Piazza

- Ask your questions there not through email

CS111
Introduction to Computer Science

Fall 2015

- Programming Process
- Printing Statements
- Fahrenheit to Celsius

The Programming Process

1. Problem Analysis: understand what the program should do
   - Inputs, outputs, error conditions
2. Algorithm Construction: choose a sequence of action to achieve the goal
   - We'll use pseudocode
3. Coding
   - Use a programming language (Java) to express the actions
4. Testing: does the program work correctly?
   - Test case constructions, debugging

Program: Printing Data

1. Analysis
   - No inputs, outputs, errors
2. Algorithm Construction

```
print “CS111 Introduction to Computer Science”
print “Fall 2015”
```

3. Coding: skip for now
4. Testing: only one test case needed, check if output matches expected

Statements in double quotes printed verbatim

Statements in Java have different syntax

See Announcement.java

Program: Fahrenheit To Celsius Conversion

1. Analysis
   - Input: temperature in Fahrenheit
   - Output: temperature in Celsius
   - Error conditions: input less than -467.67 (absolute zero)
2. Algorithm Construction

```
print "Please, enter the temperature in Fahrenheit"
tempF ← read number
tempC ← (tempF – 32) / 9 * 5
print tempC
```

When execution gets here, it waits for user to enter data, then reads the value entered, and stores it into a memory location called tempF

The value in the memory location tempC is retrieved and printed. Retrieval does NOT wipe out the values – they are still there, and can be reused as many times as needed.

The right hand side is computed, using value retrieved from the memory location tempF, the result is stored in a memory location called tempC

Program: Fahrenheit To Celsius Conversion

1. Analysis
   - Input: temperature in Fahrenheit
   - Output: temperature in Celsius
   - Error conditions: input less than -467.67 (absolute zero)

2. Algorithm Construction

```
print "Please, enter the temperature in Fahrenheit"
tempF ← read number
tempC ← (tempF – 32) / 9 * 5
print tempC
```

4. Testing

Input	Expected Output	Output
32	0	0
100	37.78	37.78
-600	error	-351.11

Program: Fahrenheit To Celsius Conversion with Error Checking

1. Analysis: same as before
2. Algorithm Construction

```
print “Please, enter the temperature in Fahrenheit”
tempF ⟵ read number
if tempF < -459.67
    print “Not a valid temperature”
    halt
tempC ⟵ (tempF – 32) / 9 * 5
print tempC
```

Indentation expresses conditionality

4. Testing

Input	Expected Output	Output
32	0	0
100	37.78	37.78
-600	error	error

When making a code change, run all tests again.

Same Problem, Different Solutions

```
print “Please, enter the temperature in Fahrenheit”
tempF ⟵ read number
if tempF < -459.67
    print “Not a valid temperature”
    halt
tempC ⟵ (tempF – 32) / 9 * 5
print tempC
```

1-way decision

See F2C.java

```
print “Please, enter the temperature in Fahrenheit”
tempF ⟵ read number
if tempF < -459.67
    print “Not a valid temperature”
else
    tempC ⟵ (tempF – 32) / 9 * 5
    print tempC
```

2-way decision

See F2C_v2.java

Multi-way decisions

- 1-way
  - `if…`

```
if true
    - operation
```

- 2-way
  - `if… else…`

```
if true
    - operation
else
    -operation
```

- 3-way (cascaded)
  - `if… else {if… else…}`

```
if true
    - operation
else
    if true
        -operation
    else
        -operation
```

- 4-way (nested)
  - `if… {if… else…} else… {if… else…}`

```
if true
    if true
        -operation
    else
        -operation
else
    if true
        -operation
    else
        -operation
```

CS111
Introduction to Computer Science

Fall 2015

- Booleans
- Amount of Daylight
- Calculator
- Testing

Booleans

- The result of a comparison
- Can be used for combining tests
  – Hour value is invalid if hour < 0 or hour > 12
- A boolean value represents the result of a testing condition
- There are only two values
  – true or false
- Like other data booleans are stored as variables

Boolean values as data

- Since true and false are data value, we can:
  - Store them in variables: xBig ← x > y
  - Read them from input and print as output
  - Create them with operations

    x<y   x>y   x<=y   x>=y   x==y   x!=y
  - Use them as operand for boolean operators

    xBig || yBig   xBig && yBig   !xBig

Boolean Operators: or, and, not

A	B	A || B
false	false	**false**
false	true	**true**
true	false	**true**
true	true	**true**

A	B	A && B
false	false	**false**
false	true	**false**
true	false	**false**
true	true	**true**

A	!A
false	**true**
true	**false**

Boolean examples

$x \leftarrow 10$, $y \leftarrow 20$

- x < y
- x >= y
- x >= 10
- x == 9 || x == 10
- x == 9 && x == 10
- x > y || (x == 10 && y < 7)

Program: Amount of daylight

- Computes the amount of daylight from sunrise to sunset
- Example interaction:

```
Enter sunrise hour: 6
Enter sunrise minute: 30
Enter true for morning sunrise, false otherwise: true
Enter sunset hour: 8
Enter sunset minute: 45
Enter true for morning sunset, false otherwise: false
Amount of daylight  is 14 hours and 15 minutes.
```

Program: Amount of daylight

1. Analysis
   - Input: sunrise hour (integer), sunrise minute (integer), am/pm (boolean), sunset hour (integer), sunset minute (integer), am/pm (boolean)
   - Output: amount of daylight in hours (integer), minutes (integer)
   - Error conditions: input out of range 0-12 or 0-59
2. Algorithm Construction

cont.

```
print “Enter sunrise hour:”
riseHour = read integer
if riseHour < 0 or riseHour > 12
   print "sunrise hour not valid"
   halt
print “Enter sunrise minute:”
riseMinute = read integer
if riseMinute < 0 or riseMinute > 59
   print "sunrise minute not valid"
   halt
print “Enter sunrise am/pm”
riseAm = read boolean
```

```
print “Enter sunset hour:”
if setHour < 0 or setHour > 23
   print "sunset hour not valid"
   halt
print “Enter sunset minute:”
setMinute <- read integer
if setMinute < 0 or setMinute > 59
   print "sunset minute not valid"
   halt
print “Enter sunset am/pm”
setAm = read boolean
```

Program: Amount of daylight

cont.

```
if riseAm is true and riseHour is 12
     riseHour = 0
if riseAm is false and riseHour is not 12
     riseHour = riseHour + 12

if setAm is true and setHour is 12
     setHour = 0
if setAm is false and setHour is not 12
     setHour = setHour + 12

if setHour < riseHour or  (setHour == riseHour and setMin < riseMin)
     print "sunrise must be before sunset"
     halt

dayHour = setHour - riseHour
dayMin = setMin - riseMin
if dayMin < 0
     dayHour = dayHour - 1
     dayMin = dayMin + 60
print dayHour
print dayMin
```

Convert sunrise hour to 24 hour time

Convert sunset hour to 24 hour time

See DaylightTime.java

Program: Amount of daylight

4. Testing

						Expected output	
riseHour	**riseMin**	**am**	**setHour**	**setMin**	**am**	**dayHour**	**dayMin**
-1						error	
24						error	
6	-1					error	
6	60					error	
6	30	1	-1			error	
6	30	1	24			error	
6	30	1	5	-1		error	
6	30	1	5	60		error	
6	30	1	7	30	1	error	
6	30	1	8	45	0	14	15
6	30	1	8	20	0	13	50
6	30	1	6	20	1	error	
6	30	1	5	30	1	error	

Program: Calculator

- Should do 4 basic arithmetic operations.
- Example interaction:

```
Enter first number: 2
Enter second number: 3
Enter [1]addition, [2]subtraction, [3]multiplication,
[4]division: 3
Result: 2*3 = 6
```

See Calculator.java

Program: Calculator

1. Analysis
   - inputs: operand1 and operand2 (real numbers), operator (integer, choice from menu)
   - outputs: result (real number)
   - errors: invalid operation, divide by 0

Program: Calculator

2. Algorithm

```
print "Enter first number:"
op1 = read number
print "Enter second number:"
op2 = read number
print "Enter [1]addition, [2]subtraction, [3]multiplication, [4]division:"
choice = read integer
if choice == 1
    print op1 + op2
else if choice == 2
    print op1 – op2
else if choice == 3
    print op1 * op2
else if choice == 4
    if op2 != 0
        print op1/op2
    else
        print "can't divide by zero"
else
    print "invalid menu selection"
```

Program: Calculator

3. Testing

First Number	Second Number	Operation	Expected Output
2	4	1	6
2	4	2	-2
2	4	3	8
2	4	4	0.5
2	0	4	Error
2	4	7	Error

Testing Data

- Each part of the program must be executed by some set of test data
- Each if condition (decision) should be made both true and false at different times

CS111
Introduction to Computer Science

Fall 2015

- Loops
- Add Numbers
- Averaging Numbers
- Maximum of a Sequence of Numbers

Loops: Repeating things

- Loops allows repetition inside a program

```
loop condition
    operations
```

- Test the condition
  - if false, go to the end of the loop and continue on
  - otherwise, do the operation and go back to the start of the loop (do the test again)
- Repeatedly ask for good input using the *while* loop

```
riseHour = read integer
while riseHour < 0 or riseHour > 12
    print “invalid hour,  enter again:”
    riseHour = read integer
```

See DaylightTime_v2.java

Program: Letter Grade for Multiple Students

- Assigning a letter grade for more than one student
- Example interaction

```
Enter number of students: 2
Enter grade: 98.2
Letter grade is A
Enter grade: 32
Letter grade is F
```

See LetterGrade.java
LetterGrade_v2.java
LetterGrade_v3.java

Program: Adding numbers

- Example interaction

```
Enter terminating value: -1
Enter next number: 3
Enter next number: 5
Enter next number: -1
Sum is: 8
```

See SequenceSum.java

Adding numbers: analysis

- Inputs: terminator, the sequence of numbers
  - number, number, ..., -1
- Outputs:
  - Sum of numbers (not the terminator)
- Errors: none
- Test data
  - blackbox: enumerate before writing the code
  - coverage: enumerate after writing the code

Input	Expected output	
-1	0	blackbox
3 -1	3	blackbox
8 15 3 -1	26	blackbox

Adding numbers: algorithm

```
print "Enter terminating value:"
terminator = read number
sum = 0
do
   print "Enter next number:"
   num = read number
   if num != terminator
      sum = sum + num
while num != terminator
print sum
```

Only one value is entered at a time, how can we add them all? Summary variable to hold a running total.

Summary variable: initialized before loop

Summary variable: Updated inside the loop

Program: Averaging numbers

- Example interaction

```
Enter terminating value: -1
Enter next number: 3
Enter next number: 5
Enter next number: -1
Average is: 4
```

See SequenceAverage.java

Averaging numbers: analysis

- Inputs: terminator, the sequence of numbers
  - number, number, ..., -1
- Outputs:
  - Average of numbers (not the terminator)
- Errors: zero length sequence
- Test data

Input	Expected output	
-1	Error	blackbox
3 -1	3	blackbox
8 4 3 -1	5	blackbox

Averaging numbers: algorithm

```
print “Enter terminating value:”
terminator = read number
sum = 0
count = 0
do
    print “Enter next number:”
    num = read number
    if num != terminator
        sum = sum + num
        count = count + 1
while num != terminator
if count == 0
    print “Zero length sequence”
    halt
print sum/count
```

Now we need both sum and a count of the numbers to average.

Program: Maximum of a Sequence

- Example interaction

```
Enter terminating value: -1
Enter next number: 3
Enter next number: 5
Enter next number: -1
Maximum is: 5
```

See SequenceMax.java

Maximum of a Sequence: analysis

- Inputs: terminator, the sequence of numbers
  - number, number, ..., -1
- Outputs:
  - Maximum number (not the terminator)
- Errors: zero length sequence
- Test data

Input	Expected output	
-1	Error	blackbox
20 10 -1	20	blackbox

Maximum of a Sequence: algorithm

```
print “Enter terminating value:”
terminator = read number
print “Enter next number: “
num = read number
if num == terminator
    print “no maximum”
    halt
max = num
do
    print “Enter next number:”
    num = read number
    if num != terminator and num > max
            max = num
while num != terminator
print max
```

Input	Expected output
-1	Error
20 10 -1	20
5 30 -1	30 coverage

Makes num > max true

CS111
Introduction to Computer Science

Fall 2015

- Counted Loops
- Nested Loops
- Break and Continue

Counted Loops

- An alternative to marking the end of the input
- How many numbers are in sequence?
  - input length of sequence of numbers to read
  - count the numbers as you read them
  - repeat as long as you have not read enough
- Example Interaction:

```
How many numbers to sum: 2
    Enter next number: 5
    Enter next number: 1
    Sum is: 6
```

See CountedSum.java

Adding numbers: analysis

- Inputs: sequence size (integer), sequence of number
  - n, $number_0$, $number_1$, ..., $number_n$
- Outputs:
  - Sum of numbers
- Errors: negative sequence size
- Test data

Size	Sequence	Expected output	
3	5, 8, 12	25	blackbox
-1		error	blackbox

Adding numbers: algorithm

```
print “How many numbers to sum:”
size = read number
if size < 0
   print “sequence size cannot be negative
   halt
count = 0
sum = 0
while count < size
   print “Enter next number:”
   num = read number
   sum = sum + num
   count = count + 1
print sum
```

count will store how many numbers we have read

Repeat as long as you have not read enough

Increment count by 1 when you read a number

Size	Sequence	Expected output	
3	5, 8, 12	25	blackbox
-1		error	blackbox
0		0	coverage

Counting to n

0, 1, 2, ..., n-1

```
int n = IO.readInt();
count = 0;
while (count < n) {
   count = count+1;
}
```

1, 2, 3, ..., n

```
int n = IO.readInt();
count = 1;
while (count <= n) {
   count = count+1;
}
```

Loops so far

- do...while
  - operations are executed at least once
  - then condition is tested

```
do
    operations
while condition
```

- while...
  - operations are only executed if condition is true

```
while condition
    operations
```

Another kind of Loop: for

- The for loop can be used when the number of iterations is know before entering the loop.

```
for ( ⟨initialization ⟩; ⟨continuation-condition ⟩; ⟨update ⟩ ) {
        ⟨statements ⟩
}
```

- How would our Adding Numbers look like with a `for` loop?

See CountedSum_v2.java

`for` loop

```
for (int count = 0; count < 5; count = count + 1) {
    IO.outputIntAnswer(count);
}

int count = 0;
while (count < 5) {
    IO.outputIntAnswer(count);
    count = count + 1;
}
```

Nested Loops

- When the operations of a loop contain another loop

```
loop condition 1
    operations
    loop condition 2
        operations
```

Problem: Build a multiplication table

- Build a multiplication table of n x m
- Example Interaction:

Enter number of rows = 3

Enter number of columns = 5

Result:

	1	2	3	4	5
1	1	2	3	4	5
2	2	4	6	8	10
3	3	6	9	12	15

See MultiplicationTable.java
MultiplicationTable_v2.java

Build multiplication table: analysis

- Inputs: number of rows (integer), number of columns (integer)
- Output: table (text)
- Errors: negative/zero number of rows or columns

Build multiplication table: algorithm

```
print “Enter number of columns:”
numCols= read number
if numCols <= 0
    print “Invalid number of columns”
    halt
print “Enter number of rows:”
numRows= read number
if numRows <= 0
    print “Invalid number of rows”
    halt
row = 1
while row <= numRows
    col = 1
    while col <= numCols
        print row * col
        col = col + 1
    print new line
    row = row + 1
```

Outer Loop

For each iteration of the outer loop all iterations of the inner loop are executed.

