Heap Memory

Another Memory Example

```
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)
  }
}
```

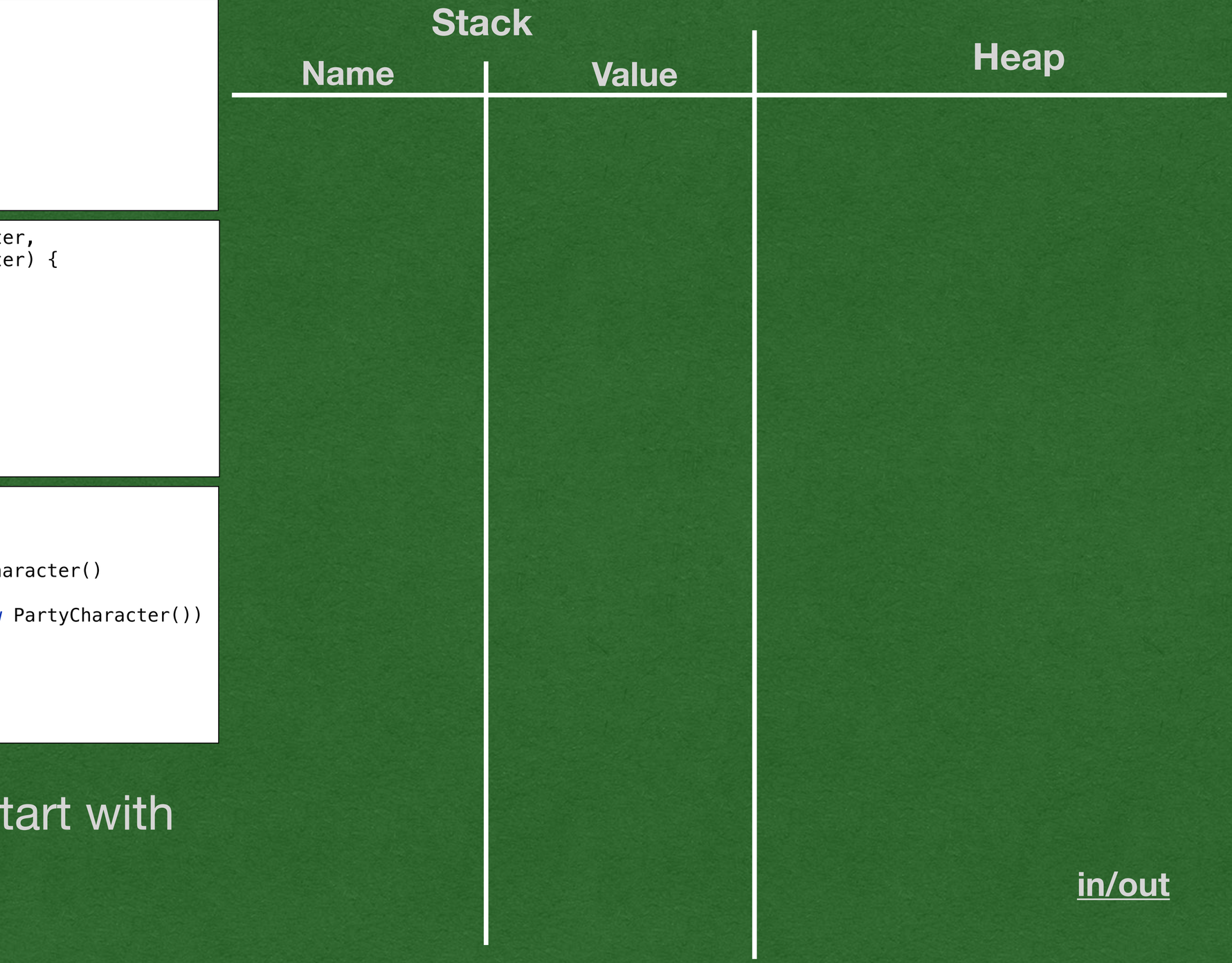
```
def main(args: Array[String]): Unit = {
  val mobXP: Int = 20
  val bossXP: Int = 100
  val hero: PartyCharacter = new PartyCharacter()
  hero.winBattle(mobXP)
