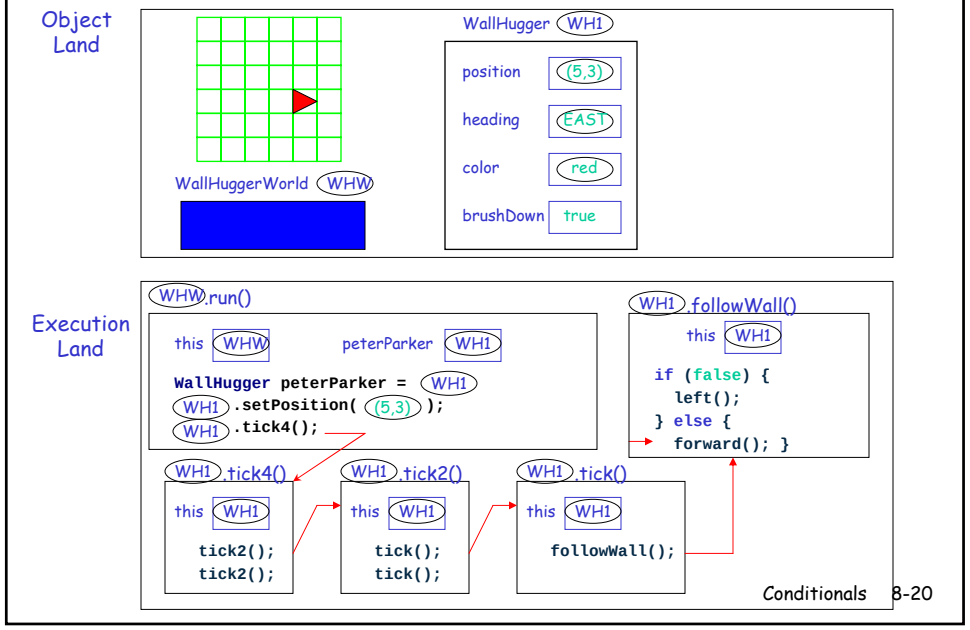
tick() invokes followWall()



