Model of Execution

Lecture Question

Question: In a package named "execution" create a Scala **class** named "Team" and a Scala **object** named "Referee".

Team will have:

- Two state values of type Int representing the strength of the team's offense and defense with a constructor to set these values. The parameters for the constructor should be offense first, then defense (Note: These values do not have defined names in this question so you cannot access them in your testing. If I use "teamOffense" and you name it "offense" and access it in your tests, your tests will crash when testing my code since the variable "offense" will not exist)
- A third state variable of type Int that is not in the constructor that represents the score of the team, is declared as a **var**, and is initialized to 0 (This variable also has not defined name)

Referee will have:

- A method named "playGame" that takes two Team objects as parameters and return type Unit. This method will alter the state of each input Team by setting their scores equal to their offense minus the other Team's defense. If a Team's offense is less than the other Team's defense their score should be 0 (no negative scores)
- A method named "declareWinner" that takes two Teams as parameters and returns the Team with the higher score. If both Teams have the same score, return a **new** Team object to indicate that neither competing team won (You may choose any values in the constructor call of this new Team)

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

More Memory Examples

- Multiple Objects on the heap

- Multiple frames on the stack

More Memory Examples

Multiple Objects on the heap

```
def winBattle(character: PartyCharacter, xp: Int): Unit = {
  character.battlesWon += 1
  character.experiencePoints += xp
}
```

```
def main(args: Array[String]): Unit = {
  val mobXP: Int = 20
  val bossXP: Int = 100
  val hero: PartyCharacter = new PartyCharacter()
  winBattle(hero, mobXP)
  val party: Party = new Party(hero, new PartyCharacter())
  party.winBattle(bossXP)
  winBattle(party.characterOne, mobXP)
  winBattle(party.characterTwo, mobXP)
}
```

```
class PartyCharacter() {
  var battlesWon: Int = 0
  var experiencePoints: Int = 0
}
```

```
class Party(val characterOne: PartyCharacter,
            val characterTwo: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.characterOne.battlesWon += 1
    this.characterTwo.battlesWon += 1
    this.characterOne.experiencePoints += xp
    this.characterTwo.experiencePoints += xp
  }

}
```

- Start at the beginning of the main method
- Command line args are on the stack

Stack
args

```
class PartyCharacter() {
  var battlesWon: Int = 0
  var experiencePoints: Int = 0
}
```

```
class Party(val characterOne: PartyCharacter,
            val characterTwo: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.characterOne.battlesWon += 1
    this.characterTwo.battlesWon += 1
    this.characterOne.experiencePoints += xp
    this.characterTwo.experiencePoints += xp
  }

}
```

```
def winBattle(character: PartyCharacter, xp: Int): Unit = {
  character.battlesWon += 1
  character.experiencePoints += xp
}
```

```
def main(args: Array[String]): Unit = {
  val mobXP: Int = 20
  val bossXP: Int = 100
  val hero: PartyCharacter = new PartyCharacter()
  winBattle(hero, mobXP)
  val party: Party = new Party(hero, new PartyCharacter())
  party.winBattle(bossXP)
  winBattle(party.characterOne, mobXP)
  winBattle(party.characterTwo, mobXP)
}
```

- Add the value mobXP to the stack with a value of 20
- Add the value bossXP to the stack with a value of 100

Stack
args
name: mobXP, value: 20
name: bossXP, value: 100

```
class PartyCharacter() {
  var battlesWon: Int = 0
  var experiencePoints: Int = 0
}
```

```
class Party(val characterOne: PartyCharacter,
            val characterTwo: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.characterOne.battlesWon += 1
    this.characterTwo.battlesWon += 1
    this.characterOne.experiencePoints += xp
    this.characterTwo.experiencePoints += xp
  }

}
```

```
def winBattle(character: PartyCharacter, xp: Int): Unit = {
  character.battlesWon += 1
  character.experiencePoints += xp
}
```

```
def main(args: Array[String]): Unit = {
  val mobXP: Int = 20
  val bossXP: Int = 100
  val hero: PartyCharacter = new PartyCharacter()
  winBattle(hero, mobXP)
  val party: Party = new Party(hero, new PartyCharacter())
  party.winBattle(bossXP)
  winBattle(party.characterOne, mobXP)
  winBattle(party.characterTwo, mobXP)
}
```

