Learning new words

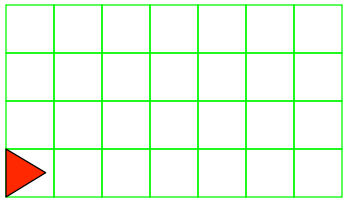
Methods
12 September 2006

CS111 Computer Programming

Department of Computer Science
Wellesley College

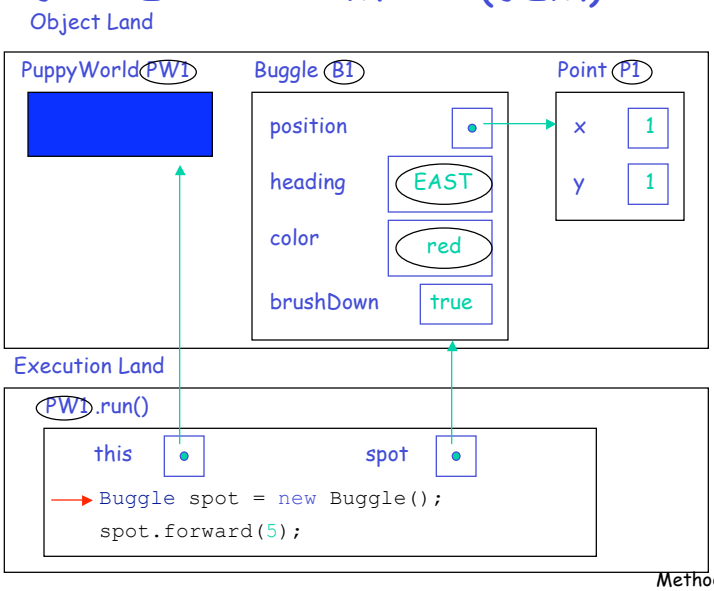
Inventing new worlds*

```
public class PuppyWorld extends BuggleWorld
{
     public void run()
     {
          Buggle spot = new Buggle();
          spot.forward(5);
     } // Run spot run
}
```

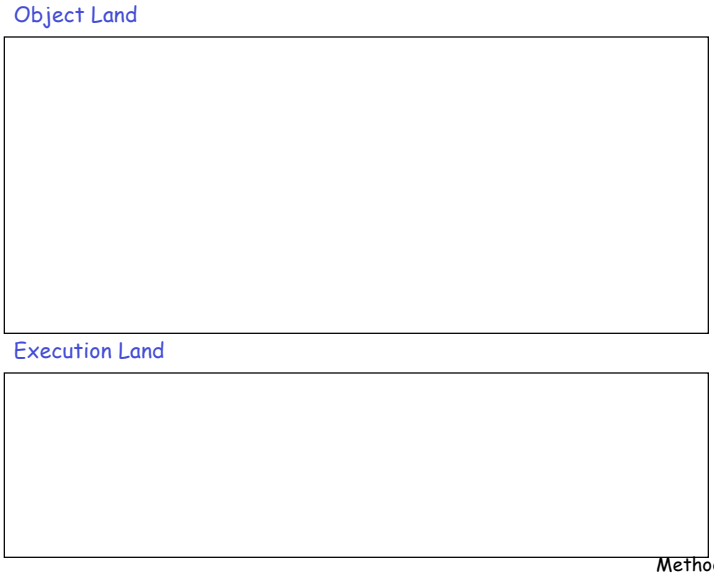


- `BuggleWorld` is itself a class.

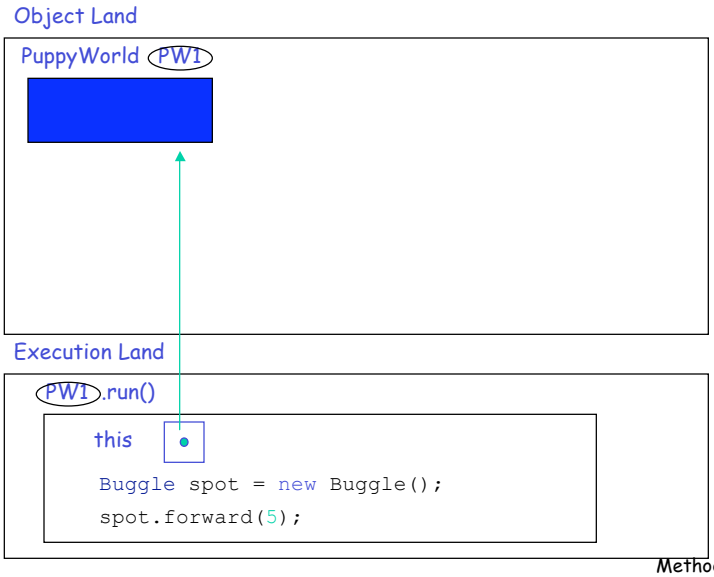
The Java Execution Model (JEM)



