

There and back again

Bugle recursion

Friday, October 5, 2007



CS111 Computer Programming

Department of Computer Science
Wellesley College

Recursion

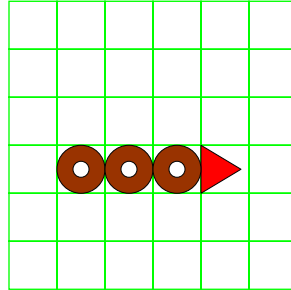
- Recursion solves a problem by solving a smaller instance of the same problem.
- Think **divide, conquer, and glue** when all the subproblems have the same "shape" as the original problem.
- A **recursive method** is a method that invokes itself



Recursion 10-2

bagelForward(3)

Write a method that teaches a buggle to leave behind a trail of bagels of a given length. Assume brushUp().



Recursion 10-3

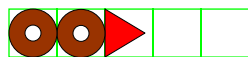
We could solve ...



`bagelForward(3);` by
`dropBagel();`
`forward();` and solving ...



`bagelForward(2);` by
`dropBagel();`
`forward();` and solving ...



`bagelForward(1);` by
`dropBagel();`
`forward();` and solving ...



`bagelForward(0);` by
Doing nothing at all

Recursion 10-4

bagelForward(int n)

```
public void bagelForward(int n) {  
  
    if (n == 0) {                // base case  
        // do nothing  
    } else {                      // recursive case  
        dropBagel();  
        forward();  
        // Now what?  
    }  
}
```

Recursion 10-5

Putting a wrapper around it

```
public class BagelWorld extends BuggleWorld {  
    public void run() {  
        BagelBuggle betty = new BagelBuggle();  
        betty.brushUp();  
        betty.bagelForward(3);  
    }  
}  
class BagelBuggle extends Buggle {  
    public void bagelForward(int n) {  
        if (n == 0) {  
            // do nothing  
        } else {  
            dropBagel();  
            forward();  
            bagelForward(n-1);  
        }  
    }  
}
```



