Spiderman

Conditional statements

CS111 Computer Programming

Department of Computer Science
Wellesley College

Spider sense

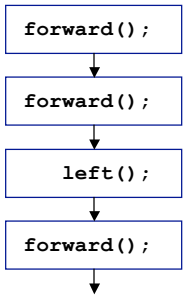
Actuators change the world:
forward(); left();
brushDown();
dropBagel();

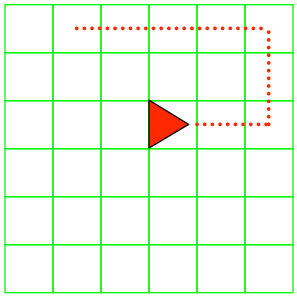
Sensors detect the world:
isFacingWall();
getColor(); isOverBagel();

Control takes input from sensors to do something through actuators

WallHuggerWorld

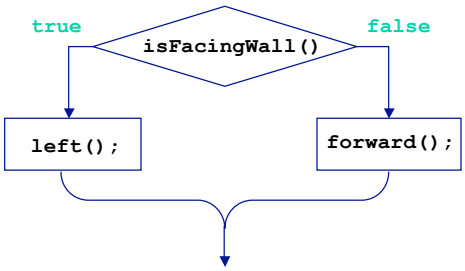
Straight-line code

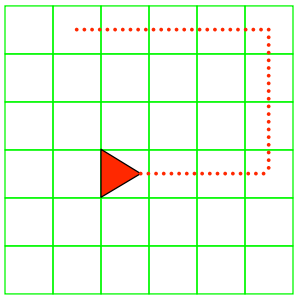




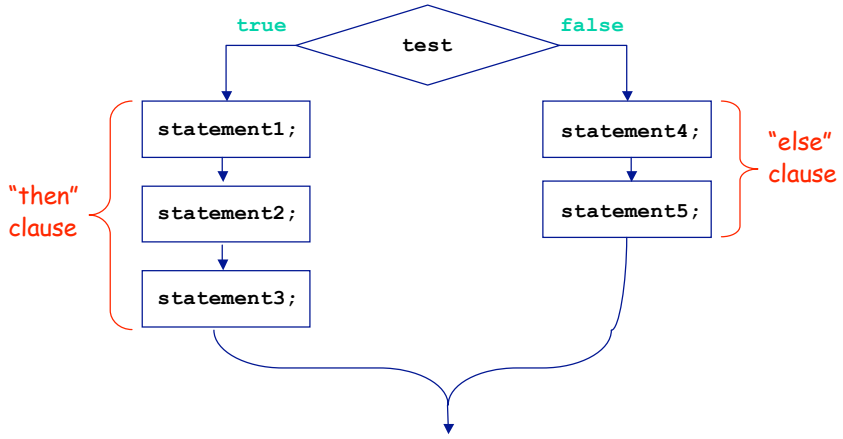
Decisions

Branching code



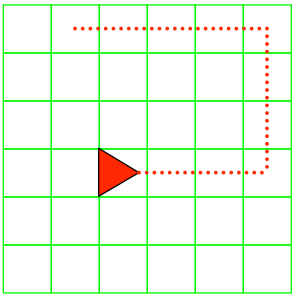


Control diagrams



Conditional statements*

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

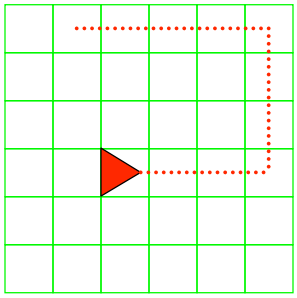


*In general, write:

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

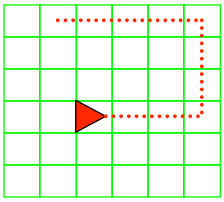
Repeating ourselves

```
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 Point(3,3));
          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();
     }
}
```



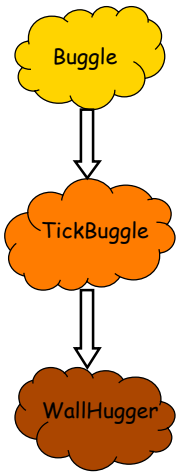
```
public class WallHuggerWorld
   extends BuggleWorld
{
     public void run ()
     {
          WallHugger peterParker =
               new WallHugger();
          peterParker.setPosition(
               new Point(3,3));
          peterParker.followWall8();
     }
}
```