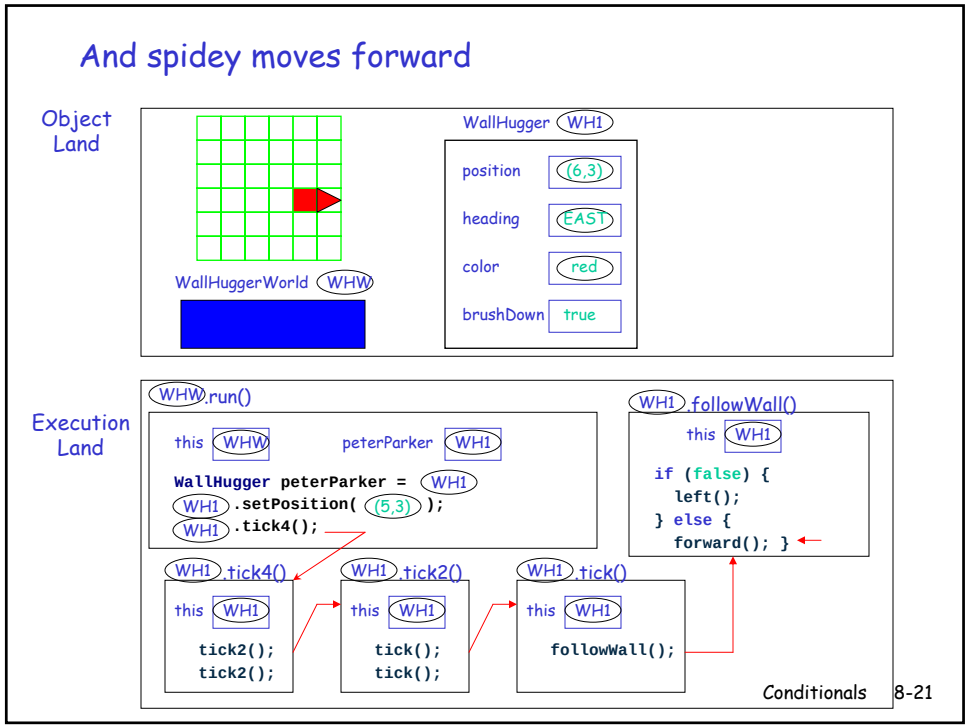
The boolean expression is evaluated



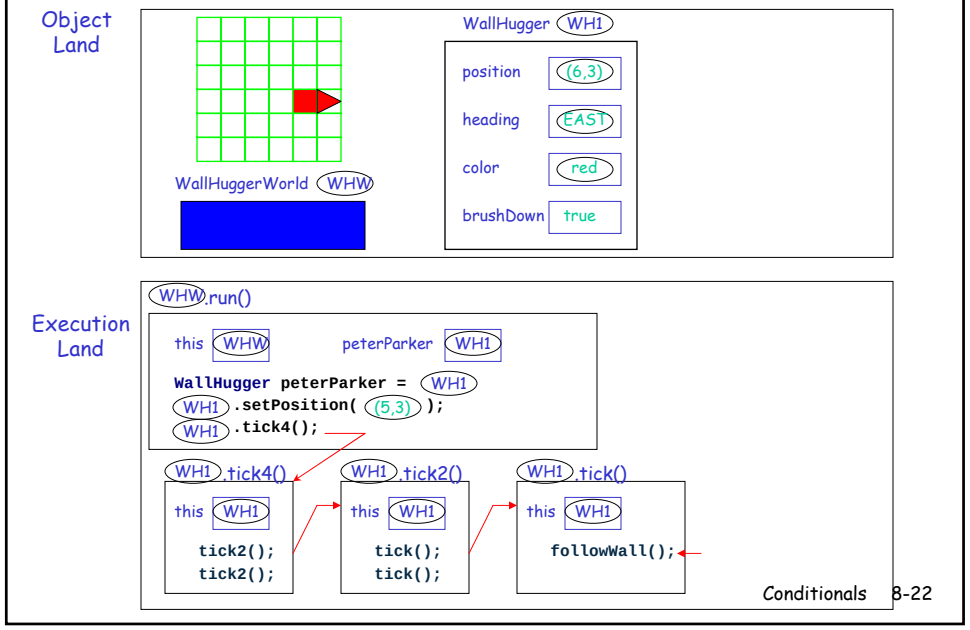
The "else" clause is executed



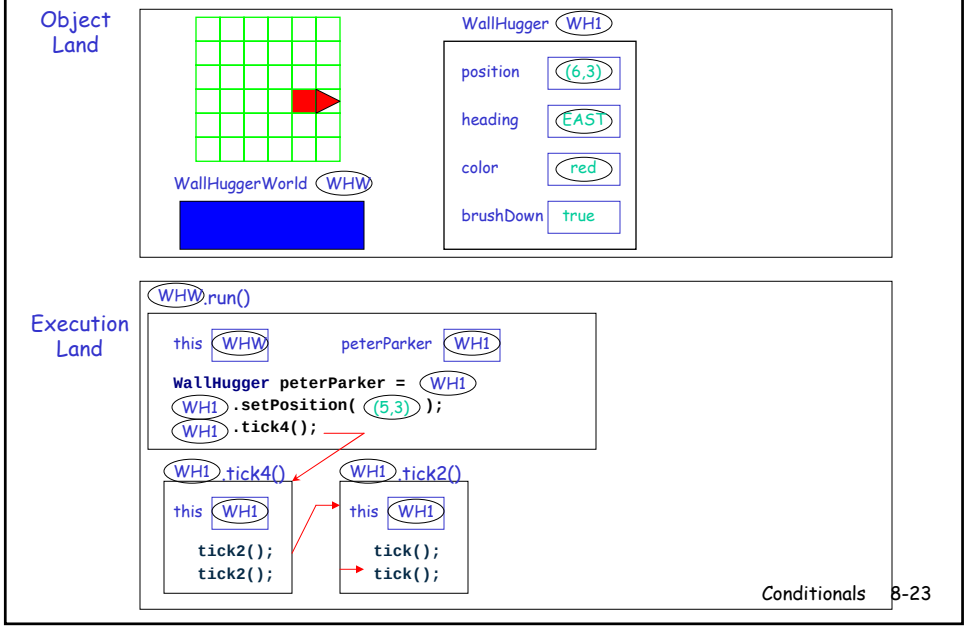
