One Last Sorting Example

Custom Sorting

- We can sort any type with any comparator
- But what if we want to sort points by their distance from a reference point
    - In general: what if the comparator needs more parameters than just the two elements?
    - [Without using global state]
- We can dynamically create a new function with the additional parameters "built-in"

Returning Functions

- We can write a function/method that takes all the needed parameters and returns a function that fits the signature of a comparator
  - The distanceComparator method returns a comparator that compares the distance to a reference point

```
def distance(v1: PhysicsVector, v2: PhysicsVector): Double = {
  Math.sqrt(Math.pow(v1.x - v2.x, 2.0) + Math.pow(v1.y - v2.y, 2.0) + Math.pow(v1.z - v2.z, 2.0))
}

def distanceComparator(referencePoint: PhysicsVector): (PhysicsVector, PhysicsVector) => Boolean = {
  (v1: PhysicsVector, v2: PhysicsVector) => {
    distance(v1, referencePoint) < distance(v2, referencePoint)
  }
}
```

Returning Functions

- Use distanceComparator to create a comparator function when needed

- Can create different comparators with different reference points

  - Global state would only allow one comparator at a time

```
val referencePoint = new PhysicsVector(0.5, 0.5, 0.0)
val sortedPoints = MergeSort.mergeSort(points, distanceComparator(referencePoint))
```

```
def distance(v1: PhysicsVector, v2: PhysicsVector): Double = {
  Math.sqrt(Math.pow(v1.x - v2.x, 2.0) + Math.pow(v1.y - v2.y, 2.0) + Math.pow(v1.z - v2.z, 2.0))
}

def distanceComparator(referencePoint: PhysicsVector): (PhysicsVector, PhysicsVector) => Boolean = {
  (v1: PhysicsVector, v2: PhysicsVector) => {
    distance(v1, referencePoint) < distance(v2, referencePoint)
  }
}
```

Recursion

Recursion Example

```
def computeGeometricSum(n: Int): Int ={
 if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
  }else{
    0
  }
}

def main(args: Array[String]): Unit = {
  val result: Int = computeGeometricSum(3)
  println(result)
}
```

Recursive Example

```
1:  def computeGeometricSum(n: Int): Int ={
2:    if(n>0) {
3:      var result: Int = computeGeometricSum(n - 1)
4:      result += n
5:      result
6:    }else{
7:      0
8:    }
9:  }
10:
11: def main(args: Array[String]): Unit = {
12:   val result: Int = computeGeometricSum(3)
13:   println(result)
14: }
```

- Each recursive calls creates a new stack frame
- Each frame remembers where it will resume running when its on the top of the stack

Program Stack	
Main Frame	args
Method Frame	name:n, value:3
	pointer -> line 3
Method Frame	name:n, value:2
	pointer -> line 3
Method Frame	name:n, value:1
	pointer -> line 3

Recursive Example

```
1:  def computeGeometricSum(n: Int): Int ={
2:    if(n>0) {
3:      var result: Int = computeGeometricSum(n - 1)
4:      result += n
5:      result
6:    }else{
7:      0
8:    }
9:  }
10:
11: def main(args: Array[String]): Unit = {
12:   val result: Int = computeGeometricSum(3)
13:   println(result)
14: }
```

- Each time the method is called a new frame is created
- New frame starts at the first line of the method

Program Stack	
Main Frame	args
Method Frame	name:n, value:3
	pointer -> line 3
Method Frame	name:n, value:2
	pointer -> line 3
Method Frame	name:n, value:1
	pointer -> line 3
Method Frame	name:n, value:0
	pointer -> line 1

Recursive Example

```
1:  def computeGeometricSum(n: Int): Int ={
2:    if(n>0) {
3:      var result: Int = computeGeometricSum(n - 1)
4:      result += n
5:      result
6:    }else{
7:      0
8:    }
9:  }
10:
11: def main(args: Array[String]): Unit = {
12:   val result: Int = computeGeometricSum(3)
13:   println(result)
14: }
```

- Top frame on the stack executes the method one line at a time

Program Stack	
Main Frame	args
Method Frame	name:n, value:3
	pointer -> line 3
Method Frame	name:n, value:2
	pointer -> line 3
Method Frame	name:n, value:1
	pointer -> line 3
Method Frame	name:n, value:0
	pointer -> line 2

