Heap Memory

Another Memory Example

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

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

```scala
class Party(val character1: PartyCharacter,
            val character2: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.character1.winBattle(xp)
    this.character2.winBattle(xp)
  }
}
```

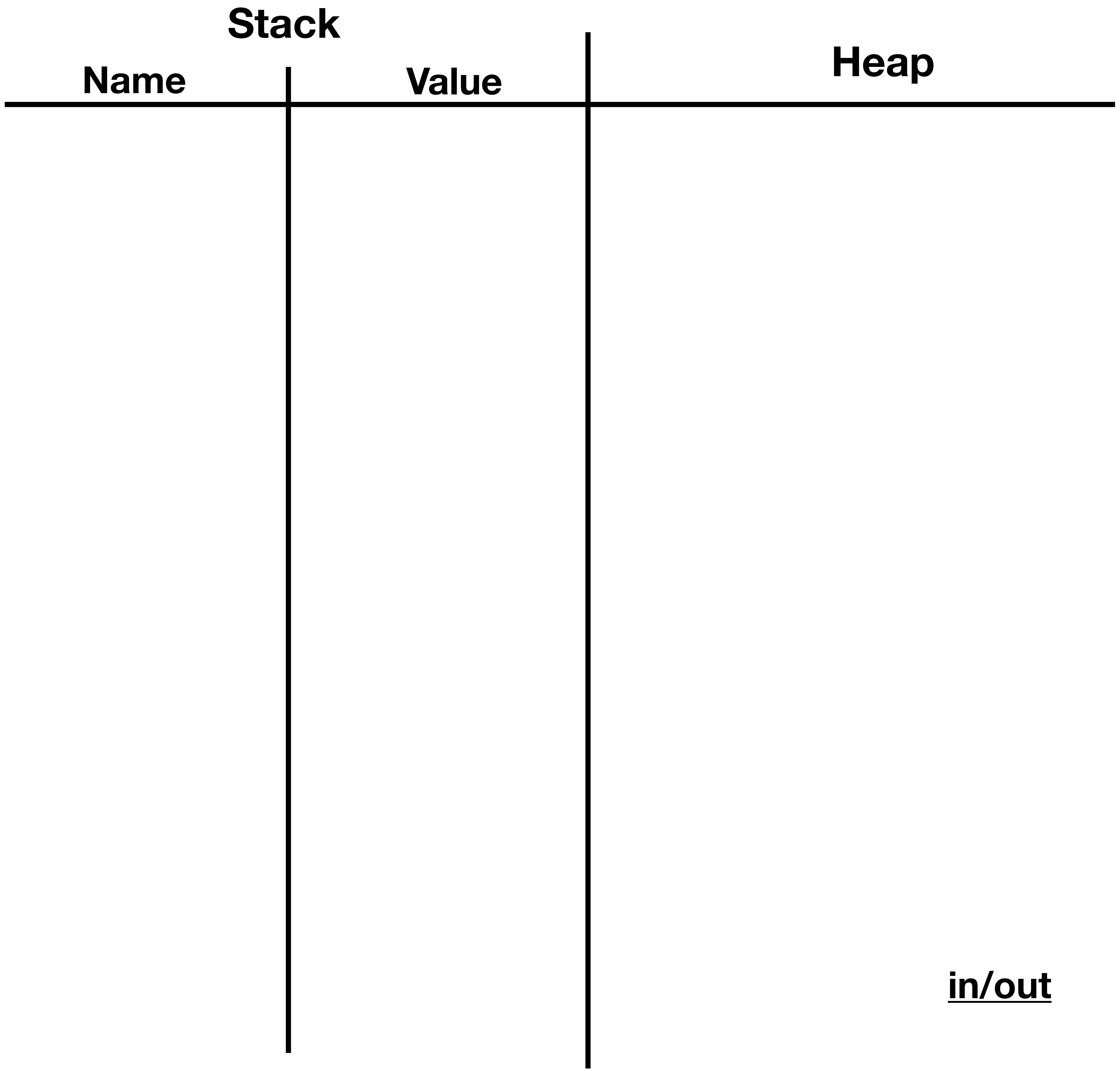
```scala
def main(args: Array[String]): Unit = {
  val mobXP: Int = 20
  val bossXP: Int = 100
  val hero: PartyCharacter = new PartyCharacter()
  hero.winBattle(mobXP)
  val party: Party = new Party(hero, new PartyCharacter())
  party.winBattle(bossXP)

  println(hero.xp)
  println(party.characterTwo.xp)
}
```

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

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

```
class Party(val character1: PartyCharacter,
            val character2: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.character1.winBattle(xp)
    this.character2.winBattle(xp)
  }
}
```

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

  println(hero.xp)
  println(party.characterTwo.xp)
}
```

- Programs always start with the main method



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

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

```
class Party(val character1: PartyCharacter,
            val character2: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.character1.winBattle(xp)
    this.character2.winBattle(xp)
  }
}
```

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

  println(hero.xp)
  println(party.characterTwo.xp)
}
```

- Add mobXP and bossXP to the stack
- Int values are added directly to the stack

Stack

Name	Value
mobXP	20
bossXP	100

Heap

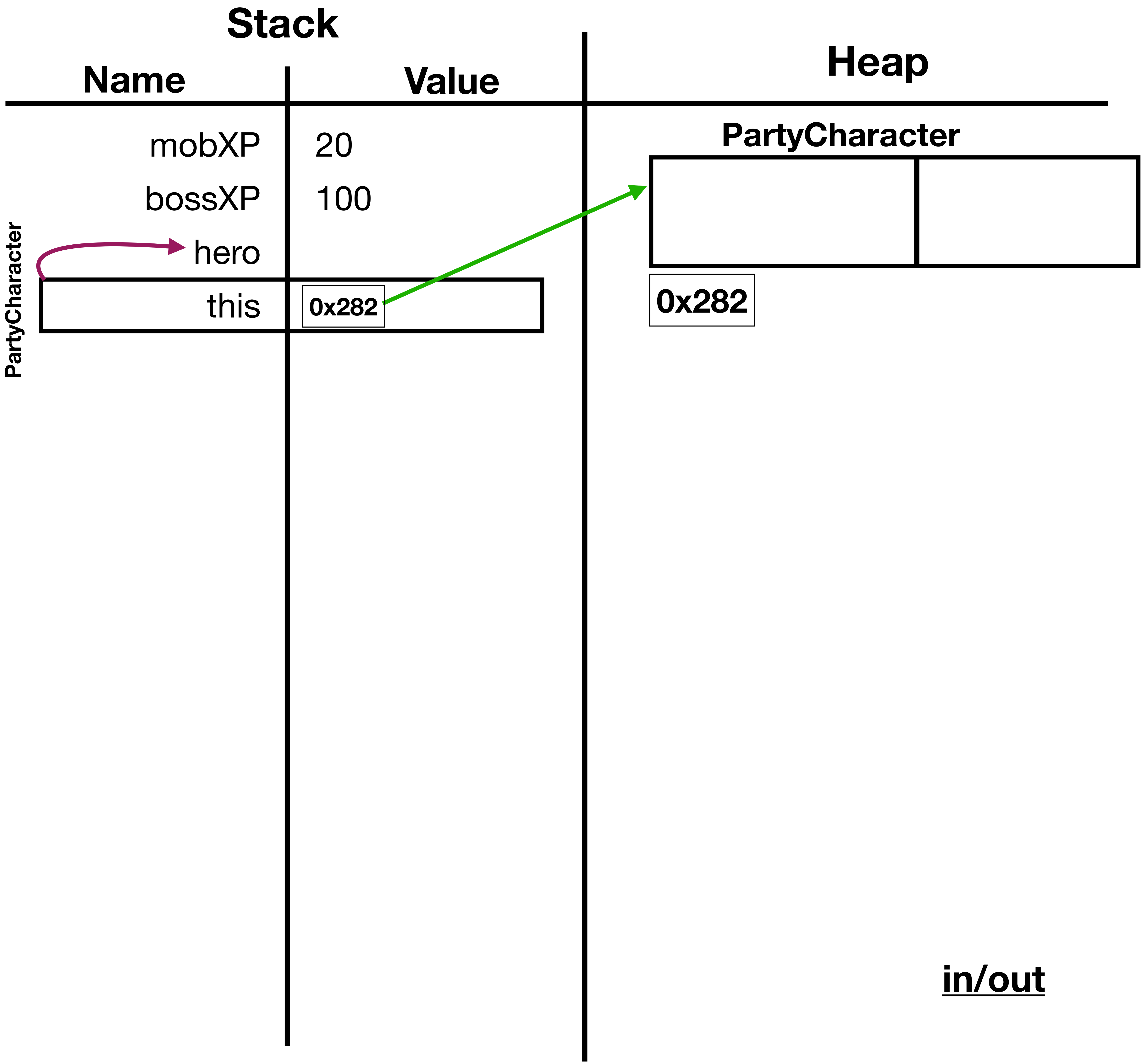
<u>in/out</u>

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

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

```
class Party(val character1: PartyCharacter,
            val character2: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.character1.winBattle(xp)
    this.character2.winBattle(xp)
  }
}
```

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

  println(hero.xp)
  println(party.characterTwo.xp)
}
```

- Create a new stack frame for the constructor call
- "this" contains a reference to the object being constructed

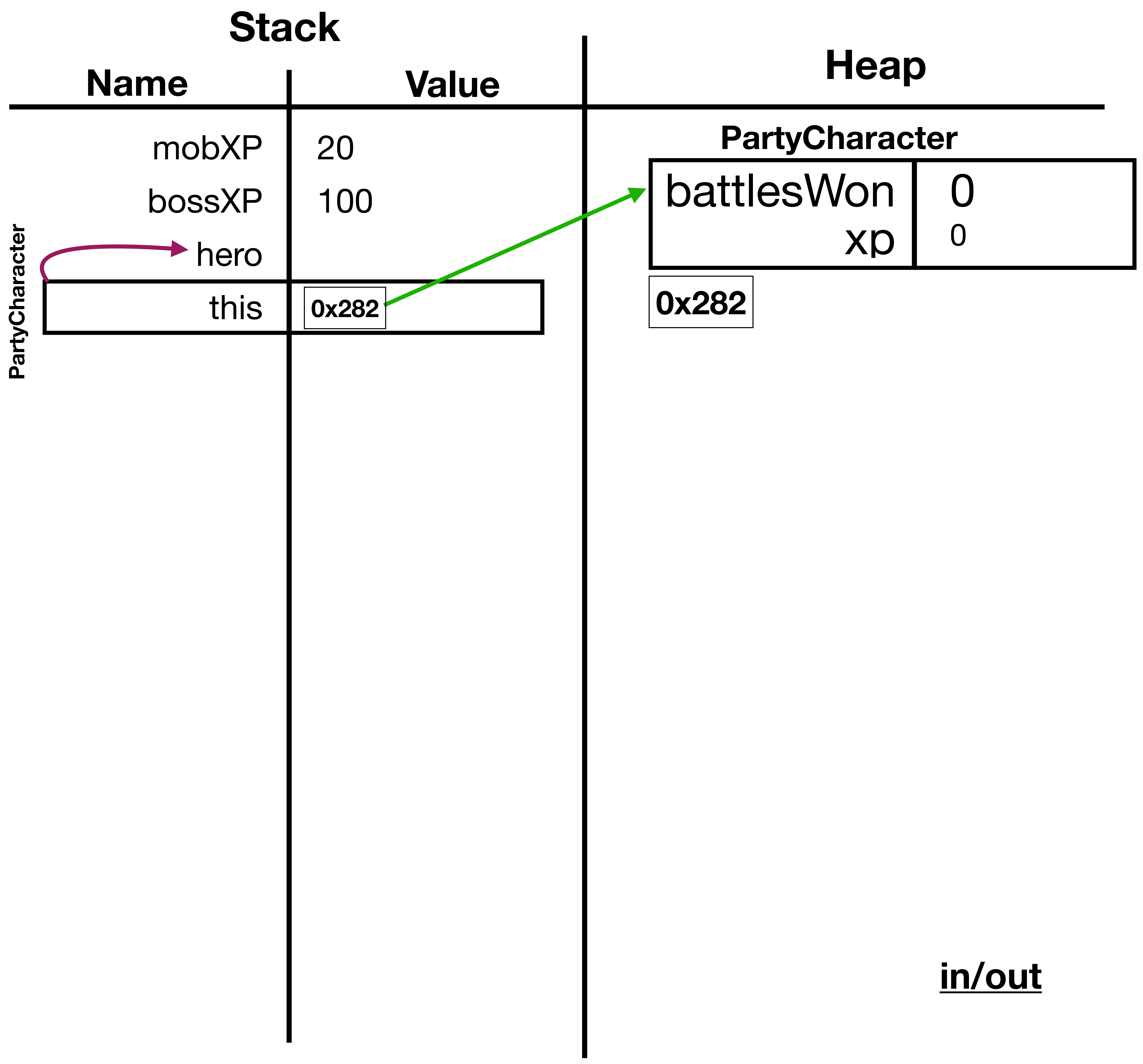


