KNOWN CLASSES

```
public class Dillo {
  public int length;
  public boolean isExtinct;

  public Dillo(int len, boolean isExtinct) {
    this.length = len;
    this.isExtinct = isExtinct;
  }

  public boolean canShelter() {
    return this.length > 60 && this.isExtinct;
  }
  public boolean isBigger(Dillo other) {
    return this.length > other.length;
  }
}
```

```
class AnimalsTest {
    Dillo babyDillo = new Dillo(8, false);
    Dillo medDillo  = new Dillo(20, true);
```

ENVIRONMENT

Names Java knows about
We say that names are "bound" to objects

babyDillo

medDillo

HEAP (OBJECTS)

Dillo
length: 8
isExtinct: false
canShelter()
isBigger()

Dillo
length: 20
isExtinct: true
canShelter()
isBigger()

Using "new" creates an object using its constructor. Here, we make two Dillos and give each one a name.

The assignment operator (=) maps a name to an object. We say that these names are "references" to individual objects that live in the heap.

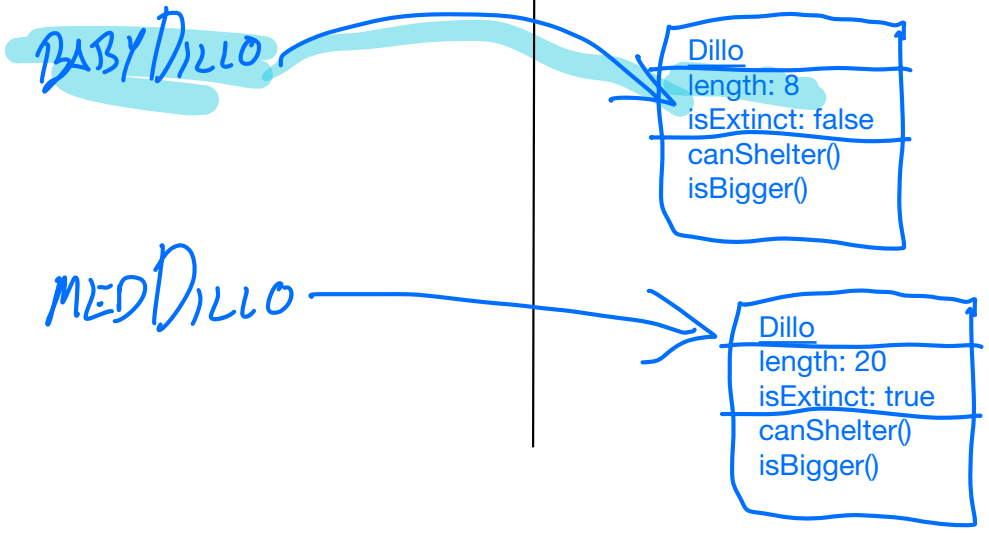
KNOWN CLASSES

```
public class Dillo {
  public int length;
  public boolean isExtinct;

  public Dillo(int len, boolean isExtinct) {
    this.length = len;
    this.isExtinct = isExtinct;
  }

  public boolean canShelter() {
    return this.length > 60 && this.isExtinct;
  }
  public boolean isBigger(Dillo other) {
    return this.length > other.length;
  }
}
```

```
class AnimalsTest {
    Dillo babyDillo = new Dillo(8, false);
    Dillo medDillo  = new Dillo(20, true);

 @Test
 public boolean testMakeDillo() {
     assertEquals(8, babyDillo.length);
 }
```

ENVIRONMENT

HEAP (OBJECTS)



When java runs testMakeDillo, it sees the name “babyDillo” and looks up this name in the environment.
Then, it uses the “.” operator to look inside the object referenced by babyDillo. From there, it can access components of this object.
Here, we access the length field.

KNOWN CLASSES

```
public class Dillo {
  public int length;
  public boolean isExtinct;

  public Dillo(int len, boolean isExtinct) {
    this.length = len;
    this.isExtinct = isExtinct;
  }

  public boolean canShelter() {
    return this.length > 60 && this.isExtinct;
  }
  public boolean isBigger(Dillo other) {
    return this.length > other.length;
  }
}
```

