(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

NAMED VALUES

<u>OBJECTS</u>

•••

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;

<u>OBJECTS</u>

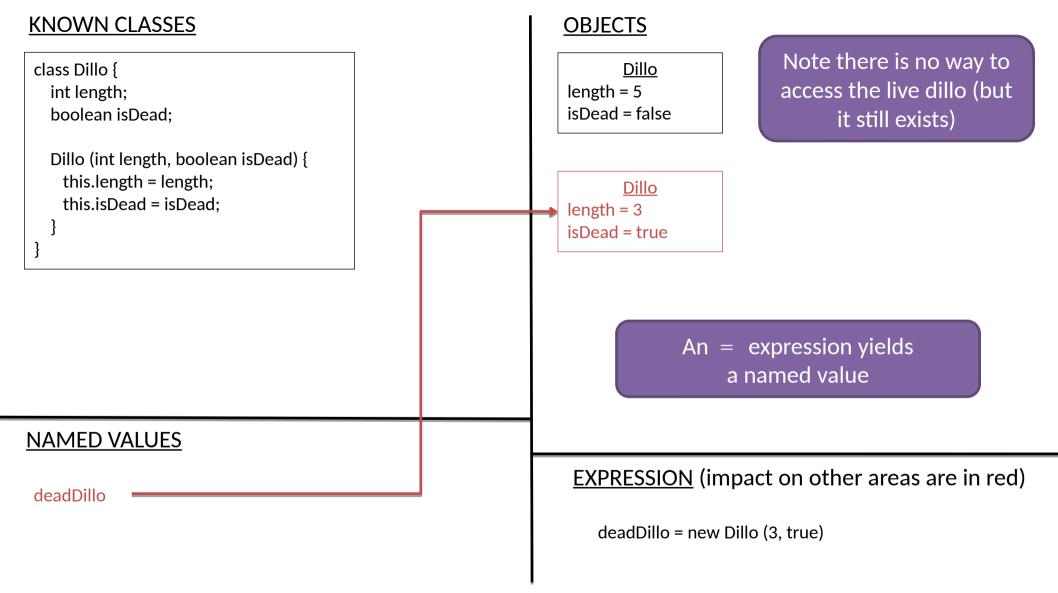
<u>Dillo</u> length = 5 isDead = false

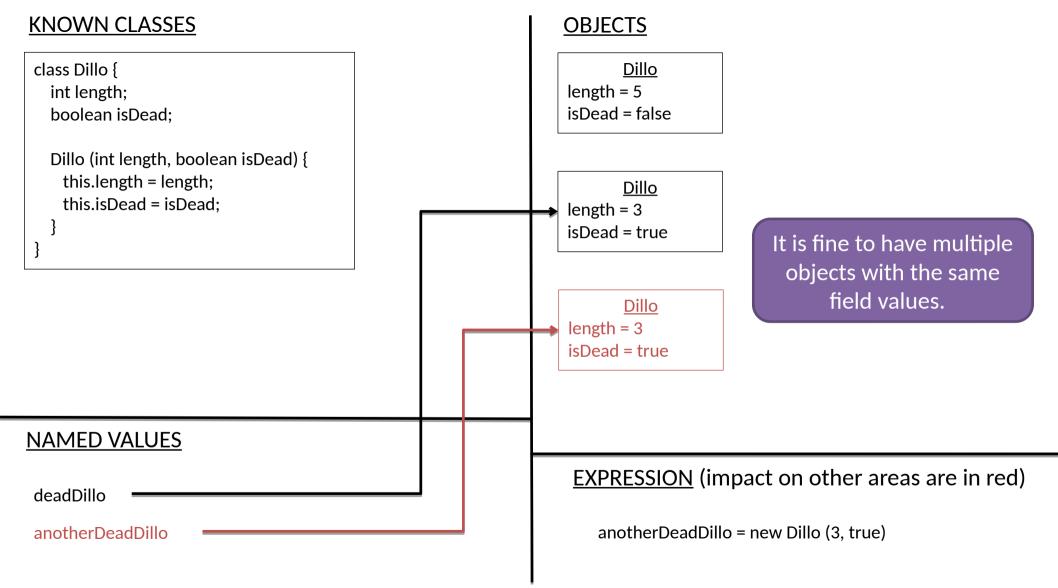
A new expression adds to the objects area

NAMED VALUES

EXPRESSION (impact on other areas are in red)

new Dillo (5, false)





(define-struct boa (name length eats))

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

(define (likes-same-food? boal 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.