Capturing the pattern

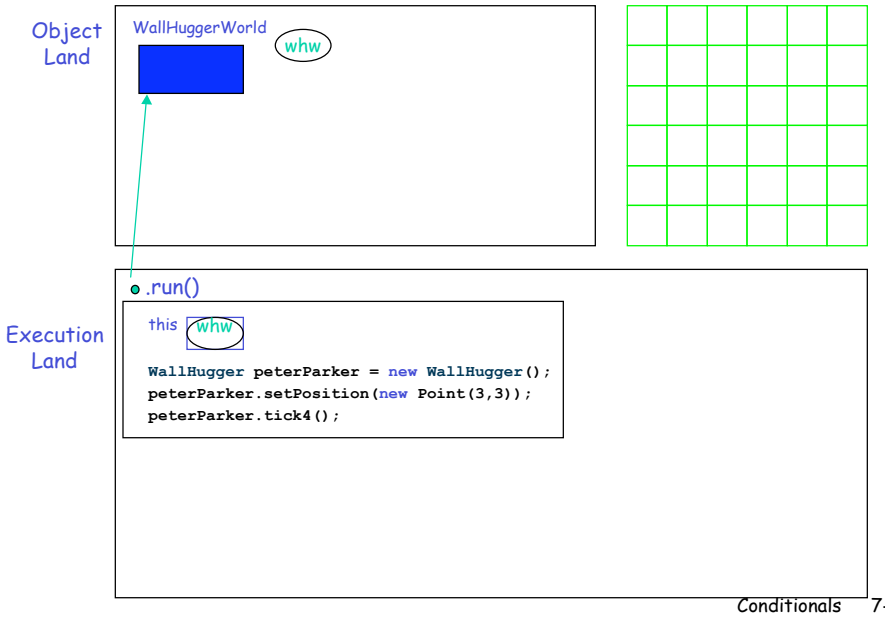
```
// Allows instances of subclasses to
// perform some action on every clock
// tick (a predetermined number of times).
public class TickBuggle extends Buggle
{
     public void tick()
     {
          // Default is to do nothing
     }

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

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

     public void tick1024()
     {
          tick512();
          tick512();
     }
} // class TickBuggle
```


Using the pattern

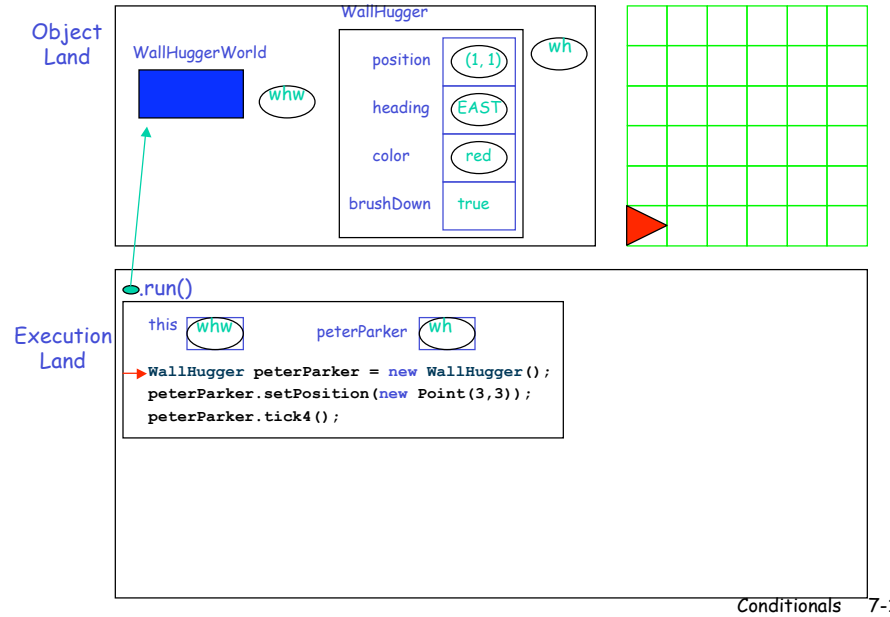
```
class WallHugger extends TickBuggle
{
     public void followWall()
     {
          …
     }

     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 Point(3,3));
          peterParker.tick128();
     }
}
```