  val party: Party = new Party(hero, new PartyCharacter())
  party.winBattle(bossXP)

  println(hero.xp)
  println(party.character2.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.character2.xp)
}
```

- Programs always start with the main method



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.character2.xp)
}
```

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

Stack

Name	Value
mobXP	20
bossXP	100

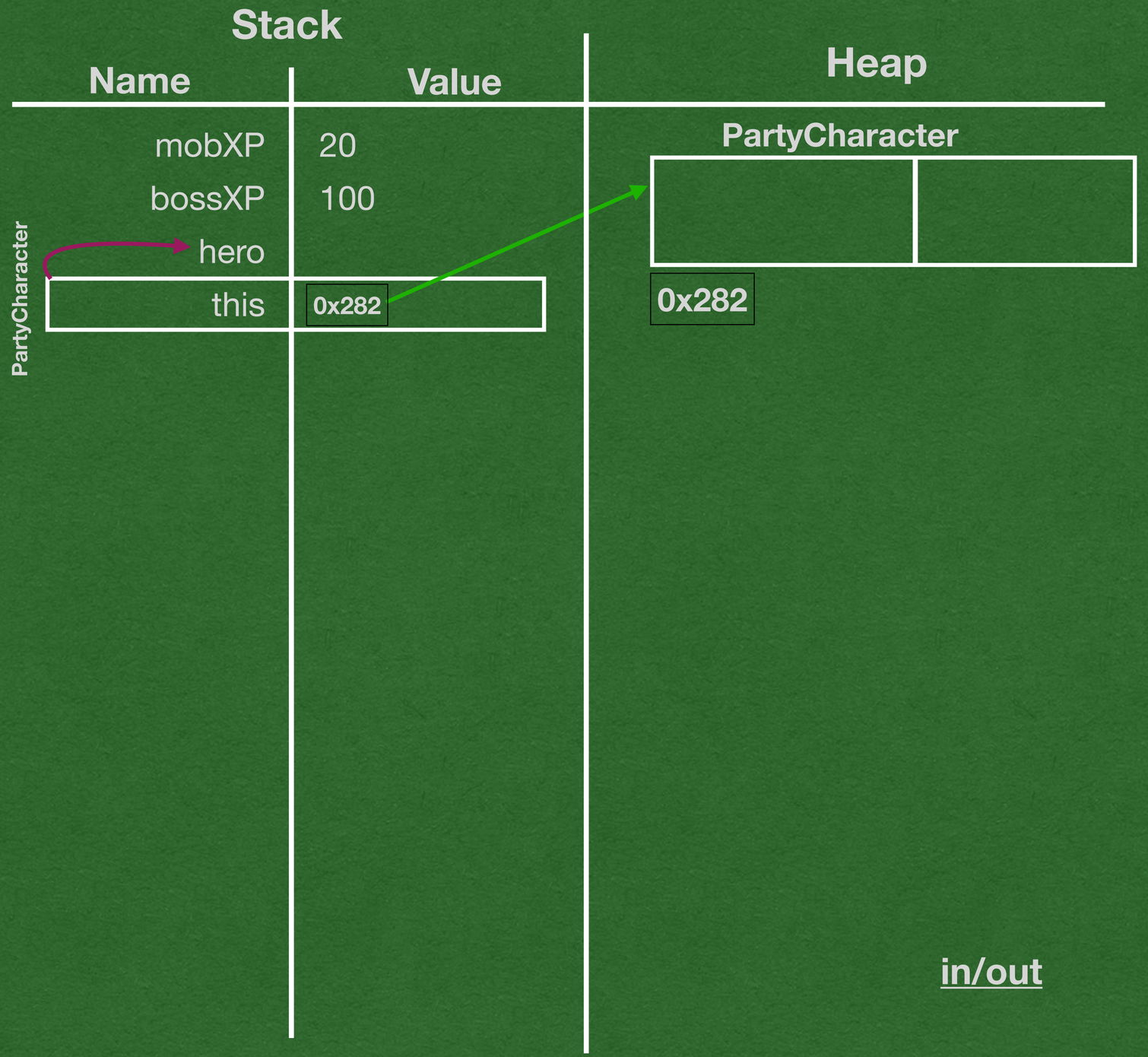
Heap

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.character2.xp)
}
```



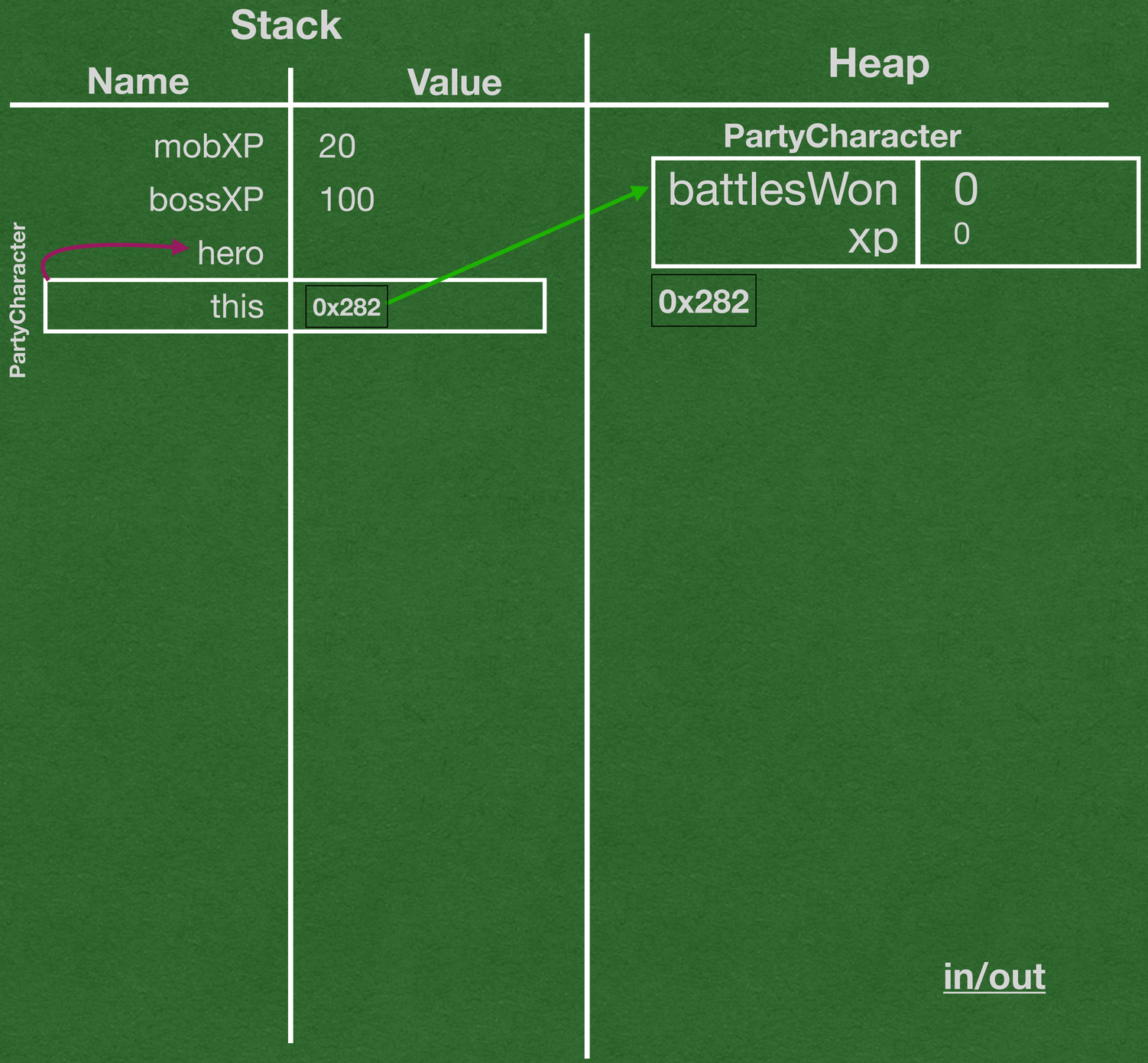
- Create a new stack frame for the constructor call
- "this" contains a reference to the object being constructed