Build multiplication table: algorithm

- Suppose we want to compute only the values bellow the diagonal

	1	2	3	4	5
1					
2	2				
3	3	6			

```
row = 1
while row <= numRows
    col = 1
    while col < row
        print row * col
        col = col + 1
    print new line
    row = row + 1
```

Outer Loop: walks over the rows.

Inner loop: walks over columns.
Restrict inner loop!

What does this print?

```
for(int j = 1; j <= 3; j++) {
   System.out.println(j);
   for(int k = 11; k<=12; k++) {
      System.out.println(k);
   }
}
```

```
1
11
12
2
11
12
3
11
12
```

Break and Continue

- Break
  - exit current loop
- Continue
  - skip the rest of current iteration
  - go directly to test

What does this print?

```
for(int j = 1; j <= 3; j++) {
   if (j == 1) { continue; }
   System.out.println(j);
   for(int k = 11; k<=12; k++) {
      System.out.println(k);
   }
}
```

```
2
11
12
3
11
12
```

What does this print?

```
for(int j = 1; j <= 3; j++) {
   if (j == 2) {break;}
   System.out.println(j);
   for(int k = 11; k<=12; k++) {
      System.out.println(k);
   }
}
```

```
1
11
12
```

What does this print?

```
for(int j = 1; j <= 3; j++) {
   System.out.println(j);
   for(int k = 11; k<=12; k++) {
      if (k==11) {break;}
      System.out.println(k);
   }
}
```

```
1
2
3
```

Increment/Decrement

Very Common	Shorthand
foo = foo + x;	foo += x;
foo = foo + 1;	foo++;
foo = foo – x;	foo -= x;
foo = foo – 1;	foo--;

Scope of Variables

- A variable lives
  - from its declaration
  - to the end of the innermost block the declaration is in

```
if(...) {
    if (...) {
        int x;
        while (...) {
        ...
        }
    }
}
```

See Scope.java

CS111
Introduction to Computer Science

Fall 2015

- Switch
- Methods

Switch Statement

- Tests the value of an expression, and depending on that value, jumps directly to some location within the switch statement.

```
switch expression
    case constant-1:
        operations
        break;
    case constant-2:
        operations
        break;
    ...
    case constant-2:
        operations;
        break;
    default:
        operations;
```

See Calculator_v3.java

Midterm

- Midterm Oct 12 (Monday) 9:40pm – 11pm
  - SC 135 (Scott Hall on College Ave)

- Practice problems posted
  - Sakai – Resources – Practice Problems

Subrotines

- A way to break a complex program into smaller pieces.
- A subrotine consists of:
  - a set of operations for carrying out a certain task that can be called from different places in the program
  - name (how the subrotine is known)
  - arguments (data for each call)
  - return value (result computed by the subrotine)

Subrotines

- A subrotine is sometimes called a black box because we don’t need to see what’s inside to interact with it. All we need is to know is its specification (interface):
  - arguments it expects and the type of value it returns

double pow(double base, double exponent);

Java Subrotines: Methods

- In Java every subrotine must be declared inside a some class
- static and non-static subrotines
  - static: belongs to the class
  - non-static: belong to the object (we'll learn later)

- From now on we'll refer to subrotines as Methods

Method Definition

- Every method must be defined somewhere (inside a class in Java)

```
<modifiers><return-type><name>(<parameters>)
{
   <statements>
}
```

```
public static double pow(double base, double exponent){}
```

```
public static int readInt(){}
```

```
public static void outputIntAnswer(int i){}
```

Method Definition

- Define a static method called factorial that receives an integer value as argument. It computes and returns the factorial of that value.

```
public static int factorial(int n){
   int result = 1;
   for (int count = 1; count <=n; count++){
      result *= count;
    }
   return result;
}
```

See the factorial method in Methods.java

Calling Static Methods

- When it returns a value

$$\texttt{double x = }\underbrace{\texttt{Math}}_{\text{What class to look for definition in}}\texttt{ . }\underbrace{\texttt{pow}}_{\text{Method name}}\texttt{ }\underbrace{\texttt{(2,3)}}_{\text{Argument list}}\texttt{;}$$

Calling Static Methods

- When it returns a value

$$\texttt{double x =}\ \underbrace{\phantom{\texttt{Math.}}}_{\text{Look in current class for definition}}\ \underbrace{\texttt{pow}}_{\text{Method name}}\ \underbrace{\texttt{(2,3)}}_{\text{Argument list}}\texttt{;}$$

Calling Static Methods

- When it does NOT return a value (void)

$$\underbrace{\texttt{IO}}_{\text{What class to look for definition in}}\ \texttt{.}\ \underbrace{\texttt{outputIntAnswer}}_{\text{Method name}}\ \underbrace{\texttt{(2)}}_{\text{Argument list}}\texttt{;}$$

Calling Static Methods

```
int x = factorial (3);
```

See the factorial method in Methods.java

Return Statement

- Returns to the caller
  - returns to where it was called from
- `return <expression>`;
  - the type of `<expression>` must match the return type specified in the definition of the function

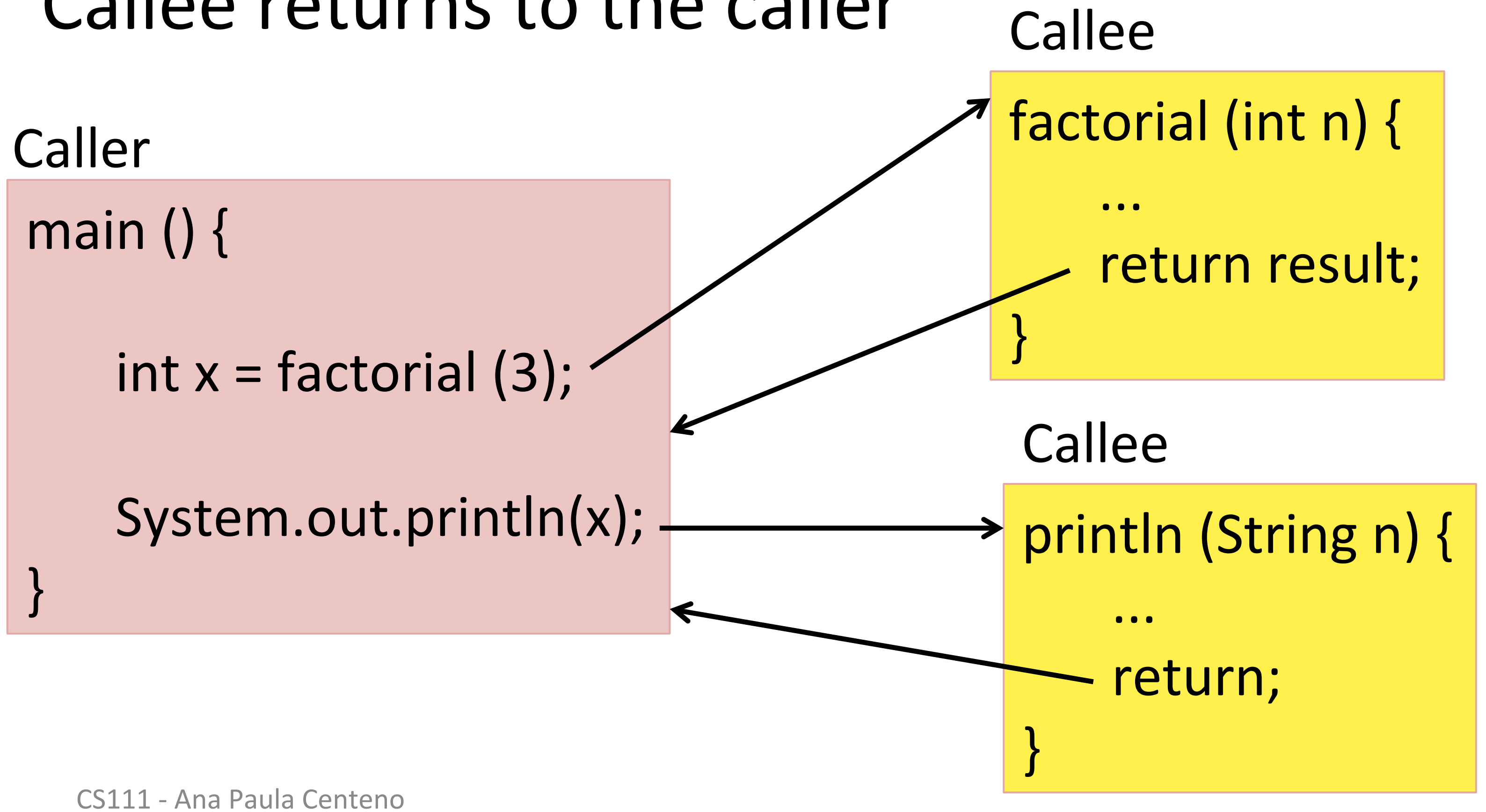
```
return result;
```

- `return;`
  - `void return type`

See the factorial method in Methods.java

Caller and Callee

- Caller calls the callee
- Callee returns to the caller



```
main () {

      int x = factorial (3);

      System.out.println(x);
}
```

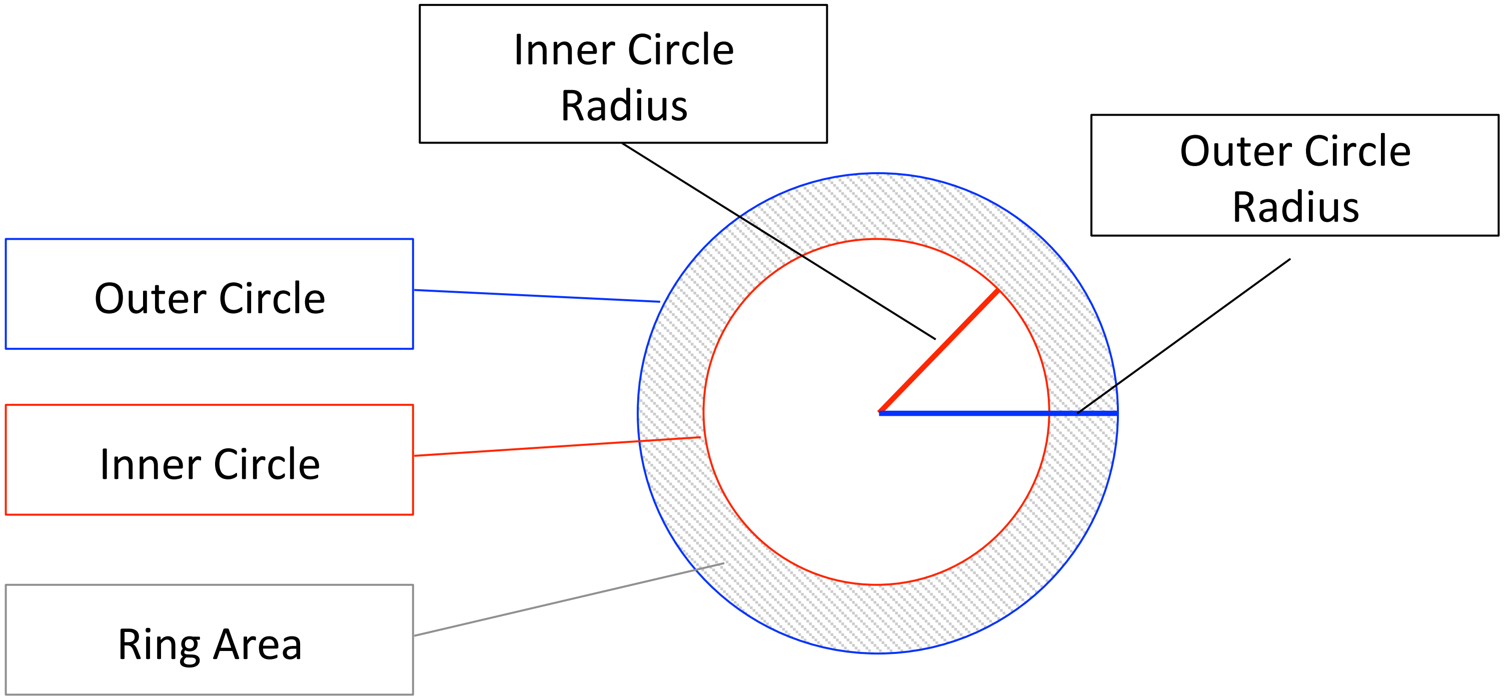
```
factorial (int n) {
      ...
      return result;
}
```

```
println (String n) {
      ...
      return;
}
```

Frames

- When a method is called (invoked), the JVM creates space to store information about that call:
  - parameters
  - local variables: declared inside the method
  - temporary variables
  - where to return to
- When a method returns, the activation record for that call is destroyed
- Also called invocation or activation records