- A **new** object of type PartyCharacter is created
  - Ask OS/JVM for enough heap space to store the object
  - OS/JVM gives us a reference to this location in heap space
- **The "hero" value only stores this reference**

Heap @428
Object of type PartyCharacter
-battlesWon value: 0
-experiencePoints value: 0

Stack
args
name: mobXP, value: 20
name: bossXP, value: 100
name: hero, value: @428

```
class PartyCharacter() {
  var battlesWon: Int = 0
  var experiencePoints: Int = 0
}
```

```
def winBattle(character: PartyCharacter, xp: Int): Unit = {
  character.battlesWon += 1
  character.experiencePoints += xp
}
```

```
class Party(val characterOne: PartyCharacter,
            val characterTwo: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.characterOne.battlesWon += 1
    this.characterTwo.battlesWon += 1
    this.characterOne.experiencePoints += xp
    this.characterTwo.experiencePoints += xp
  }

}
```

```
def main(args: Array[String]): Unit = {
  val mobXP: Int = 20
  val bossXP: Int = 100
  val hero: PartyCharacter = new PartyCharacter()
  winBattle(hero, mobXP)
  val party: Party = new Party(hero, new PartyCharacter())
  party.winBattle(bossXP)
  winBattle(party.characterOne, mobXP)
  winBattle(party.characterTwo, mobXP)
}
```

- "winBattle" is called
  - A new stack frame is created
  - Parameters are added to the stack with values equal to the arguments
- **Only a reference of PartyCharacter is passed!**

Heap @428
Object of type PartyCharacter
-battlesWon value: 0
-experiencePoints value: 0

Stack
args
name: mobXP, value: 20
name: bossXP, value: 100
name: hero, value: @428
<begin "winBattle" stack frame>
name: character, value: @428
name: xp, value: 20

```
class PartyCharacter() {
  var battlesWon: Int = 0
  var experiencePoints: Int = 0
}
```

```
class Party(val characterOne: PartyCharacter,
            val characterTwo: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.characterOne.battlesWon += 1
    this.characterTwo.battlesWon += 1
    this.characterOne.experiencePoints += xp
    this.characterTwo.experiencePoints += xp
  }

}
```

```
def winBattle(character: PartyCharacter, xp: Int): Unit = {
  character.battlesWon += 1
  character.experiencePoints += xp
}
```

```
def main(args: Array[String]): Unit = {
  val mobXP: Int = 20
  val bossXP: Int = 100
  val hero: PartyCharacter = new PartyCharacter()
  winBattle(hero, mobXP)
  val party: Party = new Party(hero, new PartyCharacter())
  party.winBattle(bossXP)
  winBattle(party.characterOne, mobXP)
  winBattle(party.characterTwo, mobXP)
}
```

- "character" stores the reference @428
- The "." (dot operator) navigates to @428 in memory
- Modify the variables found at this location on the heap

Heap @428
Object of type PartyCharacter
-battlesWon value: 1
-experiencePoints value: 20

Stack
args
name: mobXP, value: 20
name: bossXP, value: 100
name: hero, value: @428
<begin "winBattle" stack frame>
name: character, value: @428
name: xp, value: 20

```
class PartyCharacter() {
  var battlesWon: Int = 0
  var experiencePoints: Int = 0
}
```

```
def winBattle(character: PartyCharacter, xp: Int): Unit = {
  character.battlesWon += 1
  character.experiencePoints += xp
}
```

```
def main(args: Array[String]): Unit = {
  val mobXP: Int = 20
  val bossXP: Int = 100
  val hero: PartyCharacter = new PartyCharacter()
  winBattle(hero, mobXP)
  val party: Party = new Party(hero, new PartyCharacter())
  party.winBattle(bossXP)
  winBattle(party.characterOne, mobXP)
  winBattle(party.characterTwo, mobXP)
}
```

```
class Party(val characterOne: PartyCharacter,
            val characterTwo: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.characterOne.battlesWon += 1
    this.characterTwo.battlesWon += 1
    this.characterOne.experiencePoints += xp
    this.characterTwo.experiencePoints += xp
  }

}
```

- The method returns
- The stack frame and all values in it are destroyed
- **But the changes made in the heap persist!**
- Method returns Unit, but does affect the state of memory

Heap @428
Object of type PartyCharacter
-battlesWon value: 1
-experiencePoints value: 20