Recursion 10-6

JEM demonstrating bagelForward method

Object Land

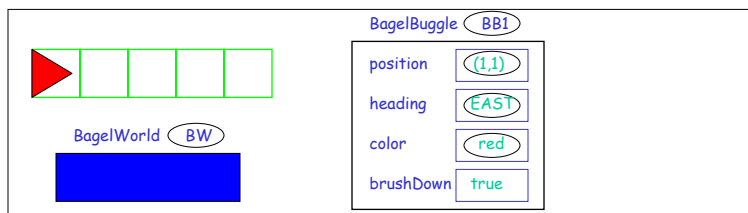


Execution Land

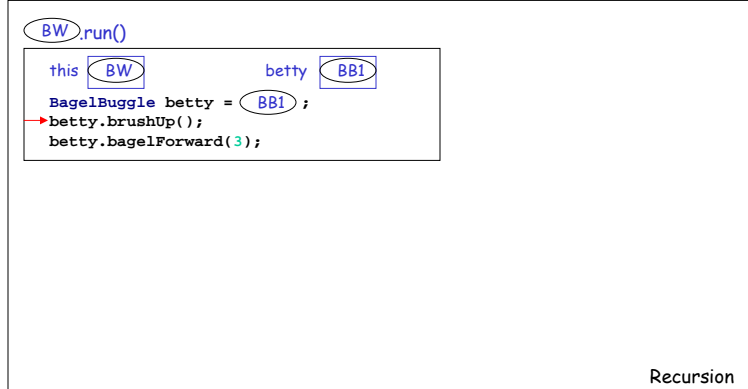


A buggle named betty is constructed

Object Land

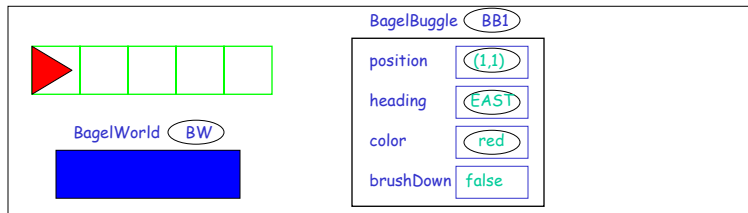


Execution Land

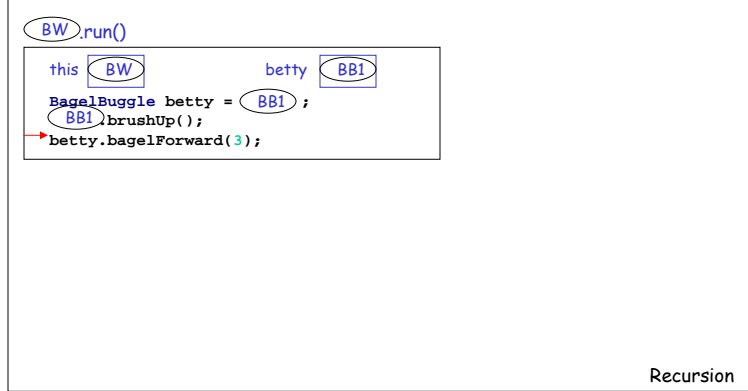


betty raises her brush

Object Land



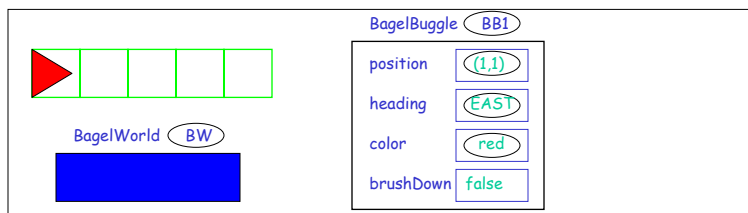
Execution Land



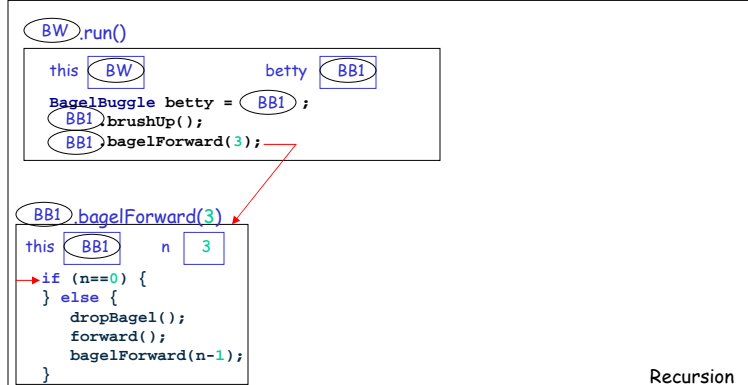
Recursion 10-9

The message bagelForward(3) is sent to betty

Object Land

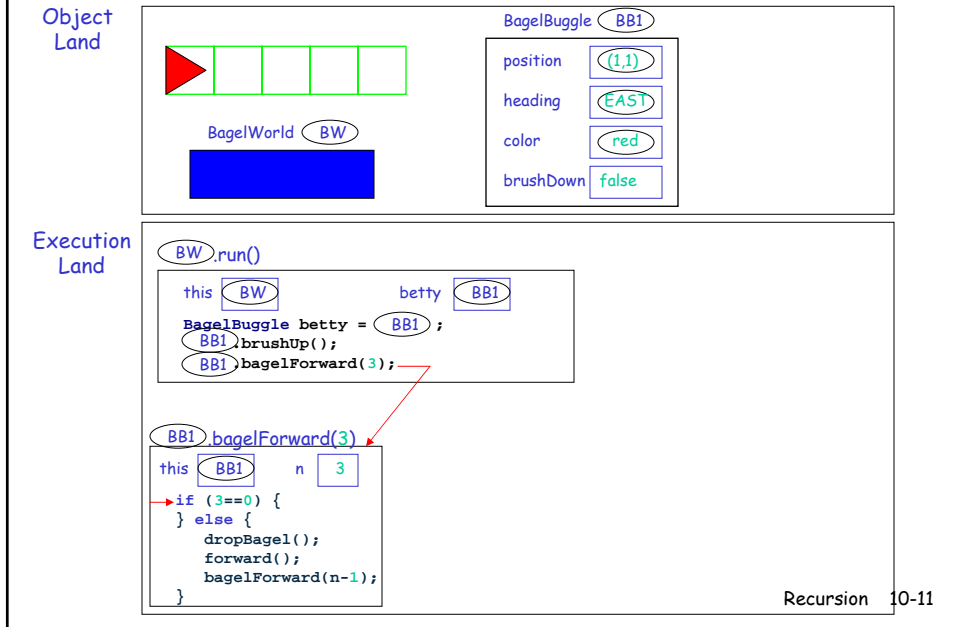


Execution Land

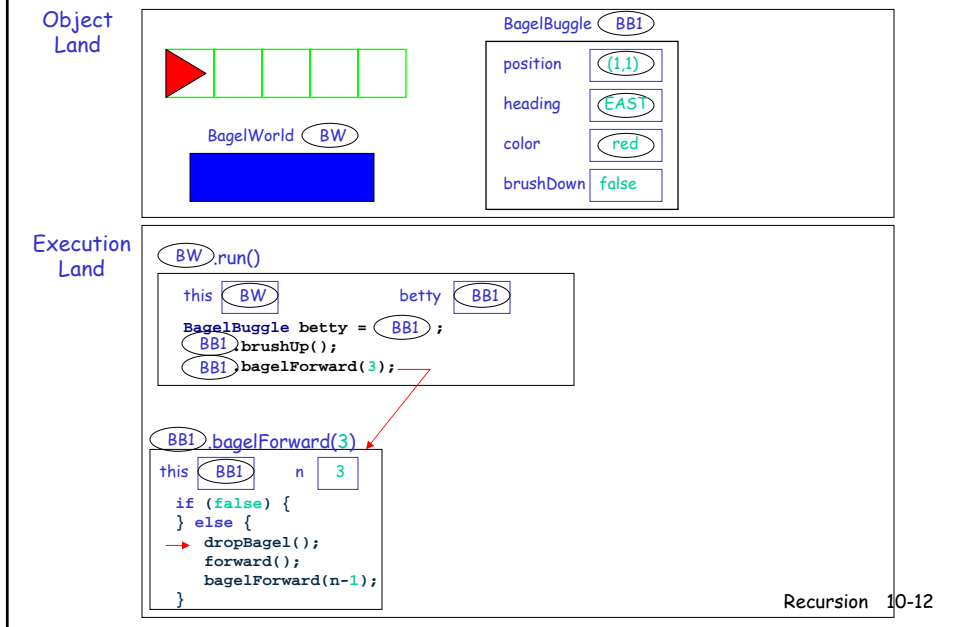


Recursion 10-10

The variable `n` is evaluated

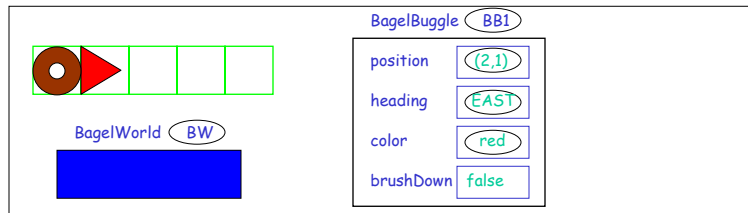


The boolean expression is evaluated: execute the `else` clause

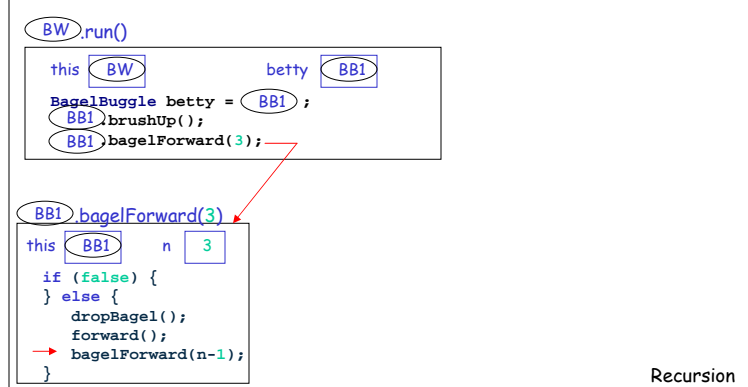


Drop a bagel and move forward

Object Land



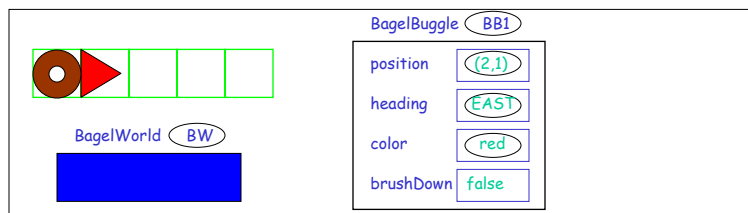
Execution Land



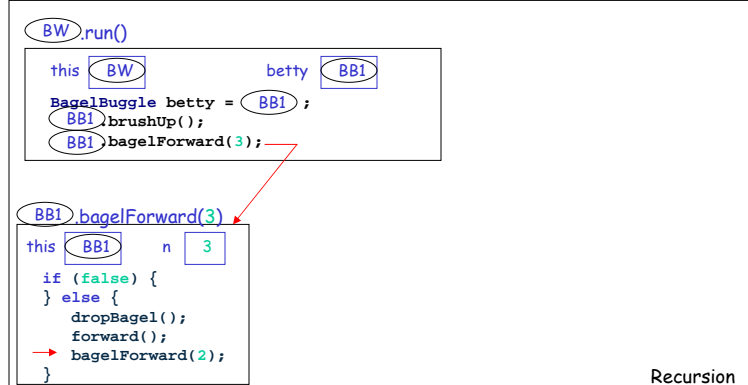
Recursion 10-13

Evaluate the method's argument

Object Land



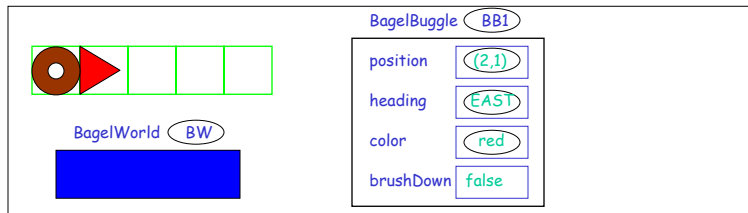
Execution Land



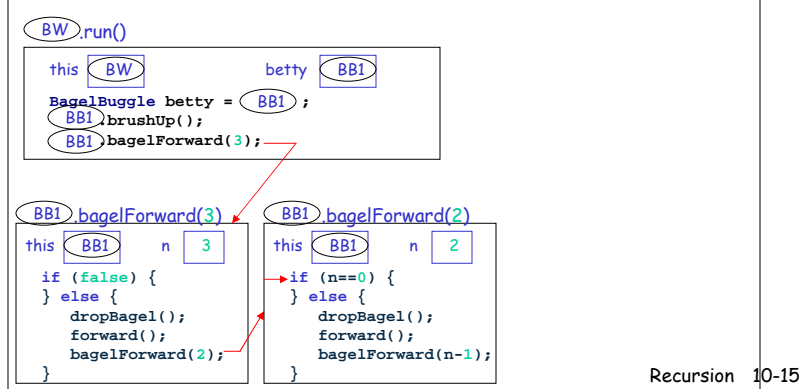
Recursion 10-14

Execute the method bagelForward(2)