RingArea


Inner Circle Radius
Outer Circle Radius
Outer Circle
Inner Circle
Ring Area

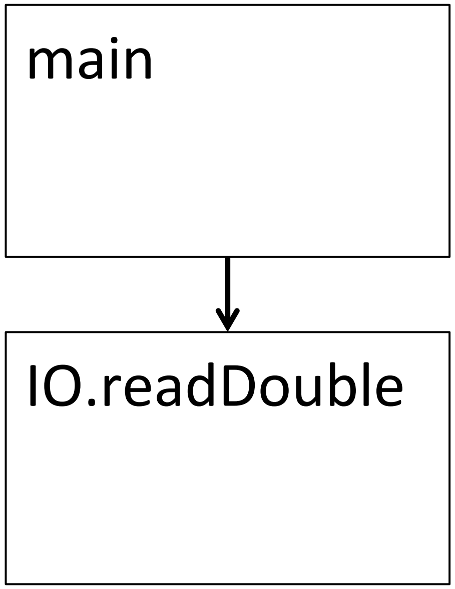
Frames

- RingArea.java

Call Sequence

Frames

- RingArea.java

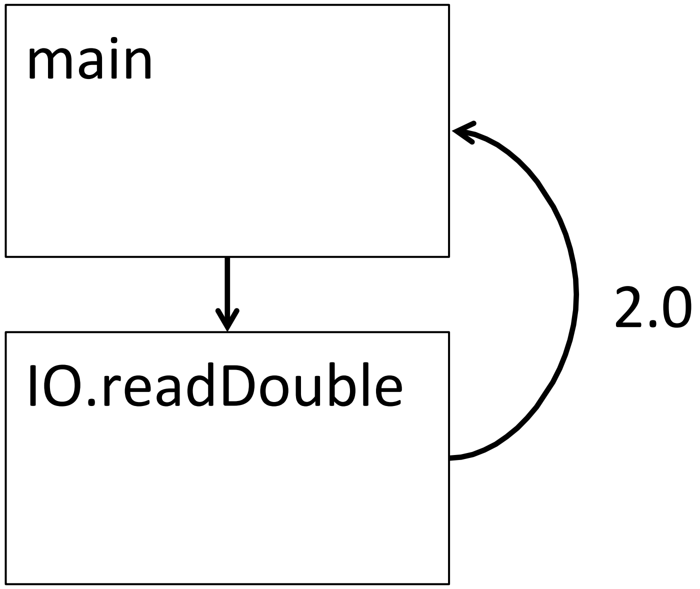


Call Sequence

```
IO.readDouble();
```

Frames

- RingArea.java



Call Sequence

```
IO.readDouble();
```

Frames

- RingArea.java

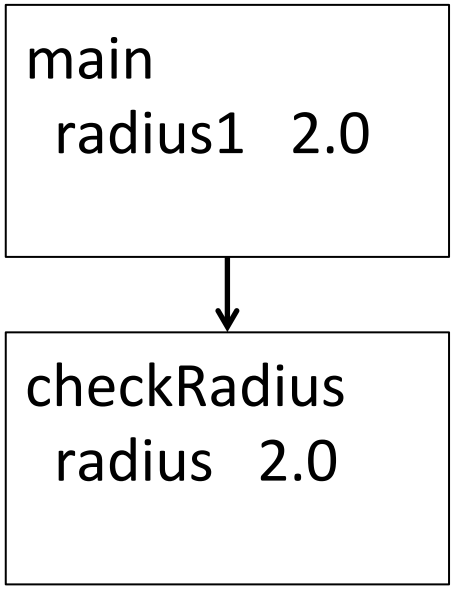
```
main
  radius1   2.0
```

Call Sequence

```
IO.readDouble();
```

Frames

- RingArea.java

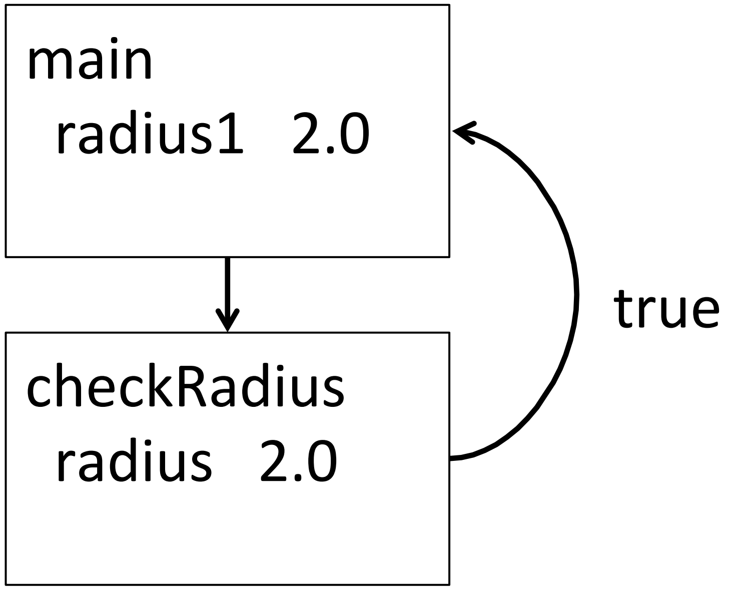


Call Sequence

```
IO.readDouble();
checkRadius(2.0);
```

Frames

- RingArea.java



Call Sequence

```
IO.readDouble();
checkRadius(2.0);
```

Frames

- RingArea.java

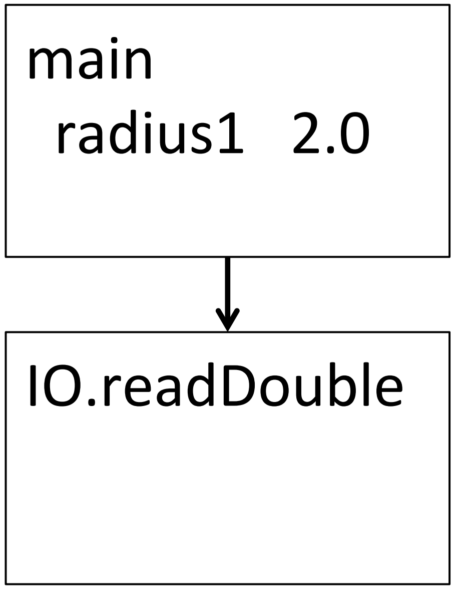
```
main
  radius1   2.0
```

Call Sequence

```
IO.readDouble();
checkRadius(2.0);
```

Frames

- RingArea.java

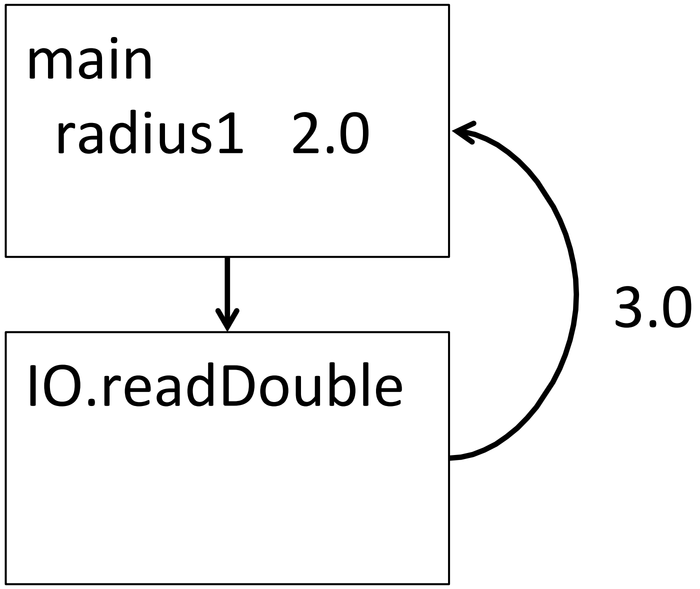


Call Sequence

```
IO.readDouble();
checkRadius(2.0);
IO.readDouble();
```

Frames

- RingArea.java



Call Sequence

```
IO.readDouble();
checkRadius(2.0);
IO.readDouble();
```

Frames

- RingArea.java

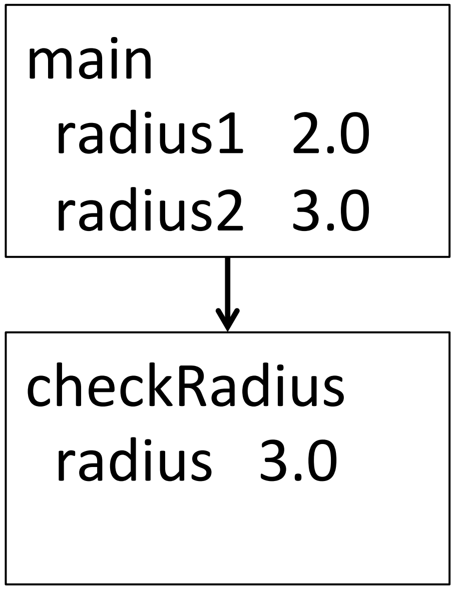
main	
radius1	2.0
radius2	3.0

Call Sequence

```
IO.readDouble();
checkRadius(2.0);
IO.readDouble();
```

Frames

- RingArea.java

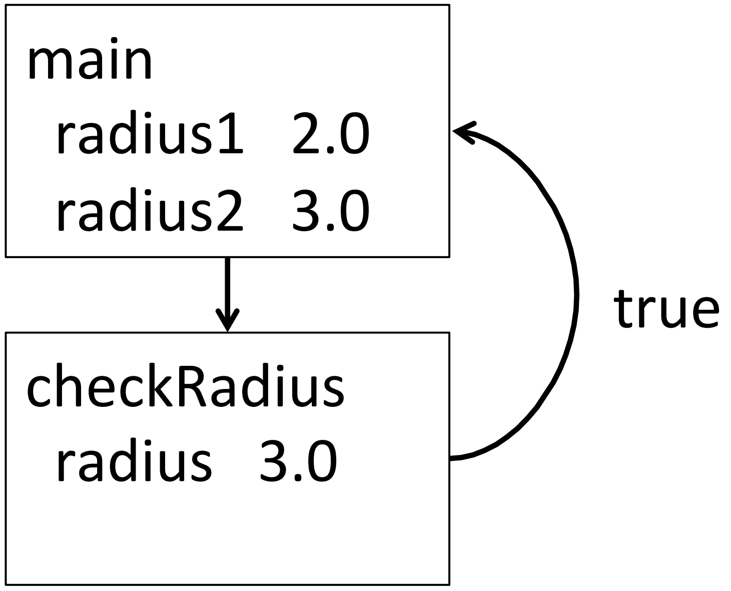


Call Sequence

```
IO.readDouble();
checkRadius(2.0);
IO.readDouble();
checkRadius(3.0);
```

Frames

- RingArea.java



Call Sequence

```
IO.readDouble();
checkRadius(2.0);
IO.readDouble();
checkRadius(3.0);
```

Frames

- RingArea.java

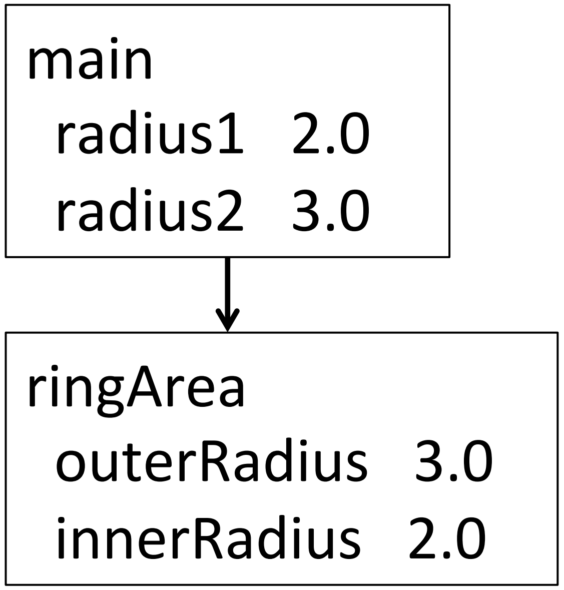
```
main
  radius1   2.0
  radius2   3.0
```

Call Sequence

```
IO.readDouble();
checkRadius(2.0);
IO.readDouble();
checkRadius(3.0);
```

Frames

- RingArea.java

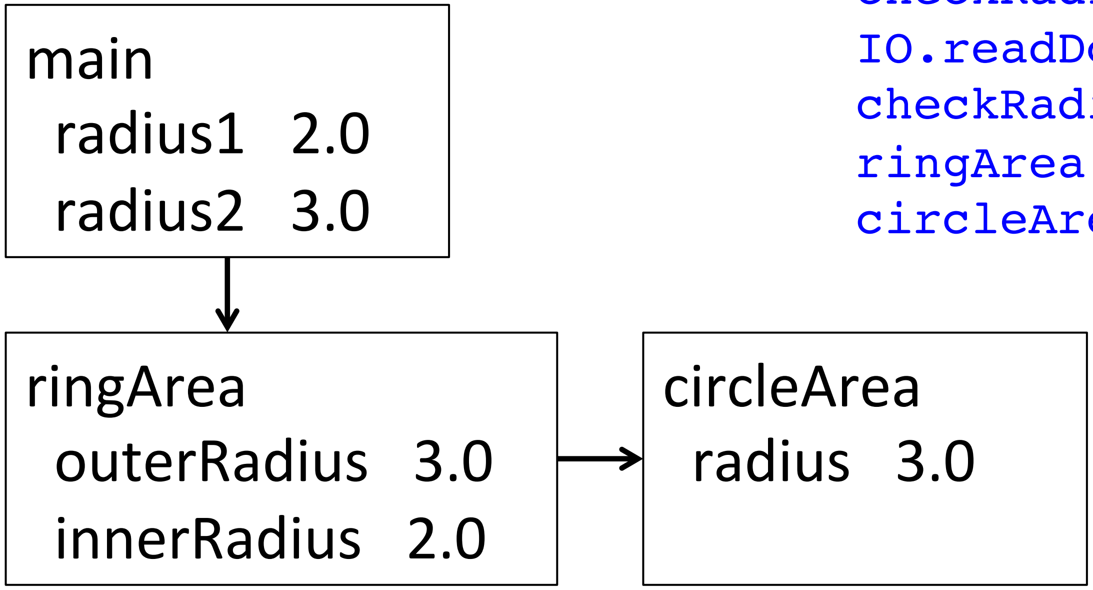


Call Sequence

```
IO.readDouble();
checkRadius(2.0);
IO.readDouble();
checkRadius(3.0);
ringArea(3.0, 2.0);
```

Frames

- RingArea.java

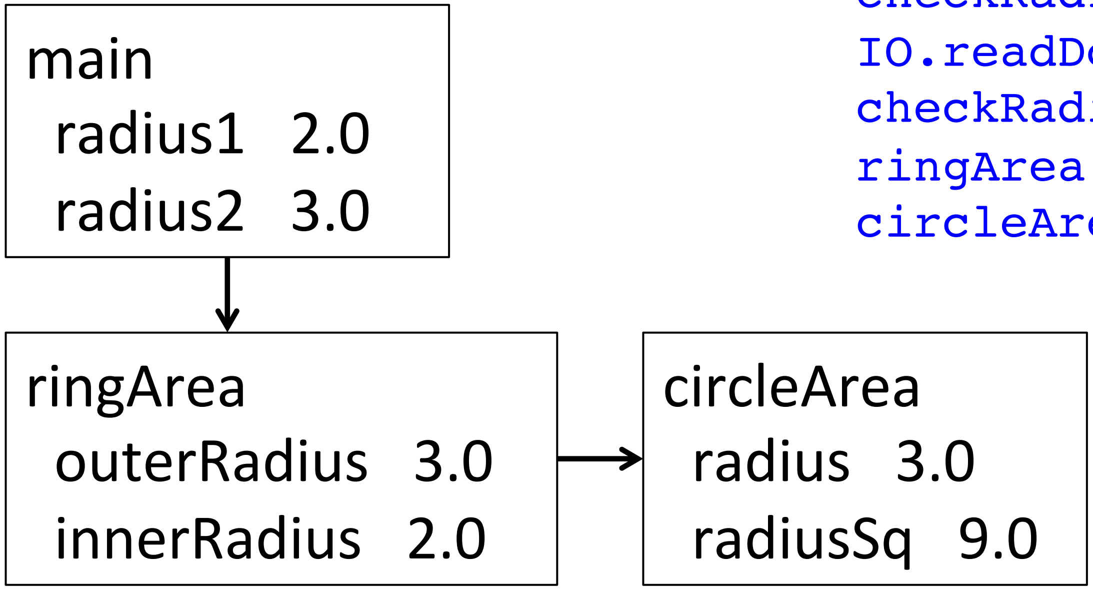


Call Sequence

```
IO.readDouble();
checkRadius(2.0);
IO.readDouble();
checkRadius(3.0);
ringArea(3.0, 2.0);
circleArea(3.0);
```

Frames

- RingArea.java

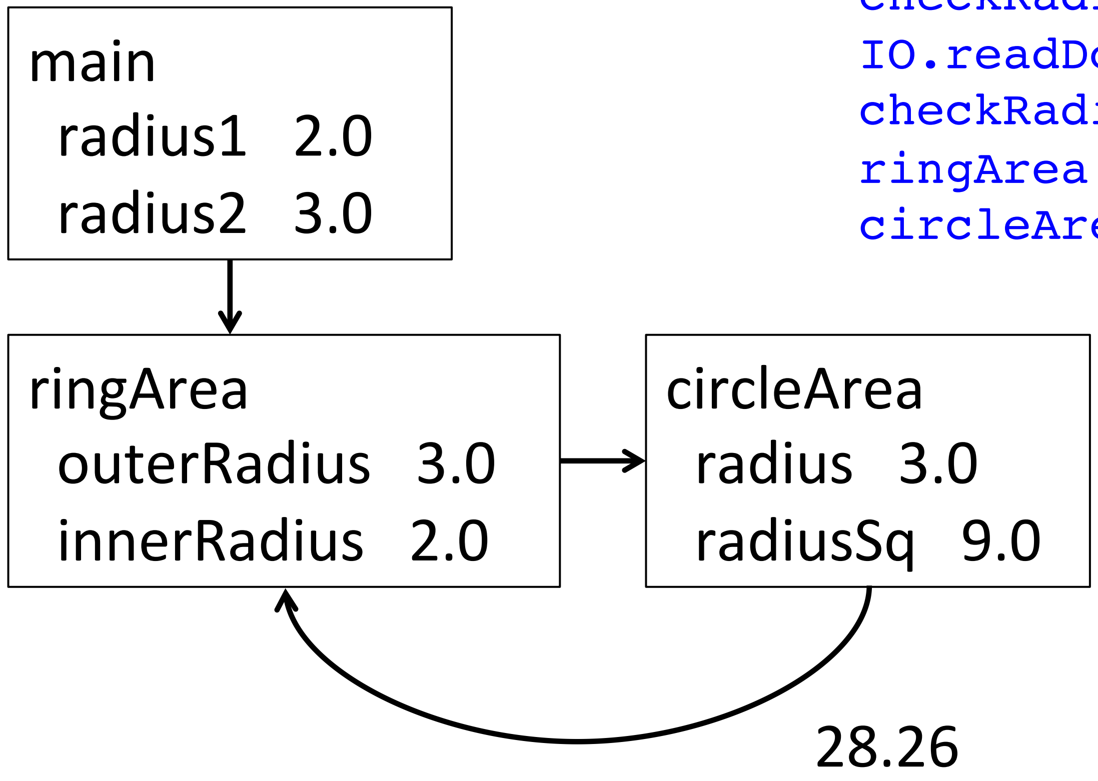


Call Sequence

```
IO.readDouble();
checkRadius(2.0);
IO.readDouble();
checkRadius(3.0);
ringArea(3.0, 2.0);
circleArea(3.0);
```

