Lecture Question

Question: Submit any file to AutoLab

Grading code today:

```
#!/bin/bash

echo '{"scores": {"score":10}}'
```

Full 50 minute lecture

* This question will be open until midnight (You shouldn't need that much time today)

Model of Execution

Interpretation v. Compilation

- Interpretation
  - Code is read and executed one statement at a time
- Compilation
  - Entire program is translated into another language
  - The translated code is interpreted

Interpretation

- Python and JavaScript are interpreted languages
- Run-time errors common
    - Program runs, but crashes when a line with an error is interpreted

This program runs without error

```
class RuntimeErrorExample:

    def __init__(self, initial_state):
        self.state = initial_state

    def add_to_state(self, to_add):
        print("adding to state")
        self.state += to_add


if __name__ == '__main__':
    example_object = RuntimeErrorExample(5)
    example_object.add_to_state(10)
    print(example_object.state)
```

This program crashes with runtime error

```
class RuntimeErrorExample:

    def __init__(self, initial_state):
        self.state = initial_state

    def add_to_state(self, to_add):
        print("adding to state")
        self.state += to_add


if __name__ == '__main__':
    example_object = RuntimeErrorExample(5)
    example_object.add_to_state("ten")
    print(example_object.state)
```

Compilation

- Scala, Java, C++ are compiled languages
- Compiler errors common
    - Program fails to be converted into the target language
    - Program never runs
    - Can make debugging much easier

Compiles and runs without error

```
class CompilerError(var state: Int) {

  def addToState(toAdd: Int): Unit ={
    this.state += toAdd
  }

}

object Main {
  def main(args: Array[String]): Unit = {
    val exampleObject = new CompilerError(5)
    exampleObject.addToState(10)
    println(exampleObject.state)
  }
}
```

Does not compile. Will not run any code

```
class CompilerError(var state: Int) {

  def addToState(toAdd: Int): Unit ={
    this.state += toAdd
  }

}

object Main {
  def main(args: Array[String]): Unit = {
    val exampleObject = new CompilerError(5)
    exampleObject.addToState("ten")
    println(exampleObject.state)
  }
}
```

Compilation

- Compilers produce efficient code
  - While translating the compiler "fixes" our code whenever it can
  - Compilers are very smart!
- Can even fix some major errors