And spidey moves forward



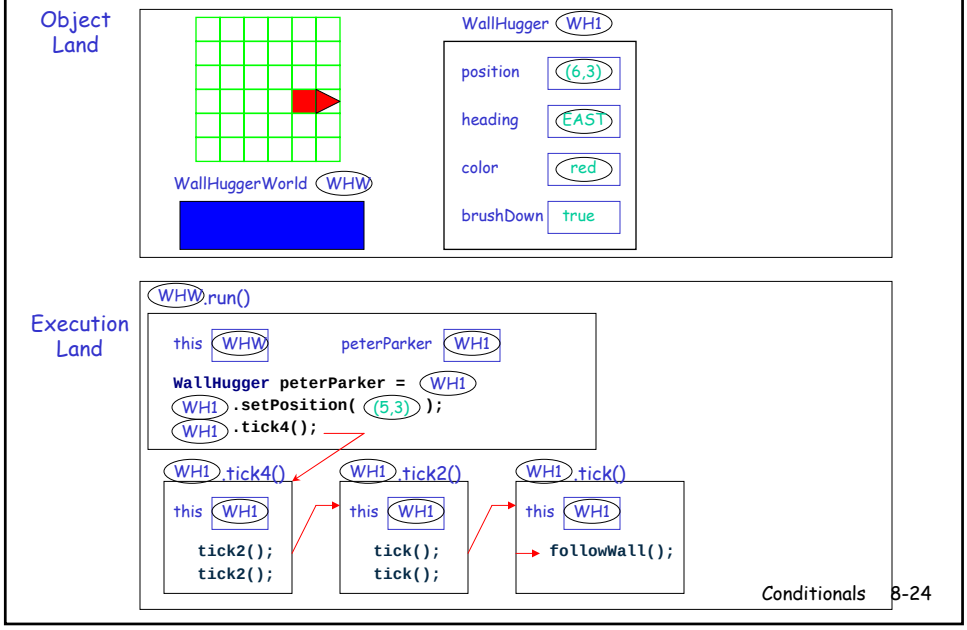
followWall() goes away



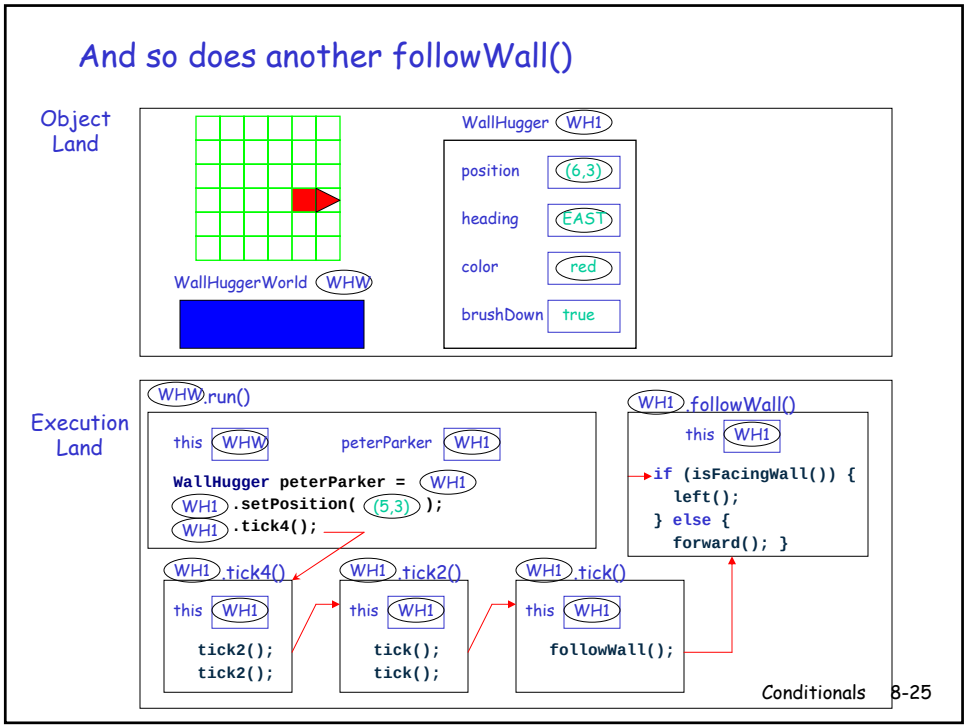
And so does tick()



