Model of Execution

Lecture Question

Question: in a package named "execution", implement the following classes:

- class Valuable
  - Constructor has no parameter
- class Trader
  - Constructor takes one variable as a parameter named "item" of type Valuable (use var)
  - This class represents someone with a valuable item they are wiling to trade. Each trader can only own 1 item at a time which is stored in the "item" state variable
- class TradeAgreement
  - Constructor takes 2 objects of type Trader
  - One method named "executeTrade" that take no parameters and returns Unit
    - When this method is called, swap the items belonging to the 2 traders from the constructor (ie. When a TradeAgreement is created, 2 people are agreeing to a trade. When executeTrade is called, they physically trade those items)
    - The agreement can only be executed once. If this method is called more than once for a single agreement, additional trades are not made (ie. The people do not trade back to their original items if this method is called twice)

Testing: In a package named "tests" create a Scala class named "TestTrading" as a test suite that tests all the functionality listed above

Note: The default behavior of == is to compare by reference. Comparing values of a type your created will return true only if the 2 values refer to the same object

More Memory Examples

- Multiple Objects on the heap

- Multiple frames on the stack

More Memory Examples

- Multiple Objects on the heap

```
def main(args: Array[String]): Unit =
{
  val bird: Bird = new Bird()
  var action: String = "Nothing"
  if(bird.inDanger()){
    val action: String = "Panic!"
  }else{
    val action: String = "Check bird"
  }
  println(action)
  val box: Box = new Box(bird, new
Bird())
  if(box.inDanger()){
    action = "Stay in the boat"
  }
  println(action)
}
```

```
class Bird {
  val timesHelpful: Int = 0
  var timesChecked: Int = 0

  def inDanger(): Boolean = {
    timesChecked += 1
    true
  }
}
```

```
class Box(val bird1: Bird, val bird2: Bird) {
  def inDanger(): Boolean = {
    bird1.inDanger() && bird2.inDanger()
  }
}
```

RAM
args
name:bird, value:42976

RAM @42976
Object of type Bird
-timesHelpful value:0
-timesChecked value:0

```
def main(args: Array[String]): Unit = {
  val bird: Bird = new Bird()
  var action: String = "Nothing"
  if(bird.inDanger()){
    val action: String = "Panic!"
  }else{
    val action: String = "Check bird"
  }
  println(action)
  val box: Box = new Box(bird, new Bird())
  if(box.inDanger()){
    action = "Stay in the boat"
  }
  println(action)
}
```

```
class Bird {
  val timesHelpful: Int = 0
  var timesChecked: Int = 0

  def inDanger(): Boolean = {
    timesChecked += 1
    true
  }
}
```

```
class Box(val bird1: Bird, val bird2: Bird) {
  def inDanger(): Boolean = {
    bird1.inDanger() && bird2.inDanger()
  }
}
```

- Start program with command line args on the stack
- Ask OS for heap space for 1 Bird

RAM
args
name:bird, value:42976
name:action, value:"Nothing"

RAM @42976
Object of type Bird
-timesHelpful value:0
-timesChecked value:0

```
def main(args: Array[String]): Unit = {
  val bird: Bird = new Bird()
  var action: String = "Nothing"
  if(bird.inDanger()){
    val action: String = "Panic!"
  }else{
    val action: String = "Check bird"
  }
  println(action)
  val box: Box = new Box(bird, new Bird())
  if(box.inDanger()){
    action = "Stay in the boat"
  }
  println(action)
}
```

```
class Bird {
  val timesHelpful: Int = 0
  var timesChecked: Int = 0

  def inDanger(): Boolean = {
    timesChecked += 1
    true
  }
}
```

```
class Box(val bird1: Bird, val bird2: Bird) {
  def inDanger(): Boolean = {
    bird1.inDanger() && bird2.inDanger()
  }
}
```

- Declare variable action
- Add to stack

RAM
args
name:bird, value:42976
name:action, value:"Nothing"
<new stack frame>

RAM @42976
Object of type Bird
-timesHelpful value:0
-timesChecked value:0

```
def main(args: Array[String]): Unit = {
  val bird: Bird = new Bird()
  var action: String = "Nothing"
  if(bird.inDanger()){
    val action: String = "Panic!"
  }else{
    val action: String = "Check bird"
  }
  println(action)
  val box: Box = new Box(bird, new Bird())
  if(box.inDanger()){
    action = "Stay in the boat"
  }
  println(action)
}
```

```
class Bird {
  val timesHelpful: Int = 0
  var timesChecked: Int = 0

  def inDanger(): Boolean = {
    timesChecked += 1
    true
  }
}
```

```
class Box(val bird1: Bird, val bird2: Bird) {
  def inDanger(): Boolean = {
    bird1.inDanger() && bird2.inDanger()
  }
}
```

- Call method
- Create new stack frame
- increment timesChecked

RAM
args
name:bird, value:42976
name:action, value:"Nothing"
<if block>
name:action, value:"Panic!"