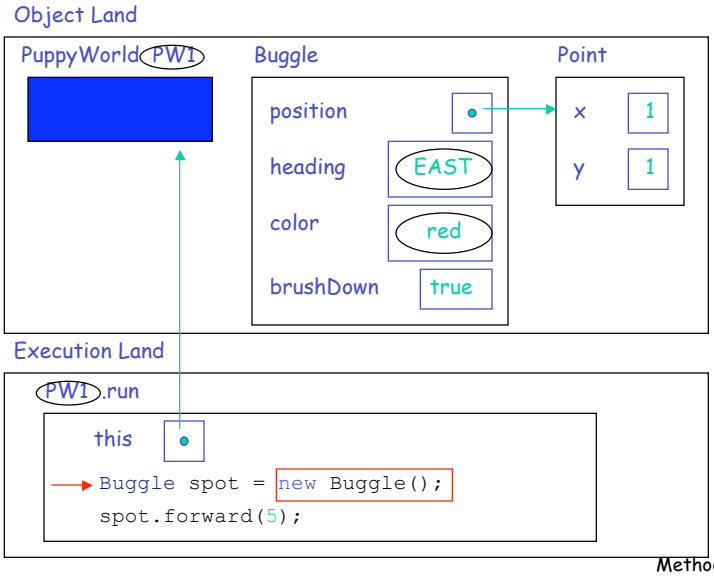
In the beginning



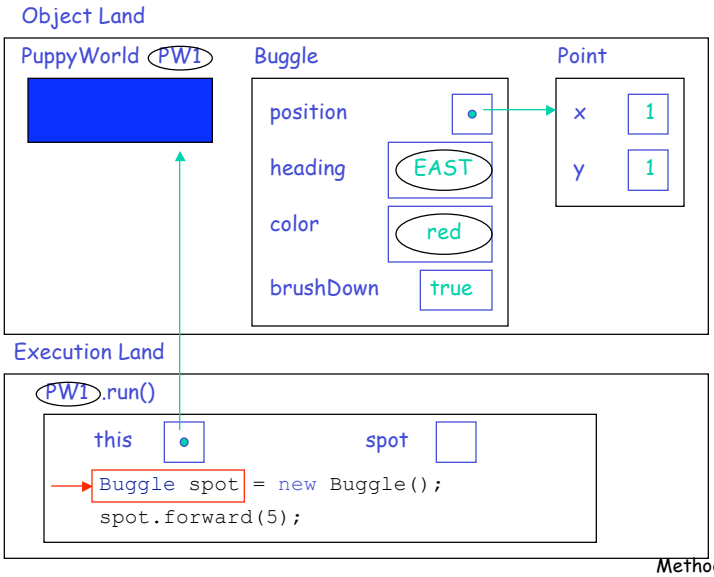
The `run()` method is invoked



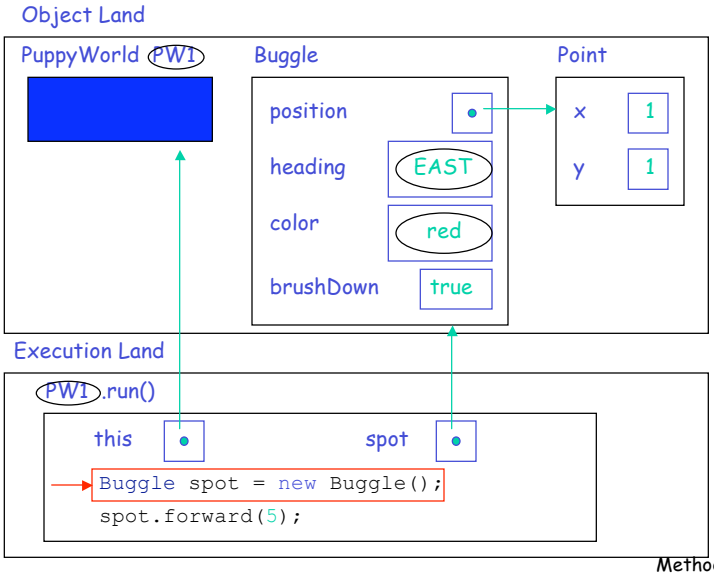
Execution of run() begins



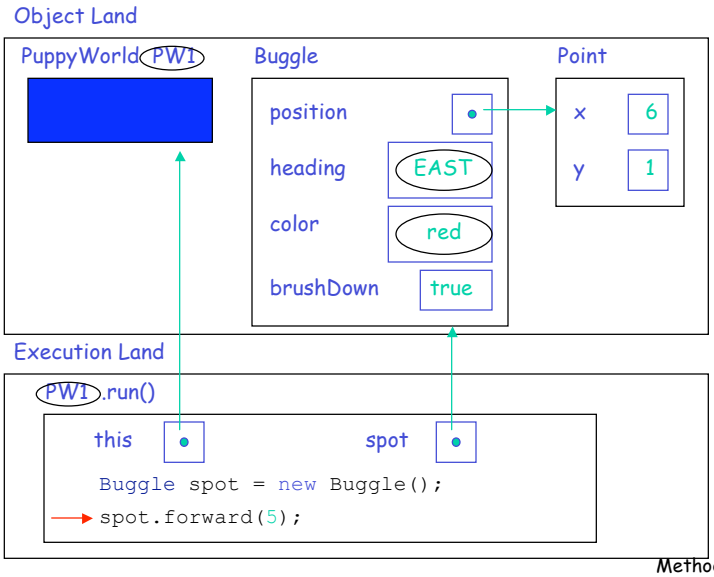
The Java Execution Model (JEM)