Frames

- RingArea.java

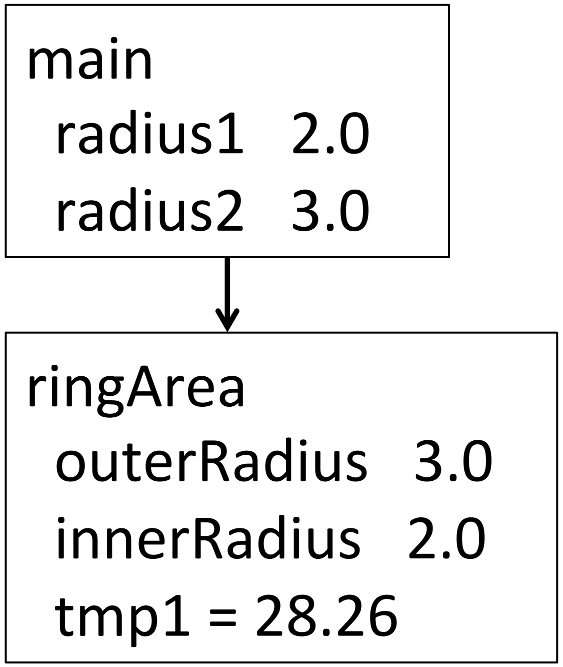


Call Sequence

```
IO.readDouble();
checkRadius(2.0);
IO.readDouble();
checkRadius(3.0);
ringArea(3.0, 2.0);
circleArea(3.0);
```

Frames

- RingArea.java

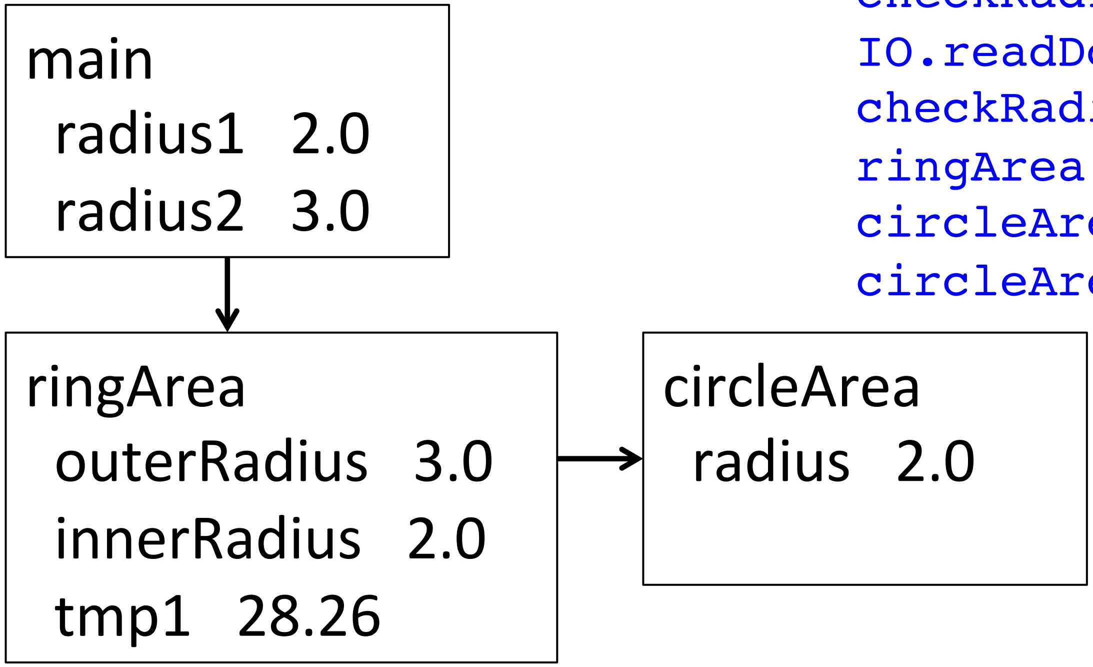


Call Sequence

```
IO.readDouble();
checkRadius(2.0);
IO.readDouble();
checkRadius(3.0);
ringArea(3.0, 2.0);
circleArea(3.0);
```

Frames

- RingArea.java

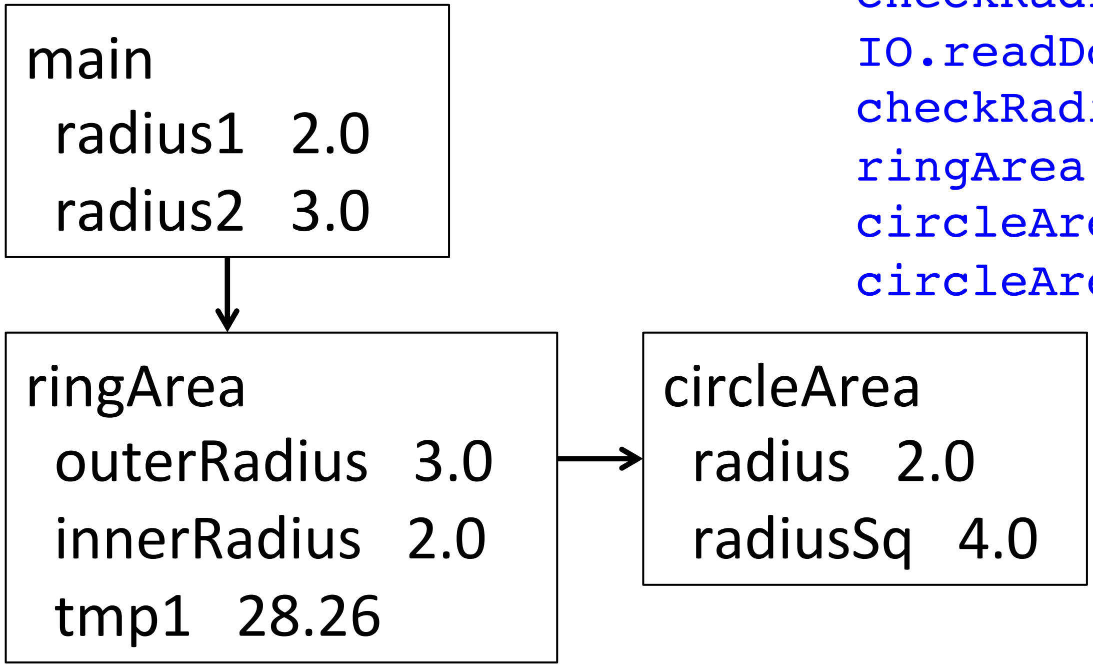


Call Sequence

```
IO.readDouble();
checkRadius(2.0);
IO.readDouble();
checkRadius(3.0);
ringArea(3.0, 2.0);
circleArea(3.0);
circleAread(2.0);
```

Frames

- RingArea.java

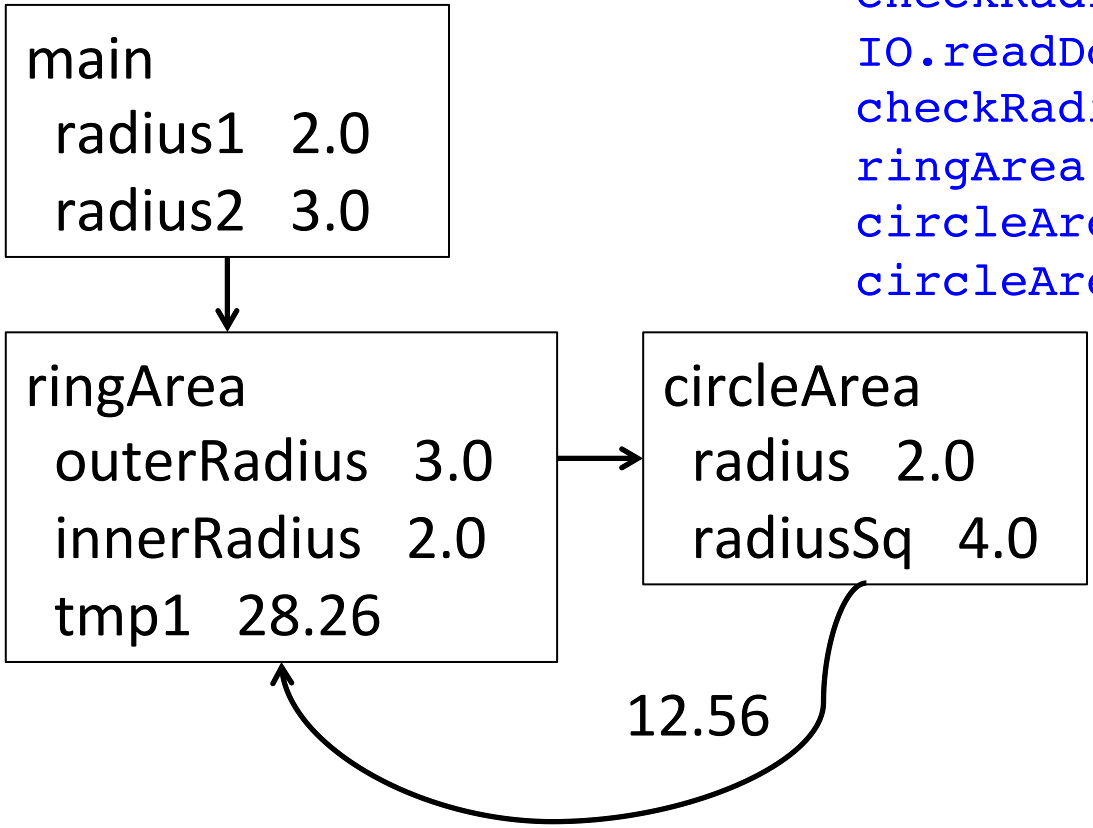


Call Sequence

```
IO.readDouble();
checkRadius(2.0);
IO.readDouble();
checkRadius(3.0);
ringArea(3.0, 2.0);
circleArea(3.0);
circleAread(2.0);
```

Frames

- RingArea.java

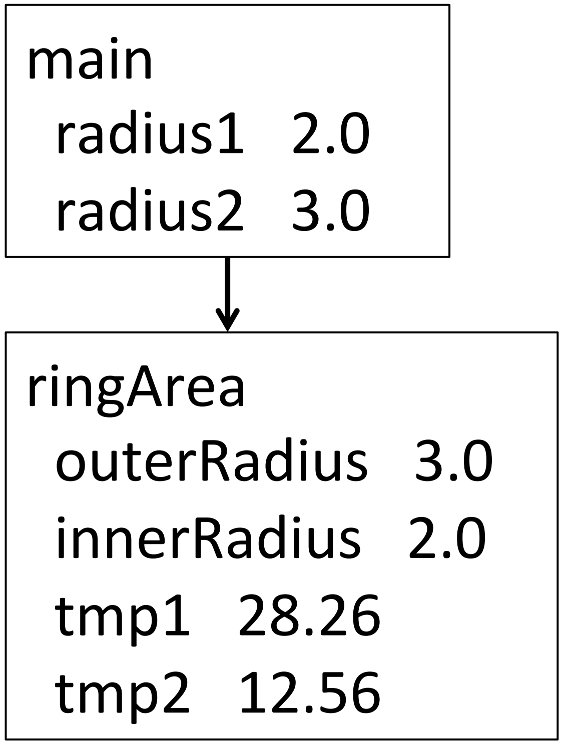


Call Sequence

```
IO.readDouble();
checkRadius(2.0);
IO.readDouble();
checkRadius(3.0);
ringArea(3.0, 2.0);
circleArea(3.0);
circleAread(2.0);
```

Frames

- RingArea.java

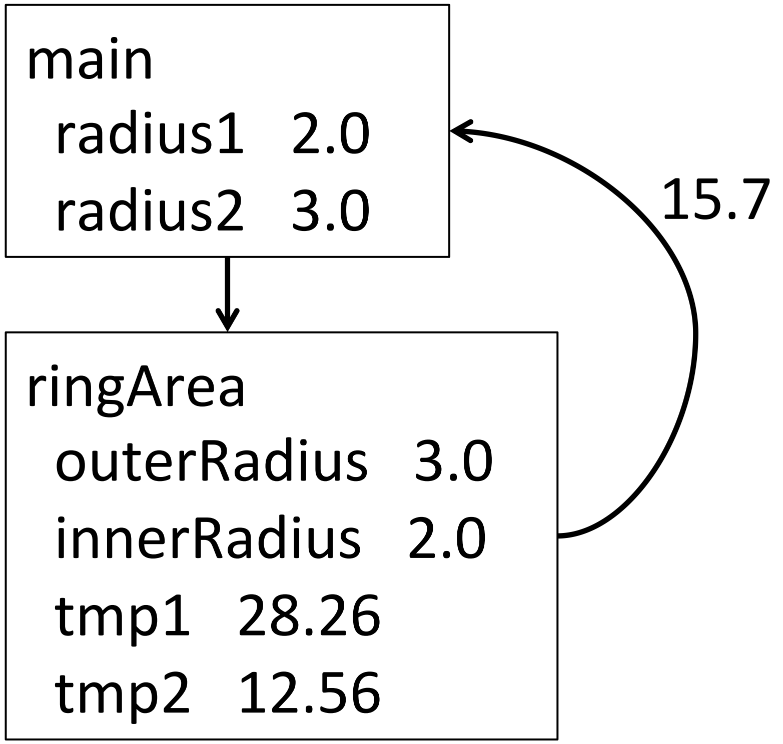


Call Sequence

```
IO.readDouble();
checkRadius(2.0);
IO.readDouble();
checkRadius(3.0);
ringArea(3.0, 2.0);
circleArea(3.0);
circleAread(2.0);
```

Frames

- RingArea.java



Call Sequence

```
IO.readDouble();
checkRadius(2.0);
IO.readDouble();
checkRadius(3.0);
ringArea(3.0, 2.0);
circleArea(3.0);
circleAread(2.0);
```