```
class PartyCharacter() {
  var battlesWon: Int = 0
  var xp: Int = 0

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

```
class Party(val character1: PartyCharacter,
            val character2: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.character1.winBattle(xp)
    this.character2.winBattle(xp)
  }
}
```

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

  println(hero.xp)
  println(party.characterTwo.xp)
}
```

- Run all the code that's outside of the methods
- All declared variables become state variables and are stored with the object

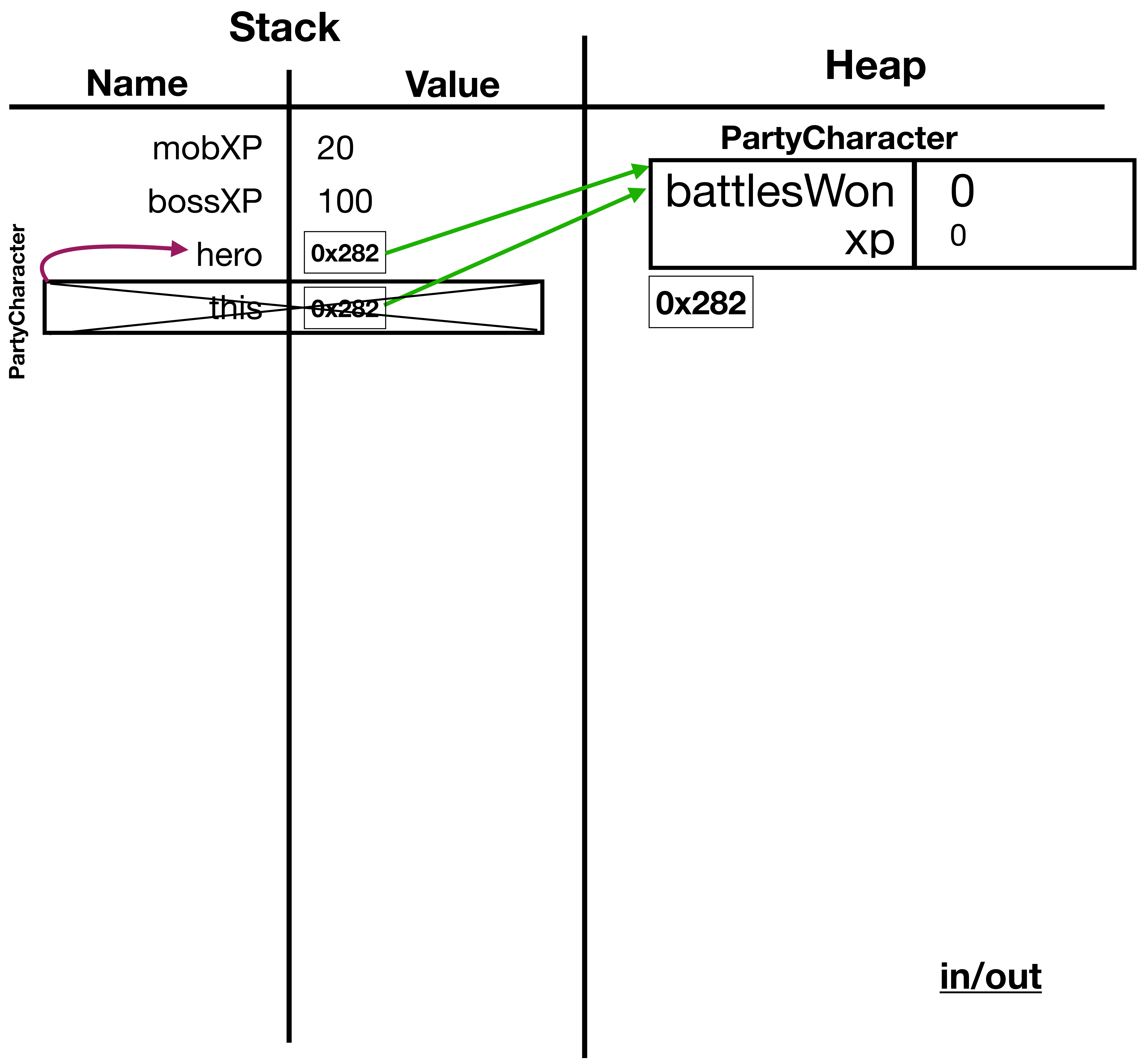


```scala
class PartyCharacter() {
  var battlesWon: Int = 0
  var xp: Int = 0

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

```scala
class Party(val character1: PartyCharacter,
            val character2: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.character1.winBattle(xp)
    this.character2.winBattle(xp)
  }
}
```

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

  println(hero.xp)
  println(party.characterTwo.xp)
}
```

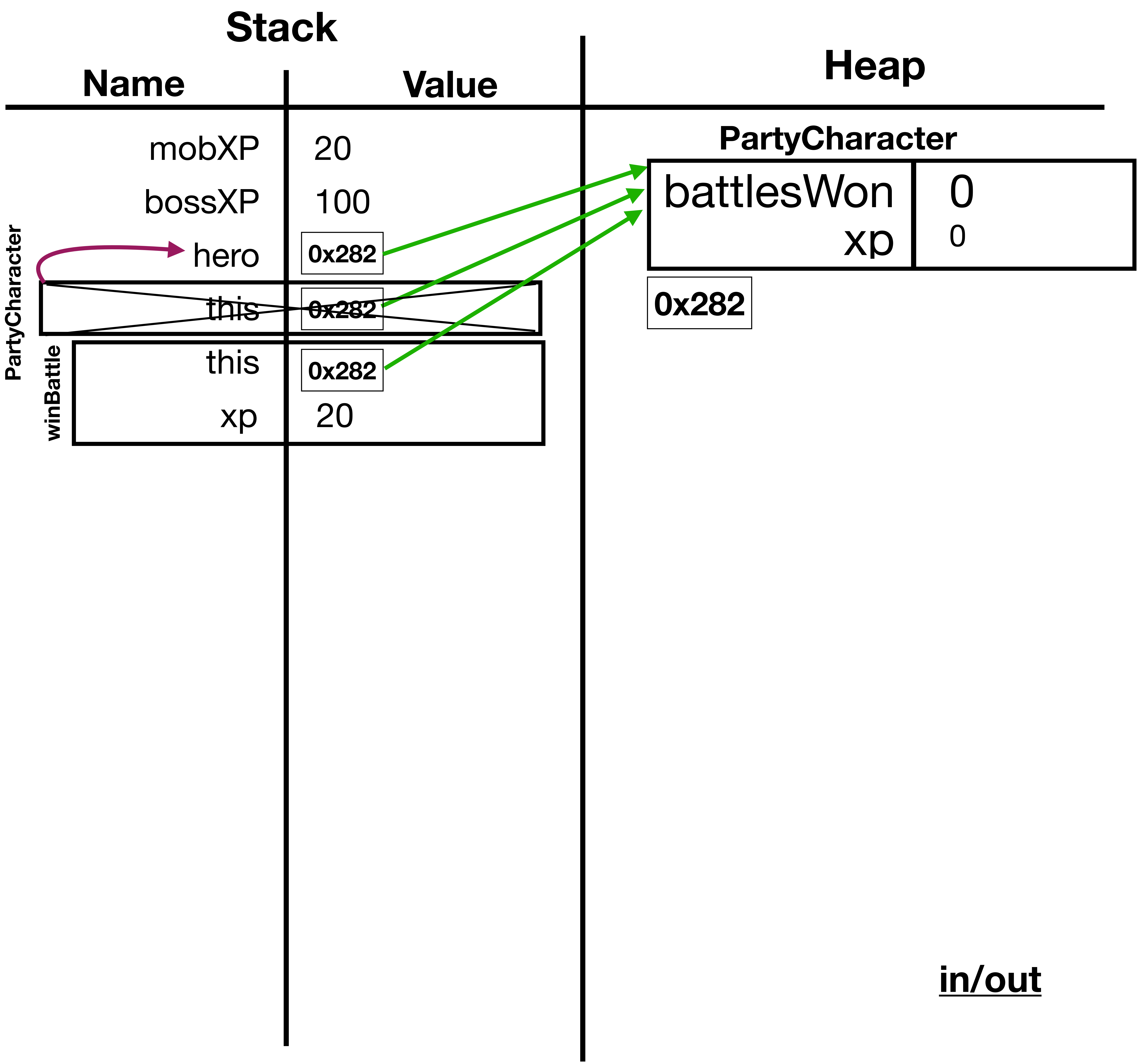
- Constructor stack frame ends and is removed from the stack
- Return a reference to the newly constructed object to hero



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

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

```
class Party(val character1: PartyCharacter,
            val character2: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.character1.winBattle(xp)
    this.character2.winBattle(xp)
  }
}
```

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

  println(hero.xp)
  println(party.characterTwo.xp)
}
```