Object Land

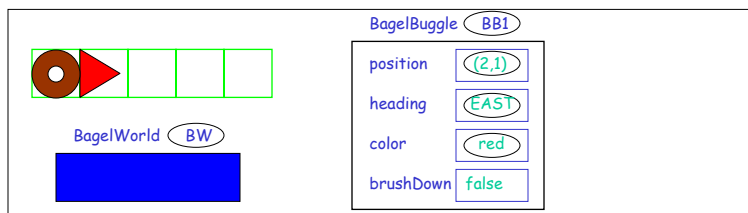


Execution Land

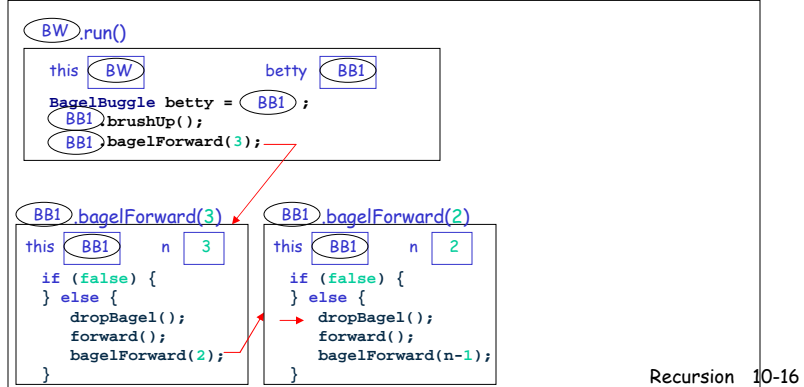


Evaluate the boolean expression and execute the else clause

Object Land

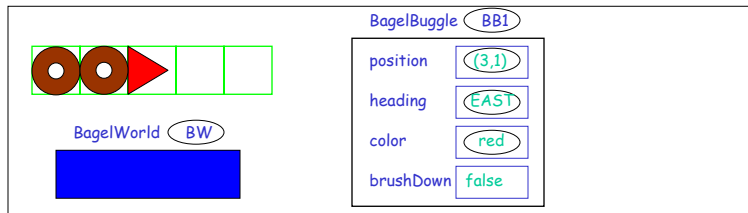


Execution Land

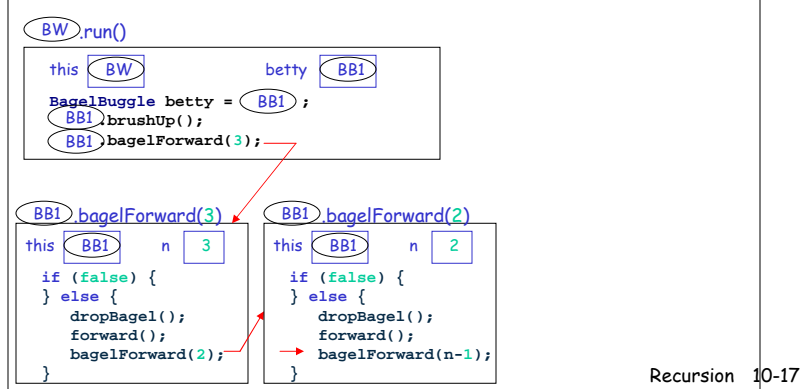


Drop a bagel and move forward

Object Land

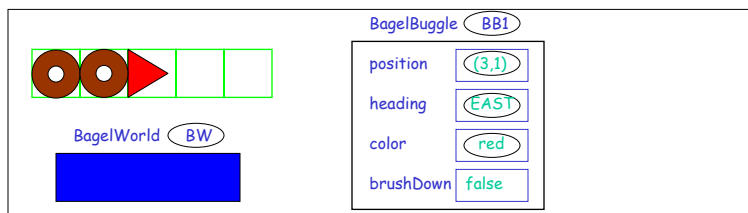


Execution Land

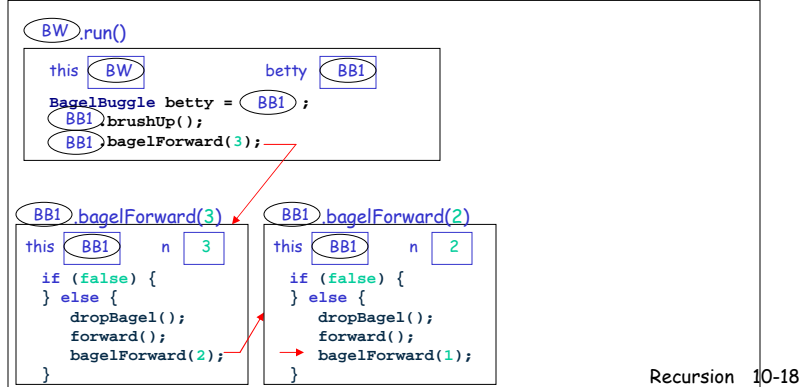


Evaluate the method's argument

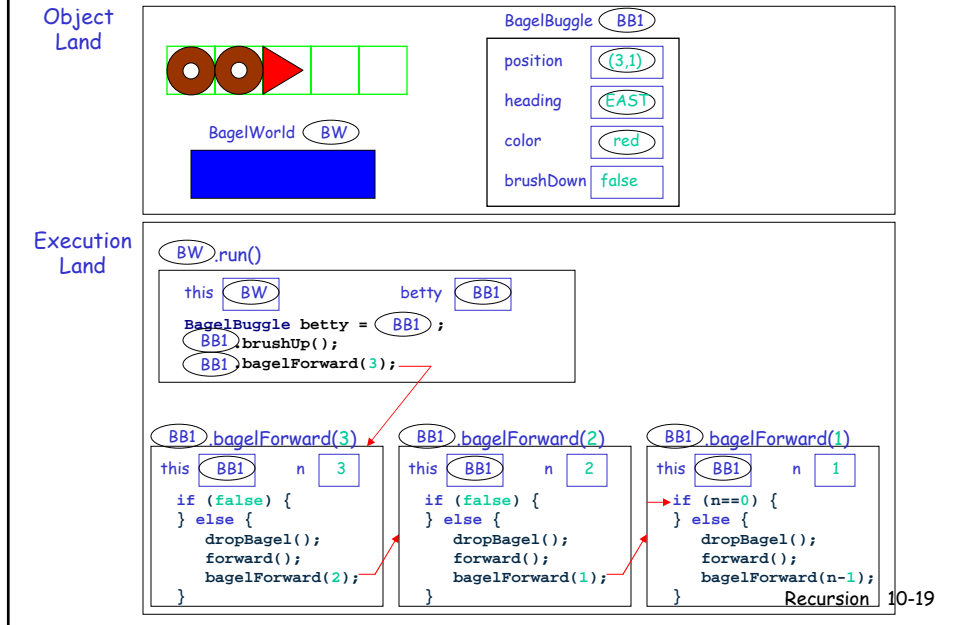
Object Land



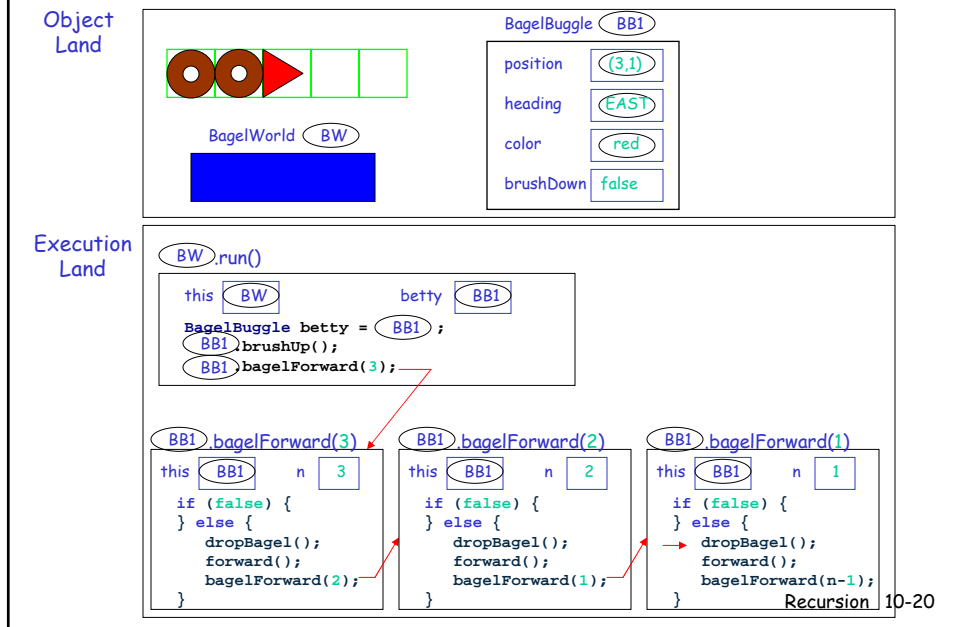
Execution Land



Execute the method bagelForward(1)