Frames

- RingArea.java

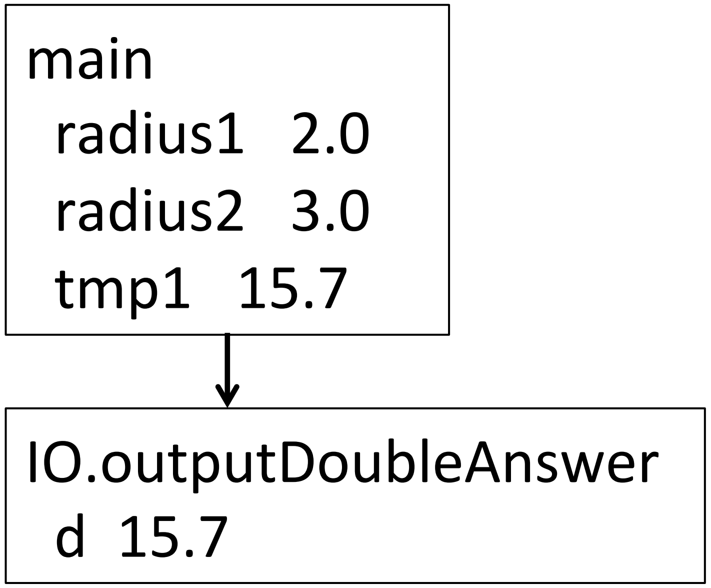
```
main
  radius1   2.0
  radius2   3.0
  tmp1   15.7
```

Call Sequence

```
IO.readDouble();
checkRadius(2.0);
IO.readDouble();
checkRadius(3.0);
ringArea(3.0, 2.0);
circleArea(3.0);
circleAread(2.0);
```

Frames

- RingArea.java



Call Sequence

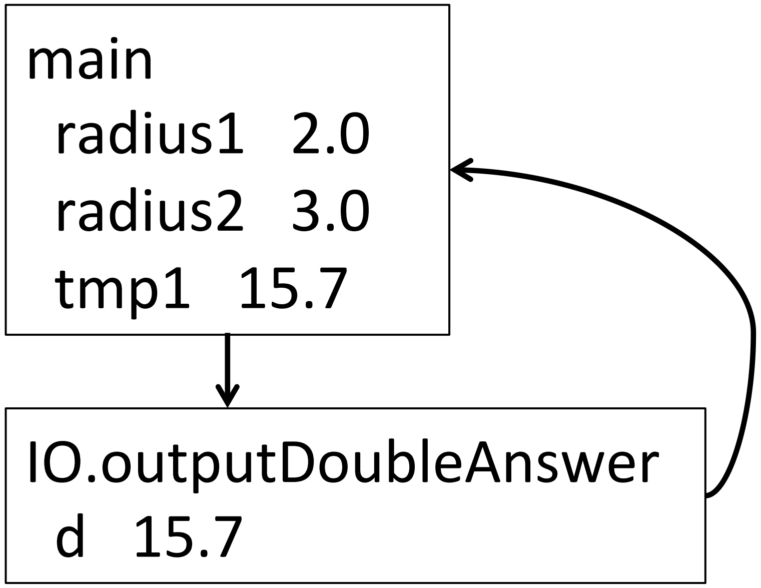
```
IO.readDouble();
checkRadius(2.0);
IO.readDouble();
checkRadius(3.0);
ringArea(3.0, 2.0);
circleArea(3.0);
circleAread(2.0);
IO.outputDoubleAnswer(15.7);
```

Output

```
15.7
```

Frames

- RingArea.java



Call Sequence

```
IO.readDouble();
checkRadius(2.0);
IO.readDouble();
checkRadius(3.0);
ringArea(3.0, 2.0);
circleArea(3.0);
circleAread(2.0);
IO.outputDoubleAnswer(15.7);
```

Output

15.7

Frames

- RingArea.java

```
main
  radius1   2.0
  radius2   3.0
  tmp1   15.7
```

Call Sequence

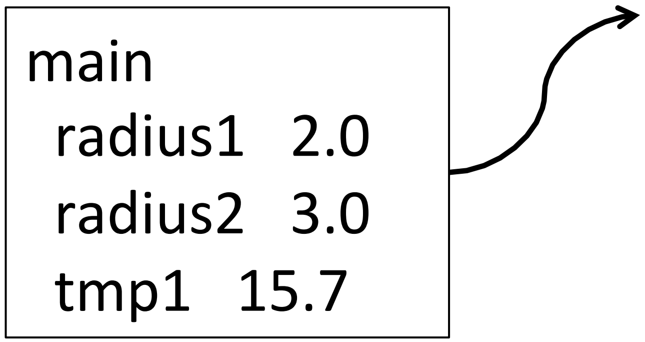
```
IO.readDouble();
checkRadius(2.0);
IO.readDouble();
checkRadius(3.0);
ringArea(3.0, 2.0);
circleArea(3.0);
circleAread(2.0);
IO.outputDoubleAnswer(15.7);
```

Output

15.7

Frames

- RingArea.java



main
radius1 2.0
radius2 3.0
tmp1 15.7

Call Sequence

```
IO.readDouble();
checkRadius(2.0);
IO.readDouble();
checkRadius(3.0);
ringArea(3.0, 2.0);
circleArea(3.0);
circleAread(2.0);
IO.outputDoubleAnswer(15.7);
```

Output

15.7

Frames

- RingArea.java

Call Sequence

```
IO.readDouble();
checkRadius(2.0);
IO.readDouble();
checkRadius(3.0);
ringArea(3.0, 2.0);
circleArea(3.0);
circleAread(2.0);
IO.outputDoubleAnswer(15.7);
```

Output

```
15.7
```

Frames

- When a method is called, it starts up from the beginning with a new Activation Record
- When a call returns to a waiting Activation, that Activation re-activates its Activation record and continues where is left off.
- When a method calls a method, the method doing the call waits, and its Activation record is saved

CS111
Introduction to Computer Science

Fall 2015

- Characters and Strings
- More frames
- Classes, objects and references

Regrading Assignment 1

If you have questions or comments regarding grading of HW1, please contact your assigned grader:

- Joseph DeVita: (1-11, 13, 14)
- Varun Sharma: (15, 16, 19-28)
- Benjamin Bancala: (29-33, 35-41)
- zhanpeng He (Adam): (42-53)
- Daehan Kwak: (54-77)

- zhanpeng he <kod.adam.ho@gmail.com>,
- Joseph DeVita <jodvita@gmail.com>,
- Varun Sharma <varun.sharma@rutgers.edu>,
- Benjamin Bancala <bdb73@scarletmail.rutgers.edu>
- Daehan Kwak <kwakno1@rutgers.edu>

Characters

- A character is any single character
  - letters ‘A’ ‘b’
  - digit ‘0’ ‘9’
  - punctuation ‘#’ ‘.’
  - special characters ‘\t’ – tab character
    ‘\n’ – newline character
- Multiple characters are not legal
  - ‘ru’

Operations on Characters

- char c = IO.readChar();
- Character.isLetter(c);
  - true if c is a letter
- Character.isDigit(c)
  - true if c is a digit
- Character.toUpperCase(c)
  - value is upper case version of c
- Character.toLowerCase(c)
  - value is lower case version of c

String

- A string is a sequence of characters
  - “cs111”
  - “”
  - “Are you listening?”
- Position of a character in the String: *index*
  - “now and then”

```
     “now and then”
      |  | | |   |
      0  3 5 7   11
```

  - length of the String: 12 characters
    - last index = length - 1

Concatenating Strings

- For Strings, + means concatenate
  - “ab” + “cd” -> “abcd”
- If one operand is a String and the other is not, Java converts the non-String into a String
  - “cs” + 111 -> “cs111”
  - 111 + “cs” -> “111cs”
  - “abcd” + ‘e’ -> “abcde”
  - (“ab” + ‘c’) + ‘d’ -> “abcd”
  - “ab” + (3+1) -> “ab4”

See the captalize method in Methods.java

Frames

- Stars.java

```
public static void main(String[] args) {
    for (int i = 1; i <= 3; i++) {
        printNStars(i);
    }
}
public static void printNStars (int n) {
    System.out.println(nTimesChar(n, '*'));
}
public static String nTimesChar (int n, char c) {
    String result = "";
    for (int i = 1; i <= n; i++) {
        result = result + c;
    }
    return result;
}
```

Frames

- Stars.java

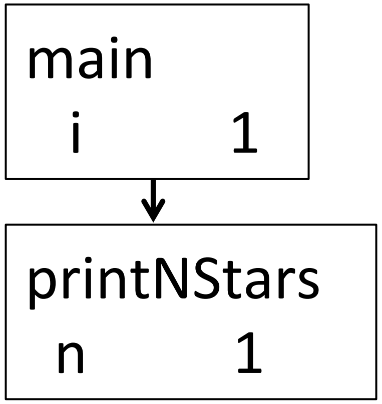
main

Call Sequence

Output

Frames

- Stars.java



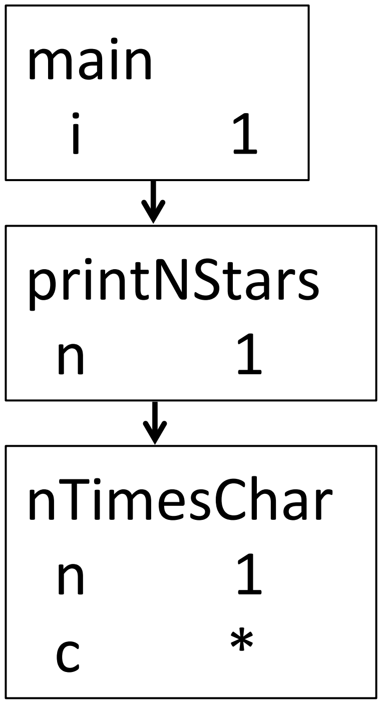
Call Sequence

```
printNStars(1);
```

Output

Frames

- Stars.java



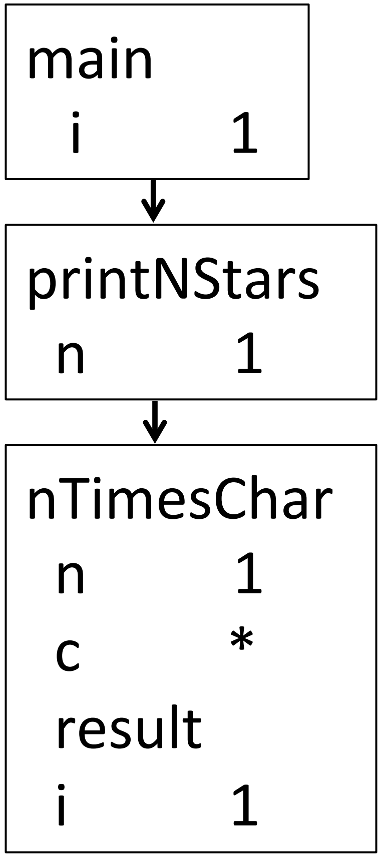
Call Sequence

```
printNStars(1);
nTimesChar(1, '*');
```

Output

Frames

- Stars.java



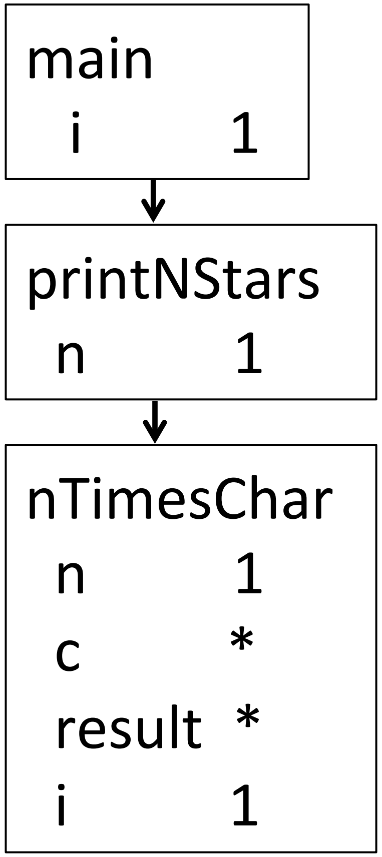
Call Sequence

```
printNStars(1);
nTimesChar(1, '*');
```

Output

Frames

- Stars.java



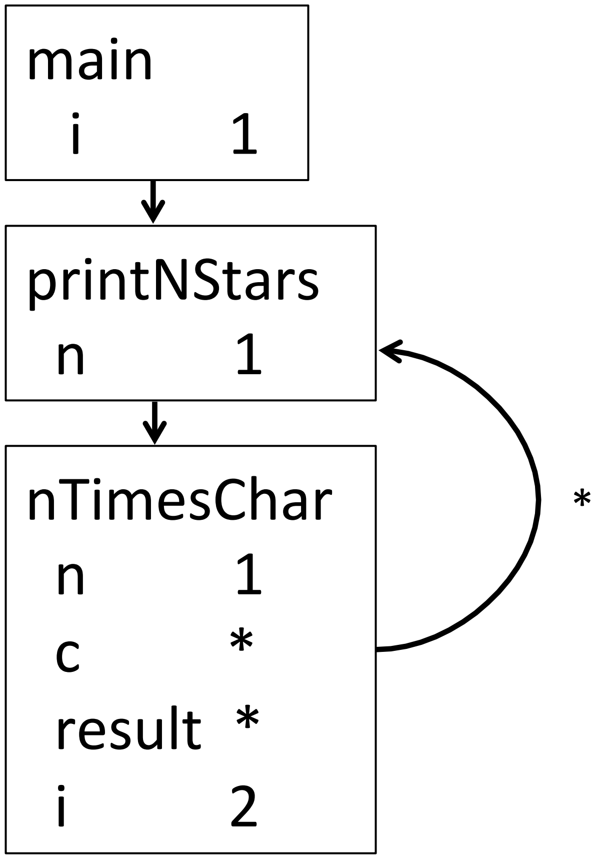
Call Sequence

```
printNStars(1);
nTimesChar(1, '*');
```

Output

Frames

- Stars.java



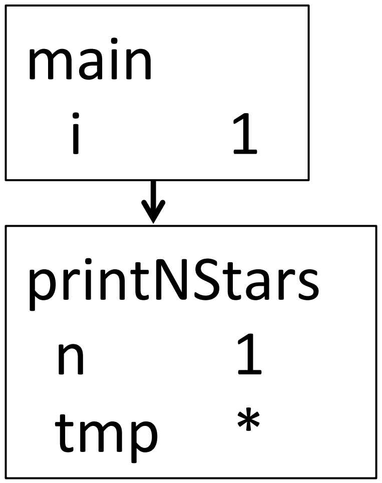
Call Sequence

```
printNStars(1);
nTimesChar(1, '*');
```

Output

Frames

- Stars.java



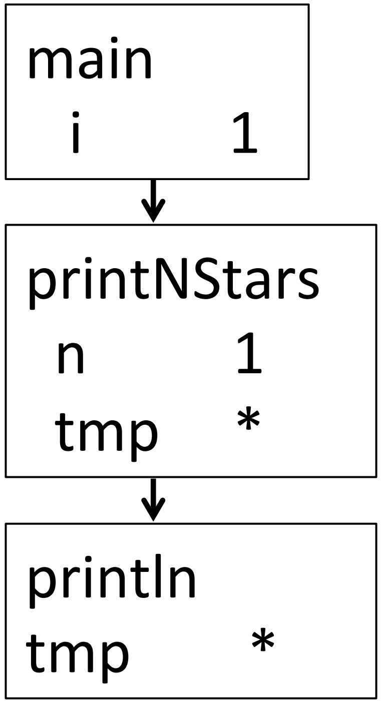
Call Sequence

```
printNStars(1);
nTimesChar(1, '*');
```