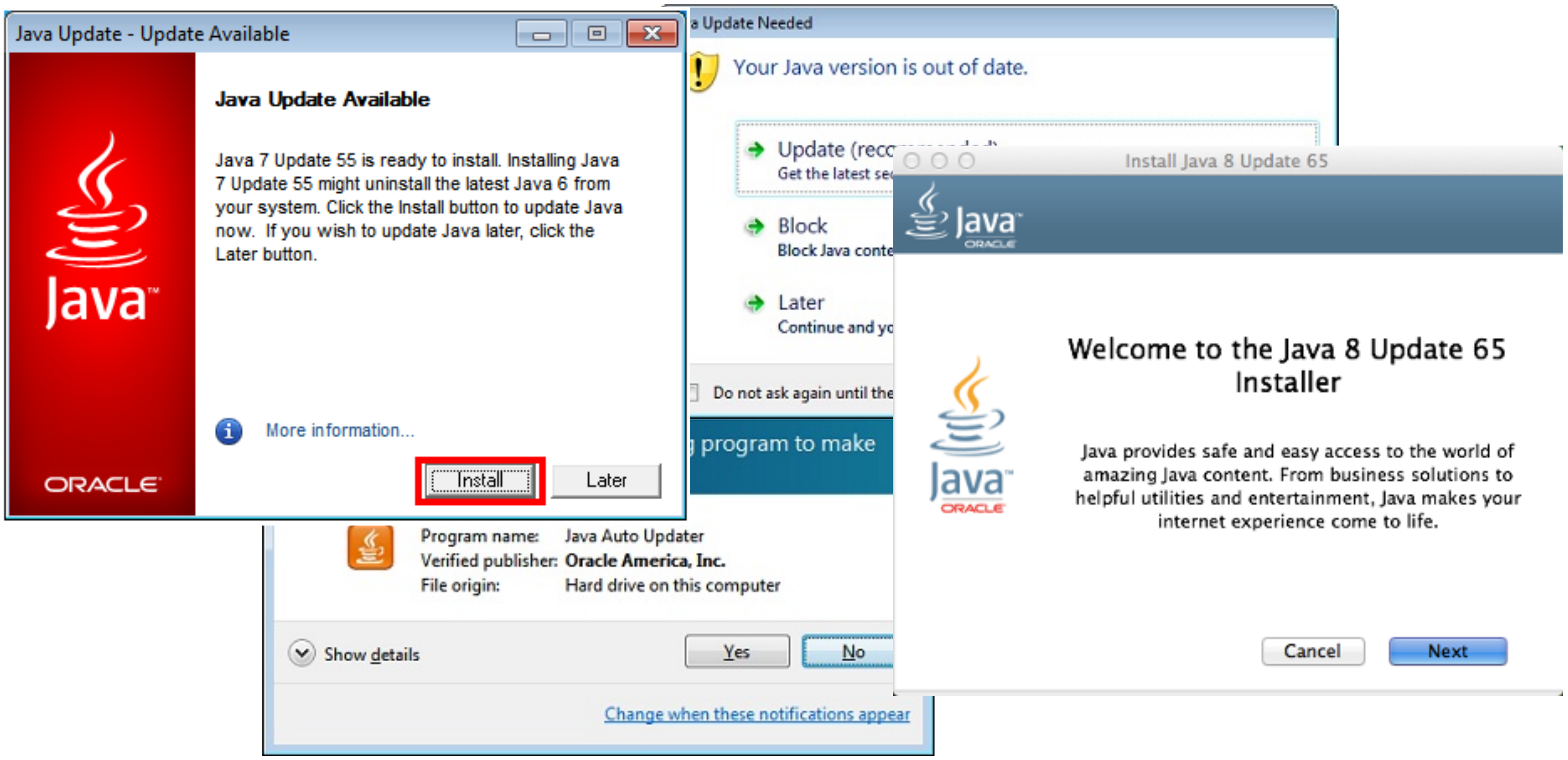
This code runs.. forever, but it doesn't overflow the stack. Thanks compiler!

```scala
object StackFlow {

  def recursiveFunction(n: Int): Int ={
    recursiveFunction(n)
  }

  def main(args: Array[String]): Unit = {
    recursiveFunction(1)
  }

}
```

***If you are interested in more details about this example, search for Tail Recursion**

Compilation - Scala

- Scala compiles to Java Byte Code
- Executed by the Java Virtual Machine (JVM)
  - Installed on Billions of devices!



Java Update - Update Available

Java Update Available

Java 7 Update 55 is ready to install. Installing Java 7 Update 55 might uninstall the latest Java 6 from your system. Click the Install button to update Java now. If you wish to update Java later, click the Later button.

a Update Needed

Your Java version is out of date.

Program name: Java Auto Updater
Verified publisher: Oracle America, Inc.
File origin: Hard drive on this computer

Install Java 8 Update 65

Welcome to the Java 8 Update 65 Installer

Java provides safe and easy access to the world of amazing Java content. From business solutions to helpful utilities and entertainment, Java makes your internet experience come to life.