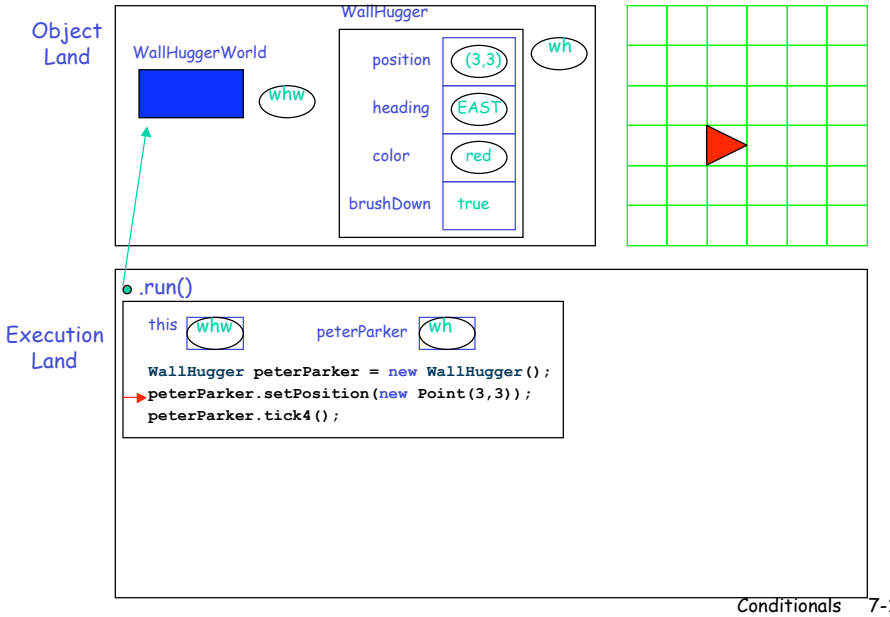
JEM traces Spiderman



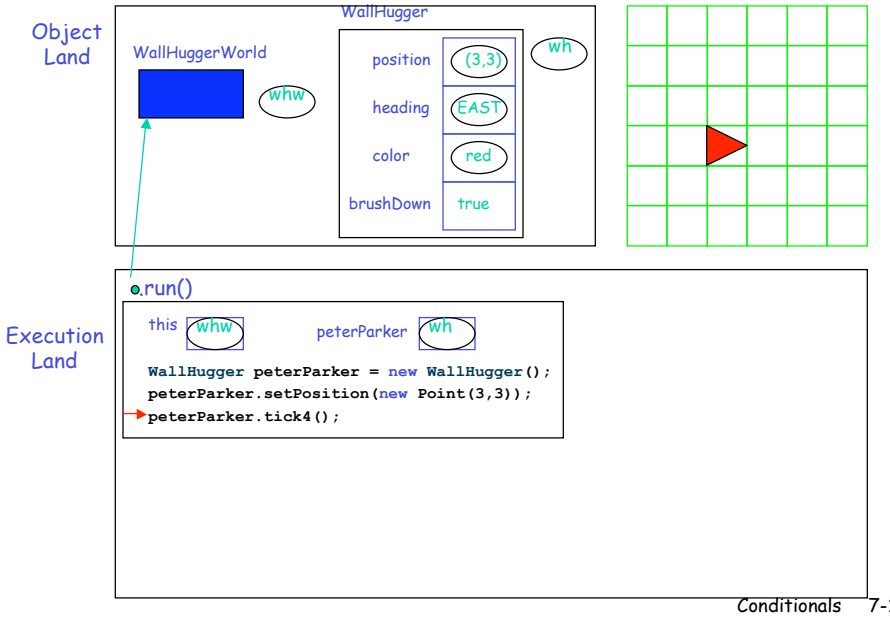
A spider bits Peter



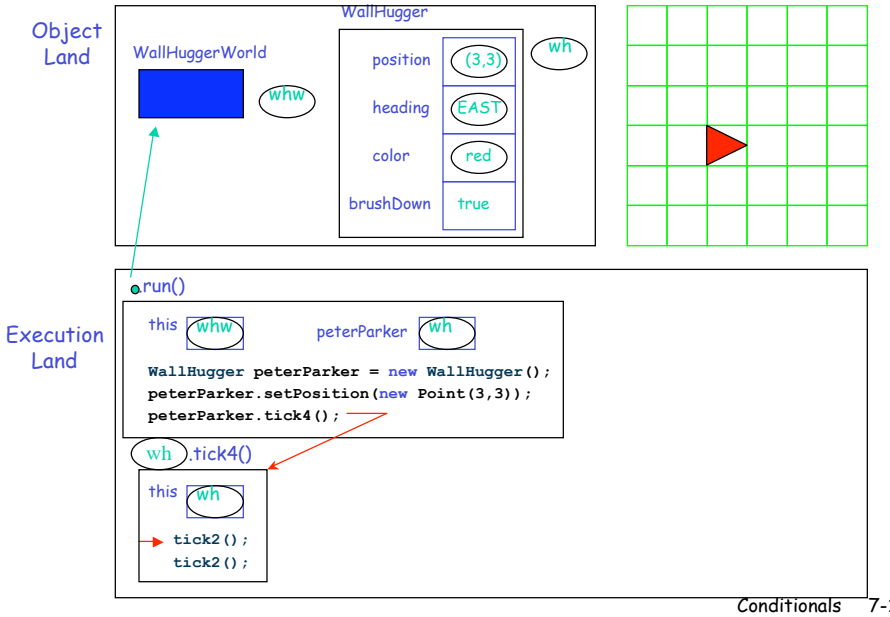
Peter jumps



