Heap Memory

Lecture Task

- Point of Sale: Lecture Task 2 -

Functionality: In the store.model.items package, write a class named “**Sale**” with the following functionality:

- A constructor that takes a variable of type Double named “percentOff” representing the percentage of the sale
    - The parameter must be declared using **var** and be named exactly “percentOff”
- A method named “updatePrice” that takes a price as a Double and returns the price with the sale applied

In the store.model.items package, write a class named “**SaleTestingItem**” with the following functionality:

- A constructor that takes a description String a price as a Double (Same as the item class)
- A method named “addSale” that takes a **reference** to a Sale object and returns Unit. This method should store the Sale in a data structure so it can be applied to the price later
- A method named “price” that doesn’t take any parameters and returns the price of the item as a Double with all sales applied

Testing: In the tests package, create a test suite named LectureTask2 that tests this functionality.

Another Memory Example

Multiple Objects on the heap

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

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

```scala
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
  }

}
```

```scala
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)
}
```

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

Stack
args

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

```scala
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)
}
```

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

```scala
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
  }

}
```

- 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

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

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

```scala
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
  }

}
```

```scala
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
}
```

```
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
  }

}
```

- "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
}
```

```
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)
}
```

- "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
}
```

```
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)
}
```

- 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

```
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)
}
```

- 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

```
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)
}
```

- 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
}
```

```
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
  }

}
```

- 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

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

```scala
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)
}
```

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

```scala
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

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

```scala
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)
}
```

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

```scala
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

```
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)
}
```

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

```
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

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

```scala
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)
}
```

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

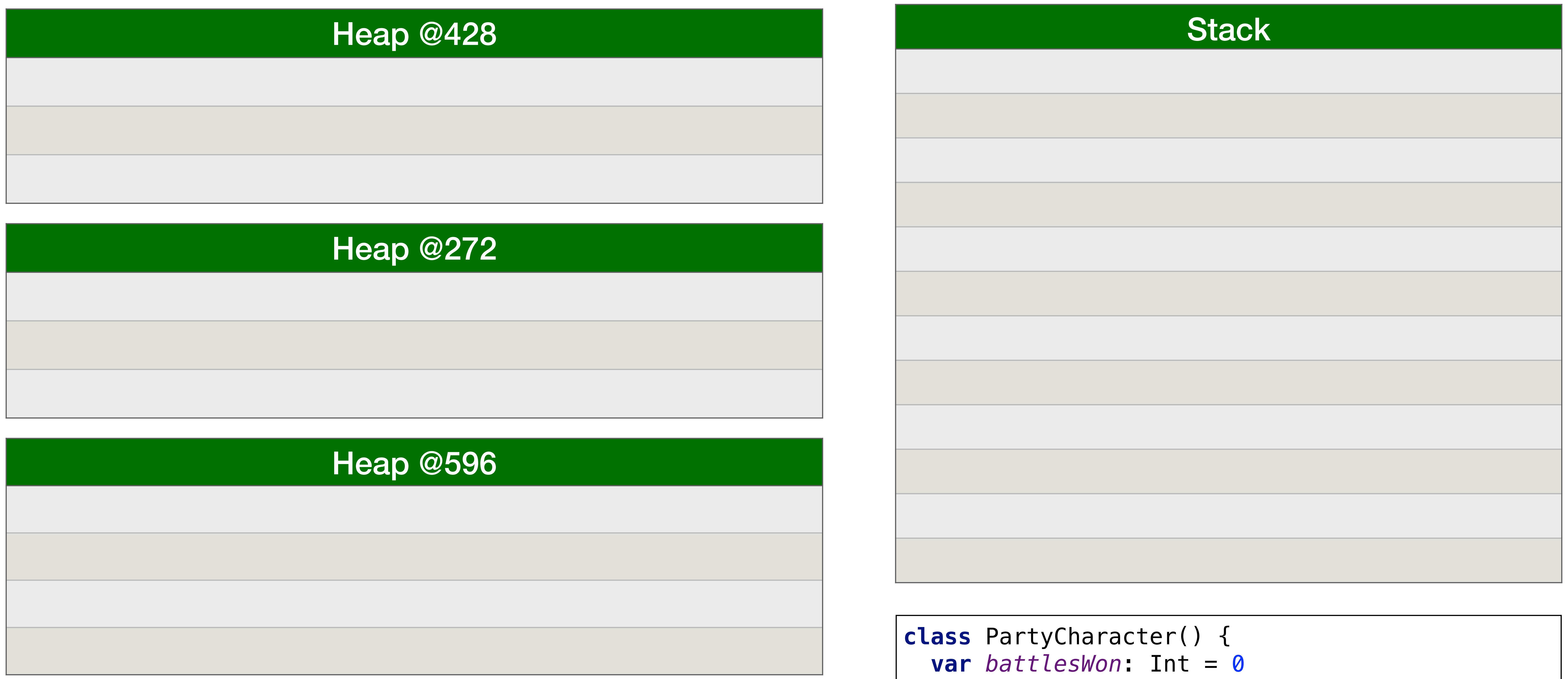
```scala
class Party(val characterOne: PartyCharacter,
            val characterTwo: PartyCharacter) {

  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



```
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
  }

}
```

Lecture Task

- Point of Sale: Lecture Task 2 -

Functionality: In the store.model.items package, write a class named “**Sale**” with the following functionality:

- A constructor that takes a variable of type Double named “percentOff” representing the percentage of the sale
    - The parameter must be declared using **var** and be named exactly “percentOff”
- A method named “updatePrice” that takes a price as a Double and returns the price with the sale applied

In the store.model.items package, write a class named “**SaleTestingItem**” with the following functionality:

- A constructor that takes a description String a price as a Double (Same as the item class)
- A method named “addSale” that takes a **reference** to a Sale object and returns Unit. This method should store the Sale in a data structure so it can be applied to the price later
- A method named “price” that doesn’t take any parameters and returns the price of the item as a Double with all sales applied

Testing: In the tests package, create a test suite named LectureTask2 that tests this functionality.

Lecture Task

Sample Usage

```
val milk: SaleTestingItem = new SaleTestingItem("milk", 3.0)

val milkSale: Sale = new Sale(20.0)

milk.addSale(milkSale)

assert(compareDoubles(milk.price(), 2.4), milk.price())
```

Commentary

Your SaleTestingItem method must store references to each Sale that is added

If a sale price is updated after being added to a SaleTestingItem, the price of the item should also update

You need to write a test that will check if a solution is handling references properly (no_oob_percent in AutoLab does not handle references properly)