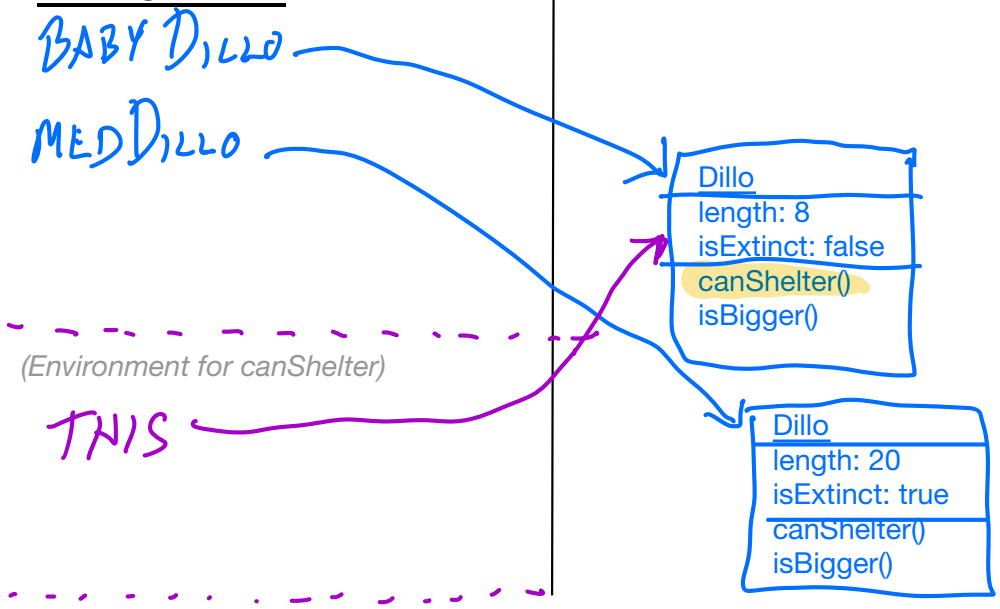
```
class AnimalsTest {
    Dillo babyDillo = new Dillo(8, false);
    Dillo medDillo  = new Dillo(20, true);

 @Test
 public boolean testMakeDillo() {
     assertEquals(8, babyDillo.length);
 }
 @Test
 public boolean testCanShelter() {
   assertEquals(false,babyDillo.canShelter());
 }
```

1

To call a method inside an object, we do the same thing—we find babyDillo in the environment and find the object it references, then we access the canShelter method inside that object.

```
c
}
```

ENVIRONMENT



HEAP (OBJECTS)

2

We can think of the environment as divided into parts. When we call any method, it can add names to its own part of the environment—these go away when the method returns. This purple box is canShelter's part of the environment.

canShelter uses the name "this". When we call canShelter, Java sets up the name "this" to point to the Dillo object where the method lives. In this way, "this" means "the current object where we are in right now" and not any other Dillo object.

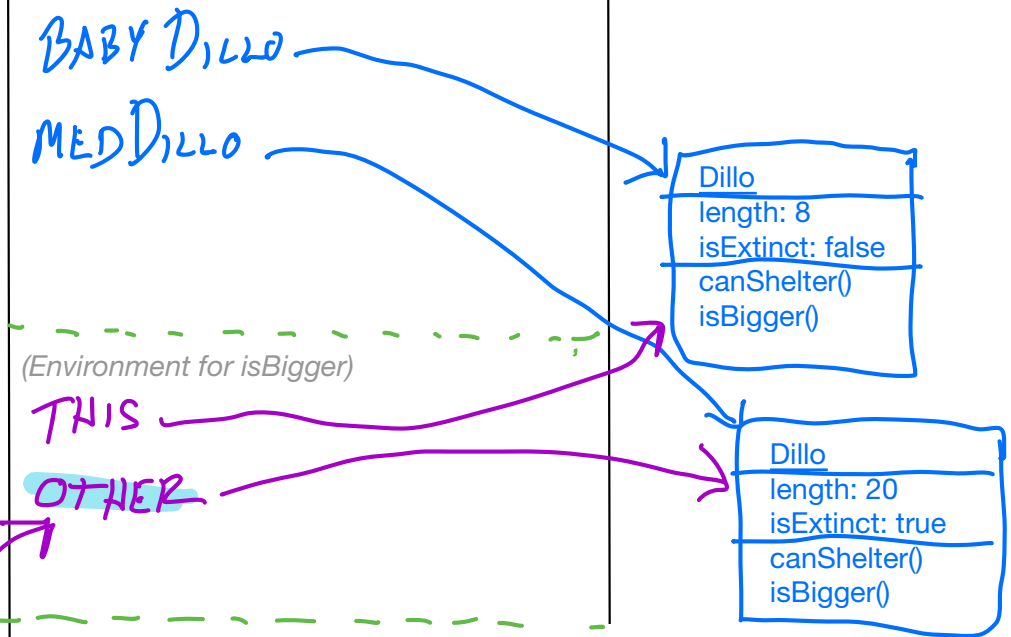
Note: canShelter and babyDillo aren't special. All code must live inside a class, so every method has a "this". We sometimes don't write or draw it in test files to save on typing. For example, we could refer to babyDillo in AnimalsTest as this.babyDillo

KNOWN CLASSES

```
public class Dillo {
  public int length;
  public boolean isExtinct;

  public Dillo(int len, boolean isExtinct) {
    this.length = len;
    this.isExtinct = isExtinct;
  }

  public boolean canShelter() {
    return this.length > 60 && this.isExtinct;
  }
  public boolean isBigger(Dillo other) {
    return this.length > other.length;
  }
}
```

```
class AnimalsTest {
  Dillo babyDillo = new Dillo(8, false);
  Dillo medDillo = new Dillo(20, true);

 @Test
 public void testMakeDillo() {
    assertEquals(8, babyDillo.length)
 }
 @Test
 public boolean testCanShelter() {
  assertEquals(false,
               babyDillo.canShelter());
 }
 @Test
 public boolean testIsBigger() {
    assertEquals(true,
                 babyDillo.isBigger(medDillo));
 }
}
```