Stack
args
name: mobXP, value: 20
name: bossXP, value: 100
name: hero, value: @428

```
class PartyCharacter() {
  var battlesWon: Int = 0
  var experiencePoints: Int = 0
}
```

```
class Party(val characterOne: PartyCharacter,
            val characterTwo: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.characterOne.battlesWon += 1
    this.characterTwo.battlesWon += 1
    this.characterOne.experiencePoints += xp
    this.characterTwo.experiencePoints += xp
  }

}
```

```
def winBattle(character: PartyCharacter, xp: Int): Unit = {
  character.battlesWon += 1
  character.experiencePoints += xp
}
```

```
def main(args: Array[String]): Unit = {
  val mobXP: Int = 20
  val bossXP: Int = 100
  val hero: PartyCharacter = new PartyCharacter()
  winBattle(hero, mobXP)
  val party: Party = new Party(hero, new PartyCharacter())
  party.winBattle(bossXP)
  winBattle(party.characterOne, mobXP)
  winBattle(party.characterTwo, mobXP)
}
```

- Create a **new** PartyCharacter on the heap
  - Reference is not stored on the stack
  - Only "party" will be able to access this object
- Create a **new** Party on the heap
  - Store reference in the main stack frame

Heap @428
Object of type PartyCharacter
-battlesWon value: 1
-experiencePoints value: 20

Heap @272
Object of type PartyCharacter
-battlesWon value: 0
-experiencePoints value: 0

Heap @596
Object of type Party
-characterOne value: @428
-characterTwo value: @272
-battlesWon value: 0

Stack
args
name: mobXP, value: 20
name: bossXP, value: 100
name: hero, value: @428
name: party, value: @596

```
def winBattle(character: PartyCharacter, xp: Int): Unit = {
  character.battlesWon += 1
  character.experiencePoints += xp
}
```

```
def main(args: Array[String]): Unit = {
  val mobXP: Int = 20
  val bossXP: Int = 100
  val hero: PartyCharacter = new PartyCharacter()
  winBattle(hero, mobXP)
  val party: Party = new Party(hero, new PartyCharacter())
  party.winBattle(bossXP)
  winBattle(party.characterOne, mobXP)
  winBattle(party.characterTwo, mobXP)
}
```

```
class PartyCharacter() {
  var battlesWon: Int = 0
  var experiencePoints: Int = 0
}
```

```
class Party(val characterOne: PartyCharacter,
            val characterTwo: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.characterOne.battlesWon += 1
    this.characterTwo.battlesWon += 1
    this.characterOne.experiencePoints += xp
    this.characterTwo.experiencePoints += xp
  }

}
```

- Begin a stack frame for the Party.winBattle method call
- A reference to the calling object is accessible using the keyword **this**

Heap @428
Object of type PartyCharacter
-battlesWon value: 1
-experiencePoints value: 20

Heap @272
Object of type PartyCharacter
-battlesWon value: 0
-experiencePoints value: 0

Heap @596
Object of type Party
-characterOne value: @428
-characterTwo value: @272
-battlesWon value: 0

Stack
args
name: mobXP, value: 20
name: bossXP, value: 100
name: hero, value: @428
name: party, value: @596
<begin "Party.winBattle" stack frame>
this, value: @596
name: xp, value: 100

```
def winBattle(character: PartyCharacter, xp: Int): Unit = {
  character.battlesWon += 1
  character.experiencePoints += xp
}
```

```
def main(args: Array[String]): Unit = {
  val mobXP: Int = 20
  val bossXP: Int = 100
  val hero: PartyCharacter = new PartyCharacter()
  winBattle(hero, mobXP)
  val party: Party = new Party(hero, new PartyCharacter())
  party.winBattle(bossXP)
  winBattle(party.characterOne, mobXP)
  winBattle(party.characterTwo, mobXP)
}
```

```
class PartyCharacter() {
  var battlesWon: Int = 0
  var experiencePoints: Int = 0
}
```

```
class Party(val characterOne: PartyCharacter,
            val characterTwo: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.characterOne.battlesWon += 1
    this.characterTwo.battlesWon += 1
    this.characterOne.experiencePoints += xp
    this.characterTwo.experiencePoints += xp
  }

}
```

- Use **this** to access the location on the heap of the calling object
- Follow the references to find the variables on the heap to modify

Heap @428
Object of type PartyCharacter
-battlesWon value: 2
-experiencePoints value: 120