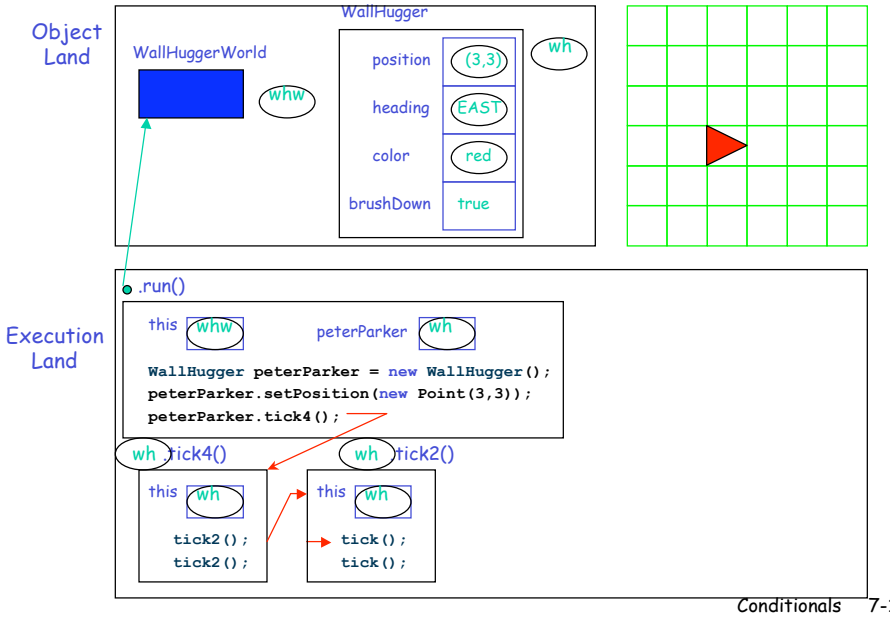
And the clock starts ticking



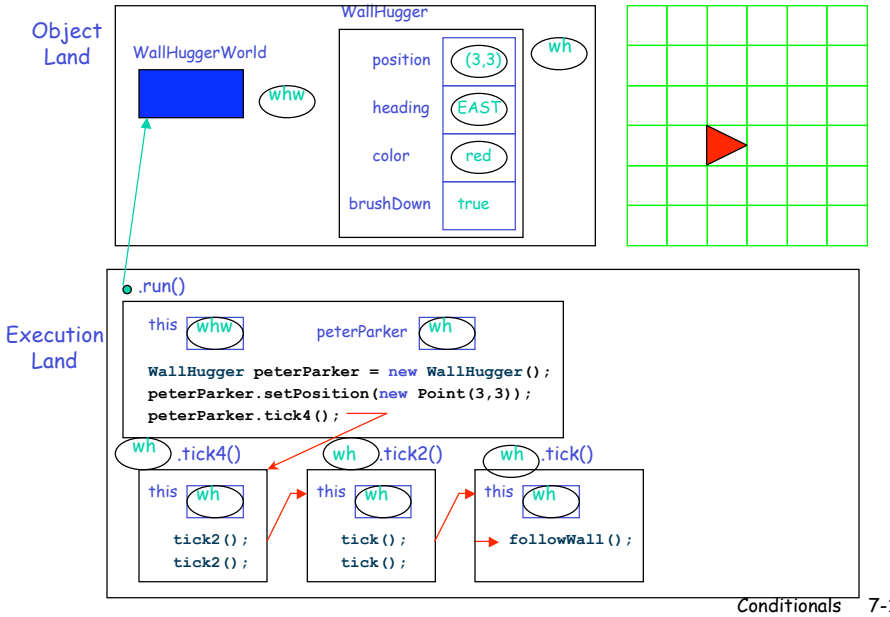
tick4() is invoked



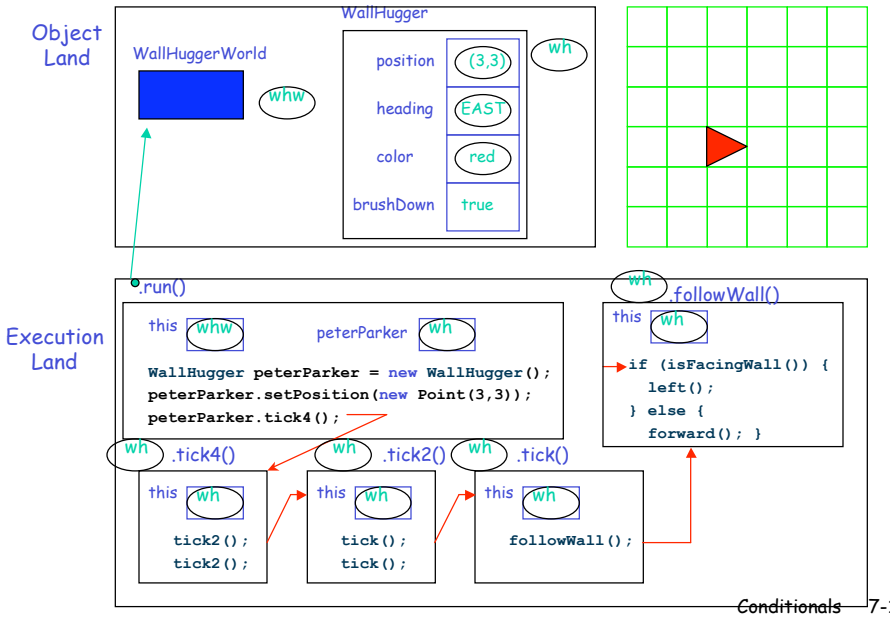
tick4() invokes tick2()