Output

Frames

- Stars.java



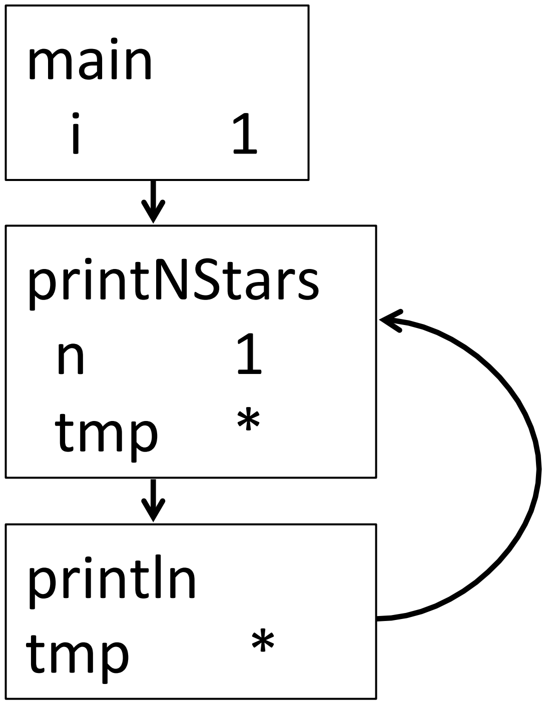
Call Sequence

```
printNStars(1);
nTimesChar(1, '*');
println("*");
```

Output

Frames

- Stars.java



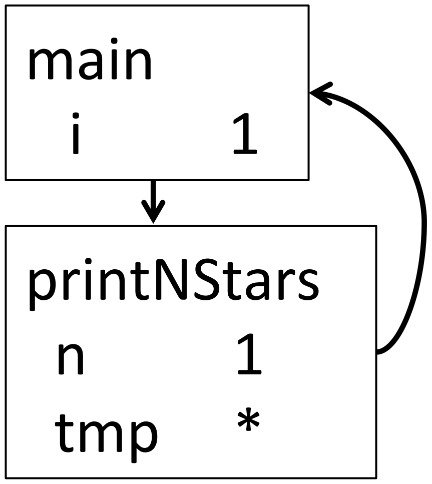
Call Sequence

```
printNStars(1);
nTimesChar(1, '*');
println("*");
```

Output *

Frames

- Stars.java



Call Sequence

```
printNStars(1);
nTimesChar(1, '*');
println("*");
```

Output *

Frames

- Stars.java

```
main
  i       1
```

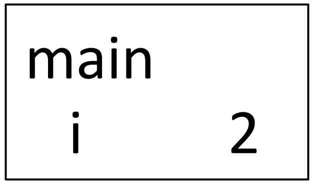
Call Sequence

```
printNStars(1);
nTimesChar(1, '*');
println("*");
```

Output *

Frames

- Stars.java



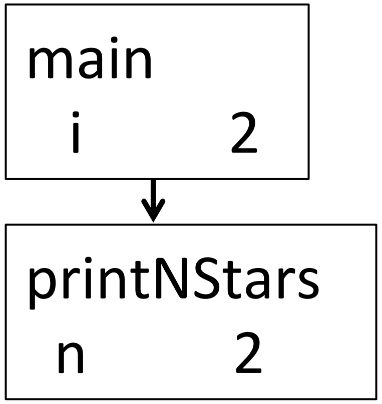
Call Sequence

```
printNStars(1);
nTimesChar(1, '*');
println("*");
```

Output *

Frames

- Stars.java



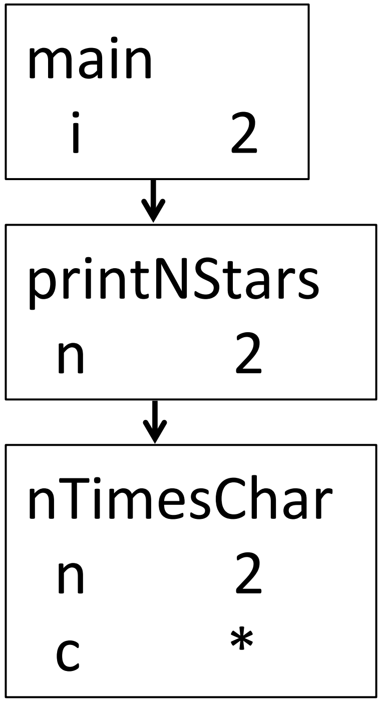
Call Sequence

```
printNStars(1);
nTimesChar(1, '*');
println("*");
printNStars(2);
```

Output *

Frames

- Stars.java



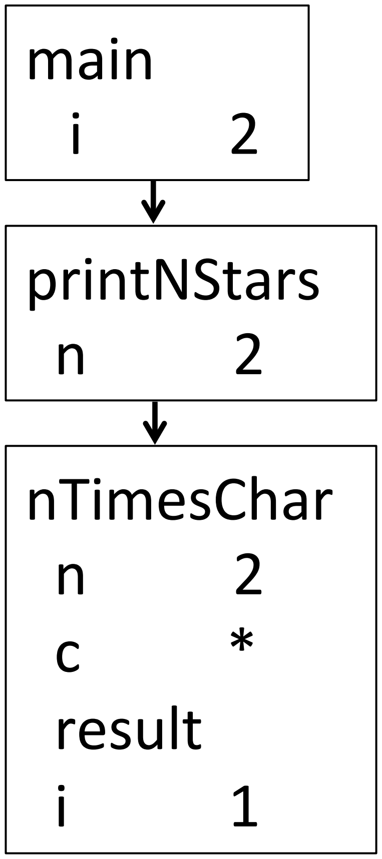
Call Sequence

```
printNStars(1);
nTimesChar(1, '*');
println("*");
printNStars(2);
nTimesChar(2, '*');
```

Output *

Frames

- Stars.java



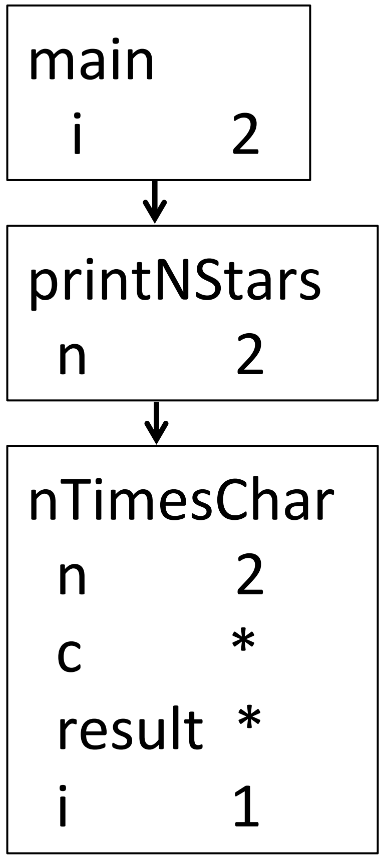
Call Sequence

```
printNStars(1);
nTimesChar(1, '*');
println("*");
printNStars(2);
nTimesChar(2, '*');
```

Output *

Frames

- Stars.java



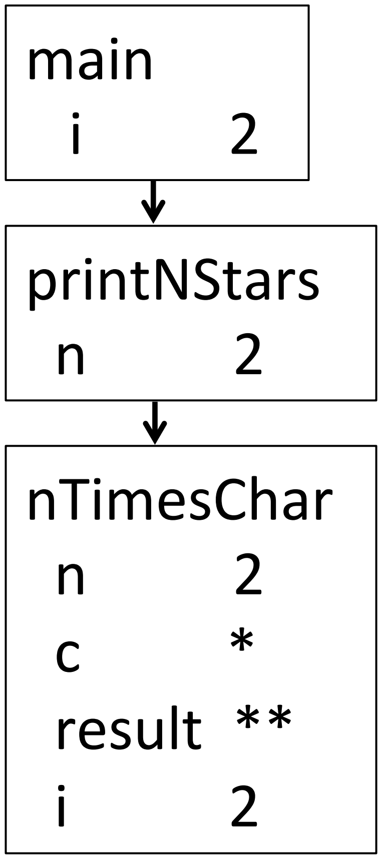
Call Sequence

```
printNStars(1);
nTimesChar(1, ‘*’);
println(“*”);
printNStars(2);
nTimesChar(2, ‘*’);
```

Output
```
*
```

Frames

- Stars.java



Call Sequence

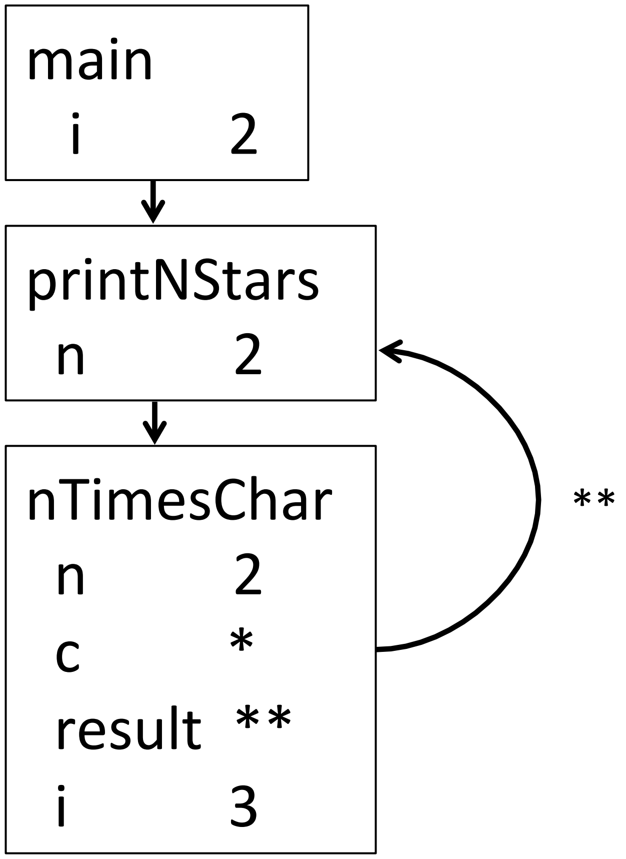
```
printNStars(1);
nTimesChar(1, ‘*’);
println(“*”);
printNStars(2);
nTimesChar(2, ‘*’);
```

Output

```
*
```

Frames

- Stars.java



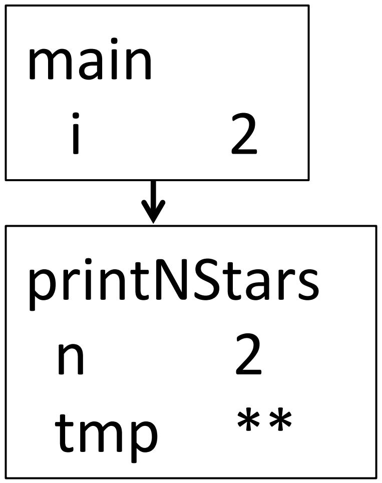
Call Sequence

```
printNStars(1);
nTimesChar(1, '*');
println("*");
printNStars(2);
nTimesChar(2, '*');
```

Output *

Frames

- Stars.java



Call Sequence

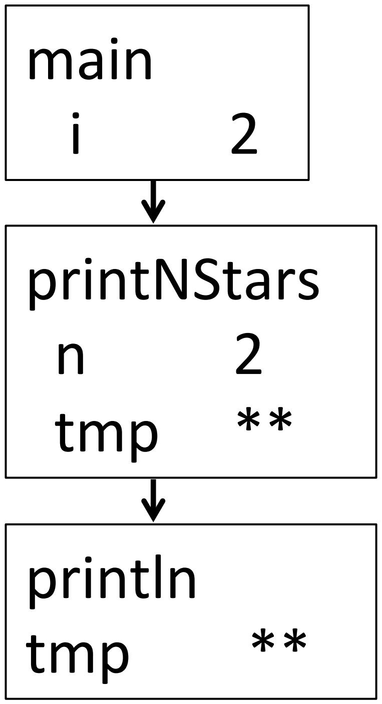
```
printNStars(1);
nTimesChar(1, '*');
println("*");
printNStars(2);
nTimesChar(2, '*');
```

Output

```
*
```

Frames

- Stars.java



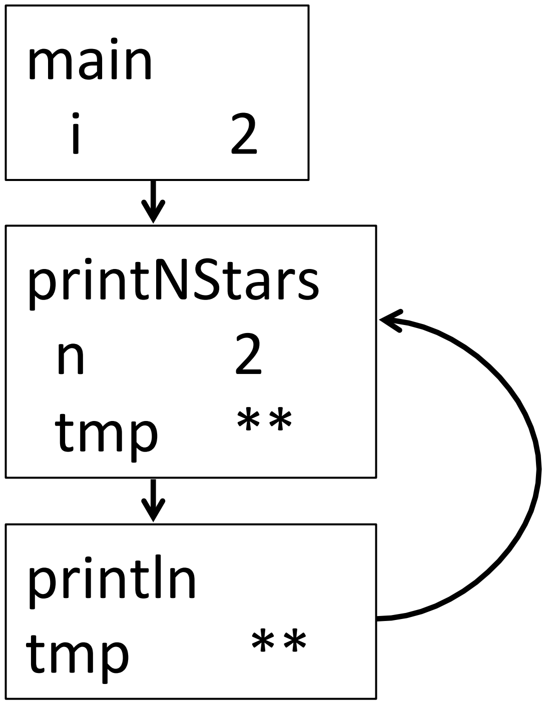
Call Sequence

```
printNStars(1);
nTimesChar(1, '*');
println("*");
printNStars(2);
nTimesChar(2, '*');
println("**");
```

Output *

Frames

- Stars.java



Call Sequence

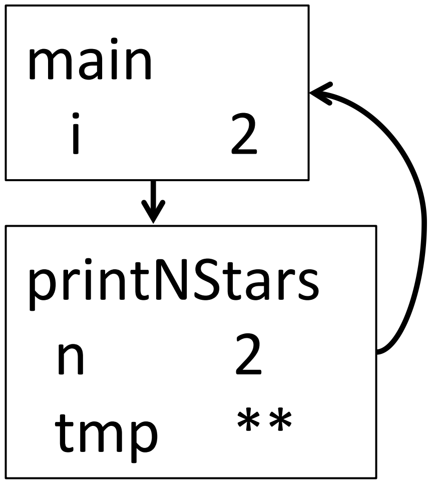
```
printNStars(1);
nTimesChar(1, '*');
println("*");
printNStars(2);
nTimesChar(2, '*');
println("**");
```

Output

```
*
**
```

Frames

- Stars.java



Call Sequence

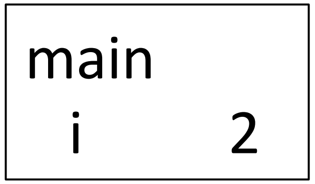
```
printNStars(1);
nTimesChar(1, '*');
println("*");
printNStars(2);
nTimesChar(2, '*');
println("**");
```

Output

```
*
**
```

Frames

- Stars.java



Call Sequence

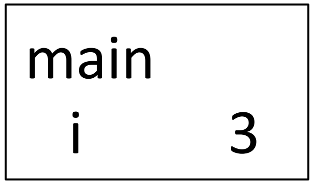
```
printNStars(1);
nTimesChar(1, '*');
println("*");
printNStars(2);
nTimesChar(2, '*');
println("**");
```

Output

```
*
**
```

Frames

- Stars.java



Call Sequence

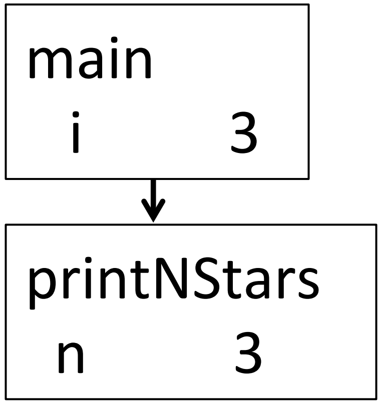
```
printNStars(1);
nTimesChar(1, '*');
println("*");
printNStars(2);
nTimesChar(2, '*');
println("**");
```