Heap @272
Object of type PartyCharacter
-battlesWon value: 1
-experiencePoints value: 100

Heap @596
Object of type Party
-characterOne value: @428
-characterTwo value: @272
-battlesWon value: 1

Stack
args
name: mobXP, value: 20
name: bossXP, value: 100
name: hero, value: @428
name: party, value: @596
<begin "Party.winBattle" stack frame>
this, value: @596
name: xp, value: 100

```
def winBattle(character: PartyCharacter, xp: Int): Unit = {
  character.battlesWon += 1
  character.experiencePoints += xp
}
```

```
class PartyCharacter() {
  var battlesWon: Int = 0
  var experiencePoints: Int = 0
}
```

```
def main(args: Array[String]): Unit = {
  val mobXP: Int = 20
  val bossXP: Int = 100
  val hero: PartyCharacter = new PartyCharacter()
  winBattle(hero, mobXP)
  val party: Party = new Party(hero, new PartyCharacter())
  party.winBattle(bossXP)
  winBattle(party.characterOne, mobXP)
  winBattle(party.characterTwo, mobXP)
}
```

```
class Party(val characterOne: PartyCharacter,
            val characterTwo: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.characterOne.battlesWon += 1
    this.characterTwo.battlesWon += 1
    this.characterOne.experiencePoints += xp
    this.characterTwo.experiencePoints += xp
  }

}
```

- Again we see a method return Unit that affects the state of heap memory
- We say the method has "**side-effects**"

Heap @428
Object of type PartyCharacter
-battlesWon value: 2
-experiencePoints value: 120

Heap @272
Object of type PartyCharacter
-battlesWon value: 1
-experiencePoints value: 100

Heap @596
Object of type Party
-characterOne value: @428
-characterTwo value: @272
-battlesWon value: 1

Stack
args
name: mobXP, value: 20
name: bossXP, value: 100
name: hero, value: @428
name: party, value: @596

```
def winBattle(character: PartyCharacter, xp: Int): Unit = {
  character.battlesWon += 1
  character.experiencePoints += xp
}
```

```
def main(args: Array[String]): Unit = {
  val mobXP: Int = 20
  val bossXP: Int = 100
  val hero: PartyCharacter = new PartyCharacter()
  winBattle(hero, mobXP)
  val party: Party = new Party(hero, new PartyCharacter())
  party.winBattle(bossXP)
  winBattle(party.characterOne, mobXP)
  winBattle(party.characterTwo, mobXP)
}
```

```
class PartyCharacter() {
  var battlesWon: Int = 0
  var experiencePoints: Int = 0
}
```

```
class Party(val characterOne: PartyCharacter,
            val characterTwo: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.characterOne.battlesWon += 1
    this.characterTwo.battlesWon += 1
    this.characterOne.experiencePoints += xp
    this.characterTwo.experiencePoints += xp
  }

}
```

- We can access the characters of the part through the "party" value
- This call of winBattle has the same reference that's stored in "hero"

Heap @428
Object of type PartyCharacter
-battlesWon value: 2
-experiencePoints value: 120

Heap @272
Object of type PartyCharacter
-battlesWon value: 1
-experiencePoints value: 100

Heap @596
Object of type Party
-characterOne value: @428
-characterTwo value: @272
-battlesWon value: 1

Stack
args
name: mobXP, value: 20
name: bossXP, value: 100
name: hero, value: @428
name: party, value: @596
<begin "winBattle" stack frame>
name: character, value: @428
name: xp, value: 20

```
def winBattle(character: PartyCharacter, xp: Int): Unit = {
  character.battlesWon += 1
  character.experiencePoints += xp
}
```

```
def main(args: Array[String]): Unit = {
  val mobXP: Int = 20
  val bossXP: Int = 100
  val hero: PartyCharacter = new PartyCharacter()
  winBattle(hero, mobXP)
  val party: Party = new Party(hero, new PartyCharacter())
  party.winBattle(bossXP)
  winBattle(party.characterOne, mobXP)
  winBattle(party.characterTwo, mobXP)
}
```

```
class PartyCharacter() {
  var battlesWon: Int = 0
  var experiencePoints: Int = 0
}
```

```
class Party(val characterOne: PartyCharacter,
            val characterTwo: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.characterOne.battlesWon += 1
    this.characterTwo.battlesWon += 1
    this.characterOne.experiencePoints += xp
    this.characterTwo.experiencePoints += xp
  }

}
```