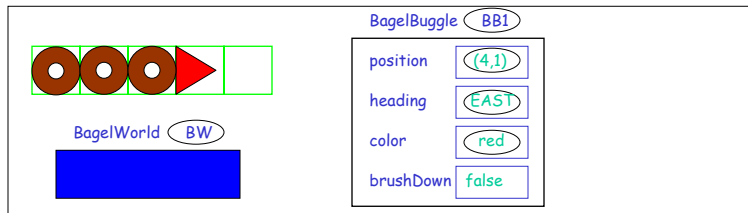


Evaluate the boolean expression and execute the else clause

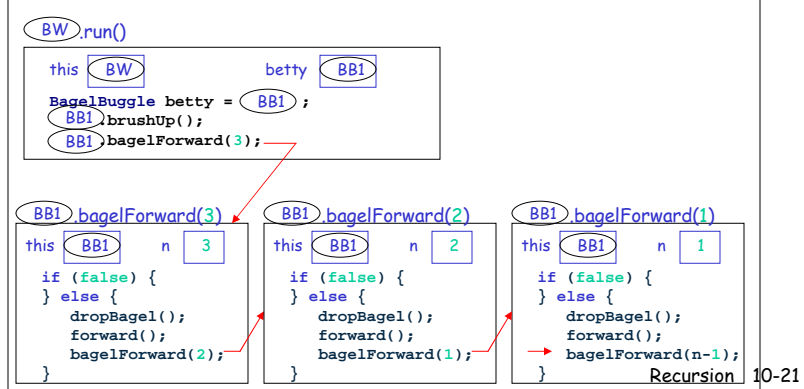


Drop a bagel and move forward

Object Land

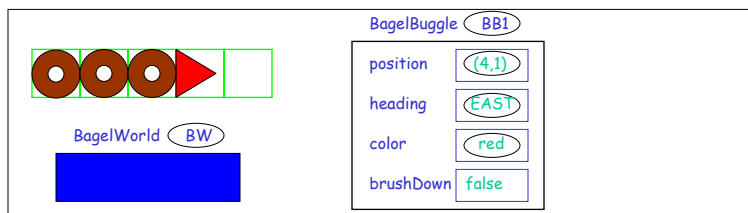


Execution Land

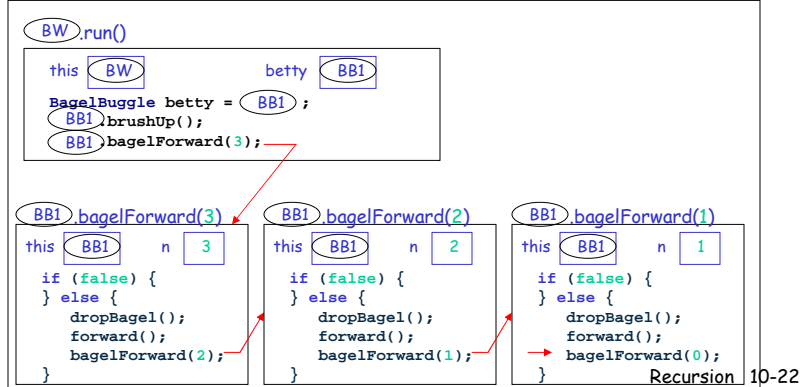


Evaluate the method's argument

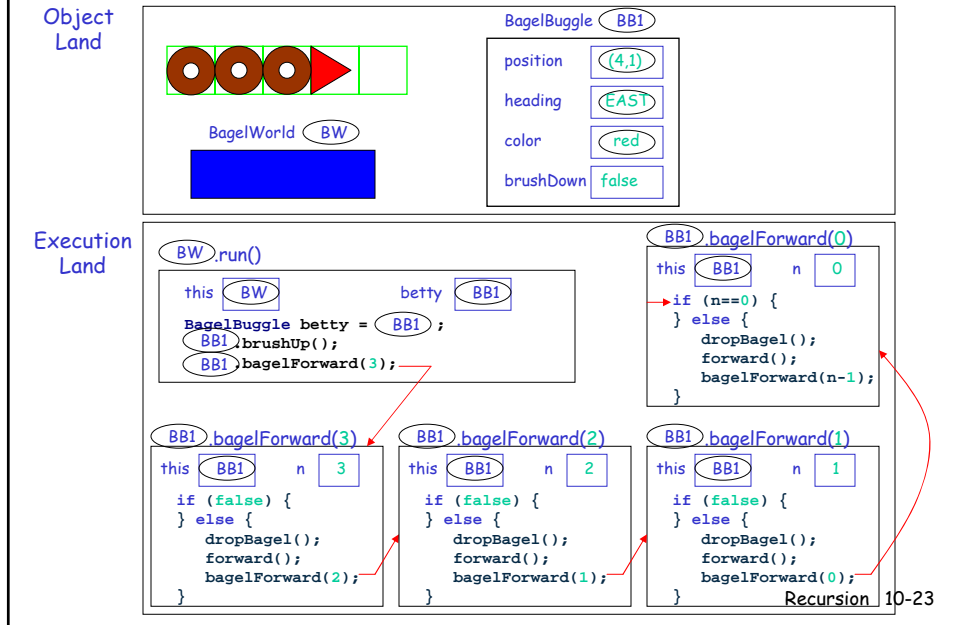
Object Land



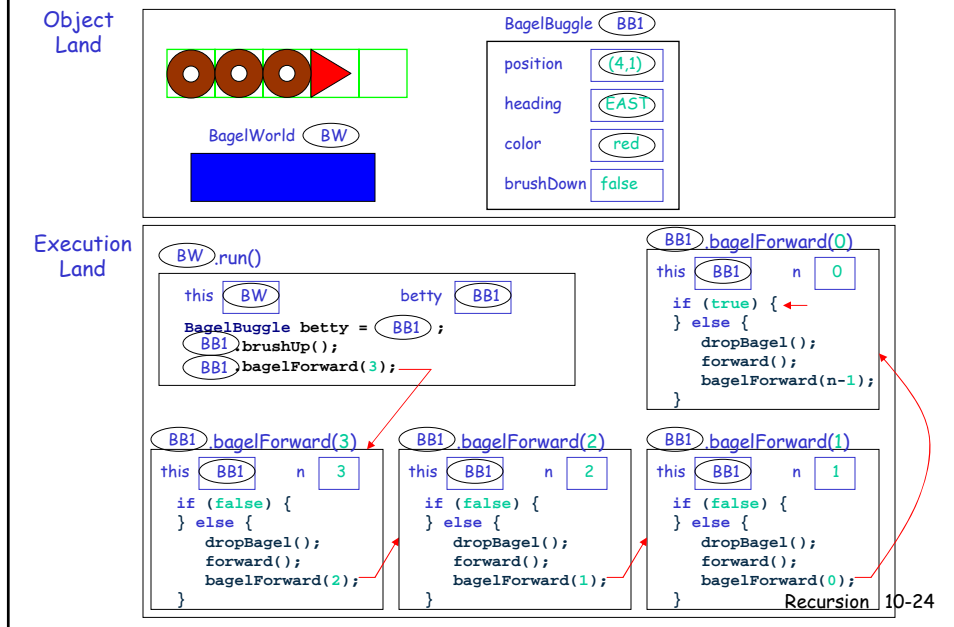
Execution Land



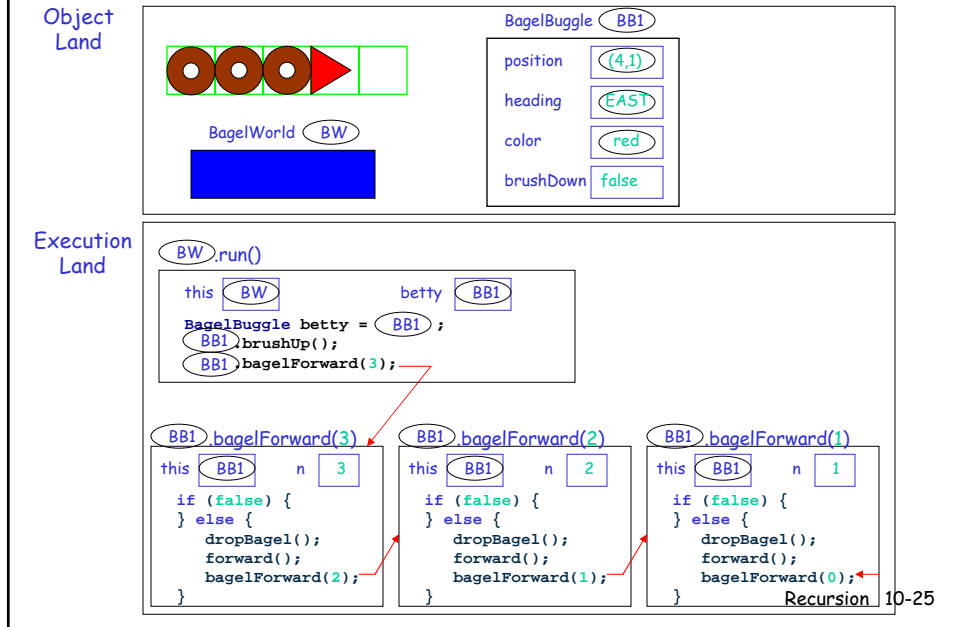
Execute the method bagelForward(0)