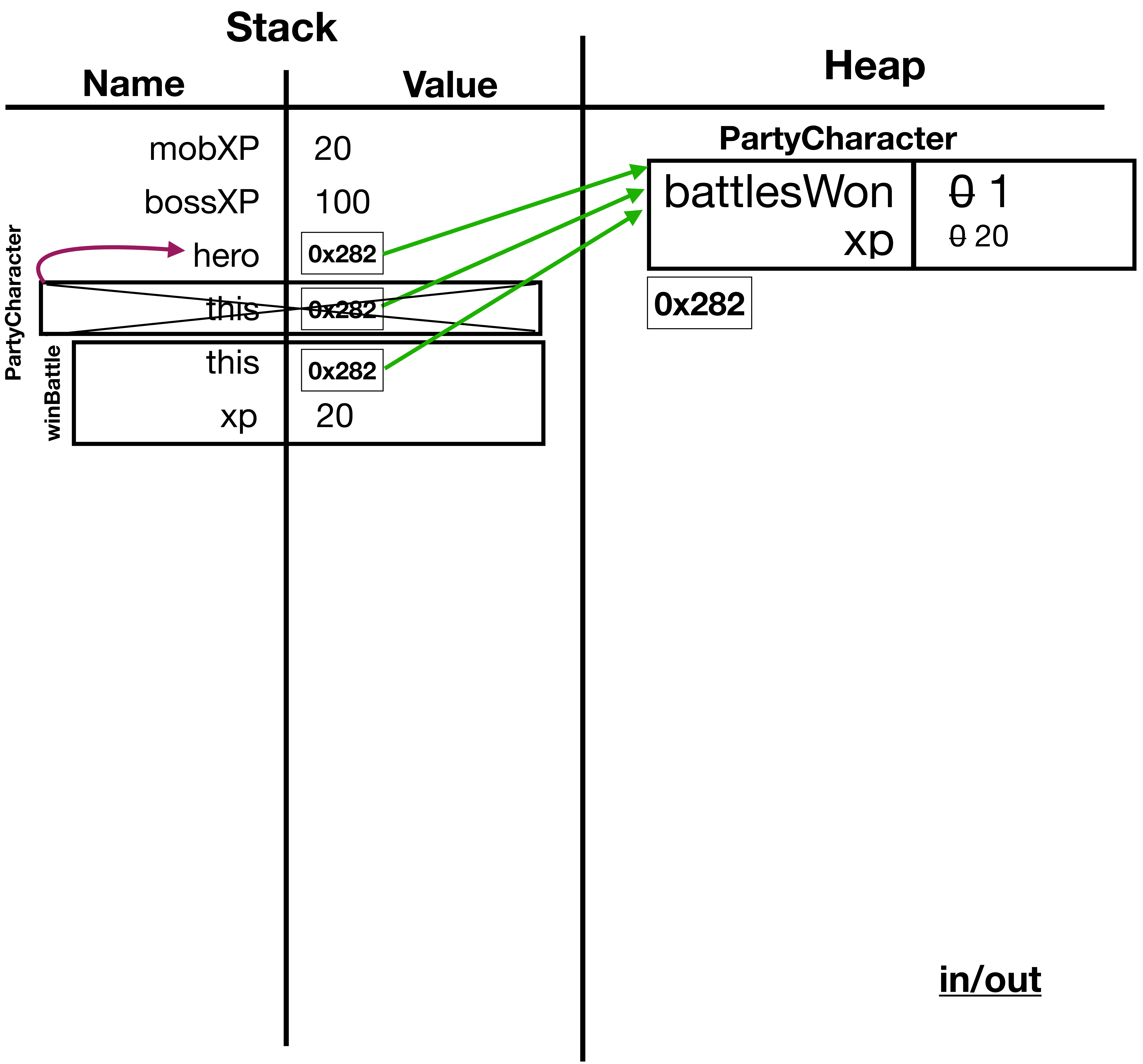
in/out

- Create a stack frame for the winBattle method call
- "this" stores a reference to the calling object

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

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

```scala
class Party(val character1: PartyCharacter,
            val character2: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.character1.winBattle(xp)
    this.character2.winBattle(xp)
  }
}
```

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

  println(hero.xp)
  println(party.characterTwo.xp)
}
```



- Update the battlesWon and xp of "this"

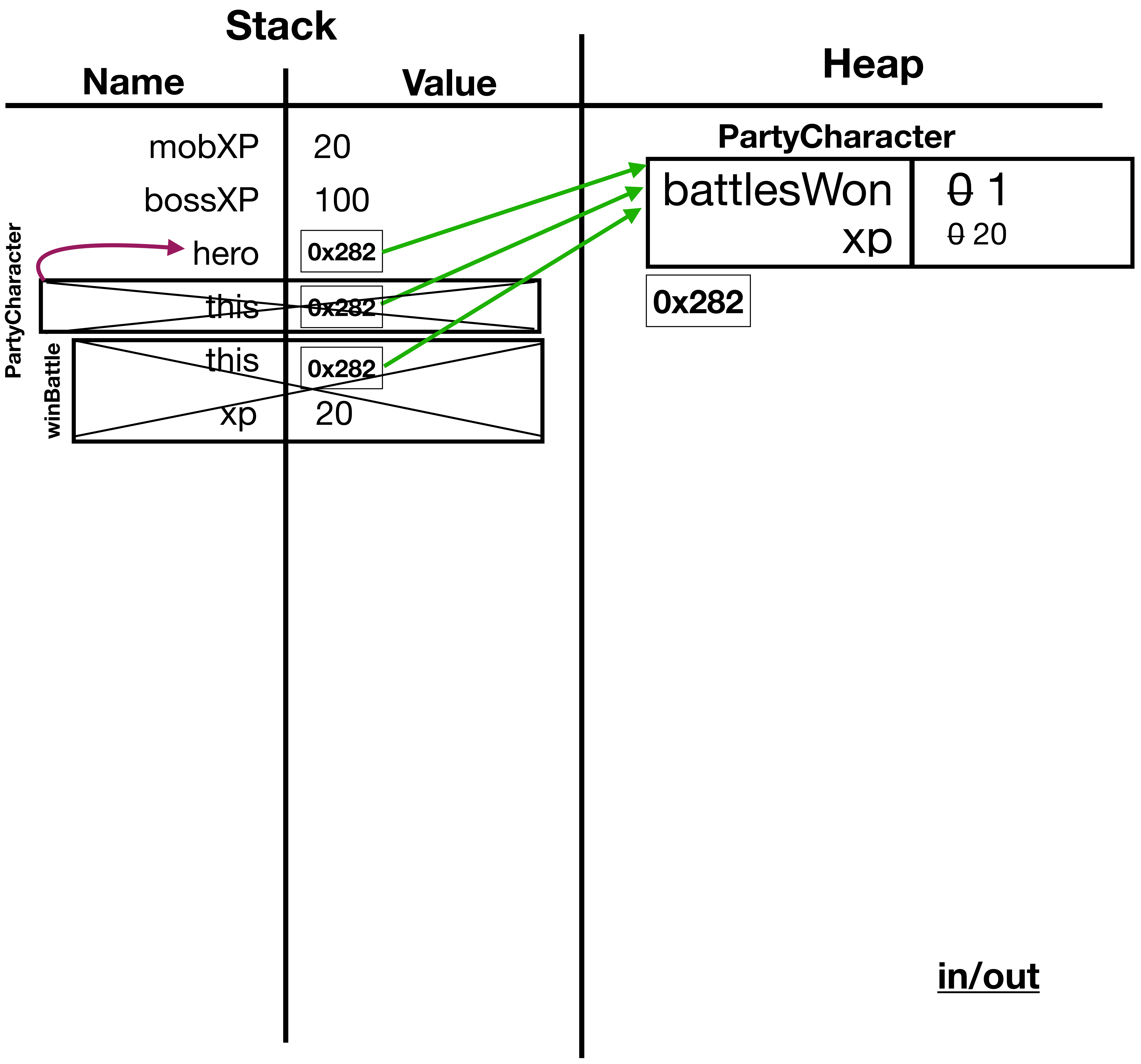
in/out

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

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

```
class Party(val character1: PartyCharacter,
            val character2: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.character1.winBattle(xp)
    this.character2.winBattle(xp)
  }
}
```

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

  println(hero.xp)
  println(party.characterTwo.xp)
}
```

- The stack frame ends and is removed from the stack
- The changes made to the heap persist!

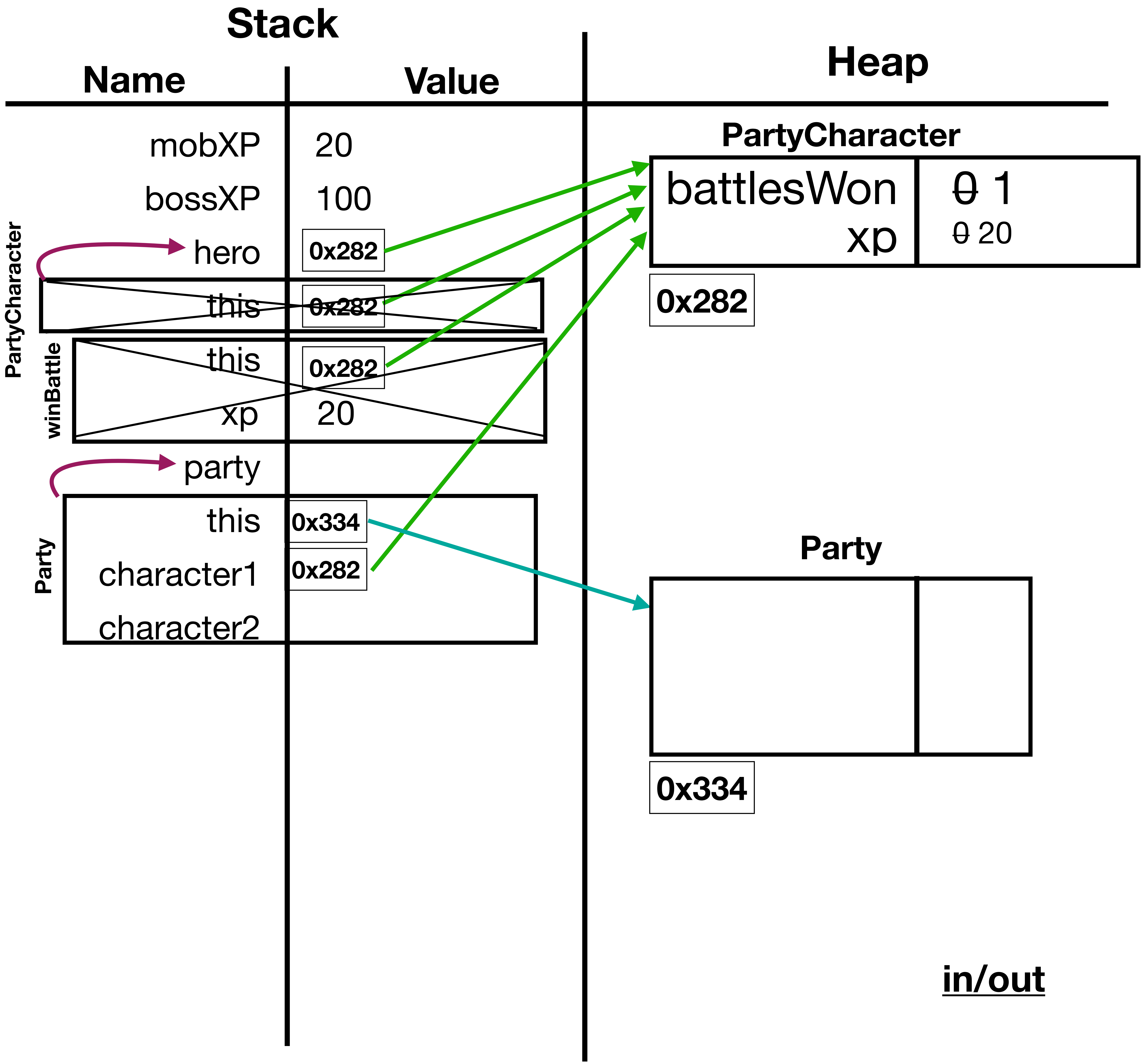


in/out

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

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

```
class Party(val character1: PartyCharacter,
            val character2: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.character1.winBattle(xp)
    this.character2.winBattle(xp)
  }
}
```