- We can access the characters of the part through the "party" value
- This call of winBattle has the same reference that's stored in "hero"

Heap @428
Object of type PartyCharacter
-battlesWon value: 3
-experiencePoints value: 140

Heap @272
Object of type PartyCharacter
-battlesWon value: 1
-experiencePoints value: 100

Heap @596
Object of type Party
-characterOne value: @428
-characterTwo value: @272
-battlesWon value: 1

Stack
args
name: mobXP, value: 20
name: bossXP, value: 100
name: hero, value: @428
name: party, value: @596
<begin "winBattle" stack frame>
name: character, value: @428
name: xp, value: 20

```
def winBattle(character: PartyCharacter, xp: Int): Unit = {
  character.battlesWon += 1
  character.experiencePoints += xp
}
```

```
def main(args: Array[String]): Unit = {
  val mobXP: Int = 20
  val bossXP: Int = 100
  val hero: PartyCharacter = new PartyCharacter()
  winBattle(hero, mobXP)
  val party: Party = new Party(hero, new PartyCharacter())
  party.winBattle(bossXP)
  winBattle(party.characterOne, mobXP)
  winBattle(party.characterTwo, mobXP)
}
```

```
class PartyCharacter() {
  var battlesWon: Int = 0
  var experiencePoints: Int = 0
}
```

```
class Party(val characterOne: PartyCharacter,
            val characterTwo: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.characterOne.battlesWon += 1
    this.characterTwo.battlesWon += 1
    this.characterOne.experiencePoints += xp
    this.characterTwo.experiencePoints += xp
  }

}
```

- Changes made to the PartyCharacter @428

Same effect as calling

winBattle(hero, mobXP)

since the same reference is passed

Heap @428
Object of type PartyCharacter
-battlesWon value: 3
-experiencePoints value: 140

Heap @272
Object of type PartyCharacter
-battlesWon value: 1
-experiencePoints value: 100

Heap @596
Object of type Party
-characterOne value: @428
-characterTwo value: @272
-battlesWon value: 1

Stack
args
name: mobXP, value: 20
name: bossXP, value: 100
name: hero, value: @428
name: party, value: @596
<begin "winBattle" stack frame>
name: character, value: @428
name: xp, value: 20

```
def winBattle(character: PartyCharacter, xp: Int): Unit = {
  character.battlesWon += 1
  character.experiencePoints += xp
}
```

```
def main(args: Array[String]): Unit = {
  val mobXP: Int = 20
  val bossXP: Int = 100
  val hero: PartyCharacter = new PartyCharacter()
  winBattle(hero, mobXP)
  val party: Party = new Party(hero, new PartyCharacter())
  party.winBattle(bossXP)
  winBattle(party.characterOne, mobXP)
  winBattle(party.characterTwo, mobXP)
}
```

```
class PartyCharacter() {
  var battlesWon: Int = 0
  var experiencePoints: Int = 0
}
```

```
class Party(val characterOne: PartyCharacter,
            val characterTwo: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.characterOne.battlesWon += 1
    this.characterTwo.battlesWon += 1
    this.characterOne.experiencePoints += xp
    this.characterTwo.experiencePoints += xp
  }

}
```

- The PartyCharacter @272 can be accessed in main through the "party" value

Heap @428
Object of type PartyCharacter
-battlesWon value: 3
-experiencePoints value: 140

Heap @272
Object of type PartyCharacter
-battlesWon value: 1
-experiencePoints value: 100

Heap @596
Object of type Party
-characterOne value: @428
-characterTwo value: @272
-battlesWon value: 1

Stack
args
name: mobXP, value: 20
name: bossXP, value: 100
name: hero, value: @428
name: party, value: @596
<begin "winBattle" stack frame>
name: character, value: @272
name: xp, value: 20

```
def winBattle(character: PartyCharacter, xp: Int): Unit = {
  character.battlesWon += 1
  character.experiencePoints += xp
}
```

```
def main(args: Array[String]): Unit = {
  val mobXP: Int = 20
  val bossXP: Int = 100
  val hero: PartyCharacter = new PartyCharacter()
  winBattle(hero, mobXP)
  val party: Party = new Party(hero, new PartyCharacter())
  party.winBattle(bossXP)
  winBattle(party.characterOne, mobXP)
  winBattle(party.characterTwo, mobXP)
}
```

```
class PartyCharacter() {
  var battlesWon: Int = 0
  var experiencePoints: Int = 0
}
```

```
class Party(val characterOne: PartyCharacter,
            val characterTwo: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.characterOne.battlesWon += 1
    this.characterTwo.battlesWon += 1
    this.characterOne.experiencePoints += xp
    this.characterTwo.experiencePoints += xp
  }

}
```