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()
  }
}
```

Index	Value
81078	args
81079	name:bird, value:42976
81080	
81081	
81082	
81083	
81084	
81085	

```
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)
}
```

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

Index	Value
42976	Object of type Bird
42977	-timesHelpful value:0
42978	-timesChecked value:0

Index	Value
27177	
27178	
27179	

Index	Value
59683	
59684	
59685	

```
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()
  }
}
```

Index	Value
81078	args
81079	name:bird, value:42976
81080	name:action, value:"Nothing"
81081	
81082	
81083	
81084	
81085	

Index	Value
42976	Object of type Bird
42977	-timesHelpful value:0
42978	-timesChecked value:0

Index	Value
27177	
27178	
27179	

Index	Value
59683	
59684	
59685	

```
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

Index	Value
81078	args
81079	name:bird, value:42976
81080	name:action, value:"Nothing"
81081	<new stack frame>
81082	
81083	
81084	
81085	

Index	Value
42976	Object of type Bird
42977	-timesHelpful value:0
42978	-timesChecked value:0

Index	Value
27177	
27178	
27179	

Index	Value
59683	
59684	
59685	

```
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

Index	Value
81078	args
81079	name:bird, value:42976
81080	name:action, value:"Nothing"
81081	<if block>
81082	name:action, value:"Panic!"
81083	
81084	
81085	

Index	Value
42976	Object of type Bird
42977	-timesHelpful value:0
42978	-timesChecked value:1

Index	Value
27177	
27178	
27179	

Index	Value
59683	
59684	
59685	

```
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

Index	Value
81078	args
81079	name:bird, value:42976
81080	name:action, value:"Nothing"
81081	
81082	
81083	
81084	
81085	

Index	Value
42976	Object of type Bird
42977	-timesHelpful value:0
42978	-timesChecked value:1

Index	Value
27177	
27178	
27179	

Index	Value
59683	
59684	
59685	

```scala
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)
}
```

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

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

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

- End of if block
- Destroy block and action

Index	Value
81078	args
81079	name:bird, value:42976
81080	name:action, value:"Nothing"
81081	
81082	
81083	
81084	
81085	

Index	Value
42976	Object of type Bird
42977	-timesHelpful value:0
42978	-timesChecked value:1

Index	Value
27177	
27178	
27179	

Index	Value
59683	
59684	
59685	

```
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"

Index	Value
81078	args
81079	name:bird, value:42976
81080	name:action, value:"Nothing"
81081	
81082	
81083	
81084	
81085	

```
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)
}
```

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

Index	Value
42976	Object of type Bird
42977	-timesHelpful value:0
42978	-timesChecked value:1

Index	Value
27177	Object of type Bird
27178	-timesHelpful value:0
27179	-timesChecked value:0

Index	Value
59683	Object of type Box
59684	-bird1 value:42976
59685	-bird2 value:27177

```
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()
  }
}
```

Index	Value
81078	args
81079	name:bird, value:42976
81080	name:action, value:"Nothing"
81081	name:box, value:59683
81082	
81083	
81084	
81085	

Index	Value
42976	Object of type Bird
42977	-timesHelpful value:0
42978	-timesChecked value:1

Index	Value
27177	Object of type Bird
27178	-timesHelpful value:0
27179	-timesChecked value:0

Index	Value
59683	Object of type Box
59684	-bird1 value:42976
59685	-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 reference to the second Bird

Index	Value
81078	args
81079	name:bird, value:42976
81080	name:action, value:"Nothing"
81081	name:box, value:59683
81082	<new stack frame box.inDanger>
81083	<new stack frame bird1.inDanger>
81084	
81085	

```
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)
}
```

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

Index	Value
42976	Object of type Bird
42977	-timesHelpful value:0
42978	-timesChecked value:1

Index	Value
27177	Object of type Bird
27178	-timesHelpful value:0
27179	-timesChecked value:0

Index	Value
59683	Object of type Box
59684	-bird1 value:42976
59685	-bird2 value:27177

```
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()
  }
}
```

Index	Value
81078	args
81079	name:bird, value:42976
81080	name:action, value:"Nothing"
81081	name:box, value:59683
81082	<new stack frame box.inDanger>
81083	<new stack frame bird2.inDanger>
81084	
81085	