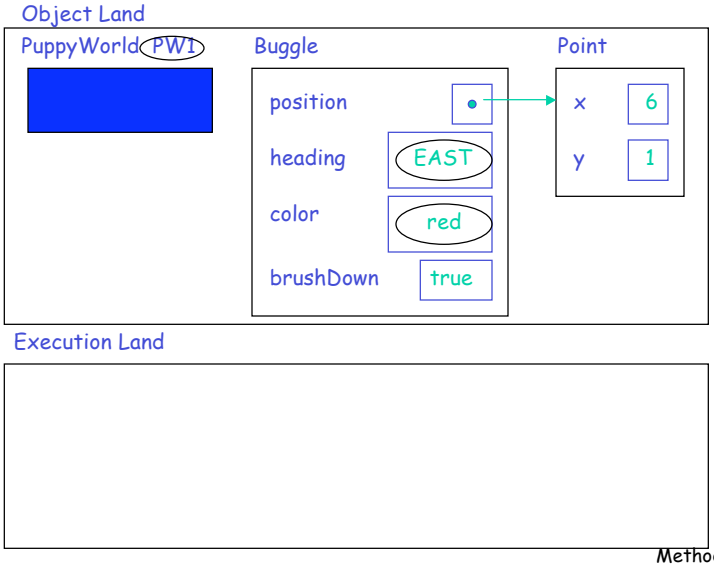
The assignment statement executes



Spot runs away

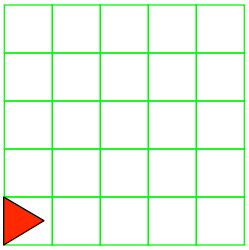


`run()` finishes execution



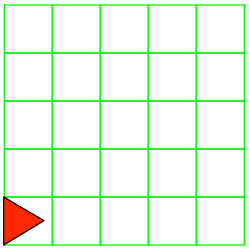
Spot does tricks

```
public class PuppyWorld extends BuggleWorld
{
     public void run()
     {
          Buggle spot = new Buggle();
          // roll over spot
          spot.left();
          spot.left();
          spot.left();
          spot.left();
          return;
     }
}
```



We would like to say . . . *

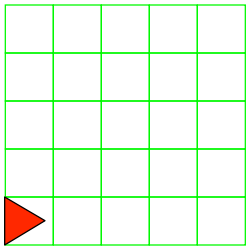
```
public class PuppyWorld extends BuggleWorld
{
     public void run()
     {
          Buggle spot = new Buggle();
          spot.rollover();
     }
}
```



*But spot would not understand. Spot is a Buggle and Buggles only understand what is in their contract.

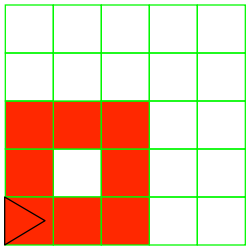
A new breed of Buggle

```
public class PuppyWorld extends BuggleWorld
{
      public void run()
      {
            PuppyBuggle spot = new PuppyBuggle();
            spot.rollover();
      }
}
class PuppyBuggle extends Buggle
{
      public void rollover()
      {
            this.left();
            this.left();
            this.left();
            this.left();
            return;
      }
}
```



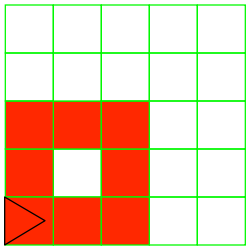
Spot learns to draw

```
public class PuppyWorld extends BuggleWorld
{
      public void run()
      {
            PuppyBuggle spot = new PuppyBuggle();
            spot.dogHouse(); // draws 3x3 doghouse
      }
}
class PuppyBuggle extends Buggle
{
      public void dogHouse()
      {
            // code to draw doghouse
      }
}
```

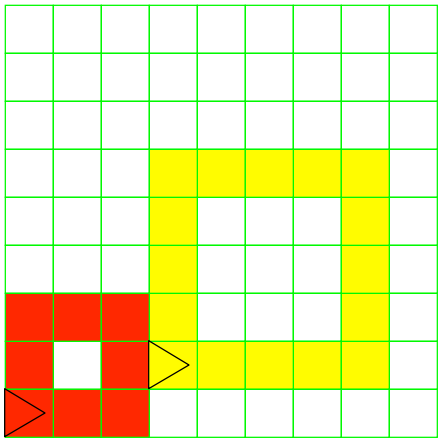


New code for PuppyBuggles

```
public class PuppyWorld extends BuggleWorld
{ … }
class PuppyBuggle extends Buggle
{
     public void dogHouse()
     {
          this.forward(2);
          this.left();
          this.forward(2);
          this.left();
          this.forward(2);
          this.left();
          this.forward(2);
          this.left();
     }
}
```



Lassie wants a dogHouse too*



*But collies are much bigger than terriers.

We could . . .

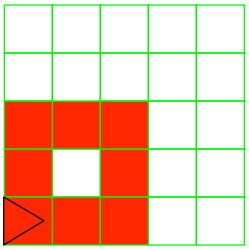
```
public class PuppyWorld extends BuggleWorld
{
      public void run()
      {
            PuppyBuggle spot = new PuppyBuggle();
            PuppyBuggle lassie = new PuppyBuggle();
            lassie.setLocation(new Point(4,2));
            lassie.setColor(Color.yellow);
            spot.dogHouse();
            lassie.bigDogHouse();
      }
}
class PuppyBuggle extends Buggle
{
      public void dogHouse() {… }
      public void bigDogHouse() {… }
}
```

A solution using parameters

```
public class PuppyWorld extends BuggleWorld
{
      public void run()
      {
            PuppyBuggle spot = new PuppyBuggle();
            PuppyBuggle lassie = new PuppyBuggle();
            lassie.setLocation(new Point(4,2));
            lassie.setColor(Color.yellow);
            spot.dogHouse(3);
            lassie.dogHouse(5);
      }
}
class PuppyBuggle extends Buggle
{
      public void dogHouse(int n) {… }
}
```

Methods with parameters

```
public class PuppyWorld extends BuggleWorld
{ … }
class PuppyBuggle extends Buggle
{
     public void dogHouse(int n)
     {
          this.forward(n-1);
          this.left();
          this.forward(n-1);
          this.left();
          this.forward(n-1);
          this.left();
          this.forward(n-1);
          this.left();
     }
}
```