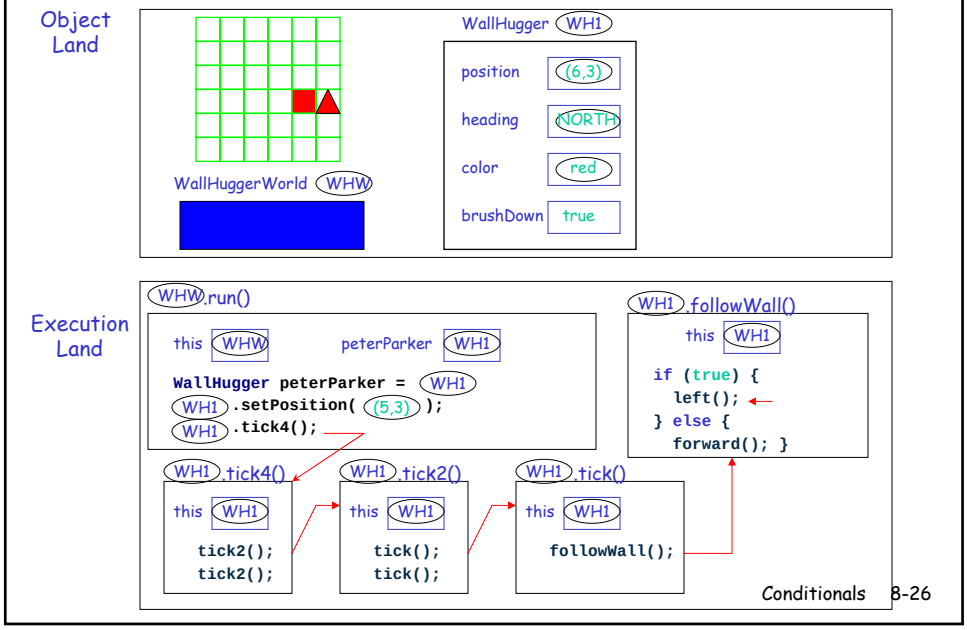
But another tick() takes its place



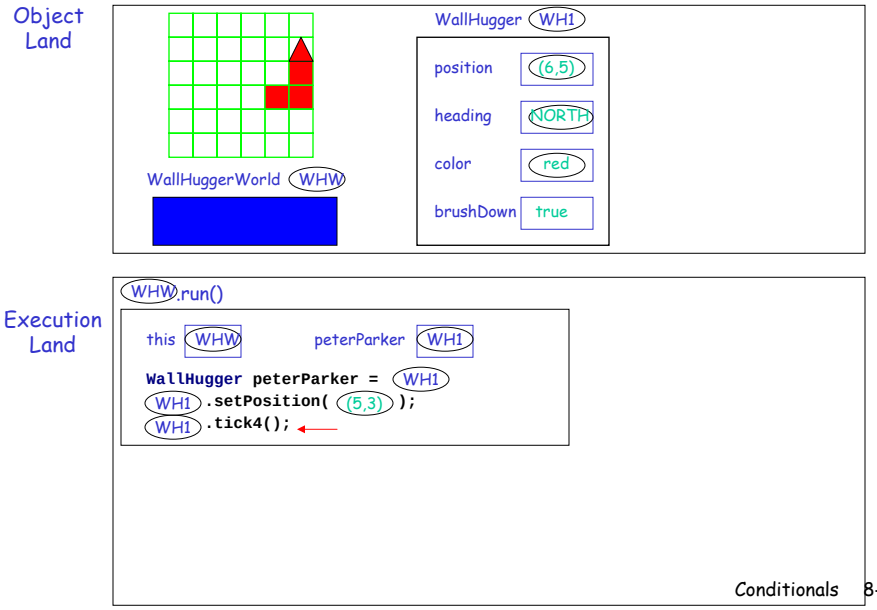
And so does another followWall()