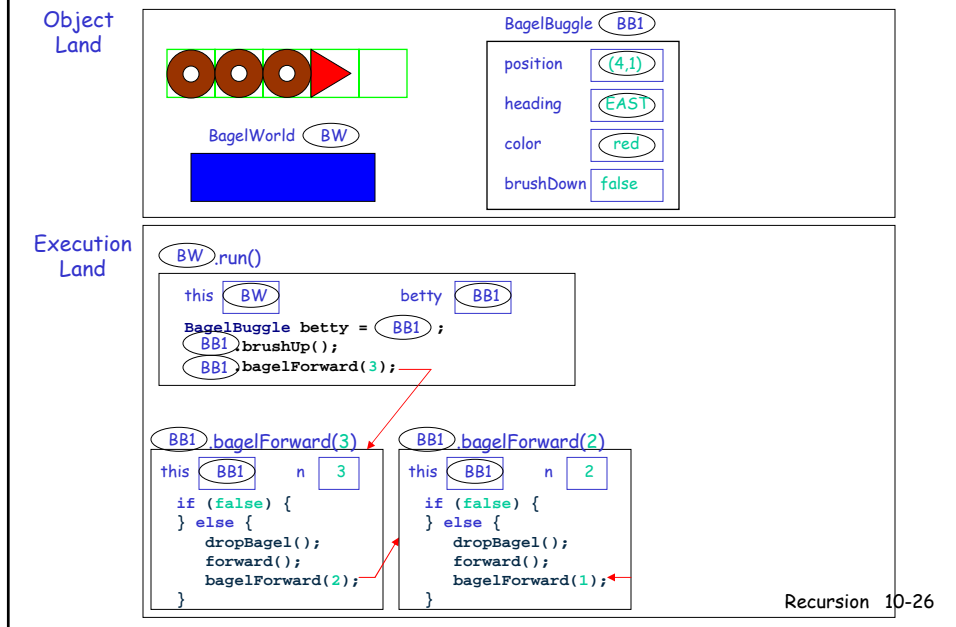
Evaluate the boolean expression and execute the then clause



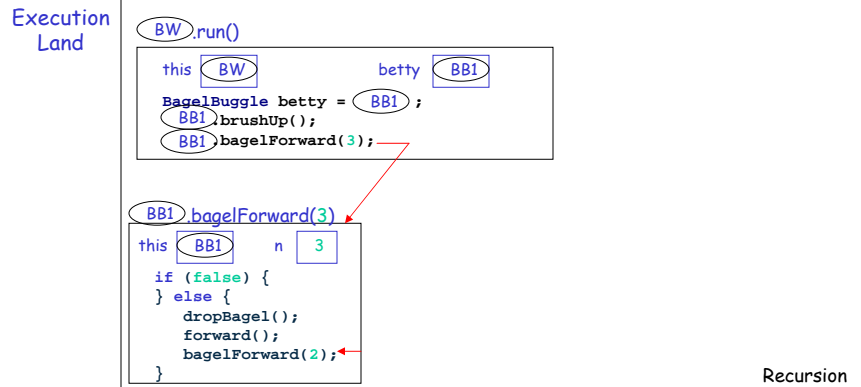
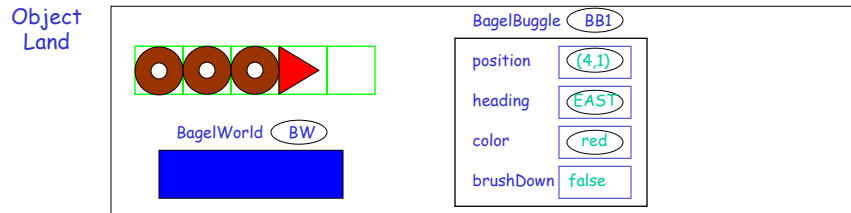
The bagelForward(0) execution frame finishes



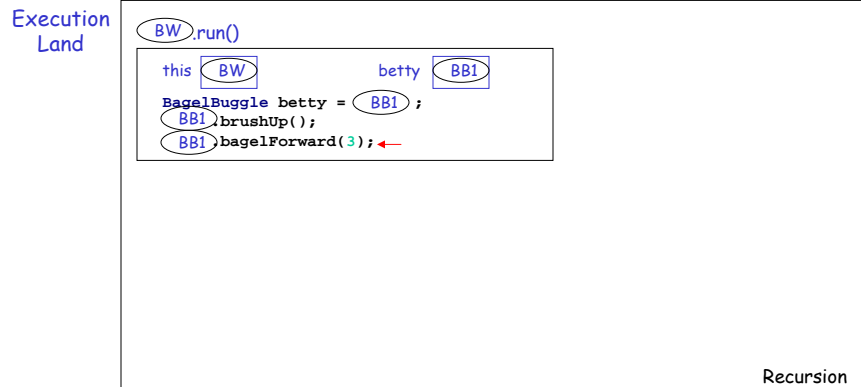
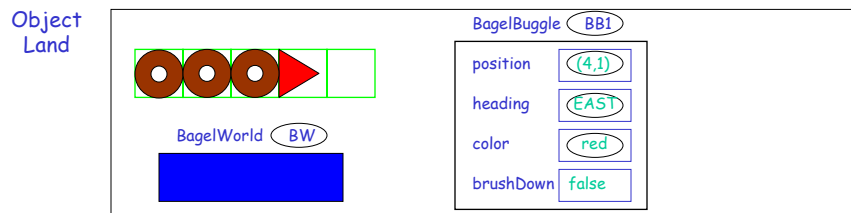
The bagelForward(1) execution frame finishes



The bagelForward(2) execution frame finishes

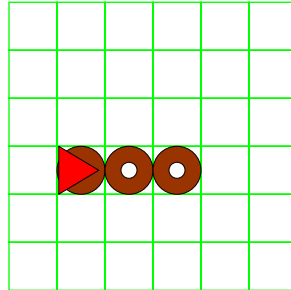


The bagelForward(3) execution frame finishes



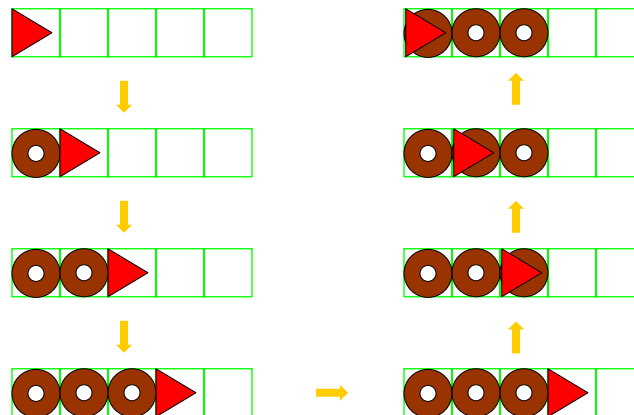
There and back: `bagelLine(3)`

Write a method that teaches a buggle to go forward n steps dropping a trail of n bagels behind it and return to the starting position. Assume `brushUp()`.



Recursion 10-29

Roundtrip BagelBuggle



Recursion 10-30

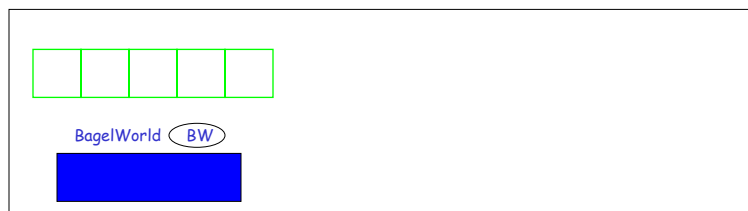
bagelLine(int n)

```
public void bagelLine(int n) {  
    if (n == 0) {  
        // do nothing  
    } else {  
        dropBagel();  
        forward();  
        bagelLine(n-1);  
        backward();  
    }  
}
```

Recursion 10-31

JEM demonstrating bagelLine method

Object
Land



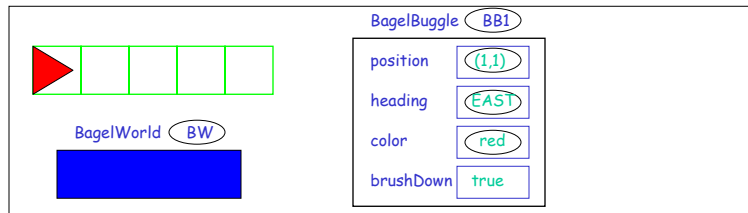
Execution
Land



Recursion 10-32

A buggle named betty is constructed

Object Land



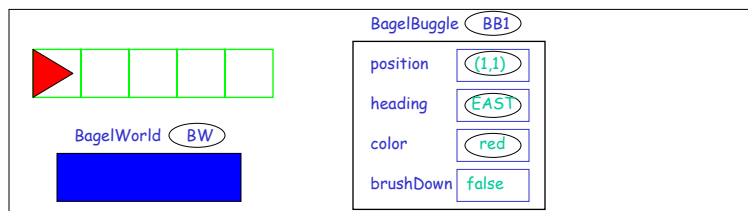
Execution Land

```
BW.run()  
  
this BW          betty BB1  
BagelBuggle betty = BB1 ;  
→ betty.brushUp();  
betty.bagelLine(3);
```