JEM traces lassie and spot

Object Land

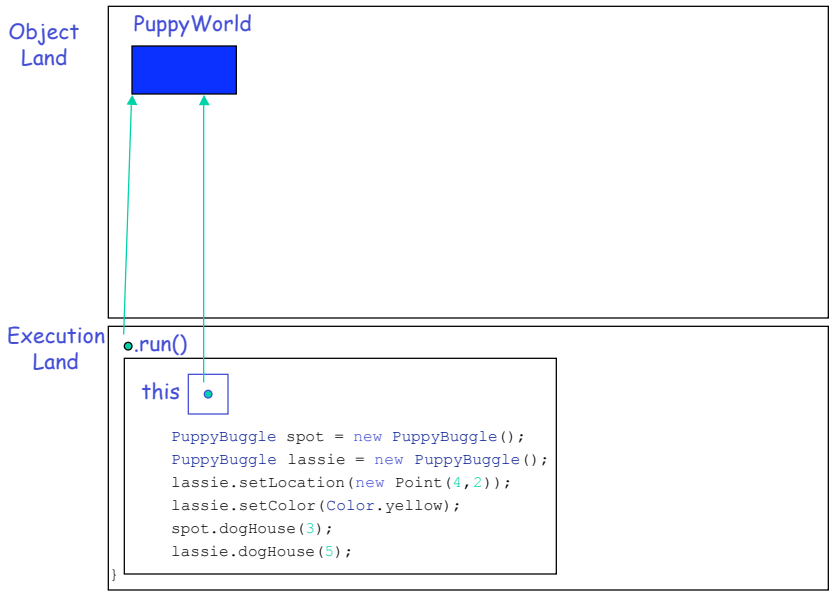
PuppyWorld PW1

Execution Land

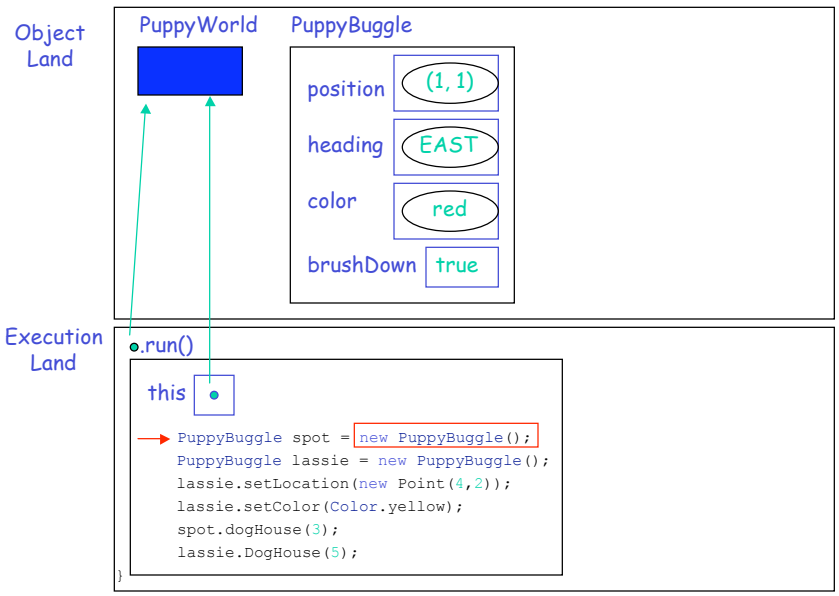
PW1.run()

```
PuppyBuggle spot = new PuppyBuggle();
PuppyBuggle lassie = new PuppyBuggle();
lassie.setLocation(new Point(4,2));
lassie.setColor(Color.yellow);
spot.dogHouse(3);
lassie.dogHouse(5);
}
```