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

  println(hero.xp)
  println(party.characterTwo.xp)
}
```



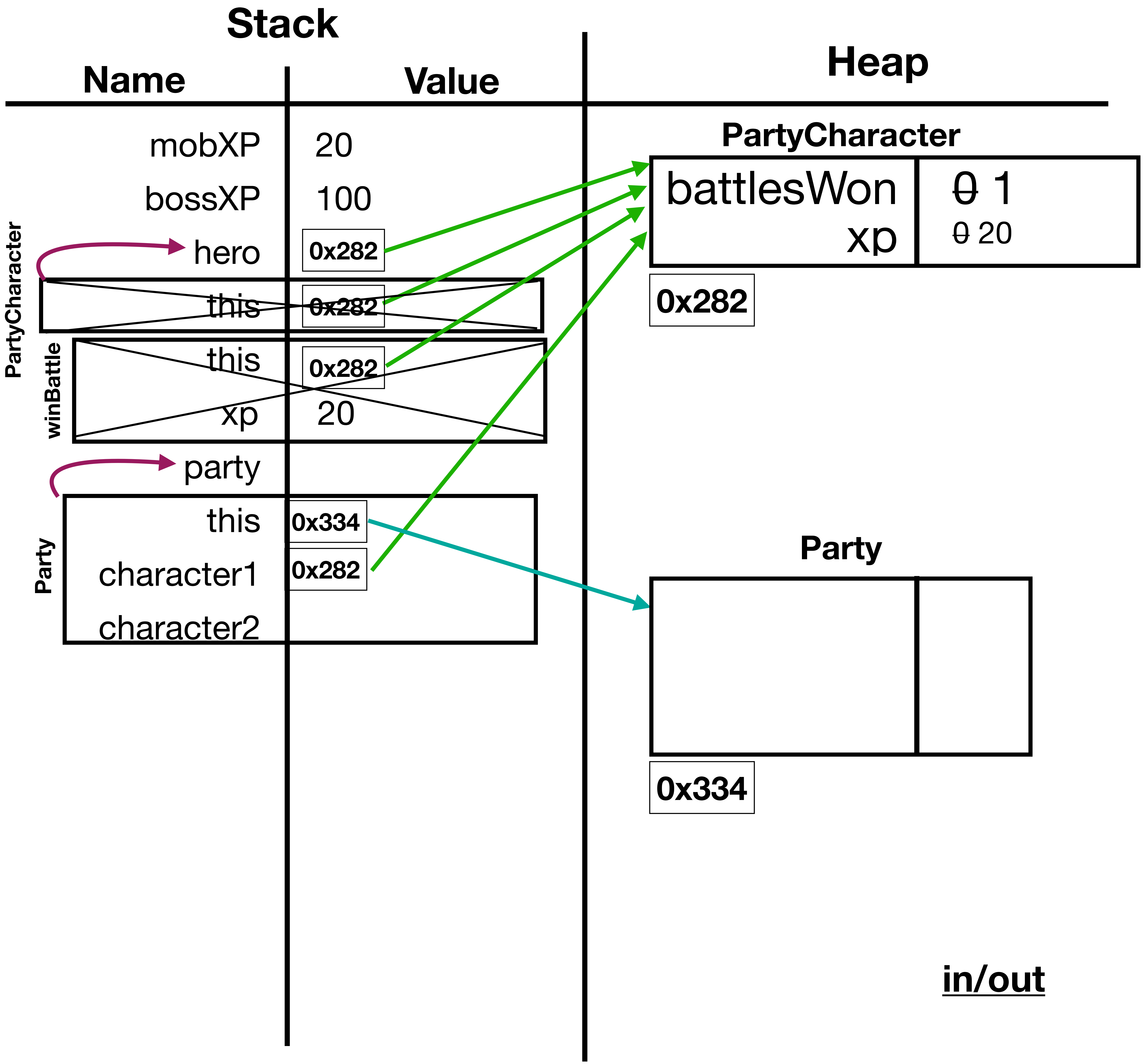
in/out

- Create a variable to store a new party
- Call the Party constructor and draw a stack frame

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

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

```
class Party(val character1: PartyCharacter,
            val character2: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.character1.winBattle(xp)
    this.character2.winBattle(xp)
  }
}
```

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

  println(hero.xp)
  println(party.characterTwo.xp)
}
```



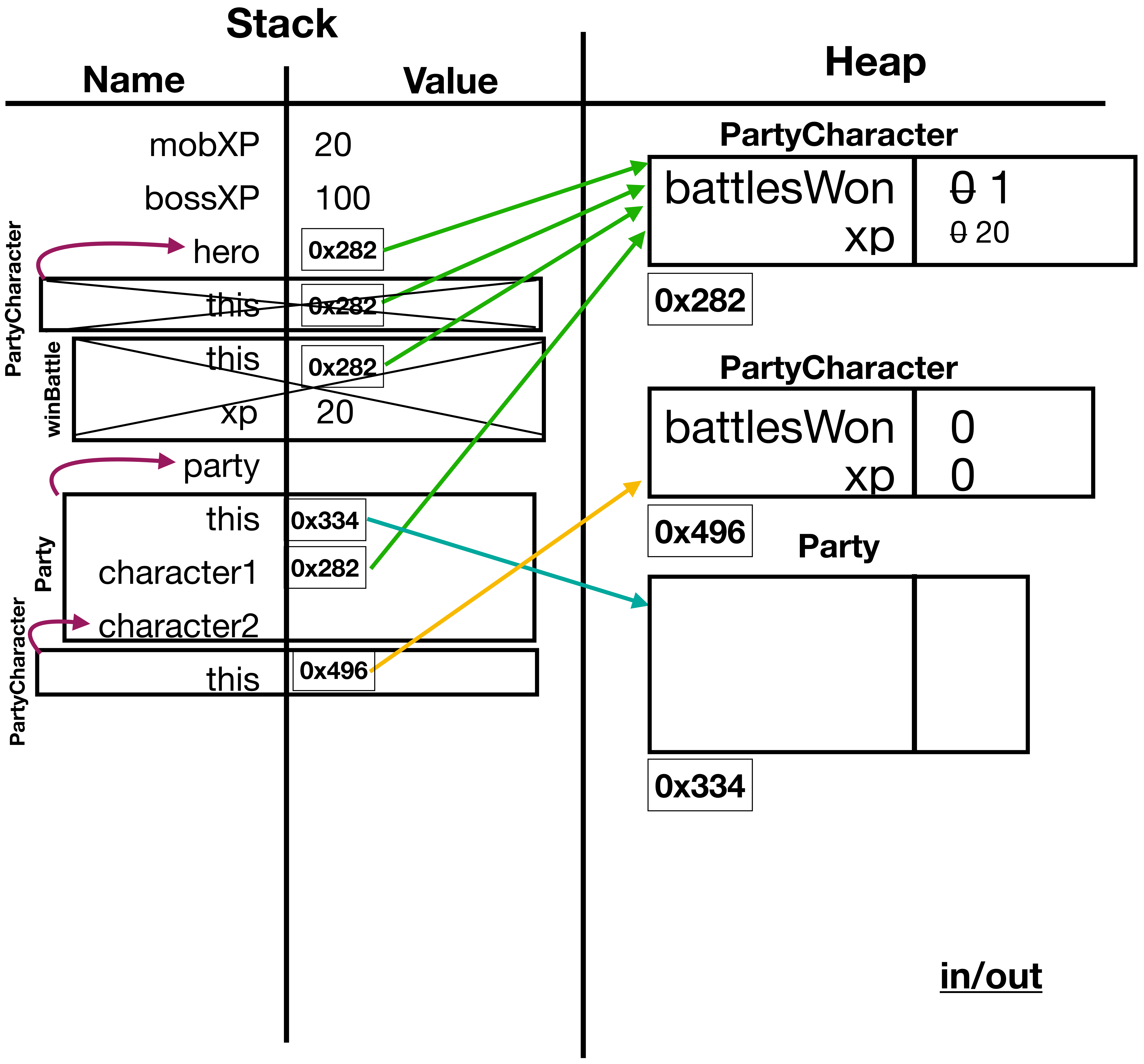
- **But what's the value of character2??**
- We need to create another stack frame for a PartyCharacter constructor

in/out

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

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

```
class Party(val character1: PartyCharacter,
            val character2: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.character1.winBattle(xp)
    this.character2.winBattle(xp)
  }
}
```

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

  println(hero.xp)
  println(party.characterTwo.xp)
}
```

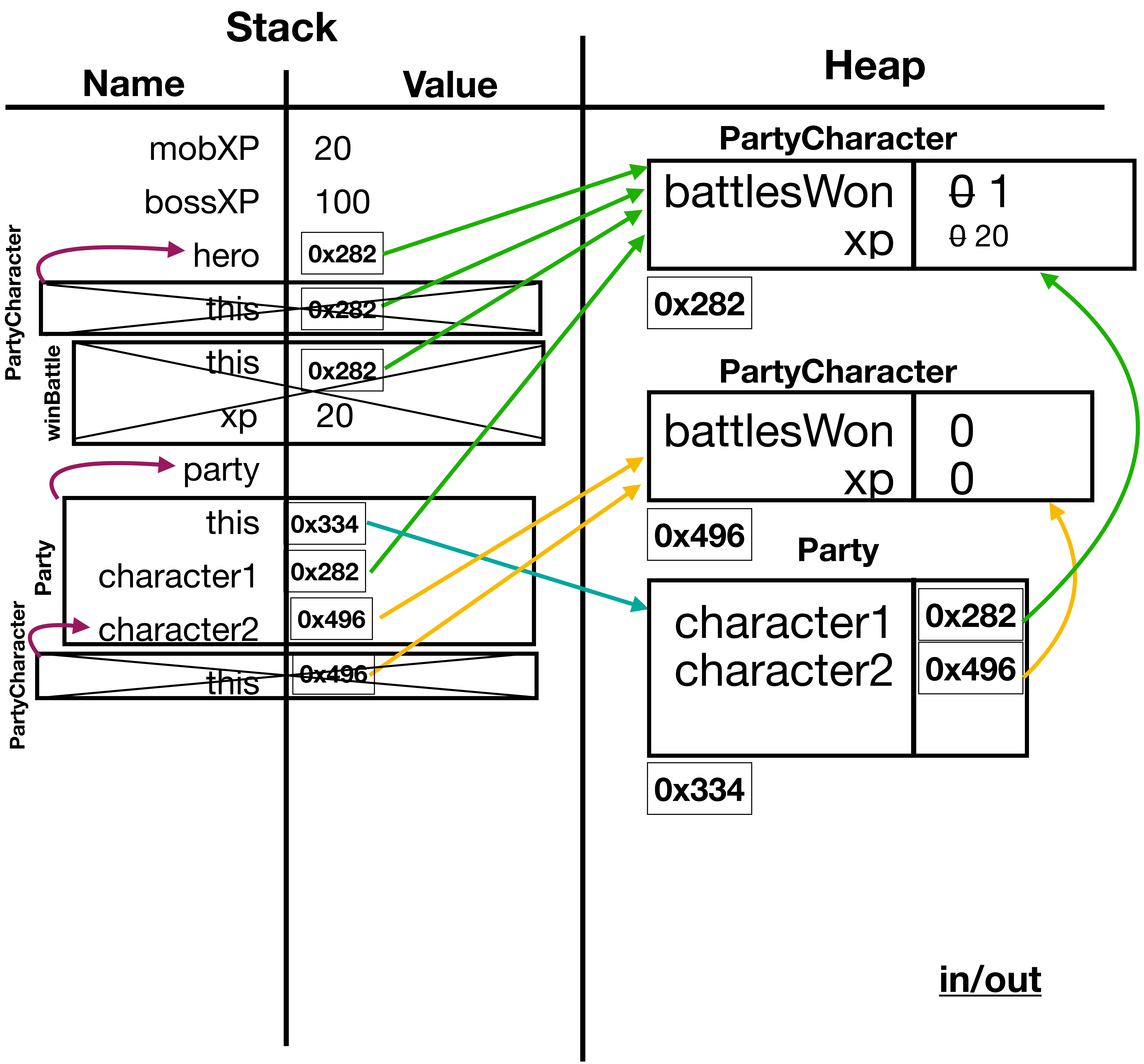


- The PartyCharacter constructor will return directly to the character2 parameter of the Party constructor

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

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