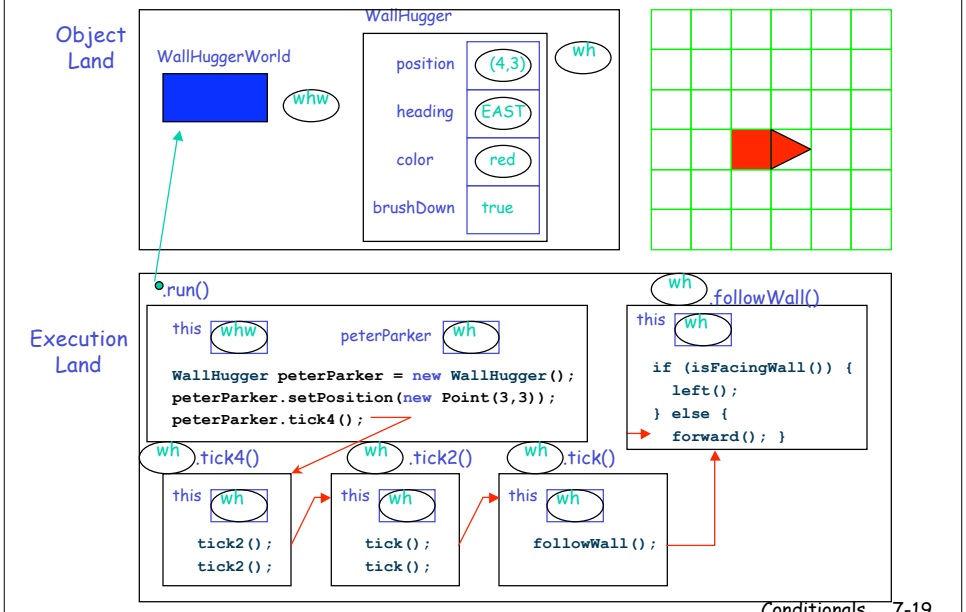
tick2() invokes tick()



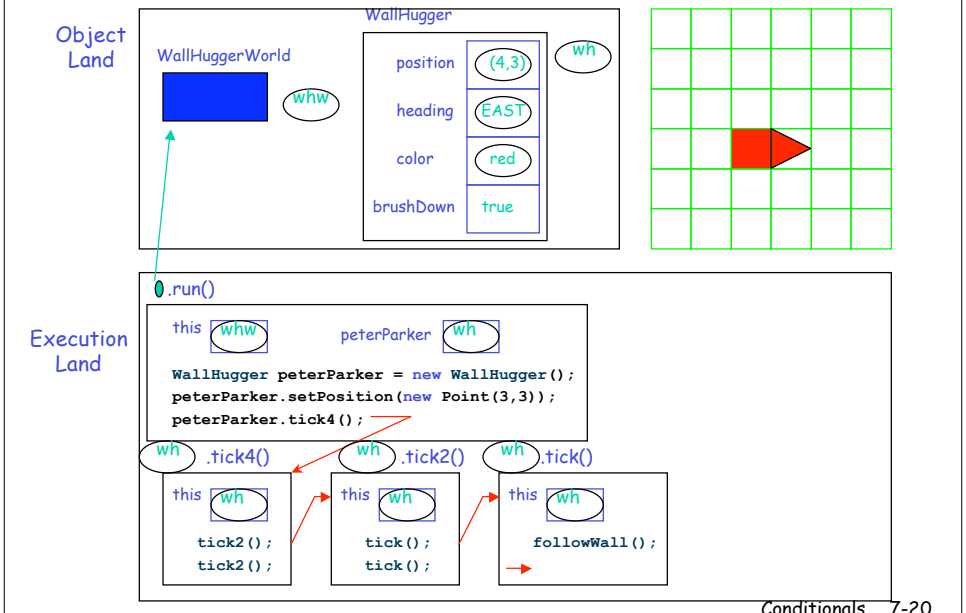
tick() invokes followWall()