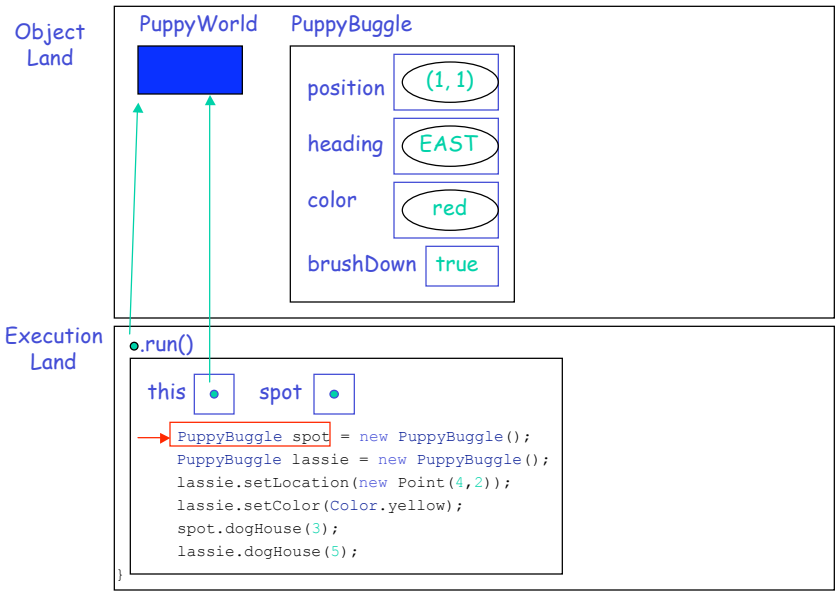
`run()` begins to run



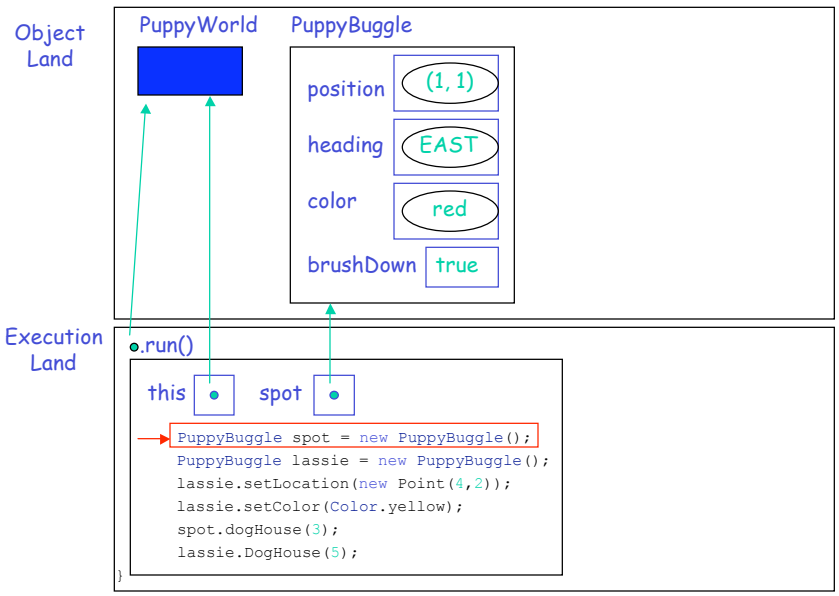
A PuppyBuggle is born



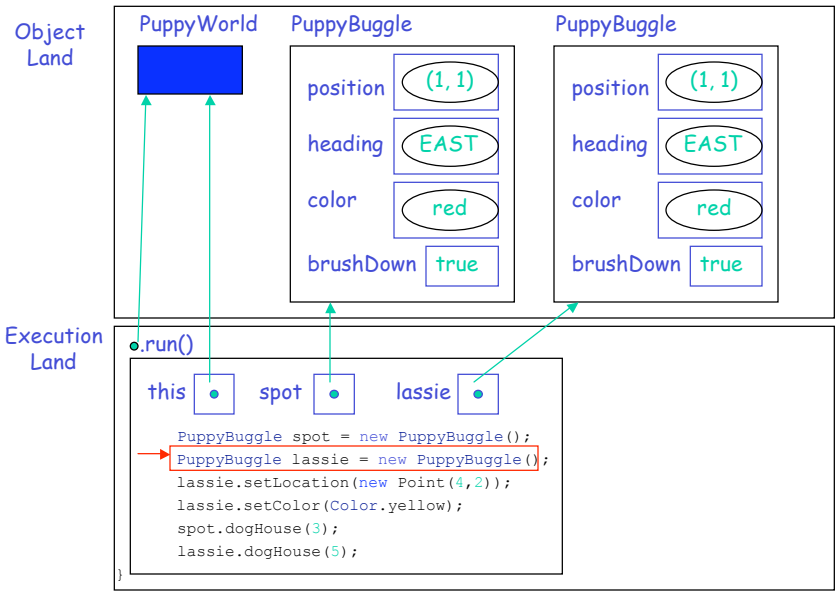
A name is chosen . . .



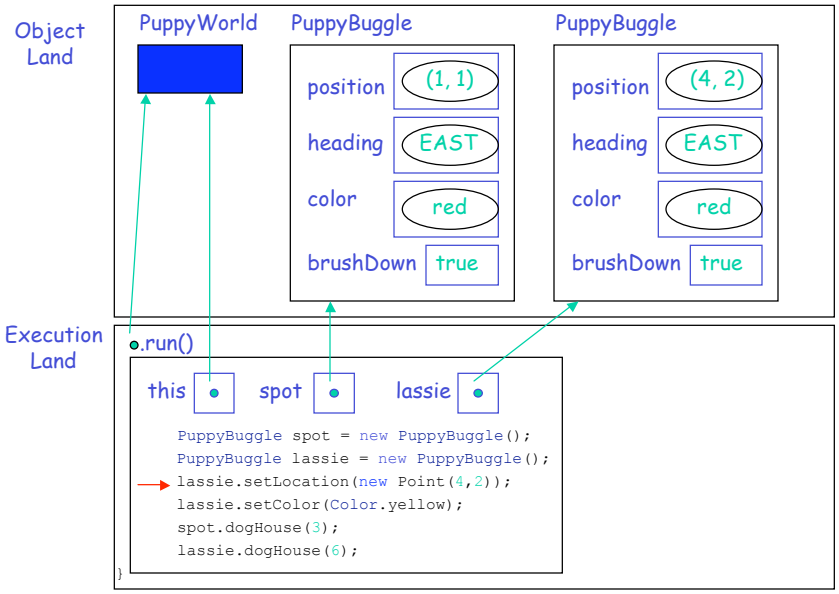
. . . and given to the new PuppyBuggle



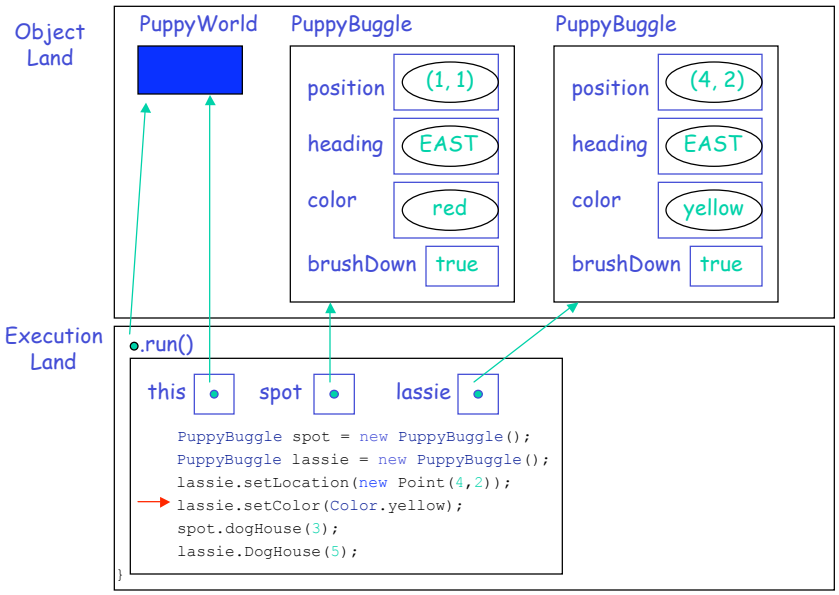
Lassie enters PuppyWorld in the same way