Recursive Example

```
1:  def computeGeometricSum(n: Int): Int ={
2:    if(n>0) {
3:      var result: Int = computeGeometricSum(n - 1)
4:      result += n
5:      result
6:    }else{
7:      0
8:    }
9:  }
10:
11: def main(args: Array[String]): Unit = {
12:   val result: Int = computeGeometricSum(3)
13:   println(result)
14: }
```

- Top frame on the stack executes the method one line at a time

Program Stack	
Main Frame	args
Method Frame	name:n, value:3
	pointer -> line 3
Method Frame	name:n, value:2
	pointer -> line 3
Method Frame	name:n, value:1
	pointer -> line 3
Method Frame	name:n, value:0
	pointer -> line 7

Recursive Example

```
1:  def computeGeometricSum(n: Int): Int ={
2:    if(n>0) {
3:      var result: Int = computeGeometricSum(n - 1)
4:      result += n
5:      result
6:    }else{
7:      0
8:    }
9:  }
10:
11: def main(args: Array[String]): Unit = {
12:   val result: Int = computeGeometricSum(3)
13:   println(result)
14: }
```

Program Stack	
Main Frame	args
Method Frame	name:n, value:3
	pointer -> line 3
Method Frame	name:n, value:2
	pointer -> line 3
Method Frame	name:n, value:1
	pointer -> line 3
Method Frame	Returning 0

- When a method call returns, its frame is destroyed
- Calling frames resumes and uses the return value

Recursive Example

```
1:  def computeGeometricSum(n: Int): Int ={
2:    if(n>0) {
3:      var result: Int = computeGeometricSum(n - 1)
4:      result += n
5:      result
6:    }else{
7:      0
8:    }
9:  }
10:
11: def main(args: Array[String]): Unit = {
12:   val result: Int = computeGeometricSum(3)
13:   println(result)
14: }
```

- When a method call returns, its frame is destroyed
- Calling frames resumes and uses the return value

Program Stack	
Main Frame	args
Method Frame	name:n, value:3
	pointer -> line 3
Method Frame	name:n, value:2
	pointer -> line 3
Method Frame	name:n, value:1
	name:result, value:0
	pointer -> line 3

Recursive Example

```
1:  def computeGeometricSum(n: Int): Int ={
2:    if(n>0) {
3:      var result: Int = computeGeometricSum(n - 1)
4:      result += n
5:      result
6:    }else{
7:      0
8:    }
9:  }
10:
11: def main(args: Array[String]): Unit = {
12:   val result: Int = computeGeometricSum(3)
13:   println(result)
14: }
```

- Method continues after the recursive call

Program Stack	
Main Frame	args
Method Frame	name:n, value:3
	pointer -> line 3
Method Frame	name:n, value:2
	pointer -> line 3
Method Frame	name:n, value:1
	name:result, value:1
	pointer -> line 3

Recursive Example

```
1:  def computeGeometricSum(n: Int): Int ={
2:    if(n>0) {
3:      var result: Int = computeGeometricSum(n - 1)
4:      result += n
5:      result
6:    }else{
7:      0
8:    }
9:  }
10:
11: def main(args: Array[String]): Unit = {
12:   val result: Int = computeGeometricSum(3)
13:   println(result)
14: }
```

- Method continues after the recursive call

Program Stack	
Main Frame	args
Method Frame	name:n, value:3
	pointer -> line 3
Method Frame	name:n, value:2
	pointer -> line 3
Method Frame	returning 1

Recursive Example

```
1:  def computeGeometricSum(n: Int): Int ={
2:    if(n>0) {
3:      var result: Int = computeGeometricSum(n - 1)
4:      result += n
5:      result
6:    }else{
7:      0
8:    }
9:  }
10:
11: def main(args: Array[String]): Unit = {
12:   val result: Int = computeGeometricSum(3)
13:   println(result)
14: }
```