- The PartyCharacter @272 can be accessed in main through the "party" value

Heap @428
Object of type PartyCharacter
-battlesWon value: 3
-experiencePoints value: 140

Heap @272
Object of type PartyCharacter
-battlesWon value: 2
-experiencePoints value: 120

Heap @596
Object of type Party
-characterOne value: @428
-characterTwo value: @272
-battlesWon value: 1

Stack
args
name: mobXP, value: 20
name: bossXP, value: 100
name: hero, value: @428
name: party, value: @596

```
def winBattle(character: PartyCharacter, xp: Int): Unit = {
  character.battlesWon += 1
  character.experiencePoints += xp
}
```

```
def main(args: Array[String]): Unit = {
  val mobXP: Int = 20
  val bossXP: Int = 100
  val hero: PartyCharacter = new PartyCharacter()
  winBattle(hero, mobXP)
  val party: Party = new Party(hero, new PartyCharacter())
  party.winBattle(bossXP)
  winBattle(party.characterOne, mobXP)
  winBattle(party.characterTwo, mobXP)
}
```

```
class PartyCharacter() {
  var battlesWon: Int = 0
  var experiencePoints: Int = 0
}
```

```
class Party(val characterOne: PartyCharacter,
            val characterTwo: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.characterOne.battlesWon += 1
    this.characterTwo.battlesWon += 1
    this.characterOne.experiencePoints += xp
    this.characterTwo.experiencePoints += xp
  }

}
```

- Program ends
- All memory freed

Heap @428

Heap @272

Heap @596

Stack

```
class PartyCharacter() {
  var battlesWon: Int = 0
  var experiencePoints: Int = 0
}
```

```
class Party(val characterOne: PartyCharacter,
            val characterTwo: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.characterOne.battlesWon += 1
    this.characterTwo.battlesWon += 1
    this.characterOne.experiencePoints += xp
    this.characterTwo.experiencePoints += xp
  }

}
```

```
def winBattle(character: PartyCharacter, xp: Int): Unit = {
  character.battlesWon += 1
  character.experiencePoints += xp
}
```

```
def main(args: Array[String]): Unit = {
  val mobXP: Int = 20
  val bossXP: Int = 100
  val hero: PartyCharacter = new PartyCharacter()
  winBattle(hero, mobXP)
  val party: Party = new Party(hero, new PartyCharacter())
  party.winBattle(bossXP)
  winBattle(party.characterOne, mobXP)
  winBattle(party.characterTwo, mobXP)
}
```

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)
}
```

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

- Call function
- Create new stack frame

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
<Used by another program>
<Used by another program>

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

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
<Used by another program>
<Used by another program>

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

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)
}
```

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>

- Conditional finally false
- return 0

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)
}
```

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>

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

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)
}
```