ENVIRONMENT



HEAP (OBJECTS)

Calling isBigger is similar. There's one more quirk, though: the isBigger method takes in an argument, "other". Just like any other variables, "other" gets a name in isBigger's environment?
Where does it point? It points to the same object that we named when calling isBigger—in this case, it's medDillo.
Then, when isBigger is run, there are two names in the environment:
- "this" refers to the current Dillo object (babyDillo)
- "other" refers to the other object passed as an argument (medDillo)

(This is a pretty advanced example so it's okay if this is tough now—we'll spend a lot more time drawing out similar things later in the semester.)

Expanding our Zoo: Finally, what if we wanted to make a class to represent a Zoo, where a zoo holds multiple animals?

We could start by making a Zoo class that holds two Dillos, like this:

```
package src;
public class Zoo {
    public Dillo animal1;
    public Dillo animal2;

    public Zoo(Dillo ani1, Dillo ani2) {
        this.animal1 = ani1;
        this.animal2 = ani2;
    }
}
```

And we could create the Zoo like this:

```
public class AnimalsTest {

    Dillo babyDillo = new Dillo(5, false);
    Dillo medDillo = new Dillo(20, true);

    // A zoo with two Dillos
    Zoo myZoo = new Zoo(babyDillo, medDillo);

    // . . .
}
```

This is an example of how to make an object that holds other objects. This is a good start.

But our Zoo is pretty boring if it holds just DIllos. What if we wanted it to hold other types of Animals like the Boa (next page)?

How could we change our Zoo to hold different kinds of animals? We'd like to have use a type for animal1 or animal2 that says "Dillo or Boa"?
This is where interfaces can help--for more about how int, see the code example and the typed notes.

```
public class Dillo {
   public int length;
   public boolean isExtinct;

   Dillo(int l, boolean isE) {
        this.length = l;
        this.isExtinct = isE;
   }

   public boolean canShelter() {
     return this.length > 60 && this.isExtinct;
   }
}
```

```
class TestRunner {
   public static void main (...)
        runClasses(AnimalsTest.class)
}
```

```
public class Boa {
    public string name;
    public int length;
    public string eats;

    public Boa (String name,
                int length,
                String eats) {
       this.name = name ;
       this.length = length ;
       this.eats = eats ;
     }
 }
```

KNOWN CLASSES

```
public class Dillo {
  public int length;
  public boolean isExtinct;

  public Dillo(int len, boolean isExtinct) {
    this.length = len;
    this.isExtinct = isExtinct;
  }

  public boolean canShelter() {
    return this.length > 60 && this.isExtinct;
  }
  public boolean isBigger(Dillo other) {
    return this.length > other.length;
  }
}
```

```
class AnimalsTest {
    Dillo babyDillo = new Dillo(8, false);
    Dillo medDillo  = new Dillo(20, true);

 @Test
 public boolean testMakeDillo() {
     assertEquals(8, babyDillo.length);
 }
 @Test
 public boolean testCanShelter() {
   assertEquals(false,babyDillo.canShelter());
 }
 @Test
 public boolean testCanShelter2() {
   Dillo lgDillo  = new Dillo(100, false);
   assertEquals(true, lgDillo.canShelter());
 }
```

ENVIRONMENT

HEAP (OBJECTS)

KNOWN CLASSES

```
public class Dillo {
  public int length;
  public boolean isExtinct;

  public Dillo(int len, boolean isExtinct) {
    this.length = len;
    this.isExtinct = isExtinct;
  }

  public boolean canShelter() {
    return this.length > 60 && this.isExtinct;
  }
  public boolean isBigger(Dillo other) {
    return this.length > other.length;
  }
}
```

```
class AnimalsTest {
    Dillo babyDillo = new Dillo(8, false);
    Dillo medDillo  = new Dillo(20, true);

 @Test
 public boolean testMakeDillo() {
     assertEquals(8, babyDillo.length);
 }
 @Test
 public boolean testCanShelter() {
   assertEquals(false,babyDillo.canShelter());
 }

 @Test
 public boolean testIsBigger() {
   assertEquals(false,
                babyDillo.isBigger(medDillo));
 }
```

ENVIRONMENT

HEAP (OBJECTS)

```java
public class Dillo {
  public int length;
  public boolean isExtinct;

  public Dillo(int len, boolean isExtinct) {
    this.length = len;
    this.isExtinct = isExtinct;
  }

  public boolean canShelter() {
    return this.length > 60 && this.isExtinct;
  }

  public boolean isBigger(Dillo other) {
    return this.length > other.length;
  }
}
```

```java
public class Boa {
   public String name;
   public int length;
   public String eats;

   public Boa(String name, int length, String eats) {
       this.name = name;
       this.length = length;
       this.eats = eats;
   }









}
```