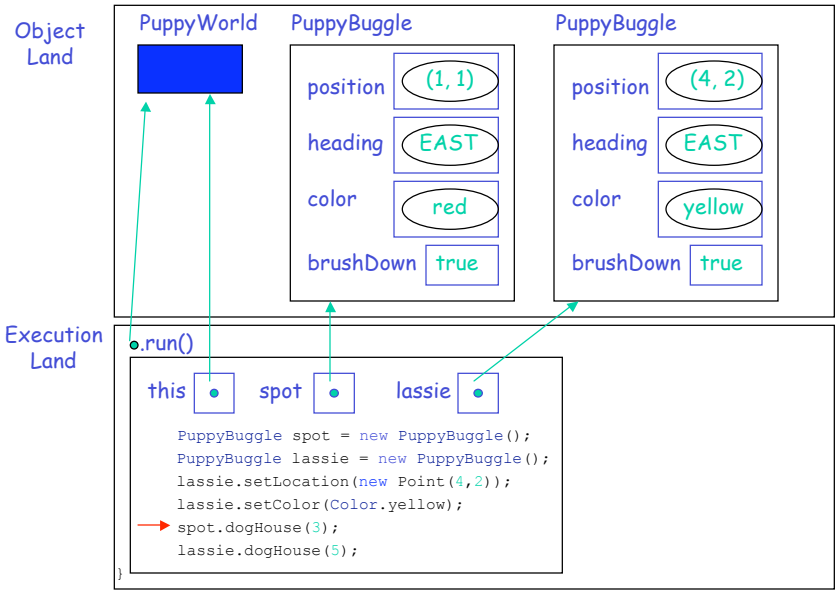
Lassie come home



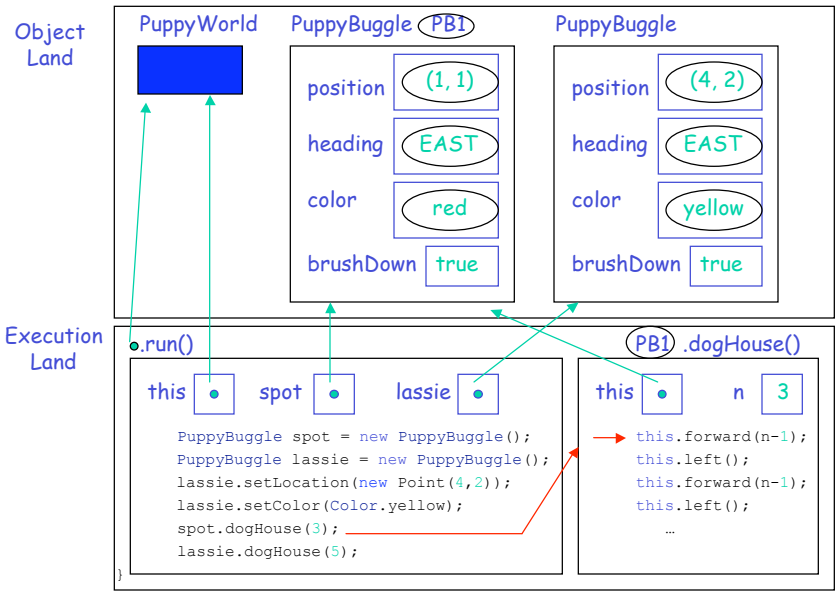
A new color for lassie



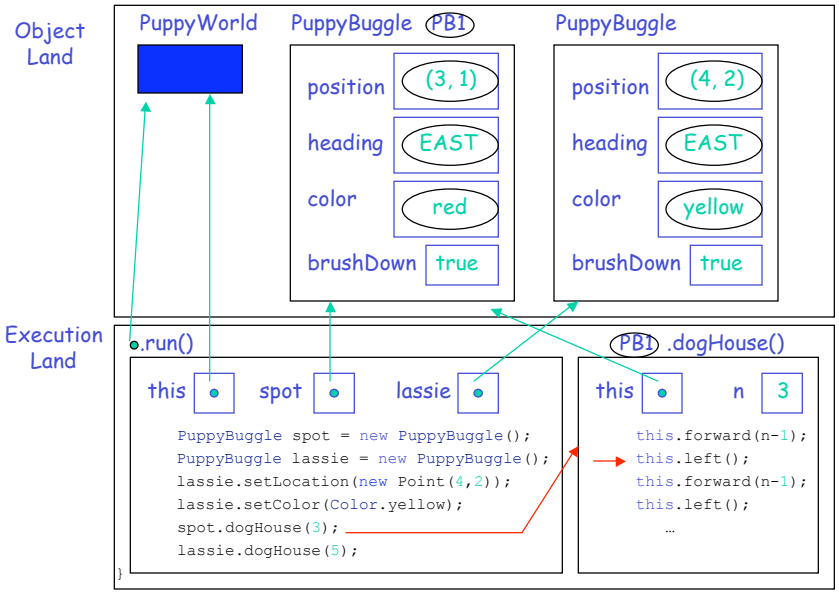
Spot is sent to the dogHouse



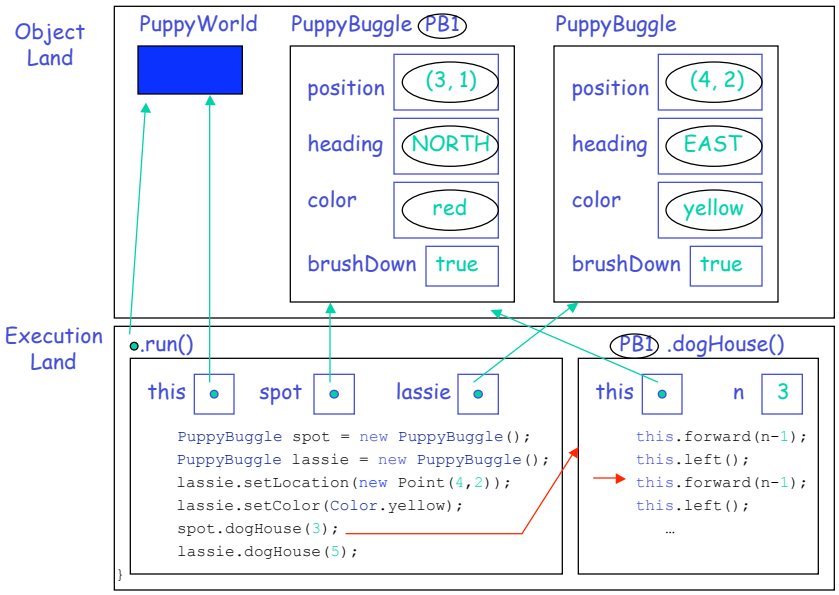
New a frame is created for `dogHouse()`



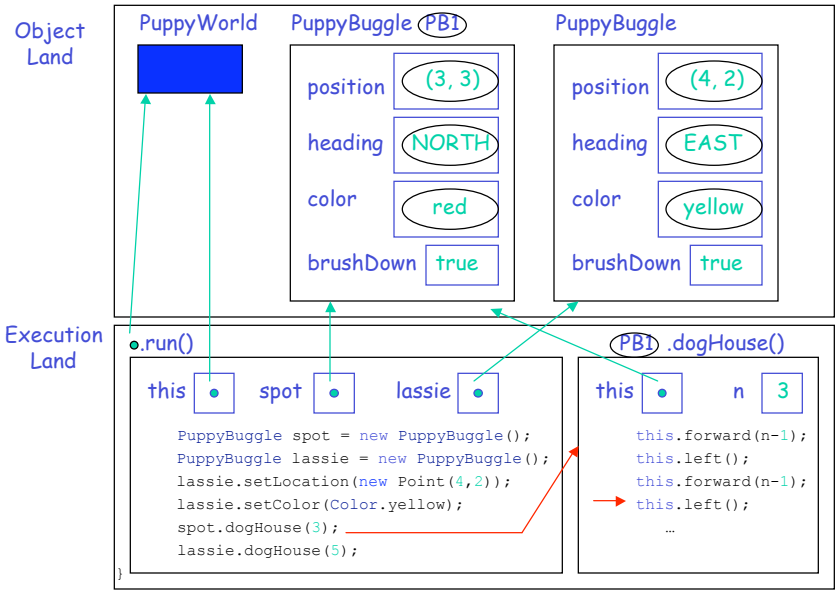
The value of n-1 is calculated `forward(2)` is executed