```
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)
}
```

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

Index	Value
42976	Object of type Bird
42977	-timesHelpful value:0
42978	-timesChecked value:2

Index	Value
27177	Object of type Bird
27178	-timesHelpful value:0
27179	-timesChecked value:0

Index	Value
59683	Object of type Box
59684	-bird1 value:42976
59685	-bird2 value:27177

```
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()
  }
}
```

Index	Value
81078	args
81079	name:bird, value:42976
81080	name:action, value:"Nothing"
81081	name:box, value:59683
81082	<if block>
81083	
81084	
81085	

Index	Value
42976	Object of type Bird
42977	-timesHelpful value:0
42978	-timesChecked value:2

Index	Value
27177	Object of type Bird
27178	-timesHelpful value:0
27179	-timesChecked value:1

Index	Value
59683	Object of type Box
59684	-bird1 value:42976
59685	-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

Index	Value
81078	args
81079	name:bird, value:42976
81080	name:action, value:"Stay in the boat"
81081	name:box, value:59683
81082	<if block>
81083	
81084	
81085	

```scala
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)
}
```

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

Index	Value
42976	Object of type Bird
42977	-timesHelpful value:0
42978	-timesChecked value:2

Index	Value
27177	Object of type Bird
27178	-timesHelpful value:0
27179	-timesChecked value:1

Index	Value
59683	Object of type Box
59684	-bird1 value:42976
59685	-bird2 value:27177

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

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

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

Index	Value
81078	args
81079	name:bird, value:42976
81080	name:action, value:"Stay in the boat"
81081	name:box, value:59683
81082	
81083	
81084	
81085	

Index	Value
42976	Object of type Bird
42977	-timesHelpful value:0
42978	-timesChecked value:2

Index	Value
27177	Object of type Bird
27178	-timesHelpful value:0
27179	-timesChecked value:1

Index	Value
59683	Object of type Box
59684	-bird1 value:42976
59685	-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"

Index	Value
81078	
81079	
81080	
81081	
81082	
81083	
81084	
81085	

Index	Value
42976	
42977	
42978	

Index	Value
27177	
27178	
27179	

Index	Value
59683	
59684	
59685	

```
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)
}
```

- Program ends
- Free all memory

```
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()
  }
}
```

More Memory Examples

- Multiple frames on the stack

```scala
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

Index	Value
96437	args
96438	<new stack frame>
96439	name:n, value:3
96440	
96441	
96442	
96443	
96444	
96445	
96446	
96447	
96448	
96449	
96450	<Used by another program>
96451	<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

Index	Value
96437	args
96438	<new stack frame>
96439	name:n, value:3
96440	<if block>
96441	<new stack frame>
96442	name:n, value:2
96443	
96444	
96445	
96446	
96447	
96448	
96449	
96450	<Used by another program>
96451	<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

Index	Value
96437	args
96438	<new stack frame>
96439	name:n, value:3
96440	<if block>
96441	<new stack frame>
96442	name:n, value:2
96443	<if block>
96444	<new stack frame>
96445	name:n, value:1
96446	
96447	
96448	
96449	
96450	<Used by another program>
96451	<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)
}
```

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

Index	Value
96437	args
96438	<new stack frame>
96439	name:n, value:3
96440	<if block>
96441	<new stack frame>
96442	name:n, value:2
96443	<if block>
96444	<new stack frame>
96445	name:n, value:1
96446	<if block>
96447	<new stack frame>
96448	name:n, value:0
96449	
96450	<Used by another program>
96451	<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)
}
```

- Conditional finally false
- return 0

Index	Value
96437	args
96438	<new stack frame>
96439	name:n, value:3
96440	<if block>
96441	<new stack frame>
96442	name:n, value:2
96443	<if block>
96444	<new stack frame>
96445	name:n, value:1
96446	<if block>
96447	<new stack frame>
96448	name:n, value:0
96449	
96450	<Used by another program>
96451	<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

