```
(define-struct dillo (length dead?))
(define baby-dillo (make-dillo 8 false))
(define adult-dillo (make-dillo 24 false))
(define huge-dead-dillo (make-dillo 65 true))
(define (can-shelter adillo)
  (and (dillo-dead? adillo)
       (> (dillo-length adillo) 60)))
(check-expect (can-shelter baby-dillo) false)
(check-expect (can-shelter huge-dead-dillo) true)
```

KNOWN CLASSES

OBJECTS

NAMED VALUES

EXPRESSION (impact on other areas are in red)

KNOWN CLASSES

```
class Dillo {
   int length;
   boolean isDead;

   Dillo (int length, boolean isDead) {
     this.length = length;
     this.isDead = isDead;
   }
}
```

A `class` expression adds to the known-classes area

OBJECTS

NAMED VALUES

EXPRESSION (impact on other areas are in red)

```
class Dillo {
   int length;
   ...
}
```

KNOWN CLASSES

```
class Dillo {
   int length;
   boolean isDead;

   Dillo (int length, boolean isDead) {
      this.length = length;
      this.isDead = isDead;
   }
}
```

OBJECTS

Dillo
length = 5
isDead = false

A `new` expression adds to the objects area

NAMED VALUES

EXPRESSION (impact on other areas are in red)

new Dillo (5, false)

KNOWN CLASSES

```
class Dillo {
   int length;
   boolean isDead;

   Dillo (int length, boolean isDead) {
      this.length = length;
      this.isDead = isDead;
   }
}
```

OBJECTS

Dillo
length = 5
isDead = false

Dillo
length = 3
isDead = true

Note there is no way to access the live dillo (but it still exists)

An = expression yields a named value

NAMED VALUES

deadDillo

EXPRESSION (impact on other areas are in red)

deadDillo = new Dillo (3, true)

KNOWN CLASSES

```
class Dillo {
   int length;
   boolean isDead;

   Dillo (int length, boolean isDead) {
      this.length = length;
      this.isDead = isDead;
   }
}
```

OBJECTS

Dillo
length = 5
isDead = false

Dillo
length = 3
isDead = true

Dillo
length = 3
isDead = true

It is fine to have multiple objects with the same field values.

NAMED VALUES

deadDillo

anotherDeadDillo

EXPRESSION (impact on other areas are in red)

anotherDeadDillo = new Dillo (3, true)

```
(define-struct boa (name length eats))

; a Boa is a (make-boa String Number String)
; interp: a boa constrictor where
;             name is the boa's name
;             length is the boa's length
;             eats is the boa's favorite food
```

```
;; likes-same-food?: Boa Boa → Boolean
;; produces true if both boas have the same
;; favorite food

(define (likes-same-food? boa1 boa2)
     (string=? (boa-eats boa1) (boa-eats boa2)))
```

Given the above Racket function, write the heading for an analogous Java method. Call your method likesSameFood. You do NOT have to write the body of the method (although you may write the method as a stub if you wish). Just figure out what the heading would be so that you can write test cases.

Write a set of Junit test cases for likesSameFood.