Output

```
*
**
```

Frames

- Stars.java



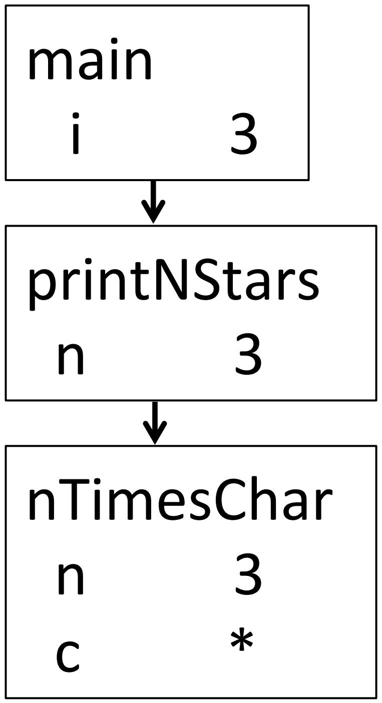
Call Sequence

```
printNStars(1);
nTimesChar(1, '*');
println("*");
printNStars(2);
nTimesChar(2, '*');
println("**");
printNStars(3);
```

Output
```
*
**
```

Frames

- Stars.java



Call Sequence

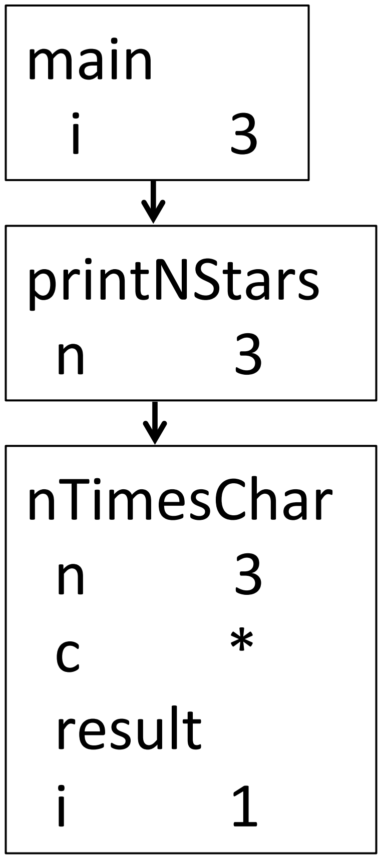
```
printNStars(1);
nTimesChar(1, '*');
println("*");
printNStars(2);
nTimesChar(2, '*');
println("**");
printNStars(3);
nTimesChar(3, '*');
```

Output

```
*
**
```

Frames

- Stars.java



Call Sequence

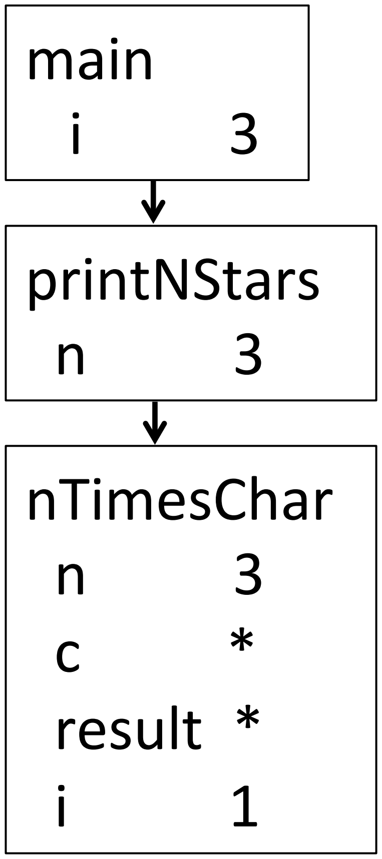
```
printNStars(1);
nTimesChar(1, ‘*’);
println(“*”);
printNStars(2);
nTimesChar(2, ‘*’);
println(“**”);
printNStars(3);
nTimesChar(3, ‘*’);
```

Output

```
*
**
```

Frames

- Stars.java



Call Sequence

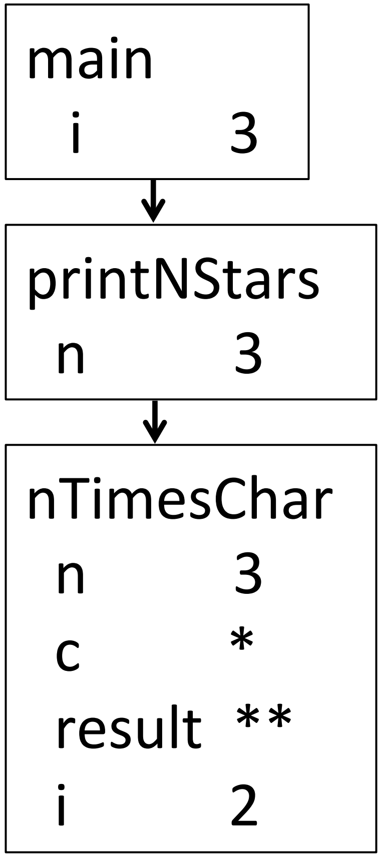
```
printNStars(1);
nTimesChar(1, '*');
println("*");
printNStars(2);
nTimesChar(2, '*');
println("**");
printNStars(3);
nTimesChar(3, '*');
```

Output

```
*
**
```

Frames

- Stars.java



Call Sequence

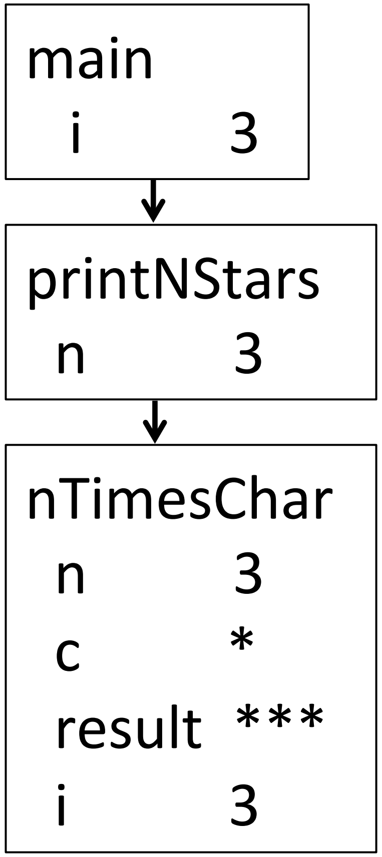
```
printNStars(1);
nTimesChar(1, '*');
println("*");
printNStars(2);
nTimesChar(2, '*');
println("**");
printNStars(3);
nTimesChar(3, '*');
```

Output

```
*
**
```

Frames

- Stars.java



Call Sequence

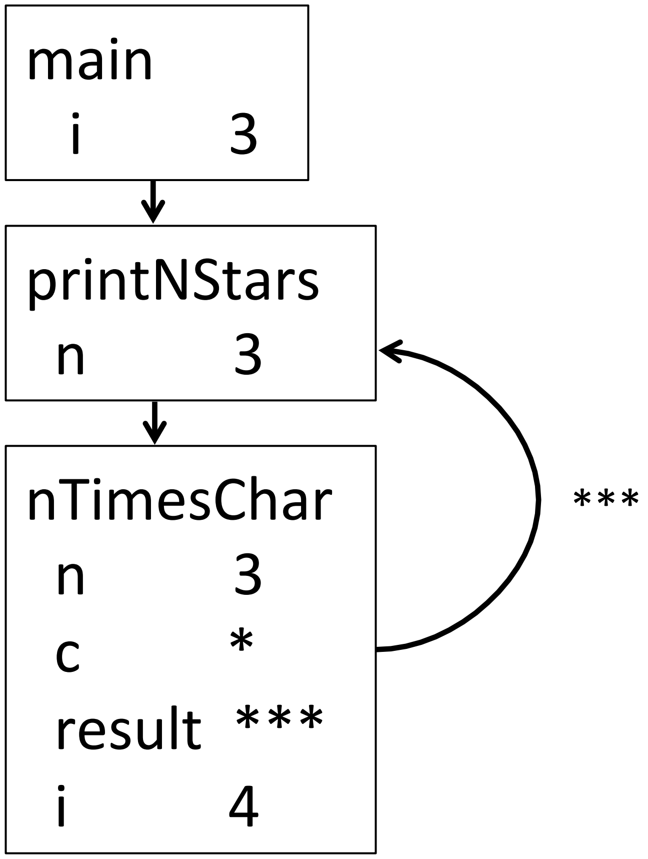
```
printNStars(1);
nTimesChar(1, '*');
println("*");
printNStars(2);
nTimesChar(2, '*');
println("**");
printNStars(3);
nTimesChar(3, '*');
```

Output

```
*
**
```

Frames

- Stars.java



Call Sequence

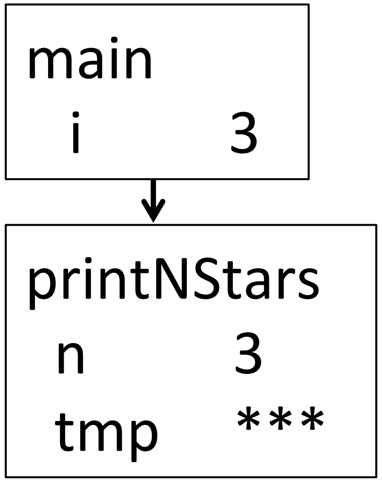
```
printNStars(1);
nTimesChar(1, '*');
println("*");
printNStars(2);
nTimesChar(2, '*');
println("**");
printNStars(3);
nTimesChar(3, '*');
```

Output

```
*
**
```

Frames

- Stars.java



Call Sequence

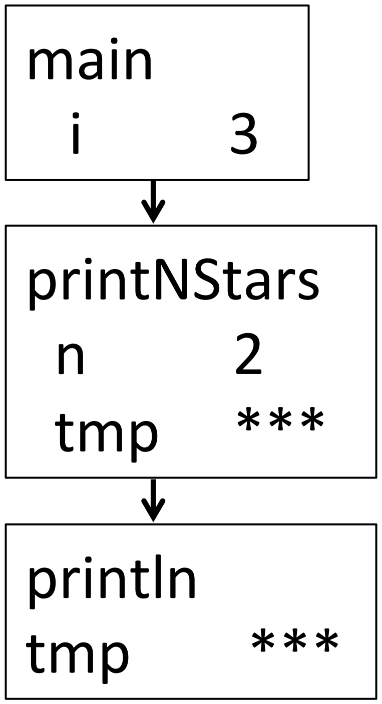
```
printNStars(1);
nTimesChar(1, '*');
println("*");
printNStars(2);
nTimesChar(2, '*');
println("**");
printNStars(3);
nTimesChar(3, '*');
```

Output

```
*
**
```

Frames

- Stars.java



Call Sequence

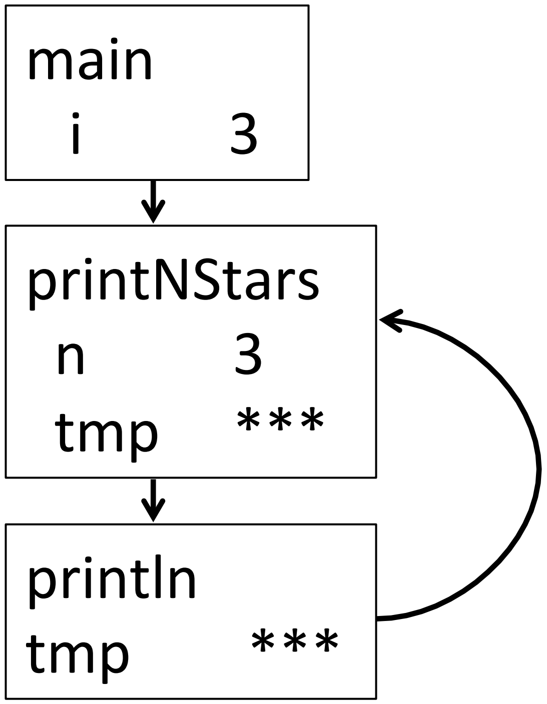
```
printNStars(1);
nTimesChar(1, '*');
println("*");
printNStars(2);
nTimesChar(2, '*');
println("**");
printNStars(3);
nTimesChar(3, '*');
println("***");
```

Output

```
*
**
```

Frames

- Stars.java



Call Sequence

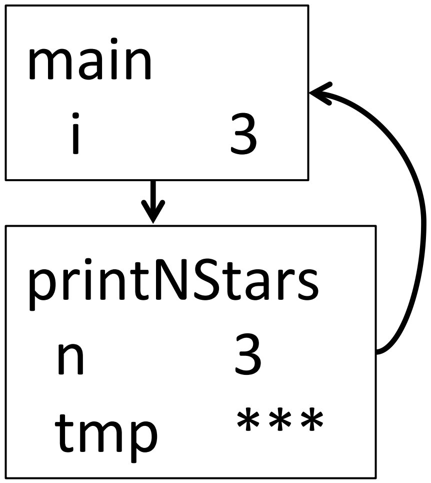
```
printNStars(1);
nTimesChar(1, '*');
println("*");
printNStars(2);
nTimesChar(2, '*');
println("**");
printNStars(3);
nTimesChar(3, '*');
println("***");
```

Output

```
*
**
***
```

Frames

- Stars.java



Call Sequence

```
printNStars(1);
nTimesChar(1, '*');
println("*");
printNStars(2);
nTimesChar(2, '*');
println("**");
printNStars(3);
nTimesChar(3, '*');
println("***");
```

Output

```
*
**
***
```

Frames

- Stars.java

```
main
  i        3
```

Call Sequence

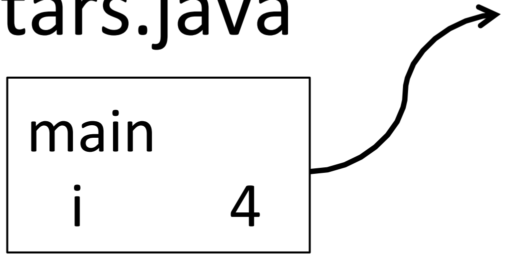
```
printNStars(1);
nTimesChar(1, ‘*’);
println(“*”);
printNStars(2);
nTimesChar(2, ‘*’);
println(“**”);
printNStars(3);
nTimesChar(3, ‘*’);
println(“***”);
```

Output

```
*
**
***
```

Frames

- Stars.java



Call Sequence

```
printNStars(1);
nTimesChar(1, ‘*’);
println(“*”);
printNStars(2);
nTimesChar(2, ‘*’);
println(“**”);
printNStars(3);
nTimesChar(3, ‘*’);
println(“***”);
```

Output

```
*
**
***
```

Frames

- Stars.java

Call Sequence

```
printNStars(1);
nTimesChar(1, '*');
println("*");
printNStars(2);
nTimesChar(2, '*');
println("**");
printNStars(3);
nTimesChar(3, '*');
println("***");
```

Output

```
*
**
***
```

String

- A string is a sequence of characters
  - "cs111"
  - ""
  - "Are you listening?"
- Position of a character in the String: *index*
  - "now and then"

```
 now and then
 |  | | |   |
 0  3 5 7   11
```