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

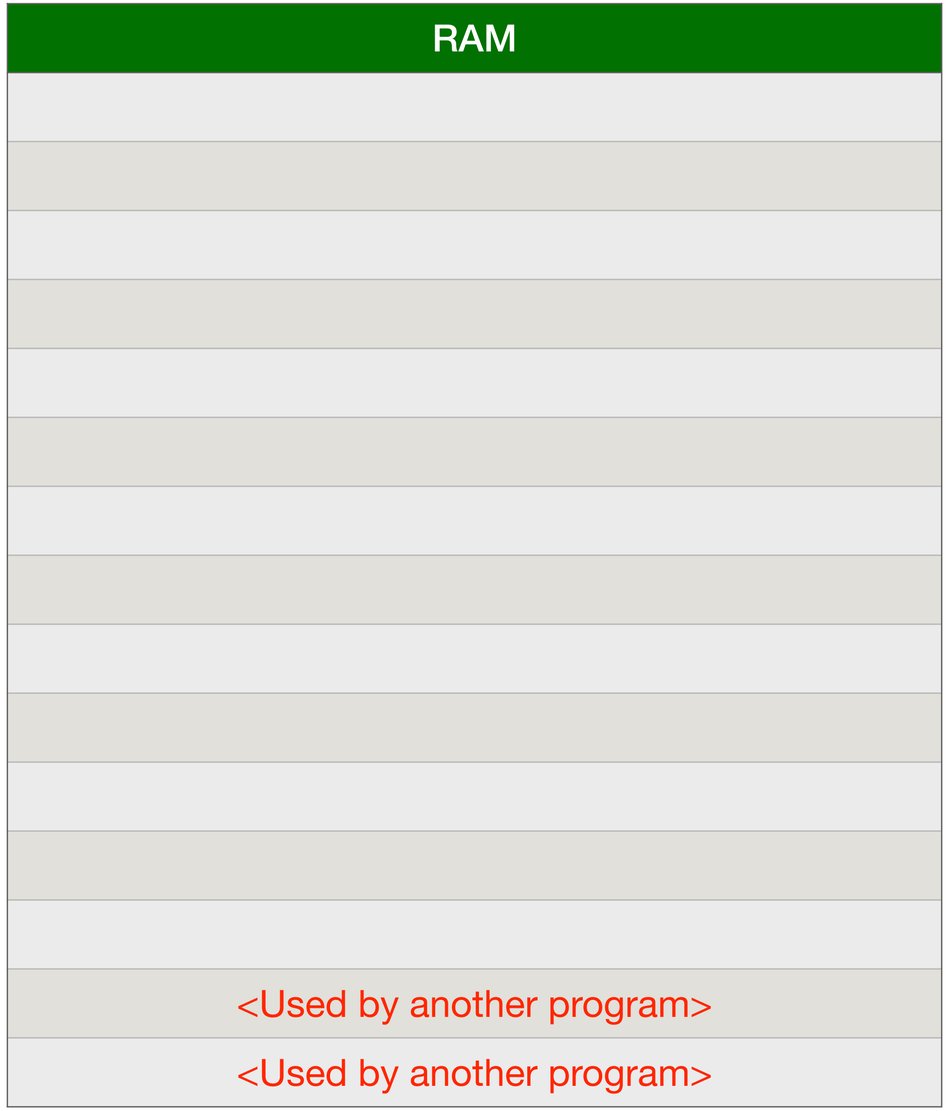
- print 6

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
  - In reality, the stack will be much larger than in this 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" create a Scala **class** named "Team" and a Scala **object** named "Referee".

Team will have:

- Two state values of type Int representing the strength of the team's offense and defense with a constructor to set these values. The parameters for the constructor should be offense first, then defense (Note: These values do not have defined names in this question so you cannot access them in your testing. If I use "teamOffense" and you name it "offense" and access it in your tests, your tests will crash when testing my code since the variable "offense" will not exist)
- A third state variable of type Int that is not in the constructor that represents the score of the team, is declared as a **var**, and is initialized to 0 (This variable also has not defined name)

Referee will have:

- A method named "playGame" that takes two Team objects as parameters and return type Unit. This method will alter the state of each input Team by setting their scores equal to their offense minus the other Team's defense. If a Team's offense is less than the other Team's defense their score should be 0 (no negative scores)
- A method named "declareWinner" that takes two Teams as parameters and returns the Team with the higher score. If both Teams have the same score, return a **new** Team object to indicate that neither competing team won (You may choose any values in the constructor call of this new Team)

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

Lecture Question

Sample Usage

```
val t1: Team = new Team(7, 3)
val t2: Team = new Team(4, 20)

Referee.playGame(t1, t2)
assert(Referee.declareWinner(t1, t2) == t2)
assert(Referee.declareWinner(t2, t1) == t2)
```

Commentary

We create Team as a **class** since we want to create many objects of type Team that will compete against each other. Each team will have different state (offense, defense, score), but will be the same type (Team)

Referee is an **object** since there only needs to be one of them and the object has no state. The same referee can officiate every game between any two teams

We pass **references** of objects of type Team to the Referee. Since the Referee has the references, when it changes the score of a Team that change is made to the state of that Team throughout the program. This change can be tested by checking the reference returned by the declareWinner method since you cannot check the score directly.

Old Example

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
<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: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()
  }
}
```

- bird2.inDanger is never called due to short-circuiting
- 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: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 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: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 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