RAM @42976
Object of type Bird
-timesHelpful value:0
-timesChecked value:1

```
def main(args: Array[String]): Unit = {
  val bird: Bird = new Bird()
  var action: String = "Nothing"
  if(bird.inDanger()){
    val action: String = "Panic!"
  }else{
    val action: String = "Check bird"
  }
  println(action)
  val box: Box = new Box(bird, new Bird())
  if(box.inDanger()){
    action = "Stay in the boat"
  }
  println(action)
}
```

```
class Bird {
  val timesHelpful: Int = 0
  var timesChecked: Int = 0

  def inDanger(): Boolean = {
    timesChecked += 1
    true
  }
}
```

```
class Box(val bird1: Bird, val bird2: Bird) {
  def inDanger(): Boolean = {
    bird1.inDanger() && bird2.inDanger()
  }
}
```

- Destroy stack frame
- Enter if block
- Declare value action

RAM
args
name:bird, value:42976
name:action, value:"Nothing"

RAM @42976
Object of type Bird
-timesHelpful value:0
-timesChecked value:1

```
def main(args: Array[String]): Unit = {
  val bird: Bird = new Bird()
  var action: String = "Nothing"
  if(bird.inDanger()){
    val action: String = "Panic!"
  }else{
    val action: String = "Check bird"
  }
  println(action)
  val box: Box = new Box(bird, new Bird())
  if(box.inDanger()){
    action = "Stay in the boat"
  }
  println(action)
}
```

```
class Bird {
  val timesHelpful: Int = 0
  var timesChecked: Int = 0

  def inDanger(): Boolean = {
    timesChecked += 1
    true
  }
}
```

```
class Box(val bird1: Bird, val bird2: Bird) {
  def inDanger(): Boolean = {
    bird1.inDanger() && bird2.inDanger()
  }
}
```

- End of if block
- Destroy block and action

RAM
args
name:bird, value:42976
name:action, value:"Nothing"

RAM @42976
Object of type Bird
-timesHelpful value:0
-timesChecked value:1

```
def main(args: Array[String]): Unit = {
  val bird: Bird = new Bird()
  var action: String = "Nothing"
  if(bird.inDanger()){
    val action: String = "Panic!"
  }else{
    val action: String = "Check bird"
  }
  println(action)
  val box: Box = new Box(bird, new Bird())
  if(box.inDanger()){
    action = "Stay in the boat"
  }
  println(action)
}
```

```
class Bird {
  val timesHelpful: Int = 0
  var timesChecked: Int = 0

  def inDanger(): Boolean = {
    timesChecked += 1
    true
  }
}
```

```
class Box(val bird1: Bird, val bird2: Bird) {
  def inDanger(): Boolean = {
    bird1.inDanger() && bird2.inDanger()
  }
}
```

- Print the string "Nothing"

RAM
args
name:bird, value:42976
name:action, value:"Nothing"

RAM @42976
Object of type Bird
-timesHelpful value:0
-timesChecked value:1

RAM @27177
Object of type Bird
-timesHelpful value:0
-timesChecked value:0

RAM @59683
Object of type Box
-bird1 value:42976
-bird2 value:27177

```
def main(args: Array[String]): Unit = {
  val bird: Bird = new Bird()
  var action: String = "Nothing"
  if(bird.inDanger()){
    val action: String = "Panic!"
  }else{
    val action: String = "Check bird"
  }
  println(action)
  val box: Box = new Box(bird, new Bird())
  if(box.inDanger()){
    action = "Stay in the boat"
  }
  println(action)
}
```

```
class Bird {
  val timesHelpful: Int = 0
  var timesChecked: Int = 0

  def inDanger(): Boolean = {
    timesChecked += 1
    true
  }
}
```

```
class Box(val bird1: Bird, val bird2: Bird) {
  def inDanger(): Boolean = {
    bird1.inDanger() && bird2.inDanger()
  }
}
```