The "if" clause is executed

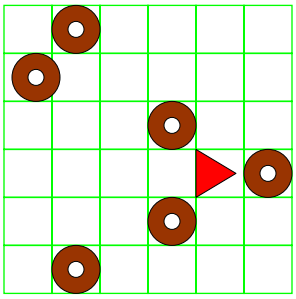


Until eventually, tick4() finishes execution

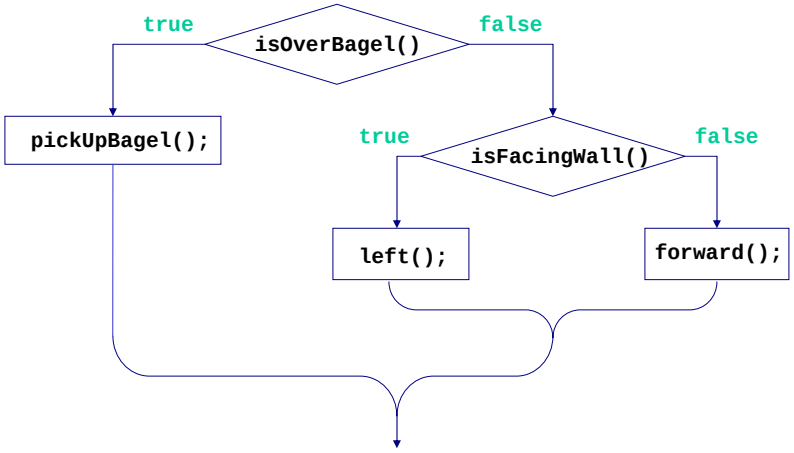


Bagels, bagels everywhere

```
public class HungryWallHuggerWorld
                          extends RandomBagelWorld {

     public void run () {
          HungryWallHugger peterParker =
                              new HungryWallHugger();
          peterParker.setPosition(new Location(5,3));
          peterParker.tick128();
     }
}
```

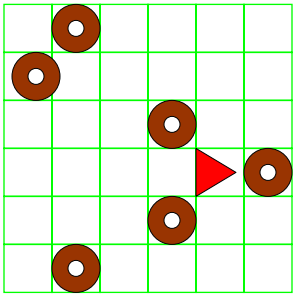


HungryWallHugger



HungryWallHugger pigs out*

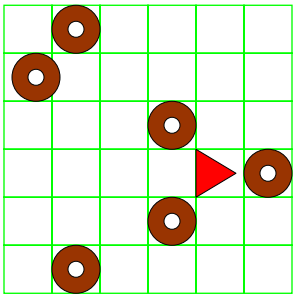
```
class HungryWallHugger extends TickBuggle {
     // If over a bagel, eat it.
     // Otherwise, turn left if
     // facing a wall, or move
     // forward if not facing a wall.
     public void followWall() {

          if (isOverBagel()) {
               pickupBagel();
          } else {
               if (isFacingWall()) {
                    left();
               } else {
                    forward();
               }
          }
     }
}
```



*Using nested conditionals.