Recursion 10-33

betty raises her brush

Object Land

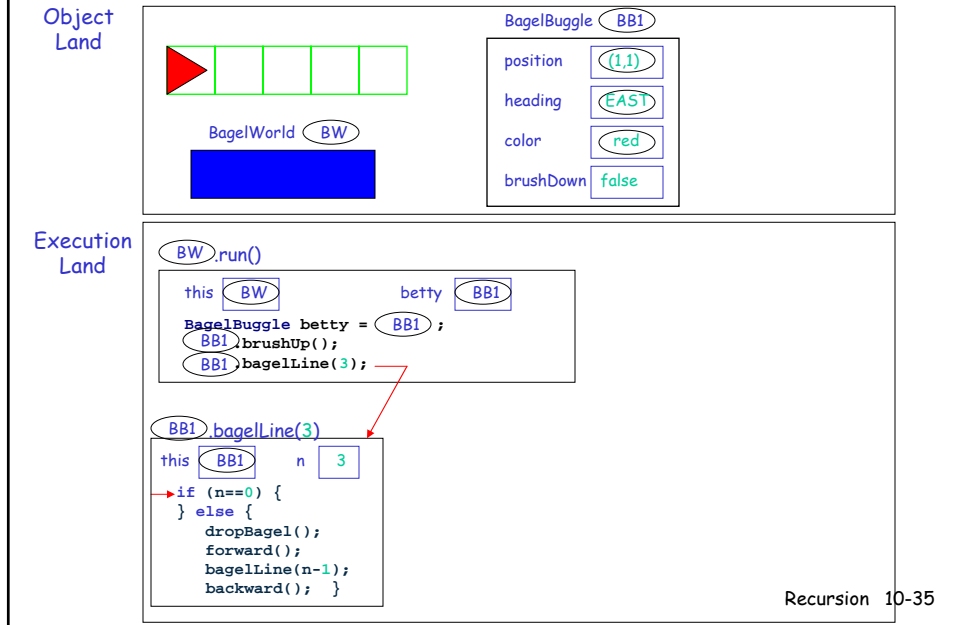


Execution Land

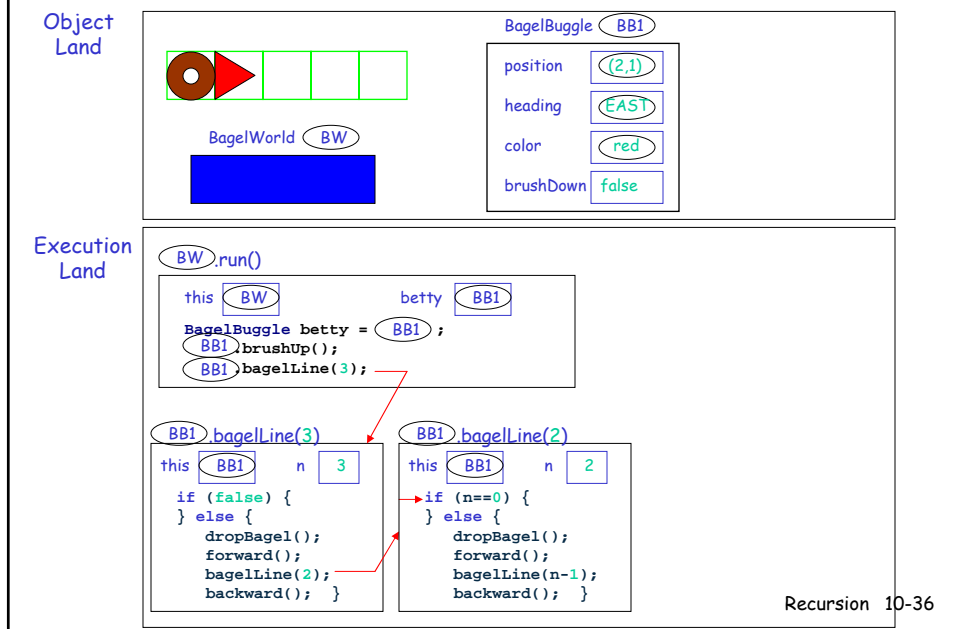
```
BW.run()  
  
this BW          betty BB1  
BagelBuggle betty = BB1 ;  
BB1.brushUp();  
→ betty.bagelLine(3);
```

Recursion 10-34

The message bagelLine(3) is sent to betty

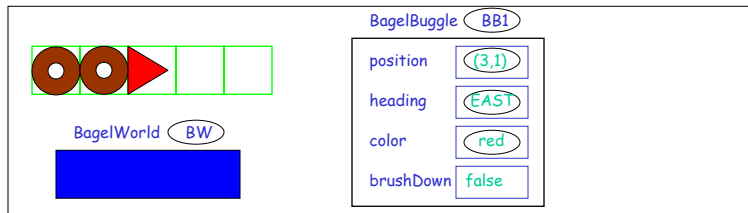


Evaluate the boolean expression, drop a bagel, move forward, and execute bagelLine(2)

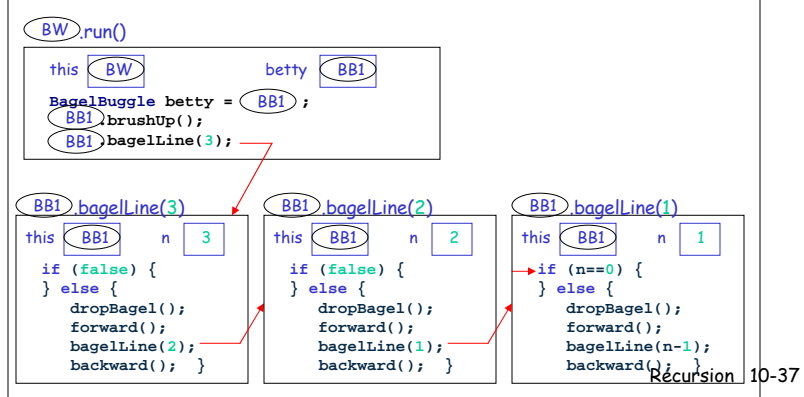


Evaluate the boolean expression, drop a bagel, move forward, and execute bagelLine(1)

Object Land

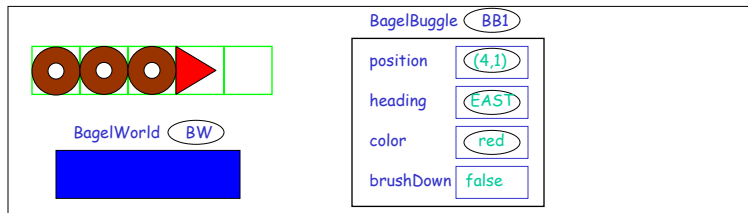


Execution Land

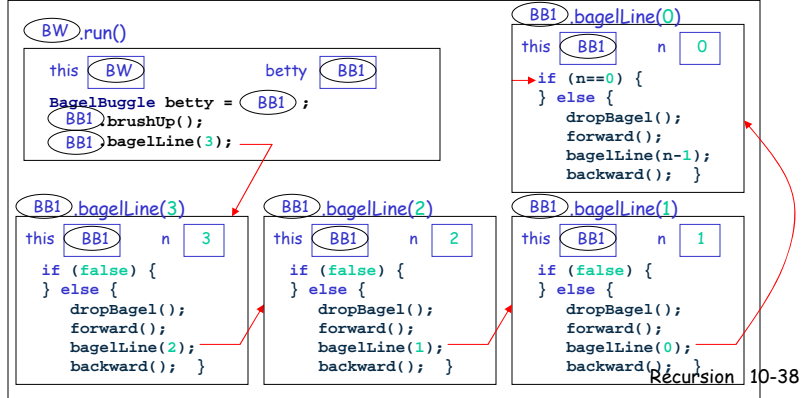


Evaluate the boolean expression, drop a bagel, move forward, and execute bagelLine(0)