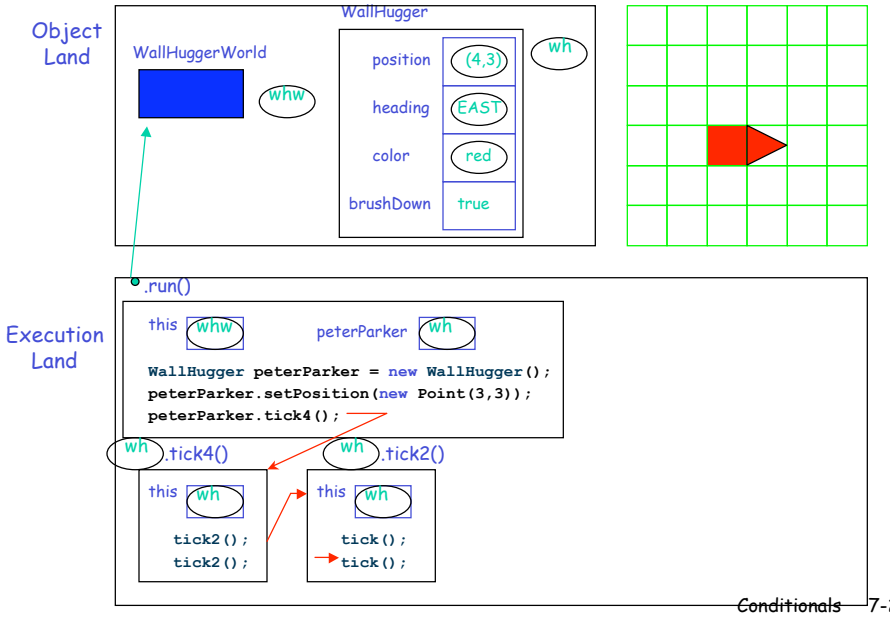
Finally, . . .