- Ask OS for heap memory for
  - Another Bird
  - A Box

RAM
args
name:bird, value:42976
name:action, value:"Nothing"
name:box, value:59683

RAM @42976
Object of type Bird
-timesHelpful value:0
-timesChecked value:1

RAM @27177
Object of type Bird
-timesHelpful value:0
-timesChecked value:0

RAM @59683
Object of type Box
-bird1 value:42976
-bird2 value:27177

```
def main(args: Array[String]): Unit = {
  val bird: Bird = new Bird()
  var action: String = "Nothing"
  if(bird.inDanger()){
    val action: String = "Panic!"
  }else{
    val action: String = "Check bird"
  }
  println(action)
  val box: Box = new Box(bird, new Bird())
  if(box.inDanger()){
    action = "Stay in the boat"
  }
  println(action)
}
```

```
class Bird {
  val timesHelpful: Int = 0
  var timesChecked: Int = 0

  def inDanger(): Boolean = {
    timesChecked += 1
    true
  }
}
```

```
class Box(val bird1: Bird, val bird2: Bird) {
  def inDanger(): Boolean = {
    bird1.inDanger() && bird2.inDanger()
  }
}
```

- Store reference to Box in value box
- main method has no direct reference to the second Bird

RAM
args
name:bird, value:42976
name:action, value:"Nothing"
name:box, value:59683
<new stack frame box.inDanger>
<new stack frame bird1.inDanger>

RAM @42976
Object of type Bird
-timesHelpful value:0
-timesChecked value:1

RAM @27177
Object of type Bird
-timesHelpful value:0
-timesChecked value:0

RAM @59683
Object of type Box
-bird1 value:42976
-bird2 value:27177

```
def main(args: Array[String]): Unit = {
  val bird: Bird = new Bird()
  var action: String = "Nothing"
  if(bird.inDanger()){
    val action: String = "Panic!"
  }else{
    val action: String = "Check bird"
  }
  println(action)
  val box: Box = new Box(bird, new Bird())
  if(box.inDanger()){
    action = "Stay in the boat"
  }
  println(action)
}
```

```
class Bird {
  val timesHelpful: Int = 0
  var timesChecked: Int = 0

  def inDanger(): Boolean = {
    timesChecked += 1
    true
  }
}
```

- Create stack frame for box.inDanger call
- Create stack frame for bird1.inDanger

```
class Box(val bird1: Bird, val bird2: Bird) {
  def inDanger(): Boolean = {
    bird1.inDanger() && bird2.inDanger()
  }
}
```

RAM
args
name:bird, value:42976
name:action, value:"Nothing"
name:box, value:59683
<new stack frame box.inDanger>
<new stack frame bird2.inDanger>

RAM @42976
Object of type Bird
-timesHelpful value:0
-timesChecked value:2

RAM @27177
Object of type Bird
-timesHelpful value:0
-timesChecked value:0

RAM @59683
Object of type Box
-bird1 value:42976
-bird2 value:27177

```
def main(args: Array[String]): Unit = {
  val bird: Bird = new Bird()
  var action: String = "Nothing"
  if(bird.inDanger()){
    val action: String = "Panic!"
  }else{
    val action: String = "Check bird"
  }
  println(action)
  val box: Box = new Box(bird, new Bird())
  if(box.inDanger()){
    action = "Stay in the boat"
  }
  println(action)
}
```

```
class Bird {
  val timesHelpful: Int = 0
  var timesChecked: Int = 0

  def inDanger(): Boolean = {
    timesChecked += 1
    true
  }
}
```

```
class Box(val bird1: Bird, val bird2: Bird) {
  def inDanger(): Boolean = {
    bird1.inDanger() && bird2.inDanger()
  }
}
```

- Destroy stack frame for bird1.inDanger
- Create stack frame for bird2.inDanger

RAM
args
name:bird, value:42976
name:action, value:"Nothing"
name:box, value:59683
<if block>

RAM @42976
Object of type Bird
-timesHelpful value:0
-timesChecked value:2

RAM @27177
Object of type Bird
-timesHelpful value:0
-timesChecked value:1

RAM @59683
Object of type Box
-bird1 value:42976
-bird2 value:27177

```
def main(args: Array[String]): Unit = {
  val bird: Bird = new Bird()
  var action: String = "Nothing"
  if(bird.inDanger()){
    val action: String = "Panic!"
  }else{
    val action: String = "Check bird"
  }
  println(action)
  val box: Box = new Box(bird, new Bird())
  if(box.inDanger()){
    action = "Stay in the boat"
  }
  println(action)
}
```

```
class Bird {
  val timesHelpful: Int = 0
  var timesChecked: Int = 0

  def inDanger(): Boolean = {
    timesChecked += 1
    true
  }
}
```

```
class Box(val bird1: Bird, val bird2: Bird) {
  def inDanger(): Boolean = {
    bird1.inDanger() && bird2.inDanger()
  }
}
```