Put another way*

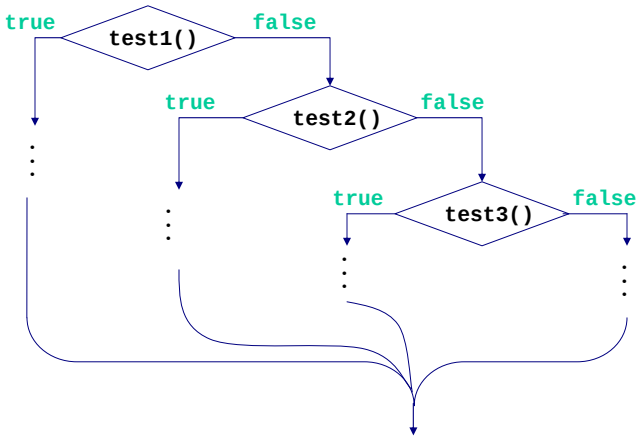
```
class HungryWallHugger extends TickBuggle {
     // If over a bagel, eat it.
     // Otherwise, turn left if
     // facing a wall, or move
     // forward if not facing a wall.
     public void followWall() {

          if (isOverBagel()) {
               pickupBagel();
          } else if (isFacingWall()) {
               left();
          } else {
               forward();
          }
     }
}
```



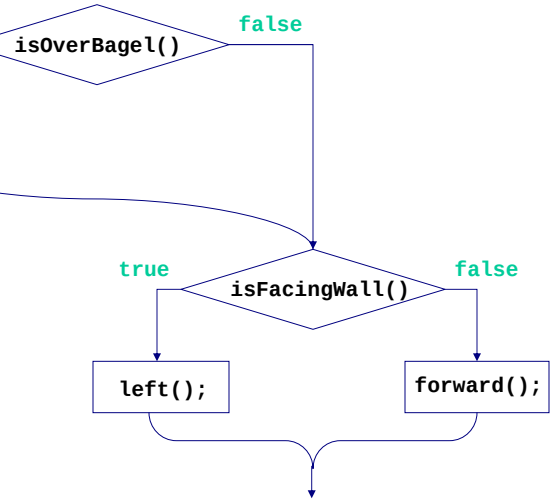
*Multi-way conditionals.

Multi-way conditional idiom

```
if (test1)) {
  …
} else if (test2) {
  …
} else if (test3) {
  …
} else {
  …
}
```

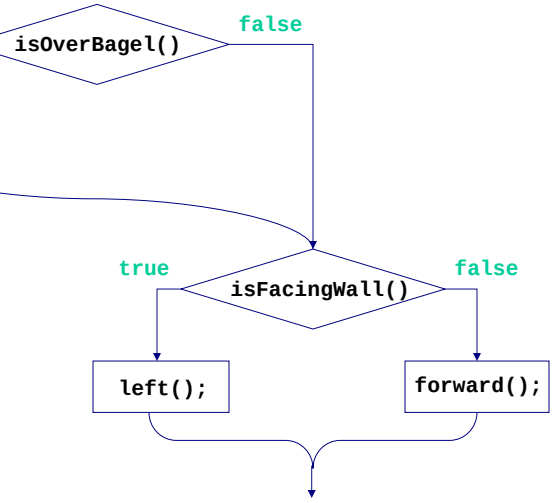


Eat and run




```
// If over a bagel, eat it.
// Also, turn left if facing
// a wall, or move forward
// if not facing a wall.
public void eatAndRun() {
     if (isOverbagel()) {
          pickupBagel();
     } else {
     } // do nothing
     if (isFacingWall()) {
          left();
     } else {
          forward();
     }
}
```


It's okay to leave off the "else"




```
// If over a bagel, eat it.
// Also, turn left if facing
// a wall, or move forward
// if not facing a wall.
public void eatAndRun() {

     if (isOverbagel()) {
          pickupBagel();
     }
     if (isFacingWall()) {
          left();
     } else {
          forward();
     }
}
```

General form of the **if** statement

The **if** statement has the form:

```
if (<test expression>) {
     <statements>
} else {
     <statements>
}
```

Three cases

```
if (<test expression>) {
     <statements>
} else {
     <statements>
}

if (<test expression>) {
     <statements>
}

if (<test expression>) {
     <statement>
} else if (<test expression>) {
     <statements>
}
```