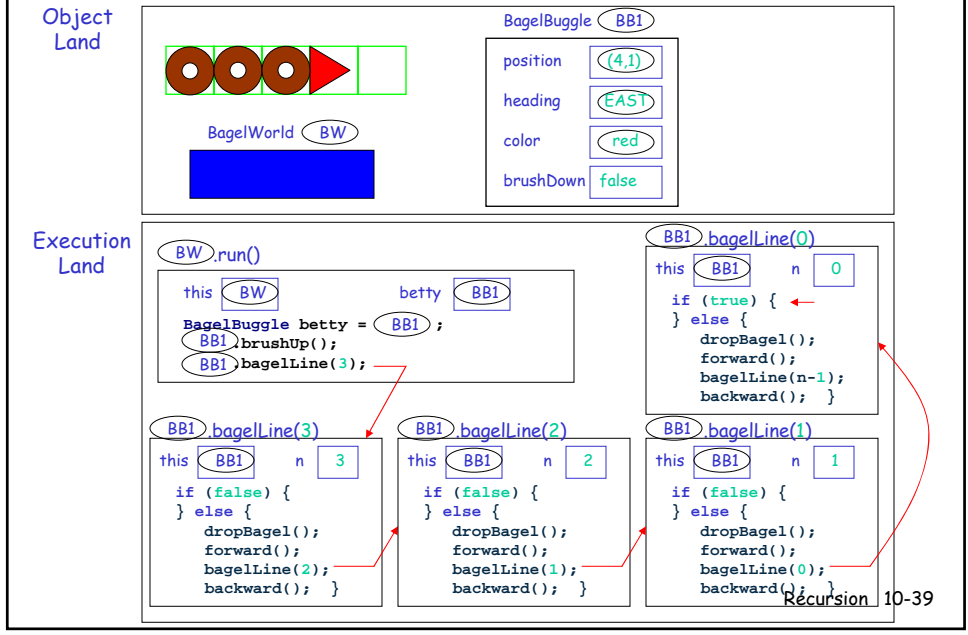
Object Land



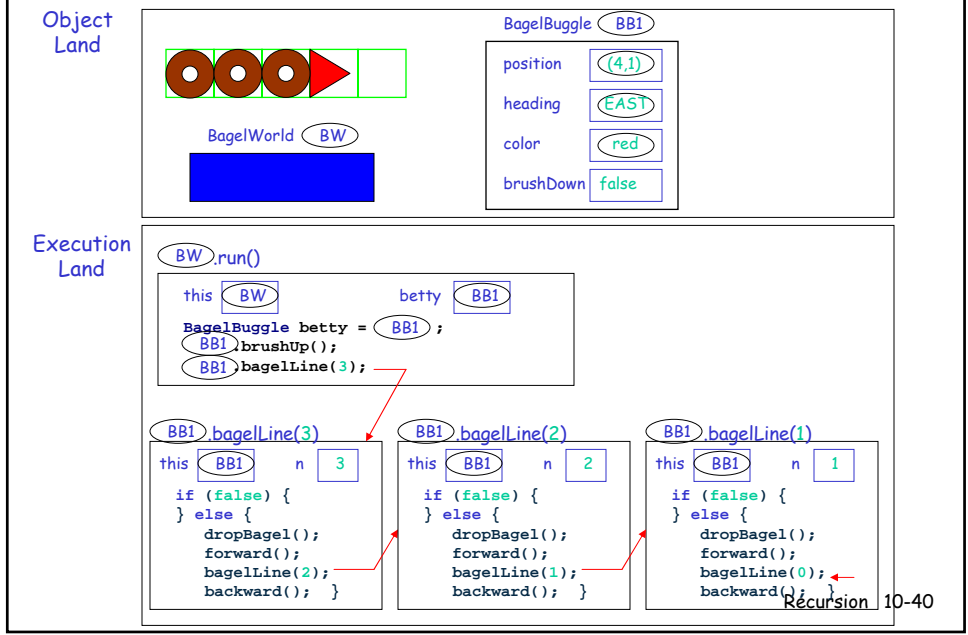
Execution Land



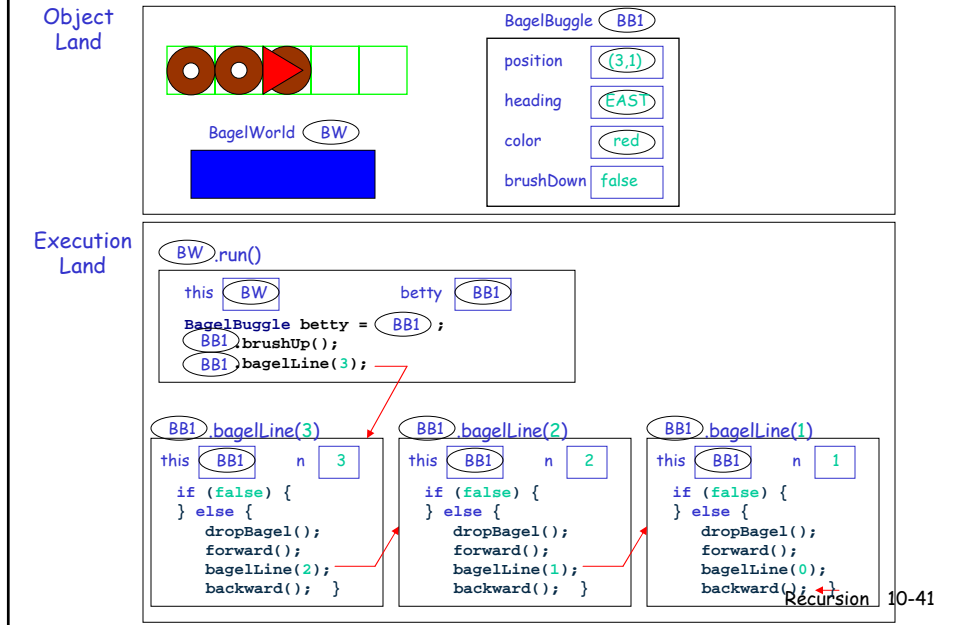
Evaluate the boolean expression and execute the then clause



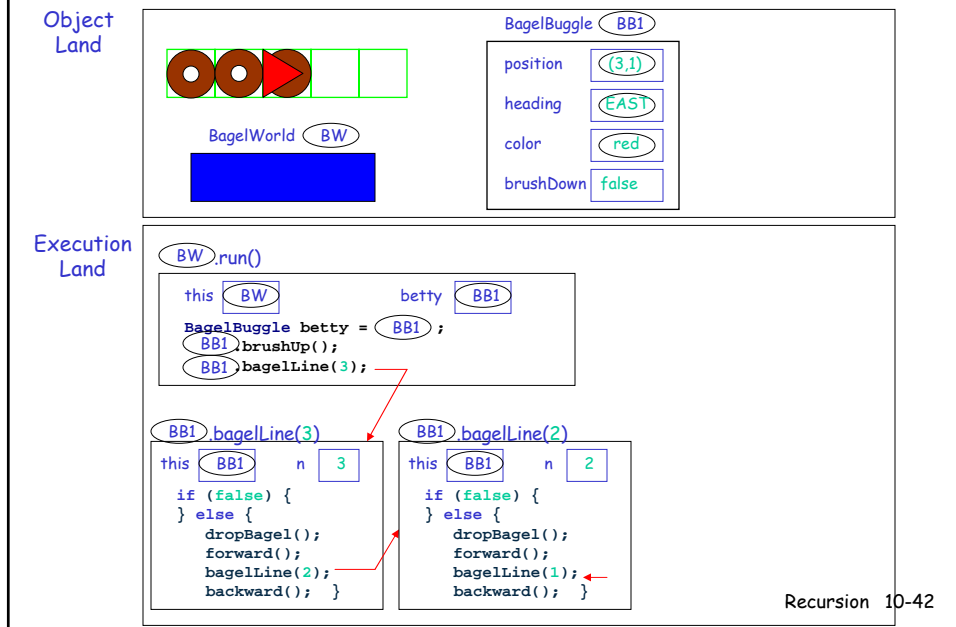
The bagelLine(0) execution frame finishes