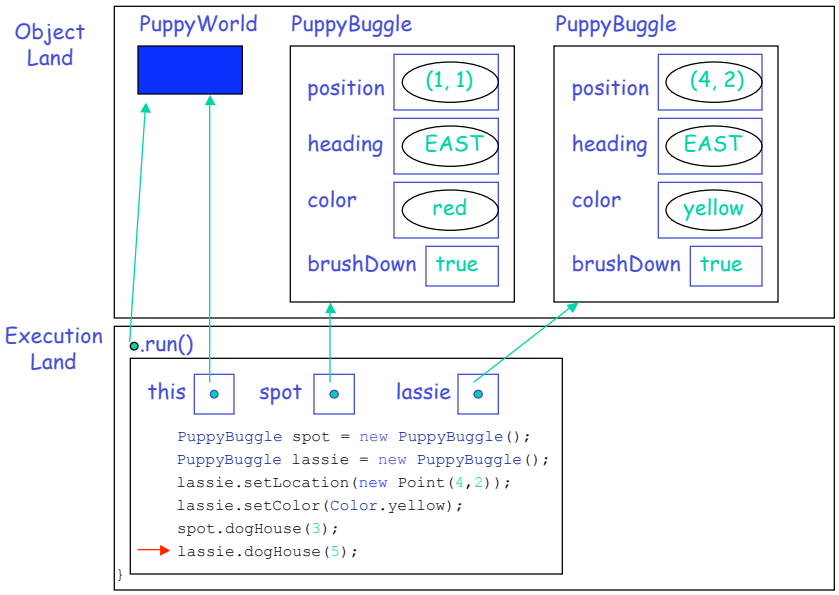
Spot turns left



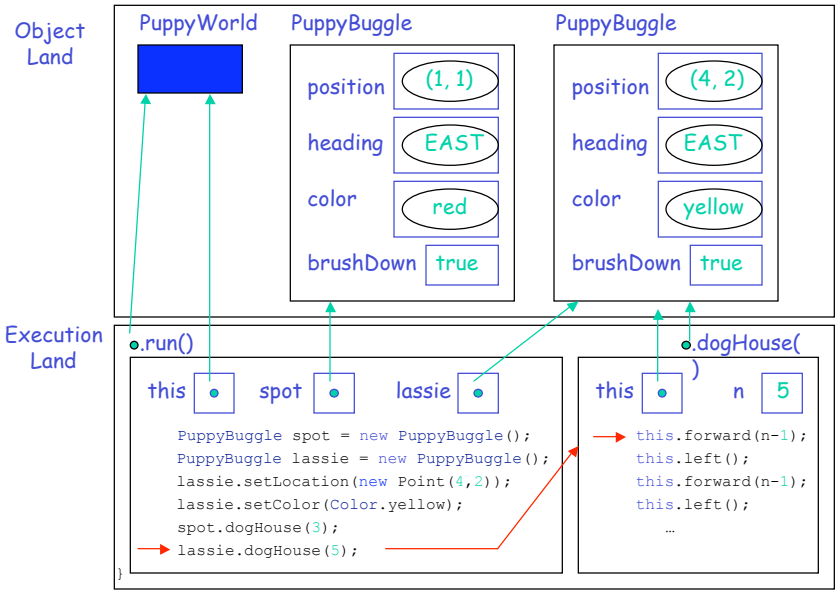
Spot and heads NORTH



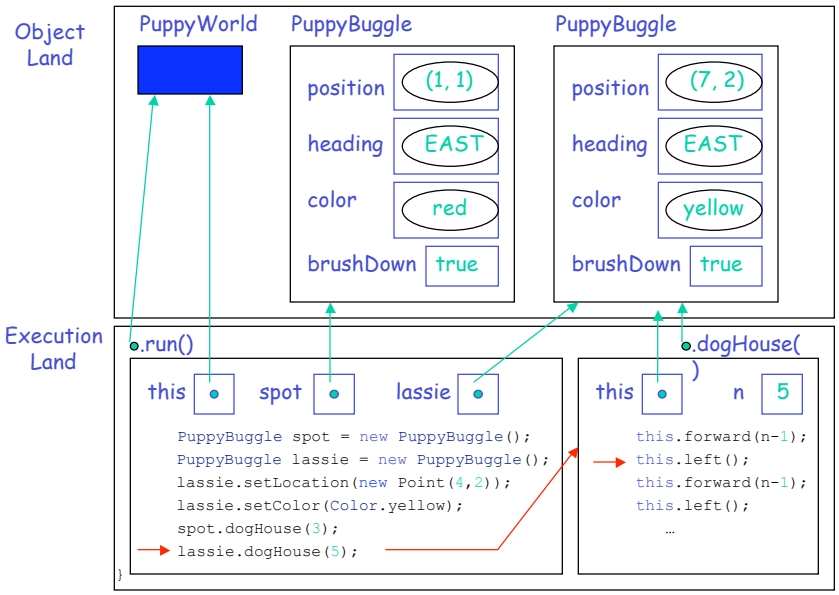
After a while `spot.dogHouse(3)` finishes



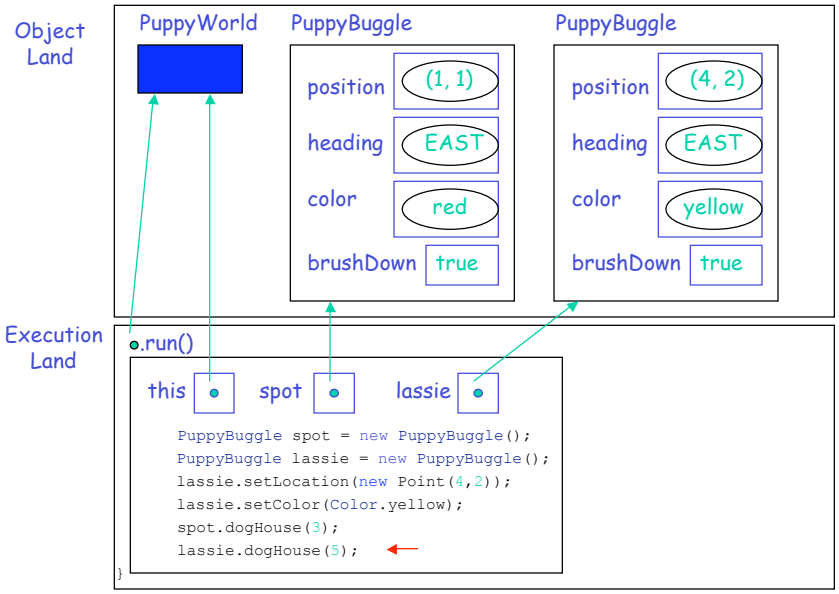
And `lassie.dogHouse(3)` begins



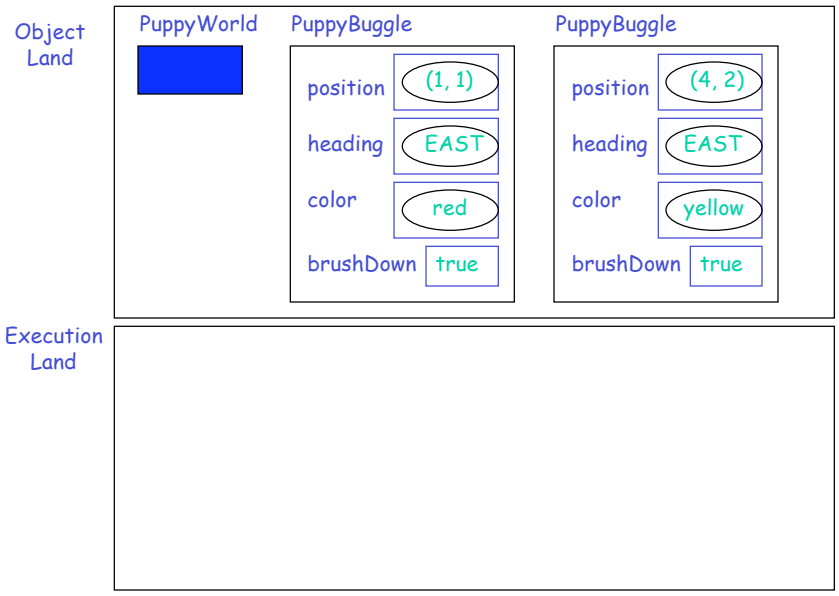
The show goes on . . .



Until lassie finishes her dogHouse



run() finishes and goes away too



Eventually the garbage collector makes her rounds

By the way, we could keep both versions*

```
public class PuppyWorld extends BuggleWorld
{
     public void run()
     {
          PuppyBuggle spot = new PuppyBuggle();
          PuppyBuggle lassie = new PuppyBuggle();
          lassie.setLocation(new Point(4,2));
          lassie.setColor(Color.yellow);
          spot.dogHouse();
          lassie.dogHouse(5);
     }
}
class PuppyBuggle extends Buggle
{
     public void dogHouse() {… }
     public void dogHouse(int n) {… }
}
```

*This is known as overloading.