```scala
class Party(val character1: PartyCharacter,
            val character2: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.character1.winBattle(xp)
    this.character2.winBattle(xp)
  }
}
```

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

  println(hero.xp)
  println(party.characterTwo.xp)
}
```



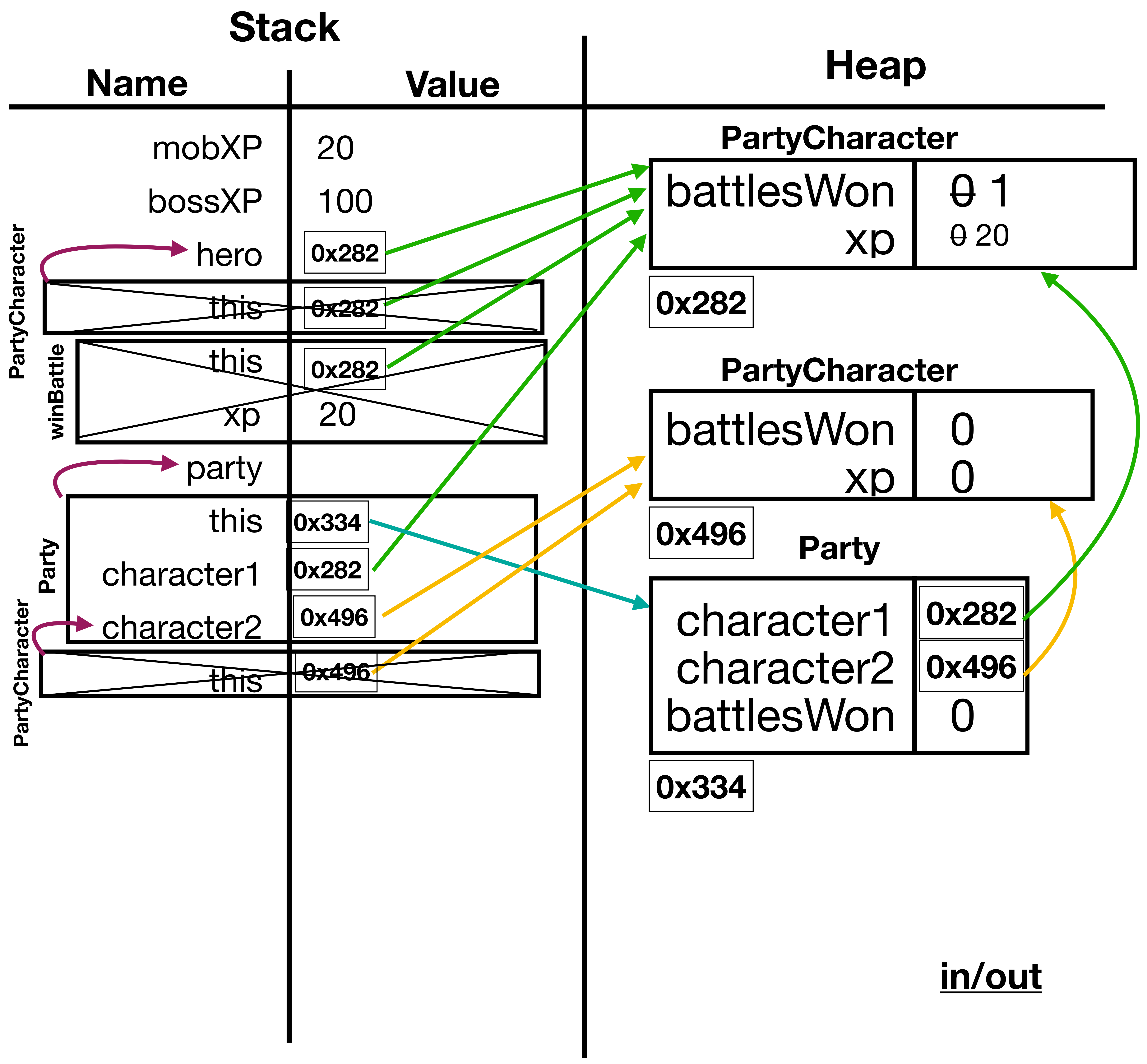
- Now that we have all the parameters
  - We can run the Party contractor

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

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

```
class Party(val character1: PartyCharacter,
            val character2: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.character1.winBattle(xp)
    this.character2.winBattle(xp)
  }
}
```

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

  println(hero.xp)
  println(party.characterTwo.xp)
}
```

- Add battlesWon to the Party object

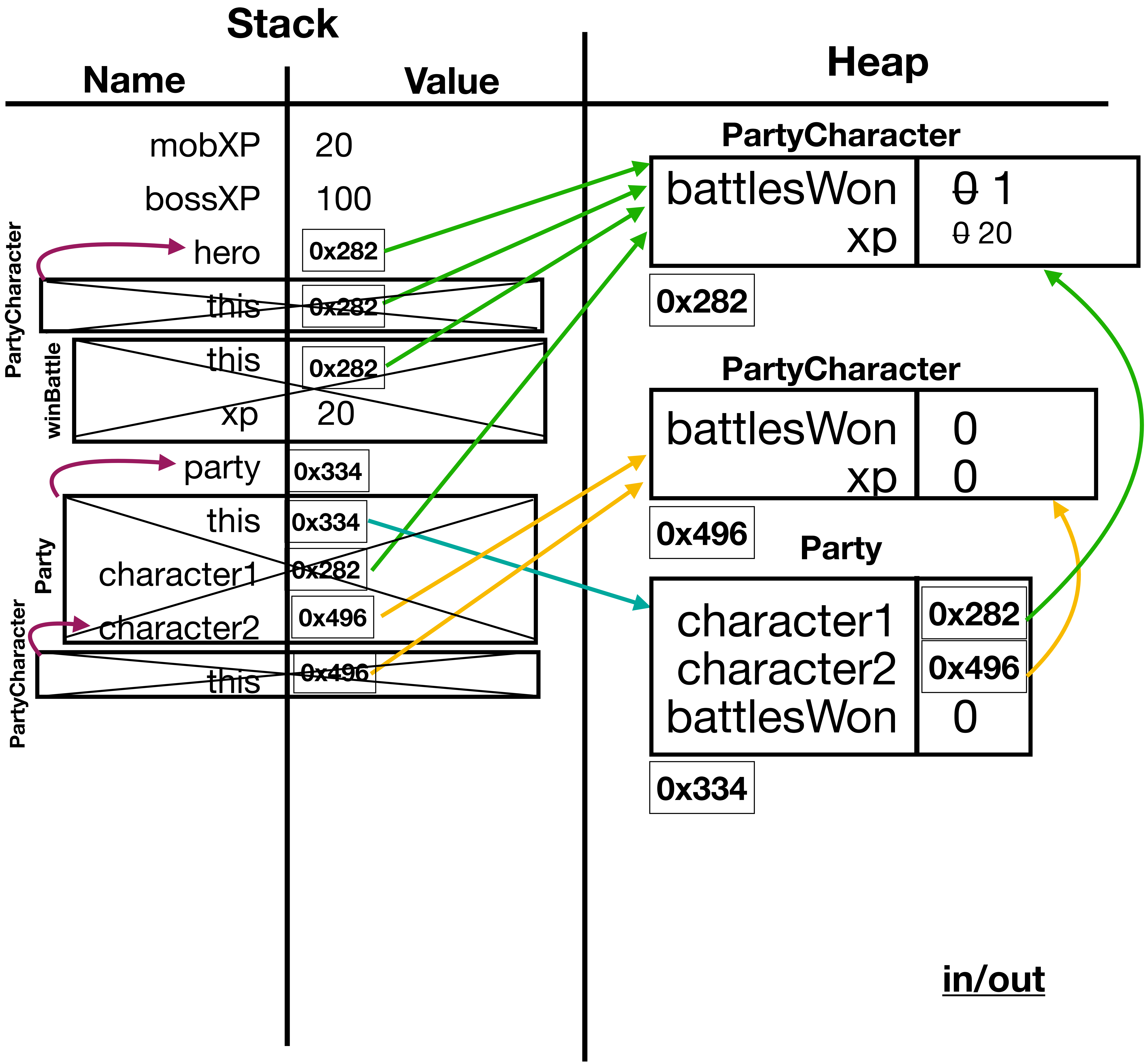


```
class PartyCharacter() {
  var battlesWon: Int = 0
  var xp: Int = 0

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

```
class Party(val character1: PartyCharacter,
            val character2: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.character1.winBattle(xp)
    this.character2.winBattle(xp)
  }
}
```