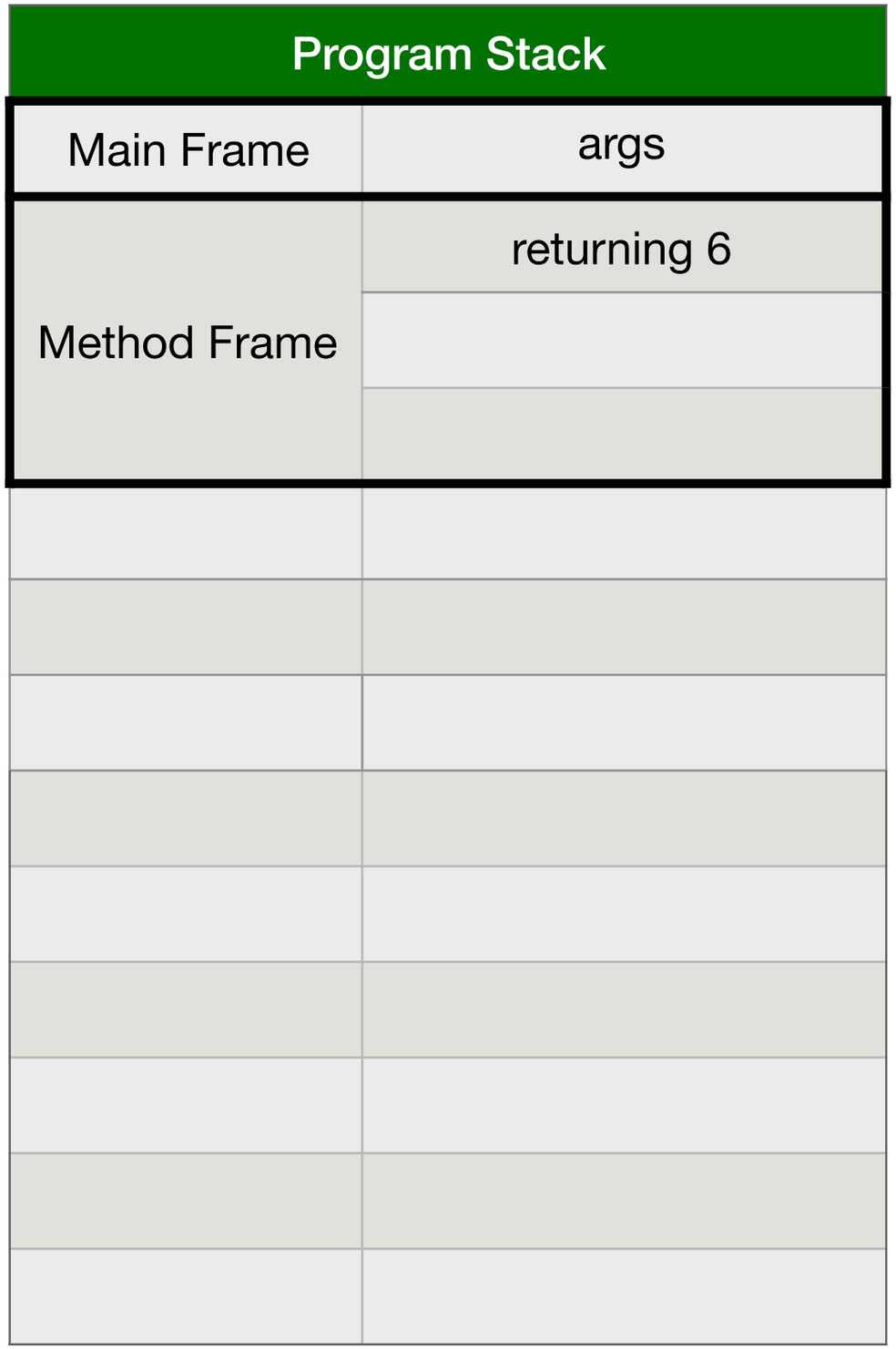
Program Stack	
Main Frame	args
Method Frame	name:n, value:3
	pointer -> line 3
Method Frame	name:n, value:2
	name:result, value:1
	pointer -> line 3

- Return value
- Pop off the stack
- Resume execution of the top frame

Recursive Example

```
1:  def computeGeometricSum(n: Int): Int ={
2:    if(n>0) {
3:      var result: Int = computeGeometricSum(n - 1)
4:      result += n
5:      result
6:    }else{
7:      0
8:    }
9:  }
10:
11: def main(args: Array[String]): Unit = {
12:   val result: Int = computeGeometricSum(3)
13:   println(result)
14: }
```