- Destroy stack frame for bird2.inDanger
- Enter if block
- Find action in outer scope

RAM
args
name:bird, value:42976
name:action, value:"Stay in the boat"
name:box, value:59683
<if block>

RAM @42976
Object of type Bird
-timesHelpful value:0
-timesChecked value:2

RAM @27177
Object of type Bird
-timesHelpful value:0
-timesChecked value:1

RAM @59683
Object of type Box
-bird1 value:42976
-bird2 value:27177

```
def main(args: Array[String]): Unit = {
  val bird: Bird = new Bird()
  var action: String = "Nothing"
  if(bird.inDanger()){
    val action: String = "Panic!"
  }else{
    val action: String = "Check bird"
  }
  println(action)
  val box: Box = new Box(bird, new Bird())
  if(box.inDanger()){
    action = "Stay in the boat"
  }
  println(action)
}
```

```
class Bird {
  val timesHelpful: Int = 0
  var timesChecked: Int = 0

  def inDanger(): Boolean = {
    timesChecked += 1
    true
  }
}
```

```
class Box(val bird1: Bird, val bird2: Bird) {
  def inDanger(): Boolean = {
    bird1.inDanger() && bird2.inDanger()
  }
}
```

- Destroy stack frame for bird2.inDanger
- Enter if block
- Find action in outer scope

RAM
args
name:bird, value:42976
name:action, value:"Stay in the boat"
name:box, value:59683

RAM @42976
Object of type Bird
-timesHelpful value:0
-timesChecked value:2

RAM @27177
Object of type Bird
-timesHelpful value:0
-timesChecked value:1

RAM @59683
Object of type Box
-bird1 value:42976
-bird2 value:27177

```
def main(args: Array[String]): Unit = {
  val bird: Bird = new Bird()
  var action: String = "Nothing"
  if(bird.inDanger()){
    val action: String = "Panic!"
  }else{
    val action: String = "Check bird"
  }
  println(action)
  val box: Box = new Box(bird, new Bird())
  if(box.inDanger()){
    action = "Stay in the boat"
  }
  println(action)
}
```

```
class Bird {
  val timesHelpful: Int = 0
  var timesChecked: Int = 0

  def inDanger(): Boolean = {
    timesChecked += 1
    true
  }
}
```

```
class Box(val bird1: Bird, val bird2: Bird) {
  def inDanger(): Boolean = {
    bird1.inDanger() && bird2.inDanger()
  }
}
```

- Destroy if block
- print "Stay in the boat"

RAM

RAM @42976

RAM @27177

RAM @59683

```
def main(args: Array[String]): Unit = {
  val bird: Bird = new Bird()
  var action: String = "Nothing"
  if(bird.inDanger()){
    val action: String = "Panic!"
  }else{
    val action: String = "Check bird"
  }
  println(action)
  val box: Box = new Box(bird, new Bird())
  if(box.inDanger()){
    action = "Stay in the boat"
  }
  println(action)
}
```

```
class Bird {
  val timesHelpful: Int = 0
  var timesChecked: Int = 0

  def inDanger(): Boolean = {
    timesChecked += 1
    true
  }
}
```

```
class Box(val bird1: Bird, val bird2: Bird) {
  def inDanger(): Boolean = {
    bird1.inDanger() && bird2.inDanger()
  }
}
```

- Program ends
- Free all memory

More Memory Examples

- Multiple frames on the stack

```
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

```
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)
}
```

- Call function
- Create new stack frame

RAM
args
<new stack frame>
name:n, value:3
<Used by another program>
<Used by another program>

Recursive 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)
}
```

- Enter if block
- Call function again
- Create new stack frame

RAM
args
<new stack frame>
name:n, value:3
<if block>
<new stack frame>
name:n, value:2
<Used by another program>
<Used by another program>