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

  println(hero.xp)
  println(party.characterTwo.xp)
}
```

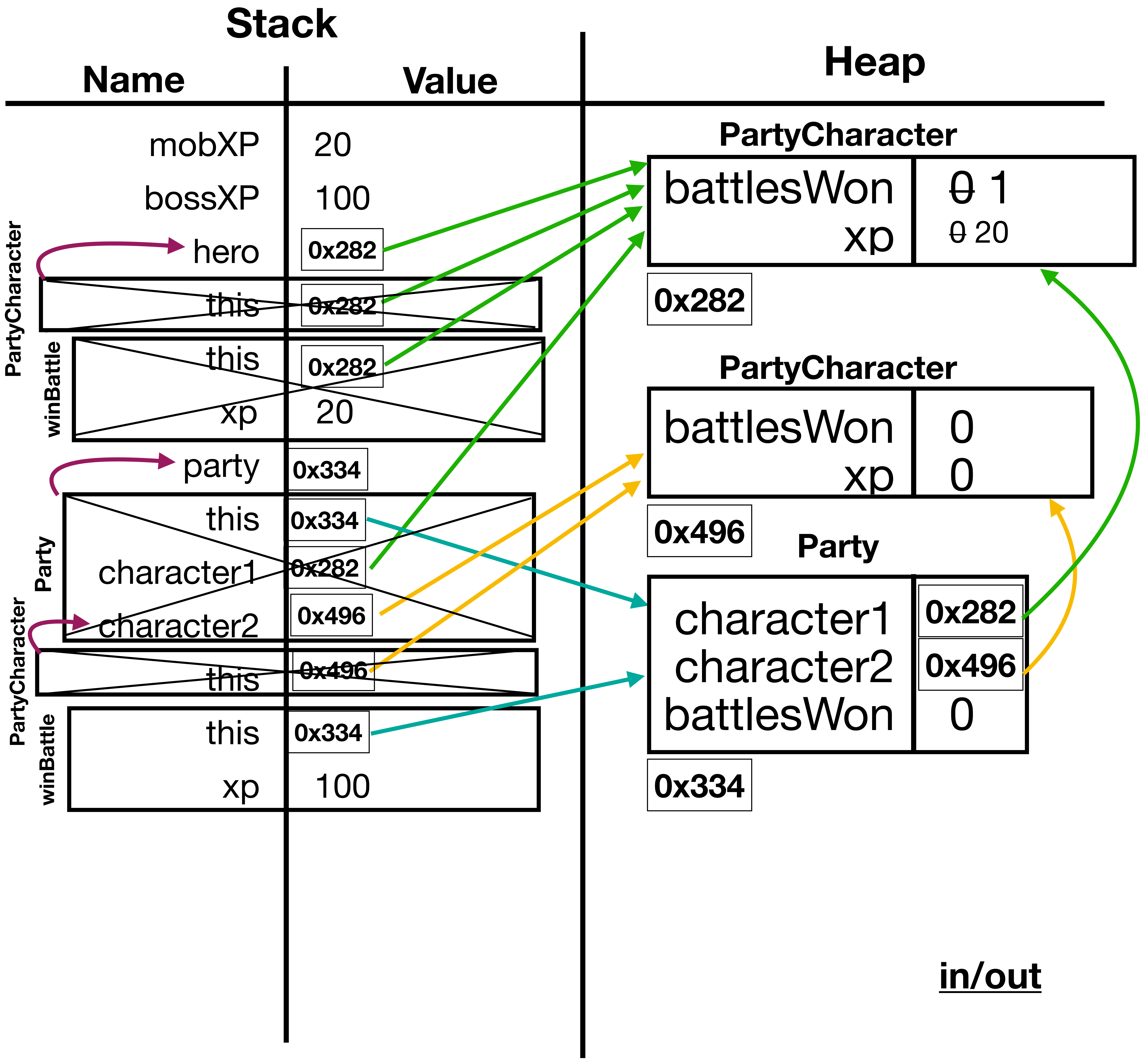
- Constructor call ends and returns a reference to the new Party



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

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

```
class Party(val character1: PartyCharacter,
            val character2: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.character1.winBattle(xp)
    this.character2.winBattle(xp)
  }
}
```

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

  println(hero.xp)
  println(party.characterTwo.xp)
}
```



- Call winBattle and add a stack frame

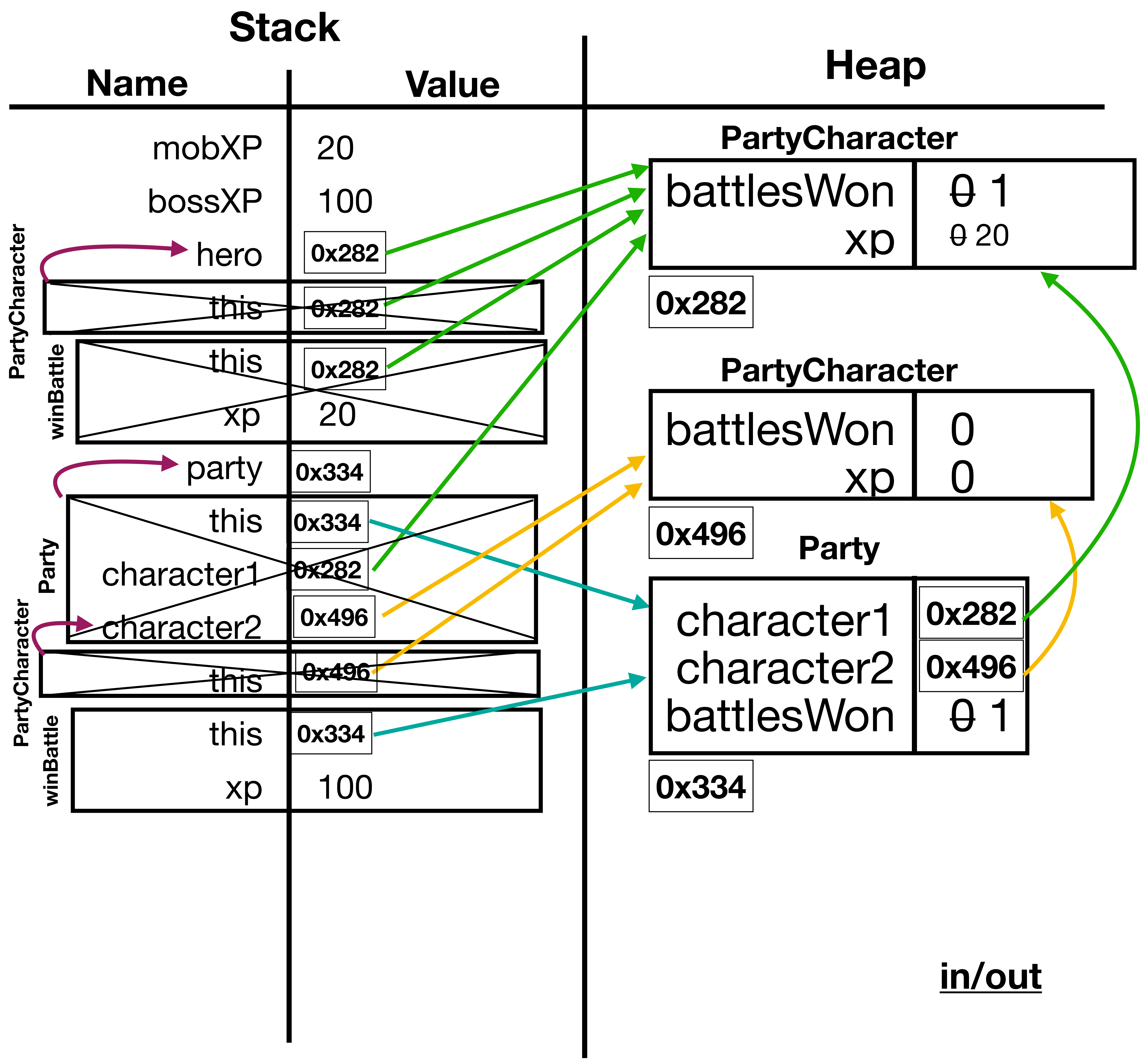
in/out

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

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

```scala
class Party(val character1: PartyCharacter,
            val character2: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.character1.winBattle(xp)
    this.character2.winBattle(xp)
  }
}
```

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

  println(hero.xp)
  println(party.characterTwo.xp)
}
```

- Increment battlesWon

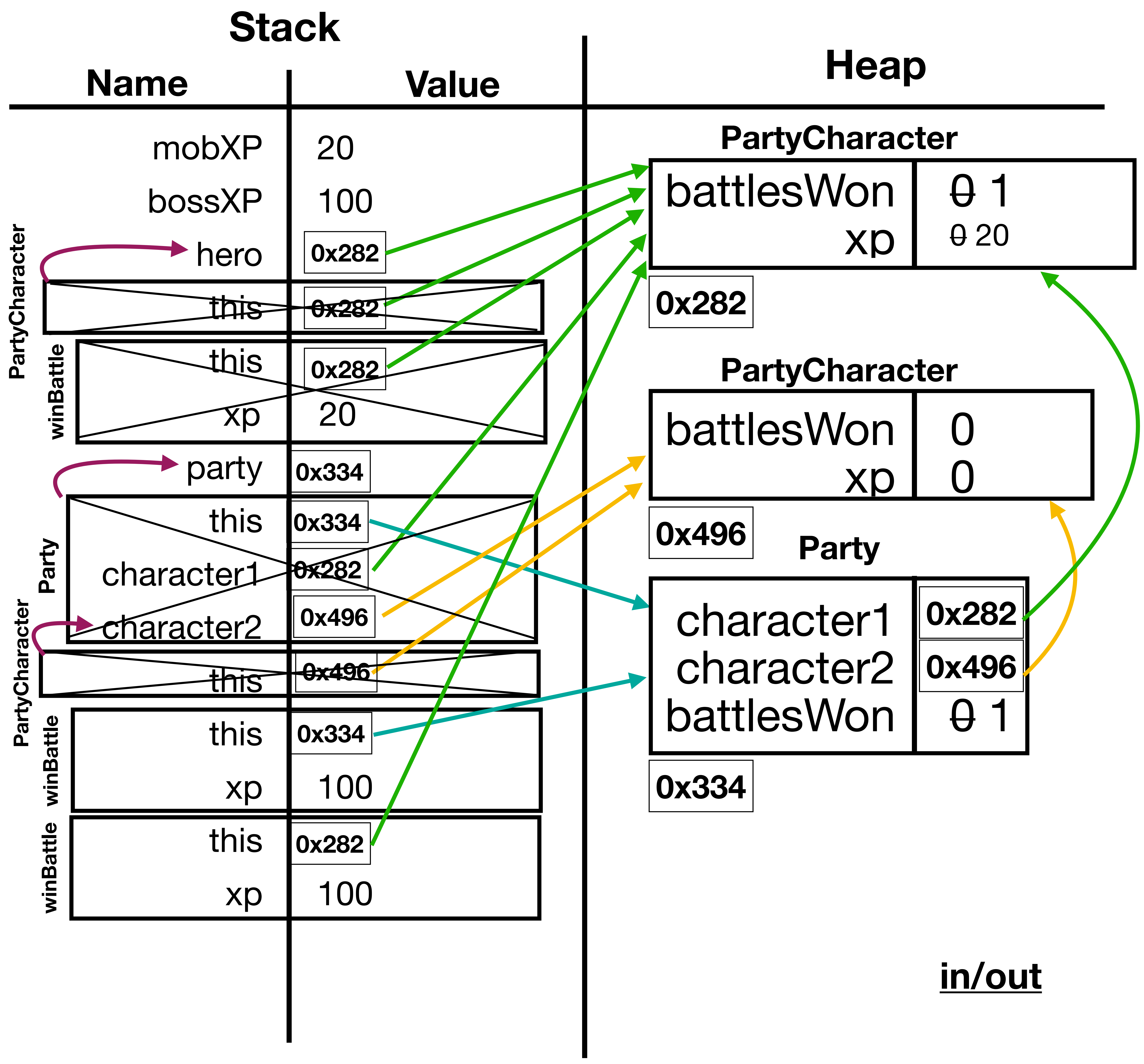