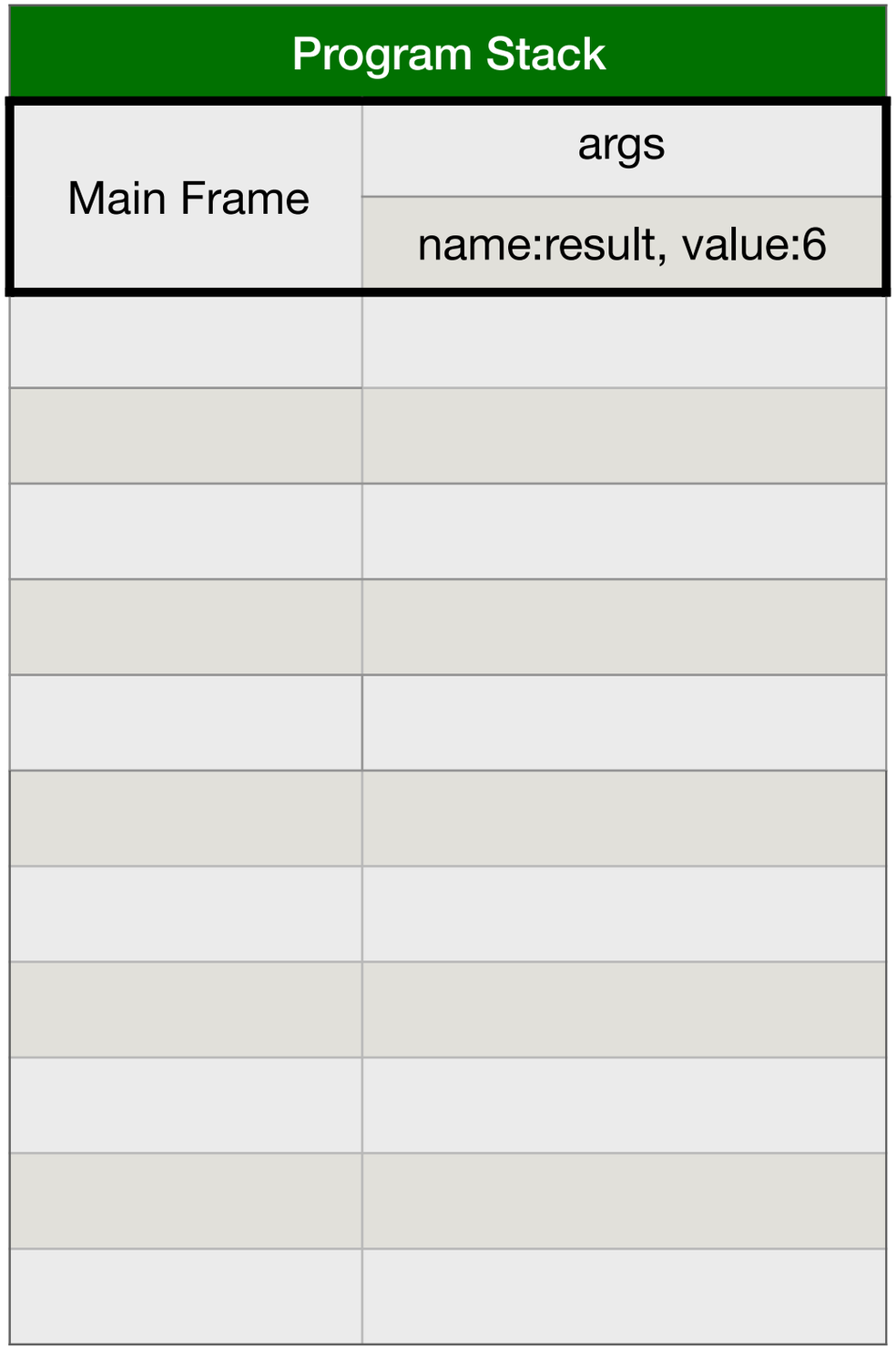
- Process continues until all recursive calls resolve
- Last frame returns to main



Recursive Example

```
1:  def computeGeometricSum(n: Int): Int ={
2:    if(n>0) {
3:      var result: Int = computeGeometricSum(n - 1)
4:      result += n
5:      result
6:    }else{
7:      0
8:    }
9:  }
10:
11: def main(args: Array[String]): Unit = {
12:   val result: Int = computeGeometricSum(3)
13:   println(result)
14: }
```

- Main continues with the result from the recursive calls



Lecture Question

Task: In a package name fp create a Scala object named ReturnFunction with a method named strangeAddition which

- Takes an Int as a parameter
- Return a function that takes an Int and returns an Int
- The returned function returns the addition of the two provided Ints
- Ex. Calling **val adder = strangeAddition(5)** will return a function that adds 5 to its input (**adder(3) == 8**)

* This question will be open until midnight