Step backward

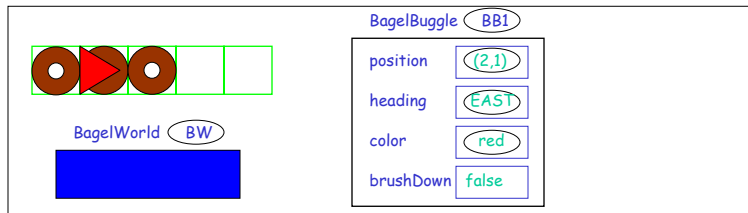


The bagelLine(1) execution frame finishes

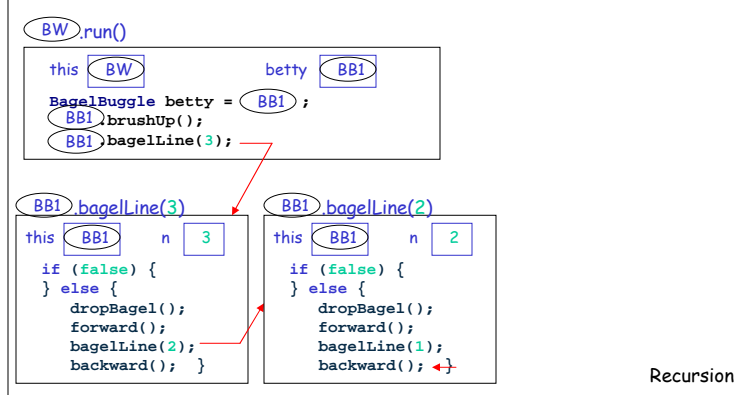


Step backward

Object Land



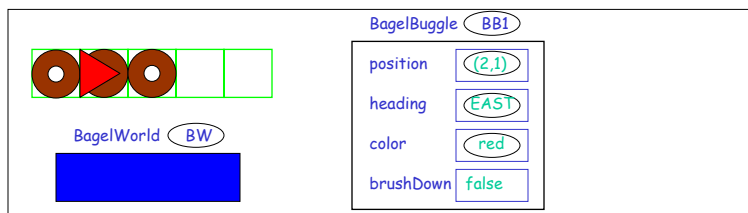
Execution Land



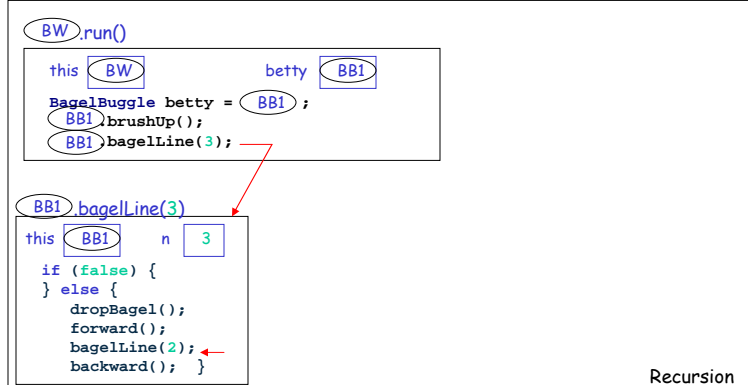
Recursion 10-43

The bagelLine(2) execution frame finishes

Object Land



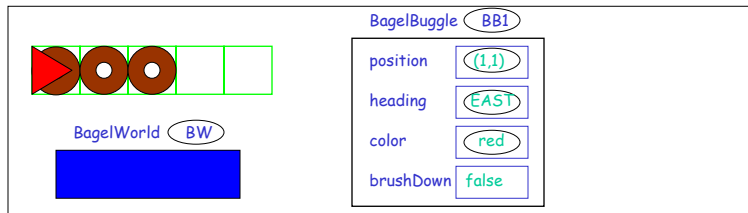
Execution Land



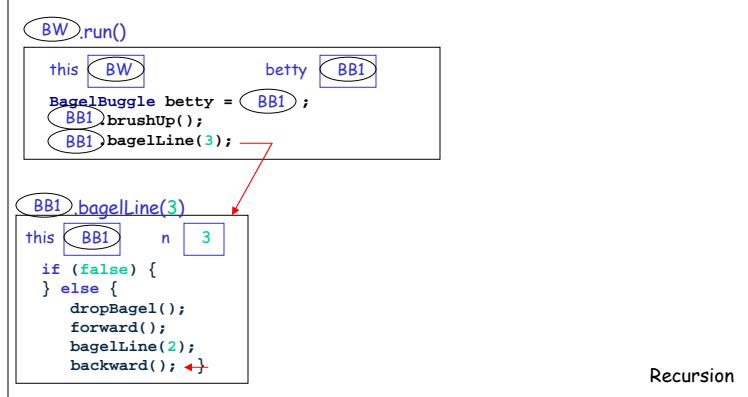
Recursion 10-44

Step backward

Object Land



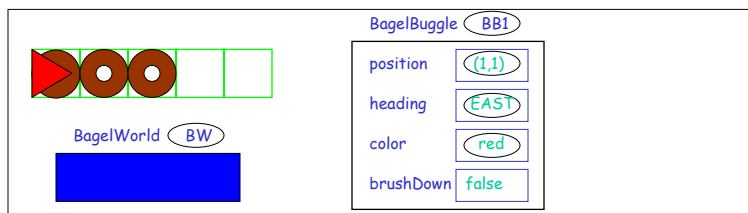
Execution Land



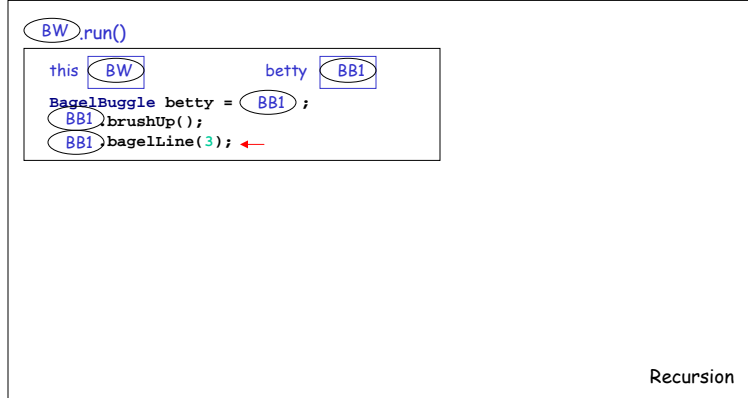
Recursion 10-45

The bagelLine(3) execution frame finishes

Object Land



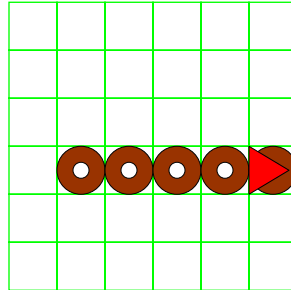
Execution Land



Recursion 10-46

bagelsToWall();

Write a method that teaches a buggle to drop a line of bagels from the buggle's current position all the way up to the wall. Assume brushUp().



Recursion 10-47

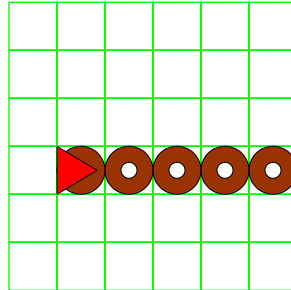
bagelsToWall();

```
public void bagelsToWall() {  
    if (isFacingWall()) {  
        dropBagel();  
    } else {  
        dropBagel();  
        forward();  
        bagelsToWall();  
    }  
}
```

Recursion 10-48

bagelsToWallAndBack();

Write a method that teaches a buggle to leave behind a trail of bagels all the way to the wall, and returns to the starting position. Assume brushUp().



Recursion 10-49

bagelsToWallAndBack();

```
public void bagelsToWallAndBack() {  
    if (isFacingWall()) {  
        dropBagel();  
    } else {  
        dropBagel();  
        forward();  
        bagelsToWallAndBack();  
        backward();  
    }  
}
```

Recursion 10-50

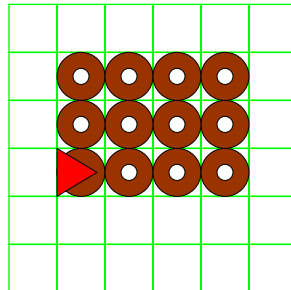
Alternatively

```
public void bagelsToWallAndBack() {  
    if (isFacingWall()) {  
        dropBagel();  
    } else {  
        forward();  
        bagelsToWallAndBack();  
        backward();  
        dropBagel();  
    }  
}
```

Recursion 10-51

Bagel Rectangle

Write a method that enables a bugle to draw a rectangle of bagels of width, w , and height, h , and return to the starting position*. Assume `brushUp()`.



*How many parameters would such a method take?

Recursion 10-52

bagelRect(int w, int h);

```
public void bagelRect(int w, int h) {  
  
    if (h == 0) {  
        // nothing to do  
    } else {  
        bagelLine(w);  
  
        left();           // move to next row  
        forward();  
        right();  
  
        bagelRect(w, h-1); // draw subrectangle  
  
        left();           // undo state changes  
        backward();  
        right();  
    }  
}
```

Recursion 10-53