```
class PartyCharacter() {
  var battlesWon: Int = 0
  var xp: Int = 0

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

```
class Party(val character1: PartyCharacter,
            val character2: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.character1.winBattle(xp)
    this.character2.winBattle(xp)
  }
}
```

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

  println(hero.xp)
  println(party.characterTwo.xp)
}
```

- Create another stack frame



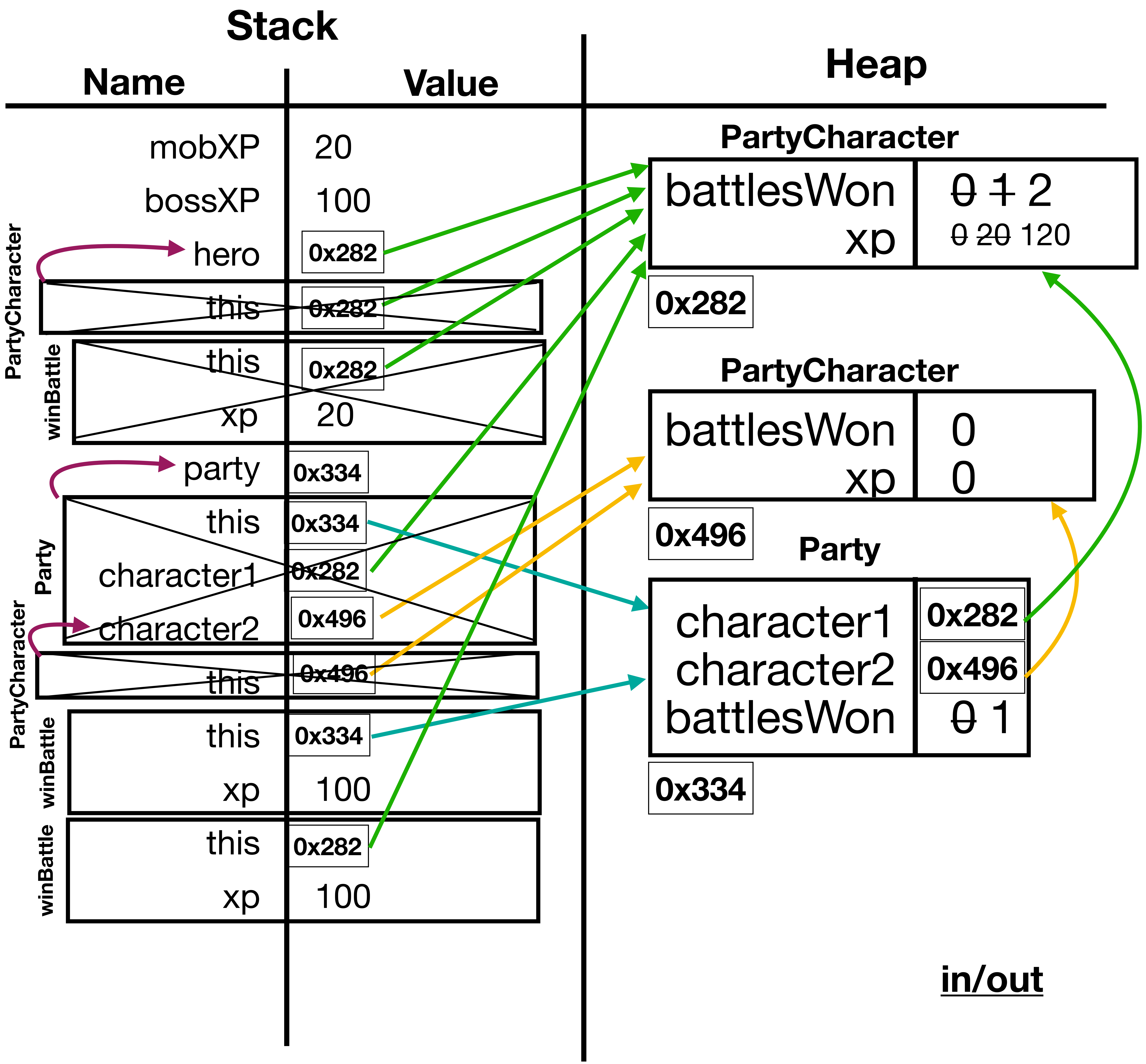
in/out

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

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

```
class Party(val character1: PartyCharacter,
            val character2: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.character1.winBattle(xp)
    this.character2.winBattle(xp)
  }
}
```

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

  println(hero.xp)
  println(party.characterTwo.xp)
}
```

- Update values



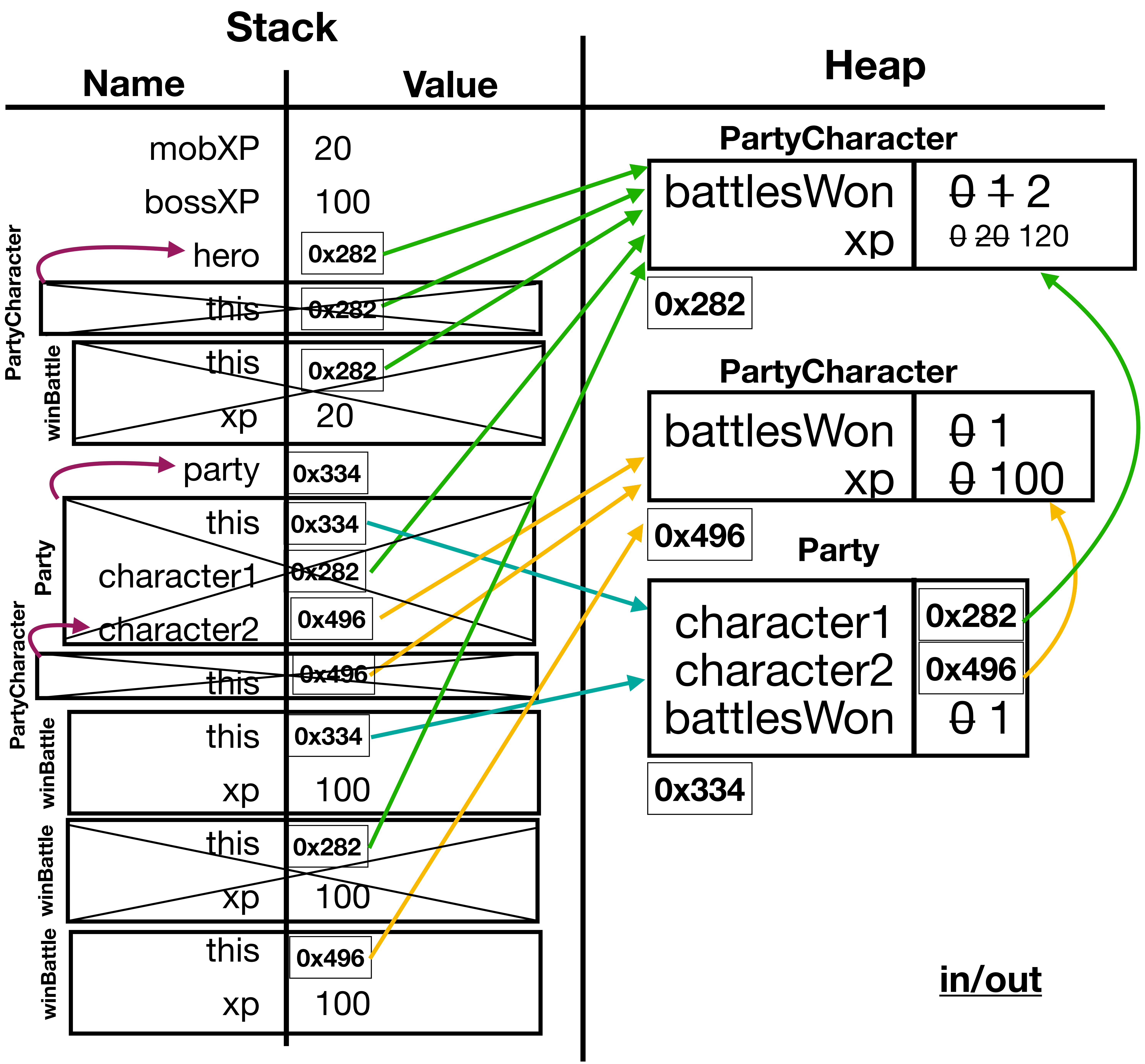
in/out

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

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

```scala
class Party(val character1: PartyCharacter,
            val character2: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.character1.winBattle(xp)
    this.character2.winBattle(xp)
  }
}
```

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

  println(hero.xp)
  println(party.characterTwo.xp)
}
```

- Stack frame ends
- Repeat the process for character2



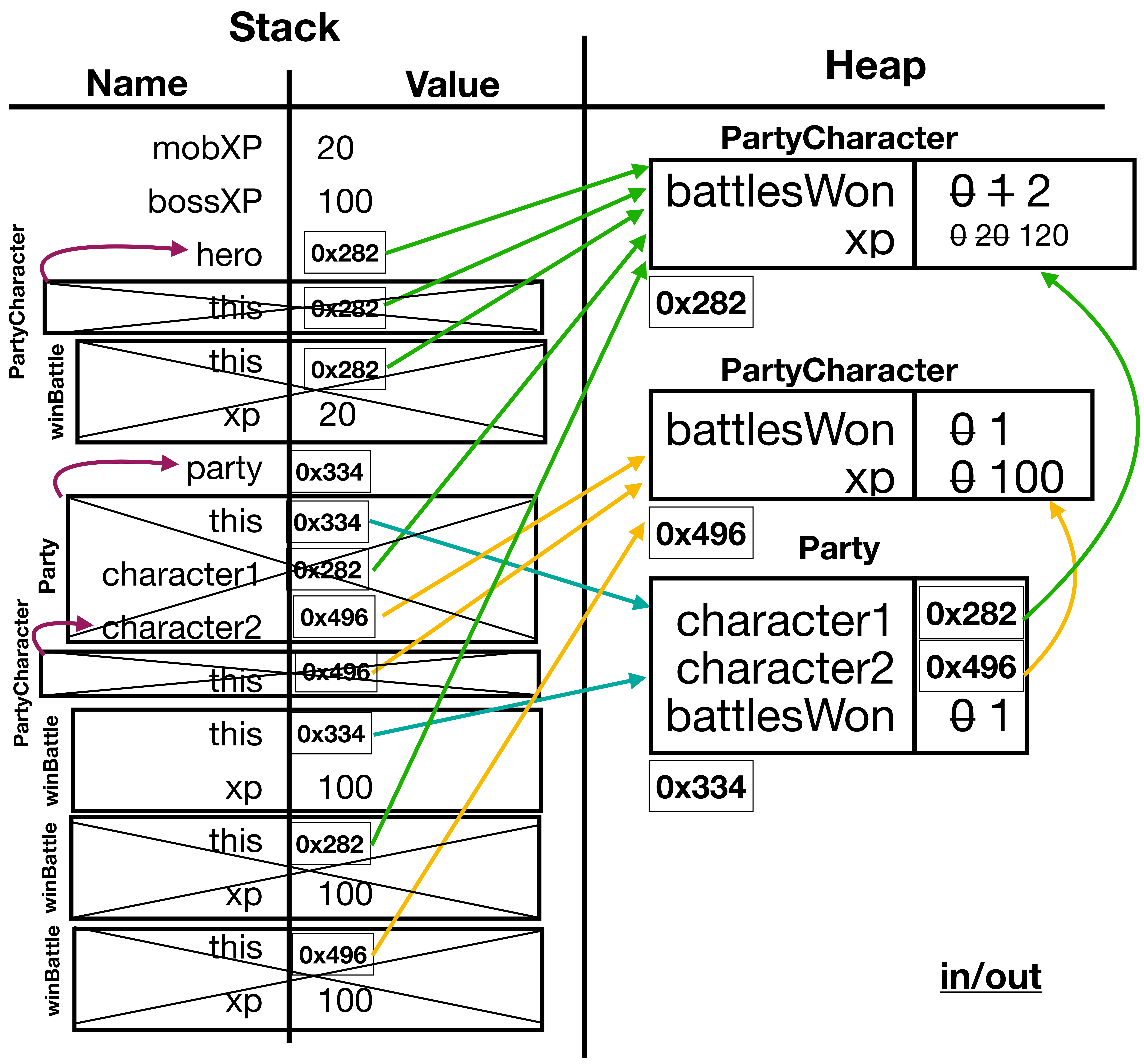
in/out

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

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