Recursive 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)
}
```

- In next function call, conditional true
- New if block
- New stack frame

RAM
args
<new stack frame>
name:n, value:3
<if block>
<new stack frame>
name:n, value:2
<if block>
<new stack frame>
name:n, value:1
<Used by another program>
<Used by another program>

Recursive 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)
}
```

RAM
args
<new stack frame>
name:n, value:3
<if block>
<new stack frame>
name:n, value:2
<if block>
<new stack frame>
name:n, value:1
<if block>
<new stack frame>
name:n, value:0
<Used by another program>
<Used by another program>

- Repeat, repeat
- Many variables named n on the stack
- Each is in different frame so it's ok

Recursive 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)
}
```

- Conditional finally false
- return 0

RAM
args
<new stack frame>
name:n, value:3
<if block>
<new stack frame>
name:n, value:2
<if block>
<new stack frame>
name:n, value:1
<if block>
<new stack frame>
name:n, value:0
<Used by another program>
<Used by another program>

Recursive 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)
}
```

- Assign return value to result

RAM
args
<new stack frame>
name:n, value:3
<if block>
<new stack frame>
name:n, value:2
<if block>
<new stack frame>
name:n, value:1
<if block>
name:result, value:0
<Used by another program>
<Used by another program>

Recursive 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)
}
```

- Add value of the n in this stack frame to result
- result is the last expression and is returned

RAM
args
<new stack frame>
name:n, value:3
<if block>
<new stack frame>
name:n, value:2
<if block>
<new stack frame>
name:n, value:1
<if block>
name:result, value:1
<Used by another program>
<Used by another program>

Recursive 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)
}
```

RAM
args
<new stack frame>
name:n, value:3
<if block>
<new stack frame>
name:n, value:2
<if block>
name:result, value:1
<Used by another program>
<Used by another program>

- Return to function call from previous frame
- Store return value in result

Recursive 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)
}
```

- Add value of n from this frame..
- Repeat

RAM
args
<new stack frame>
name:n, value:3
<if block>
<new stack frame>
name:n, value:2
<if block>
name:result, value:3
<Used by another program>
<Used by another program>

Recursive 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)
}
```

- Add value of n from this frame..
- Repeat

RAM
args
<new stack frame>
name:n, value:3
<if block>
name:result, value:3
<Used by another program>
<Used by another program>

Recursive 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)
}
```

RAM
args
<new stack frame>
name:n, value:3
<if block>
name:result, value:6
<Used by another program>
<Used by another program>

- And repeat..
- Imagine if the original input were 1000
  - This is why we use computers

Recursive 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)
}
```

- Value result in main method gets the last return value

RAM
args
name:result, value:6
<Used by another program>
<Used by another program>