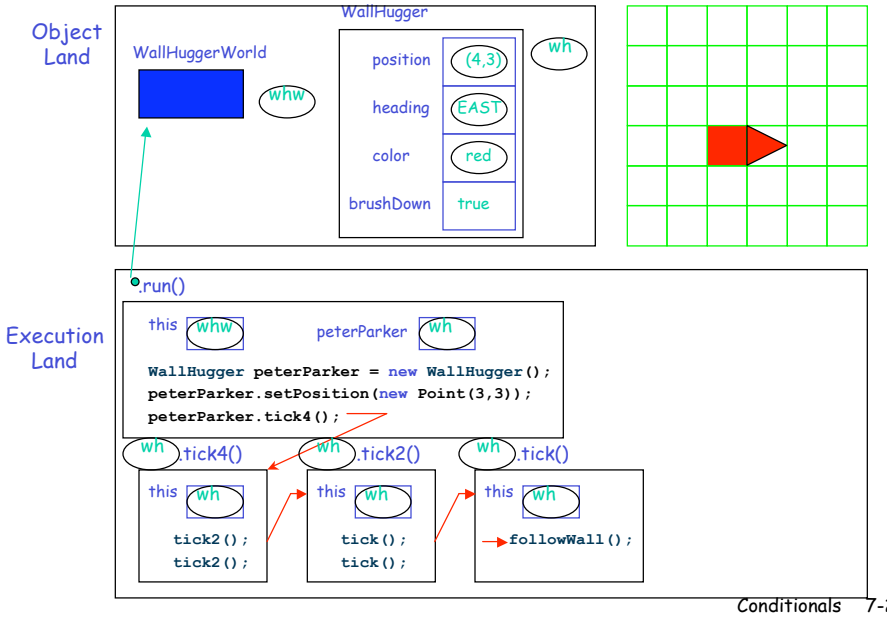
followWall() goes away



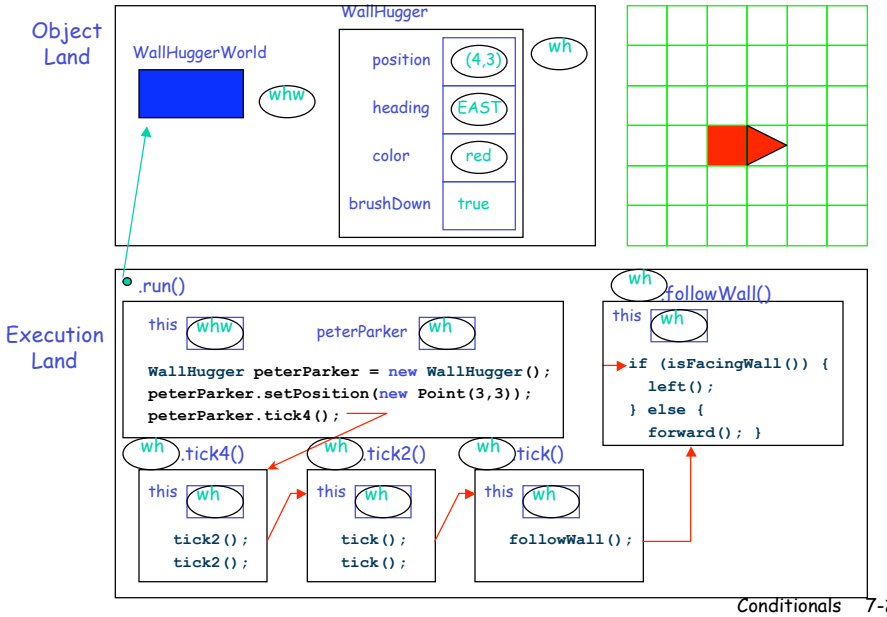
And so does tick



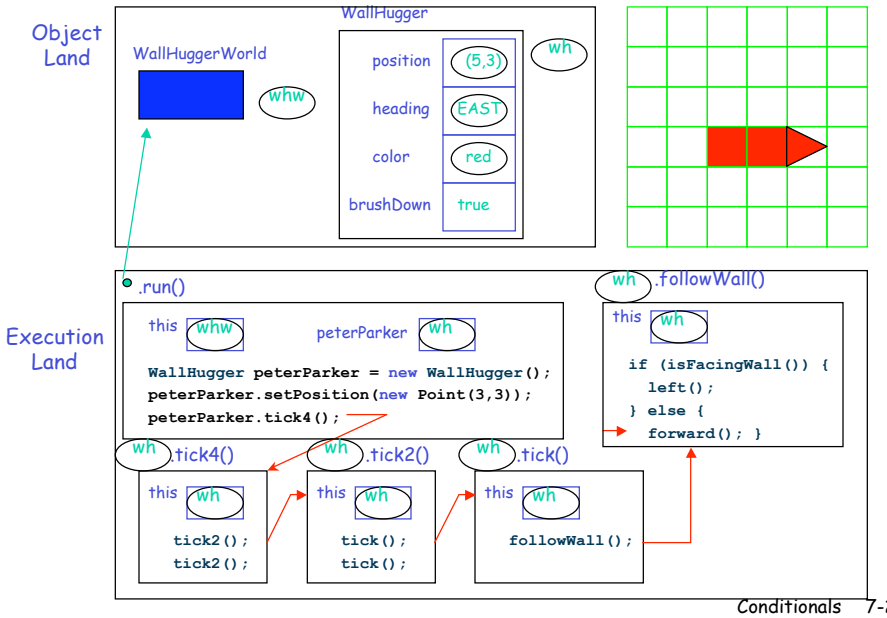
But another tick takes its place



And so does another followWall()

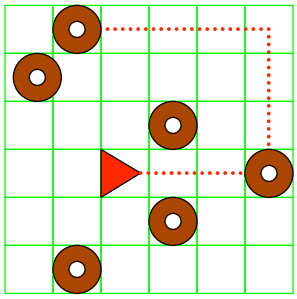


The beat goes on . . .