```
class Party(val character1: PartyCharacter,
            val character2: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.character1.winBattle(xp)
    this.character2.winBattle(xp)
  }
}
```

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

  println(hero.xp)
  println(party.characterTwo.xp)
}
```

- Top stack frame ends

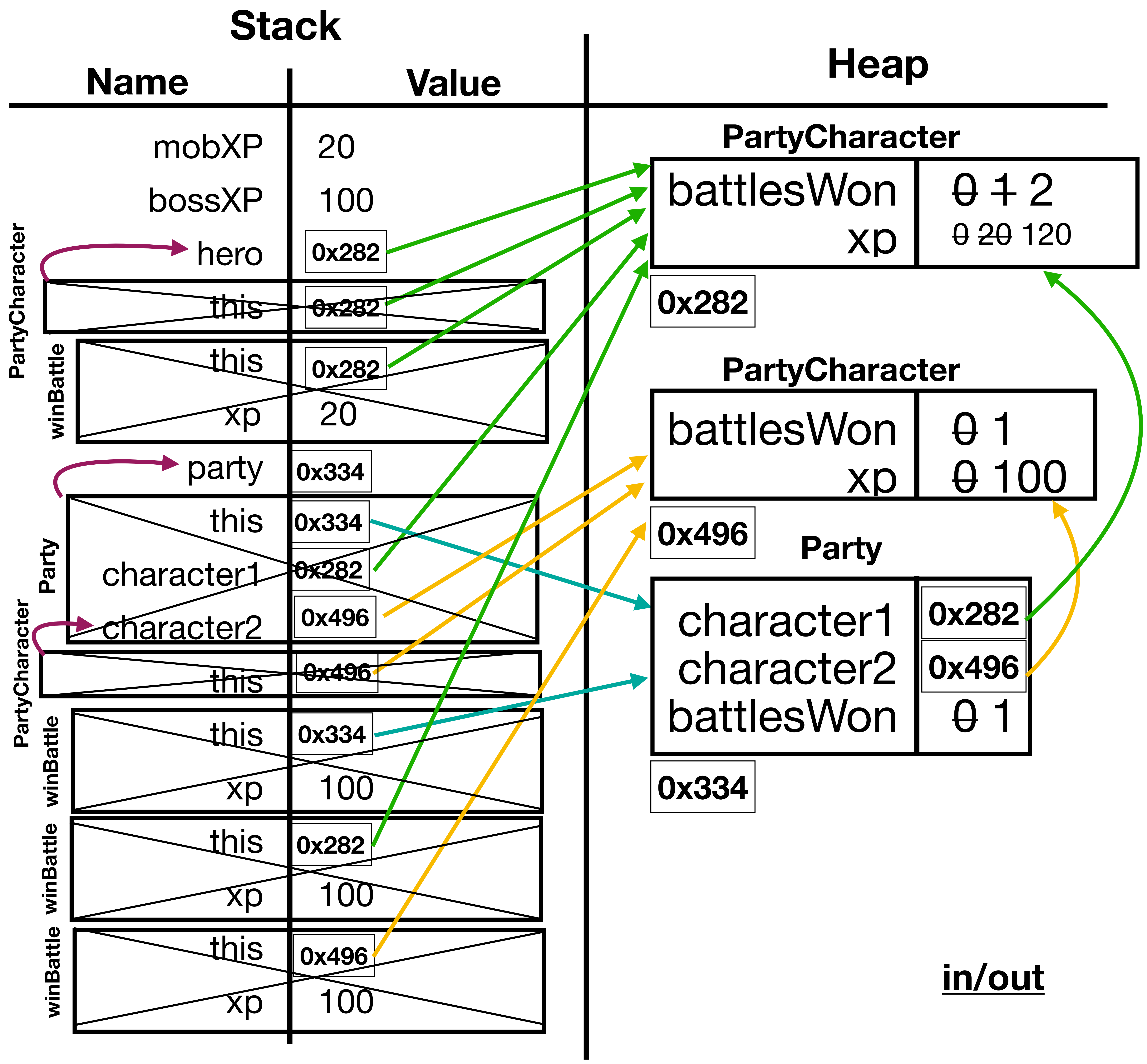


```
class PartyCharacter() {
  var battlesWon: Int = 0
  var xp: Int = 0

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

```
class Party(val character1: PartyCharacter,
            val character2: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.character1.winBattle(xp)
    this.character2.winBattle(xp)
  }
}
```

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

  println(hero.xp)
  println(party.characterTwo.xp)
}
```

- Party stack frame ends

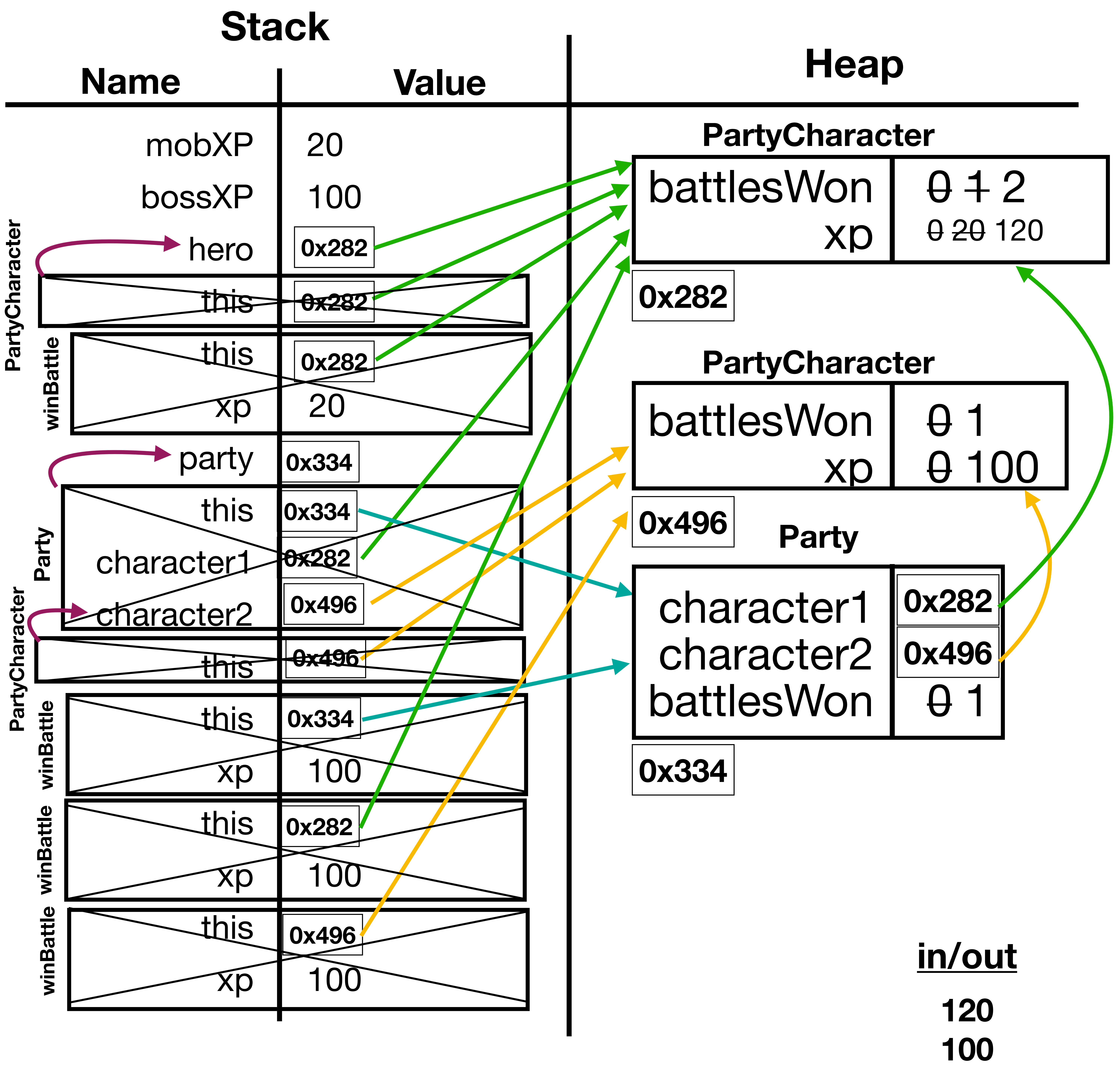


```
class PartyCharacter() {
  var battlesWon: Int = 0
  var xp: Int = 0

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

```
class Party(val character1: PartyCharacter,
            val character2: PartyCharacter) {

  var battlesWon: Int = 0

  def winBattle(xp: Int): Unit = {
    this.battlesWon += 1
    this.character1.winBattle(xp)
    this.character2.winBattle(xp)
  }
}
```

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

  println(hero.xp)
  println(party.characterTwo.xp)
}
```

- Print values to the screen
- end the program



Stack
Name
Value
Heap
mobXP 20
bossXP 100
hero 0x282
PartyCharacter
this 0x282
winBattle
this 0x282
xp 20
party 0x334
Party
this 0x334
character1 0x282
character2 0x496
PartyCharacter
this 0x496
winBattle
this 0x334
xp 100
winBattle
this 0x282
xp 100
winBattle
this 0x496
xp 100
PartyCharacter
battlesWon ~~0~~ ~~1~~ 2
xp ~~0~~ ~~20~~ 120
0x282
PartyCharacter
battlesWon ~~0~~ 1
xp ~~0~~ 100
0x496
Party
character1 0x282
character2 0x496
battlesWon ~~0~~ 1
0x334
in/out
120
100