Index	Value
96437	args
96438	<new stack frame>
96439	name:n, value:3
96440	<if block>
96441	<new stack frame>
96442	name:n, value:2
96443	<if block>
96444	<new stack frame>
96445	name:n, value:1
96446	<if block>
96447	name:result, value:0
96448	
96449	
96450	<Used by another program>
96451	<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

Index	Value
96437	args
96438	<new stack frame>
96439	name:n, value:3
96440	<if block>
96441	<new stack frame>
96442	name:n, value:2
96443	<if block>
96444	<new stack frame>
96445	name:n, value:1
96446	<if block>
96447	name:result, value:1
96448	
96449	
96450	<Used by another program>
96451	<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)
}
```

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

Index	Value
96437	args
96438	<new stack frame>
96439	name:n, value:3
96440	<if block>
96441	<new stack frame>
96442	name:n, value:2
96443	<if block>
96444	name:result, value:1
96445	
96446	
96447	
96448	
96449	
96450	<Used by another program>
96451	<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

Index	Value
96437	args
96438	<new stack frame>
96439	name:n, value:3
96440	<if block>
96441	<new stack frame>
96442	name:n, value:2
96443	<if block>
96444	name:result, value:3
96445	
96446	
96447	
96448	
96449	
96450	<Used by another program>
96451	<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

Index	Value
96437	args
96438	<new stack frame>
96439	name:n, value:3
96440	<if block>
96441	name:result, value:3
96442	
96443	
96444	
96445	
96446	
96447	
96448	
96449	
96450	<Used by another program>
96451	<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)
}
```

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

Index	Value
96437	args
96438	<new stack frame>
96439	name:n, value:3
96440	<if block>
96441	name:result, value:6
96442	
96443	
96444	
96445	
96446	
96447	
96448	
96449	
96450	<Used by another program>
96451	<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)
}
```

- Value result in main method gets the last return value

Index	Value
96437	args
96438	name:result, value:6
96439	
96440	
96441	
96442	
96443	
96444	
96445	
96446	
96447	
96448	
96449	
96450	<Used by another program>
96451	<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

Index	Value
96437	args
96438	name:result, value:6
96439	
96440	
96441	
96442	
96443	
96444	
96445	
96446	
96447	
96448	
96449	
96450	<Used by another program>
96451	<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

Index	Value
96437	
96438	
96439	
96440	
96441	
96442	
96443	
96444	
96445	
96446	
96447	
96448	
96449	
96450	<Used by another program>
96451	<Used by another program>

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)
}
```

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

Index	Value
96437	args
96438	<new stack frame>
96439	name:n, value:3
96440	<if block>
96441	<new stack frame>
96442	name:n, value:2
96443	<if block>
96444	<new stack frame>
96445	name:n, value:1
96446	<if block>
96447	<new stack frame>
96448	name:n, value:0
96449	
96450	<Used by another program>
96451	<Used by another program>

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)
}
```

- This program keeps adding frames to the stack

Index	Value
96437	args
96438	<new stack frame>
96439	name:n, value:3
96440	<new stack frame>
96441	name:n, value:2
96442	<new stack frame>
96443	name:n, value:1
96444	<new stack frame>
96445	name:n, value:0
96446	<new stack frame>
96447	name:n, value:-1
96448	<new stack frame>
96449	name:n, value:-2
96450	<Used by another program>
96451	<Used by another program>

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

Index	Value
96437	args
96438	<new stack frame>
96439	name:n, value:3
96440	<new stack frame>
96441	name:n, value:2
96442	<new stack frame>
96443	name:n, value:1
96444	<new stack frame>
96445	name:n, value:0
96446	<new stack frame>
96447	name:n, value:-1
96448	<new stack frame>
96449	name:n, value:-2
96450	<Used by another program> <new stack frame>
96451	<Used by another program> name:n, value:-3

Debugger Example