Recursive 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)
}
```

- print 6

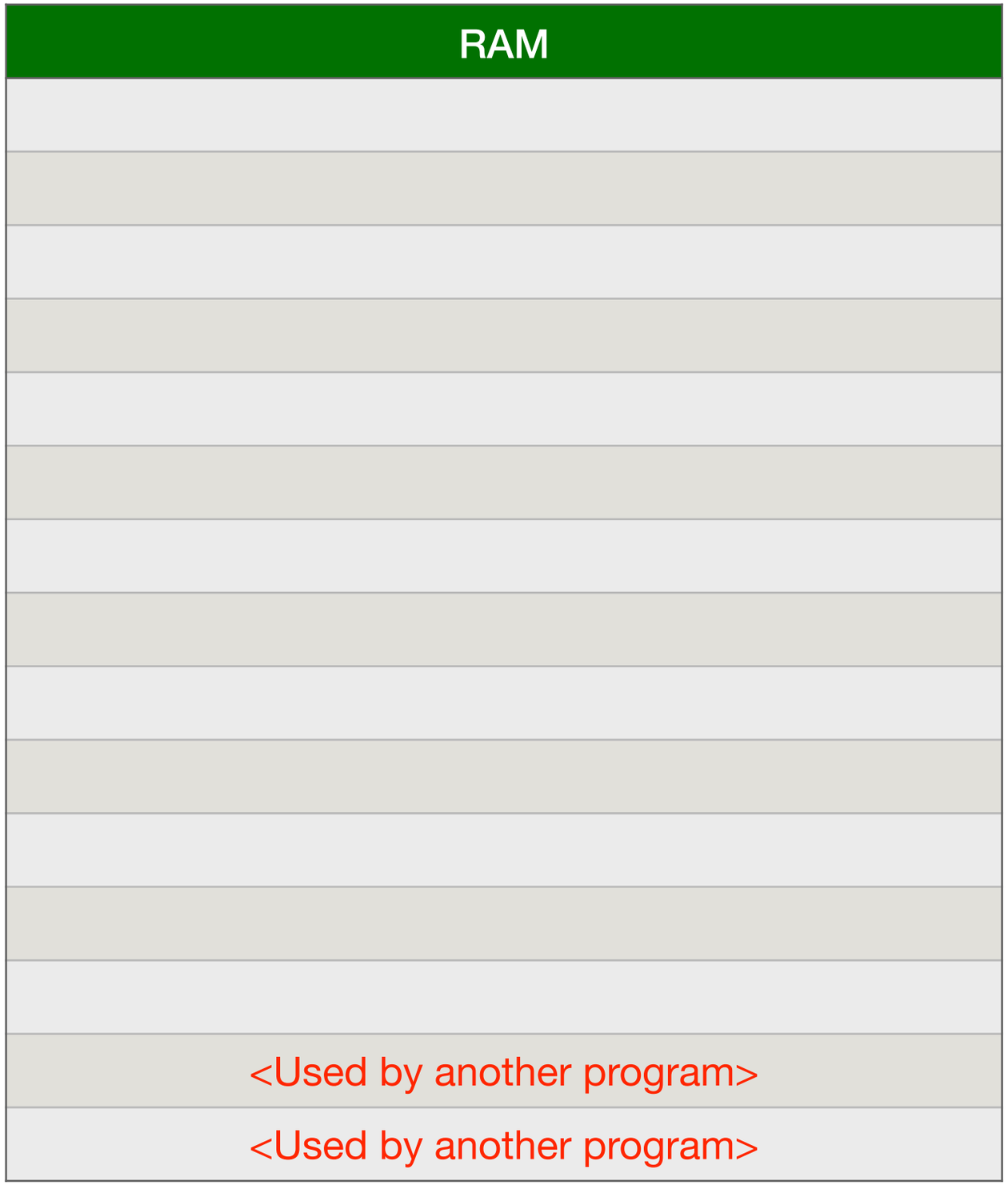
RAM
args
name:result, value:6
<Used by another program>
<Used by another program>

Recursive 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)
}
```

- Free memory



More Memory Examples

- We were close to the end of the stack on that example
- What if this were our code?

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

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

Recursive Example

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

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

RAM
args
<new stack frame>
name:n, value:3
<if block>
<new stack frame>
name:n, value:2
<if block>
<new stack frame>
name:n, value:1
<if block>
<new stack frame>
name:n, value:0
<Used by another program>
<Used by another program>

- At this point the other program was going to return 0 and return back up the stack

Recursive Example

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

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

RAM
args
<new stack frame>
name:n, value:3
<new stack frame>
name:n, value:2
<new stack frame>
name:n, value:1
<new stack frame>
name:n, value:0
<new stack frame>
name:n, value:-1
<new stack frame>
name:n, value:-2
<Used by another program>
<Used by another program>

- This program keeps adding frames to the stack

Recursive Example

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

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

- STACK OVERFLOW
- Program crashes

RAM
args
<new stack frame>
name:n, value:3
<new stack frame>
name:n, value:2
<new stack frame>
name:n, value:1
<new stack frame>
name:n, value:0
<new stack frame>
name:n, value:-1
<new stack frame>
name:n, value:-2
<Used by another program> <new stack frame>
<Used by another program> name:n, value:-3

Lecture Question

Question: in a package named "execution", implement the following classes:

- class Valuable
  - Constructor has no parameter
- class Trader
  - Constructor takes one variable as a parameter named "item" of type Valuable (use var)
  - This class represents someone with a valuable item they are wiling to trade. Each trader can only own 1 item at a time which is stored in the "item" state variable
- class TradeAgreement
  - Constructor takes 2 objects of type Trader
  - One method named "executeTrade" that take no parameters and returns Unit
    - When this method is called, swap the items belonging to the 2 traders from the constructor (ie. When a TradeAgreement is created, 2 people are agreeing to a trade. When executeTrade is called, they physically trade those items)
    - The agreement can only be executed once. If this method is called more than once for a single agreement, additional trades are not made (ie. The people do not trade back to their original items if this method is called twice)

Testing: In a package named "tests" create a Scala class named "TestTrading" as a test suite that tests all the functionality listed above

Note: The default behavior of == is to compare by reference. Comparing values of a type your created will return true only if the 2 values refer to the same object