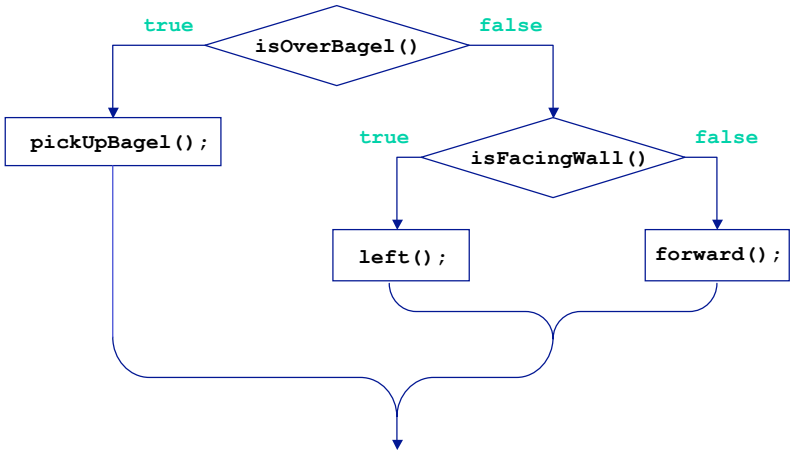


Bagels, bagels everywhere

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

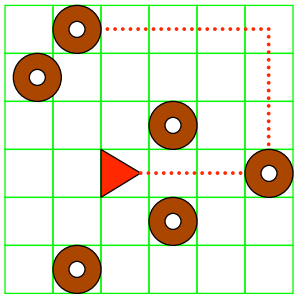


Hungry WallHugger



WallHugger pigs out*

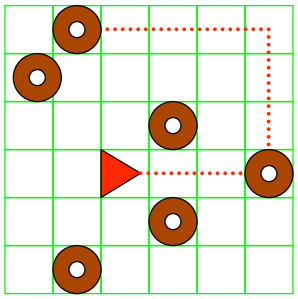
```
class WallHugger extends TickBuggle
{
     public void followWall()
     {
          if (isOverBagel()) {
               pickupBagel();
          } else {
               if (isFacingWall()) {
                    left();
               } else {
                    forward();
               }
    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



*Using nested conditionals.

Put another way*

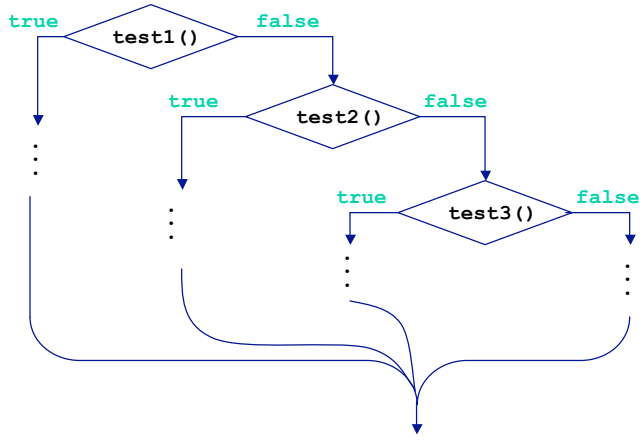
```
class WallHugger extends TickBuggle
{
     public void followWall()
     {
          if (isOverBagel()) {
               pickupBagel();
          } else if (isFacingWall()) {
               left();
          } else {
               forward();
          }
     }
}
```



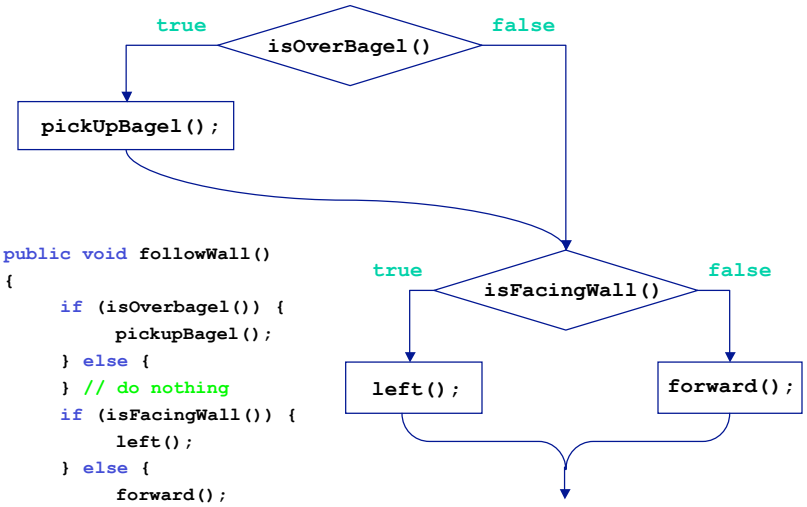
*Multi-way conditionals.

Multi-way conditional idiom

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

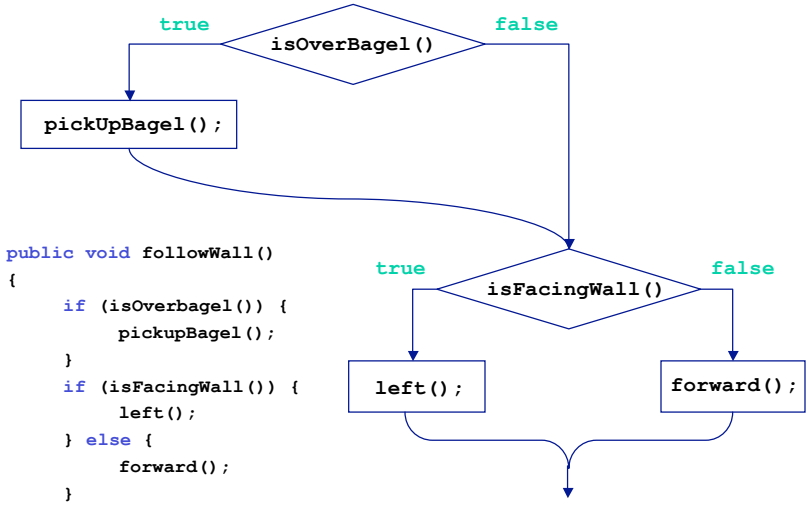


Eat and run




```
public void followWall()
{
     if (isOverbagel()) {
          pickupBagel();
     } else {
     } // do nothing
     if (isFacingWall()) {
          left();
     } else {
          forward();
     }
}
```

It's okay to leave off the "else"




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


Real general form of the **if** statement

The **if** statement has the form:

if (*<expression>***)**
  <statement>
*[***else***
  <statement>]

Three cases

if (*<expression>***)**
  <statement>
else
  <statement>

if (*<expression>***)**
  <statement>

if (*<expression>***)**
  <statement>
else if (*<expression>***)**
  <statement>
  ...