  - length of the String: 12 characters
    - last index = length - 1

Strings, Classes and Objects

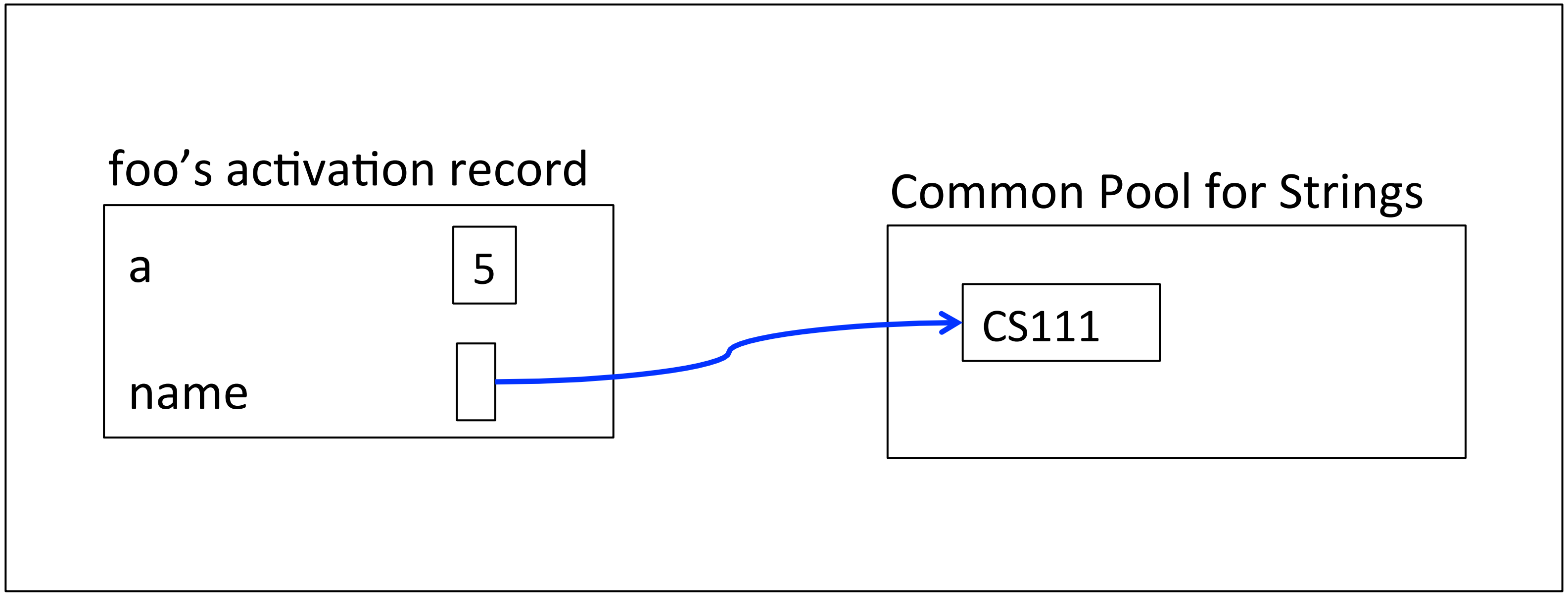
- String is not a primitive data type but a class
  - Whenever we create a string we create and object
- Classes
  - can be containers for static variables and methods
  - can be used to describe objects
    - in this role the class describes a special kind of data type
- Objects are the instantiation of a class
  - int a; $\longleftarrow$ a is of type int
  - String name; $\longleftarrow$ name is of type String

Strings, Classes, Objects and References

```
public static void foo () {
    int a = 5;
    String name = "CS111";
}
```

name **is** a reference to the String object "CS111"

Java Virtual Machine (JVM)



Variables, Primitive Data Types and References

- A variable can only hold primitive data types OR references to objects
- The objects are kept (held) elsewhere
  - the variable holds the objects' reference
  - what is a reference?
    - the address of the object

String Methods: operations

- `String name = “cs111”;`
- Length of a string

  `name.length()` $\longrightarrow$ 5
- Substring: copy a consecutive sequence of characteres

  starting at index

  up to, not including this index

  `name.substring(1,3);` $\longrightarrow$ “s1”

  `name.substring(4,7);` $\longrightarrow$ error

  `name.substring(1);` $\longrightarrow$ “s111”

String Methods: operations

- Index of the first occurrence of a character

  `“cs111”.indexof(‘1’);` $\longrightarrow$ 2

  `“cs111”.indexof(‘x’);` $\longrightarrow$ -1

- The character at index 2

  `“cs111”.charAt(3);` $\longrightarrow$ 1

- In all upper case

  `“aBcde”.toUpperCase();` $\longrightarrow$ “ ABCDE”

- In all lower case

  `“ABCde”.toLowerCase();` $\longrightarrow$ “abcde”

String Methods: operations

- Test if two strings are the same

  `name1.equals(name2);`

- Test if one string is alphabetically before another

  int c = name1.compareTo(name2);

  c < 0 means name1 before name2

  c == 0 means name1 equals name2

  c > 0 means name1 after name2

See the alphabeticalOder method in Methods.java

String Methods: operations

- None of the operations changes the existing string

```
String name = “Joe”;
String upperName = name.toUpperCase();
System.out.println(upperName); // JOE
System.out.println(name); // Joe
```

CS111
Introduction to Computer Science

Fall 2015

- More on Strings
- Classes, Objects and References

Creating Formatted Strings

- Printing a formatted string

```
System.out.printf(“The value of a float variable is %f, the value of an integer variable is %d and the string is %s\n”, floatVar, intVar, stringVar);
```

- You can also write

```
String fs = String.format(“The value of a float variable is %f, the value of an integer variable is %d and the string is %s\n”, floatVar, intVar, stringVar);
System.out.println(fs);
```

Program: Count Spaces in a String

Given a String, count how many spaces the String has:

“Are you listening?”

2

- Initialize a count variable
- Loop through all the characters of the string
- Update count if character is a space

Implement CountSpaces method in StringTest.java

Program: letter frequency

- Given a String, output the frequency of each letter:

  "Are you listening?"

  a 1, r 1, e 2, y 1, o 1, u 1, l 1, i 2, s 1, t 1, n 2, g 1

- For each letter in the alphabet
  - Output its frequency on the String

Loop over alphabet

Create a method to count character frequency

See the letterFrequency method in StringTest.java

Face Class and Objects

- Face.java
  - Class describing a Face object

- To create an instance of Face (object)

```
Face f = new Face();
```

Face Class and Objects

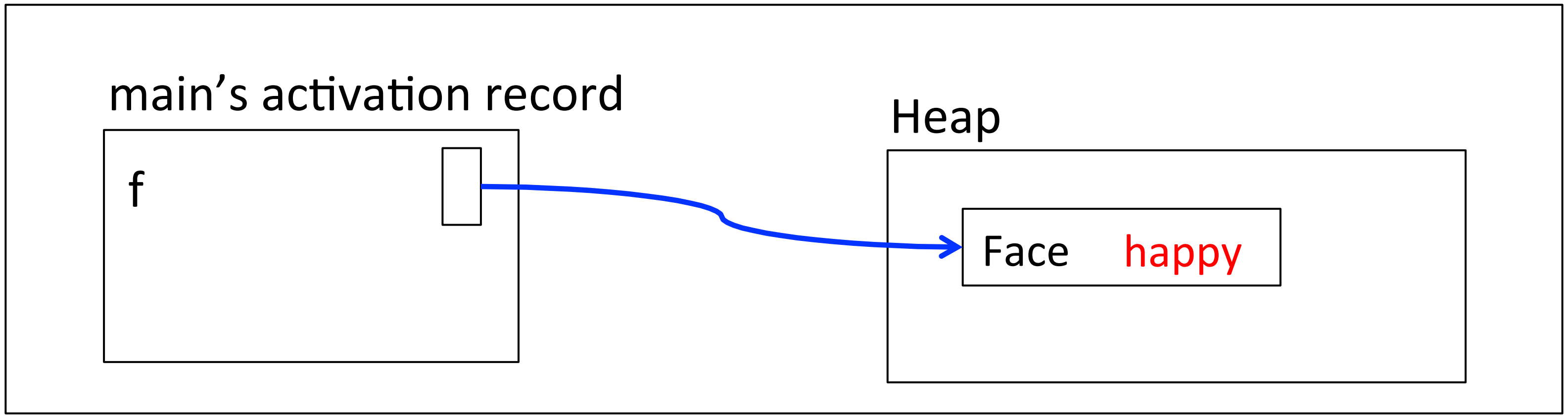
```
public static void main (String[] args) {
    Face f = new Face();
    f.setExpression("happy");
    f.setExpression("mad");
}
```

f **is** a reference to an object of type Face

new creates an instance of Face on the Heap

The **Heap** is where objects created with new live

Java Virtual Machine (JVM)

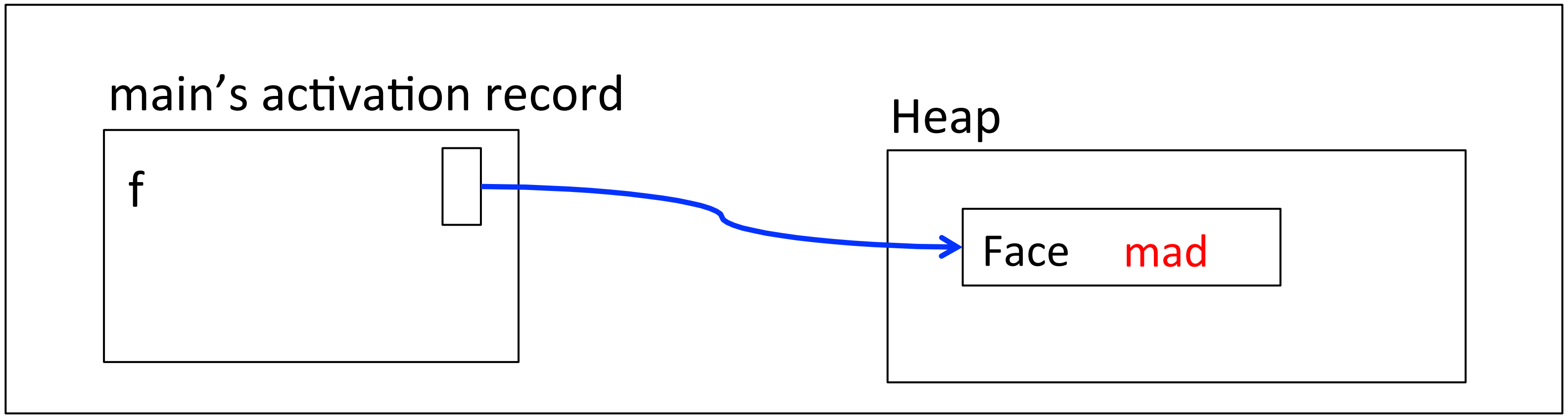


Face is a Mutable Object

```
public static void main (String[] args) {
    Face f = new Face();
    f.setExpression(“happy”);
    f.setExpression(“mad”);
}
```

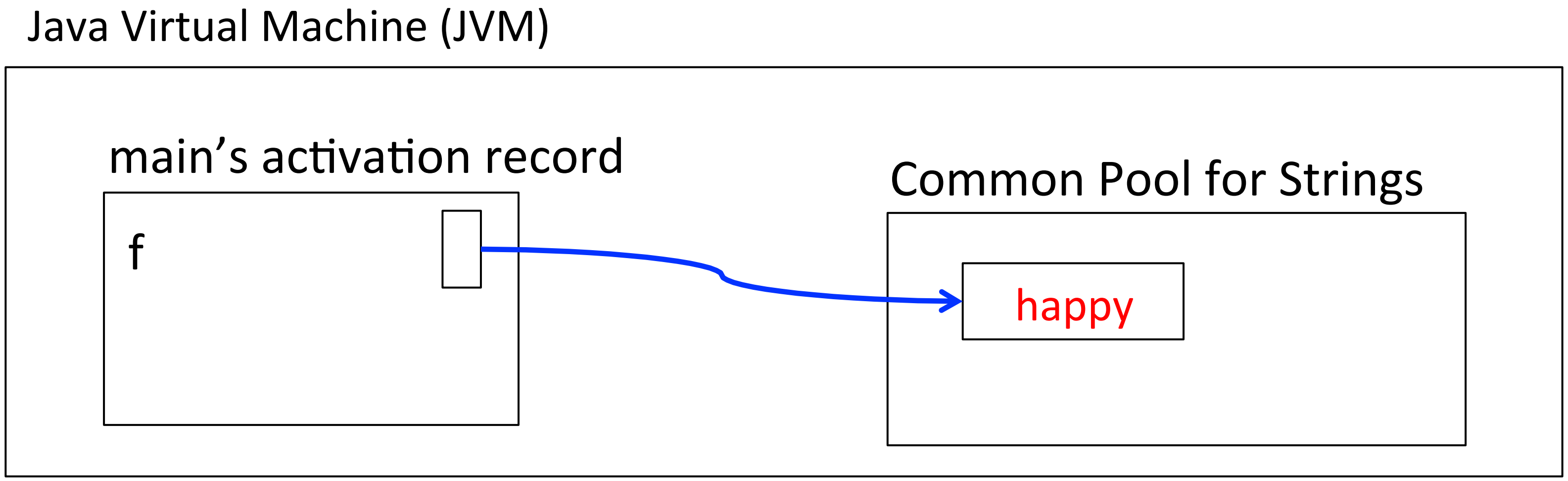
setExpression changes the object Face from happy to mad

Java Virtual Machine (JVM)



String is an Immutable Object

```
public static void main (String[] args) {
    String f = “happy”;
    f = “mad”;
}
```

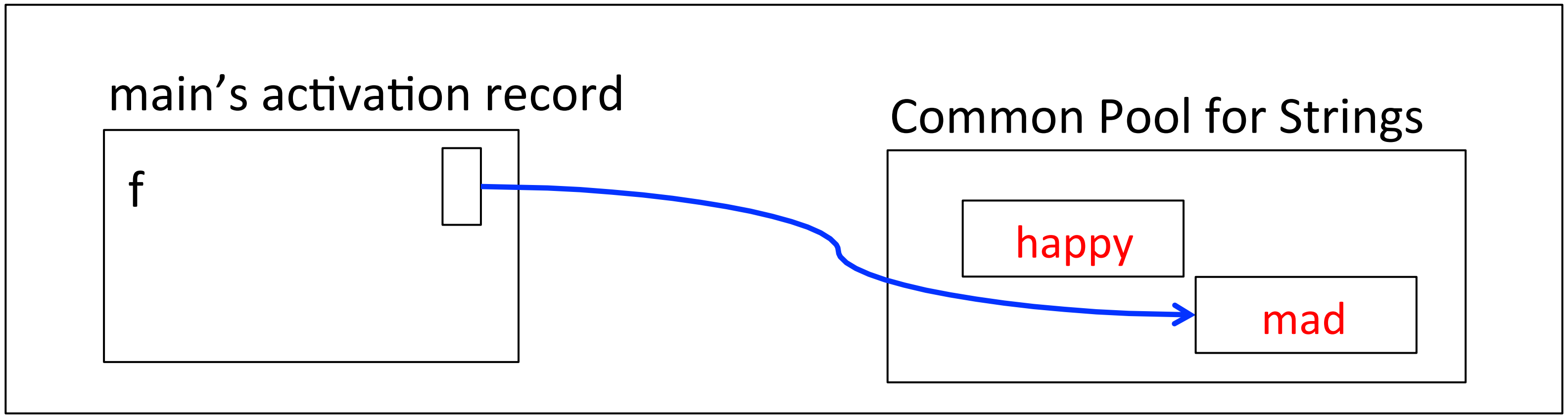


String is an Immutable Object

```
public static void main (String[] args) {
   String f = “happy”;
   f = “mad”;
}
```

This statement does not change the String object **happy**
It creates another String object **mad** and **f** now references it

Java Virtual Machine (JVM)


main’s activation record
f
Common Pool for Strings
happy
mad