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.character2.xp)
}
```



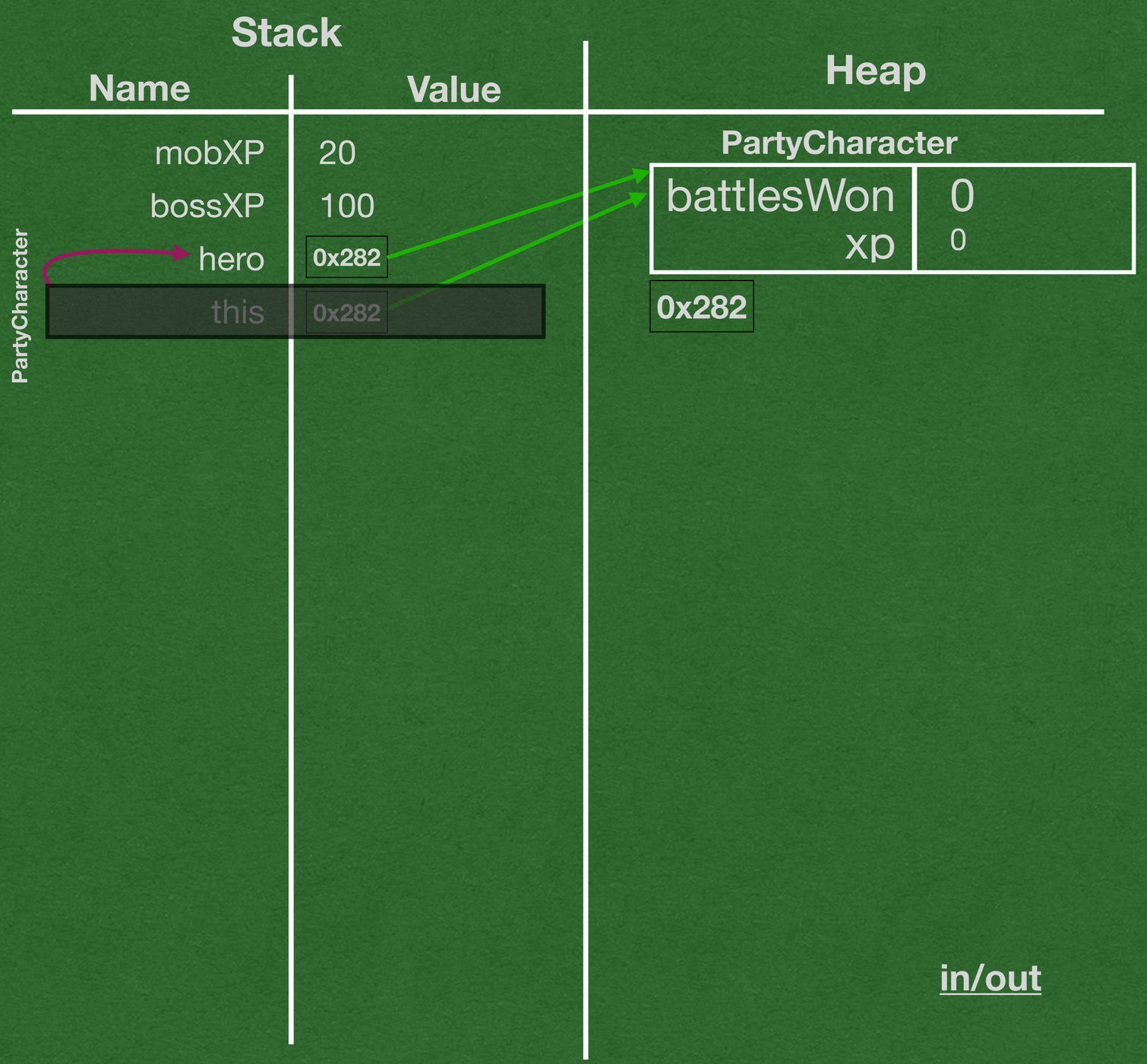
- Run all the code that's outside of the methods
- All declared variables become state variables and are stored with the object

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.character2.xp)
}
```



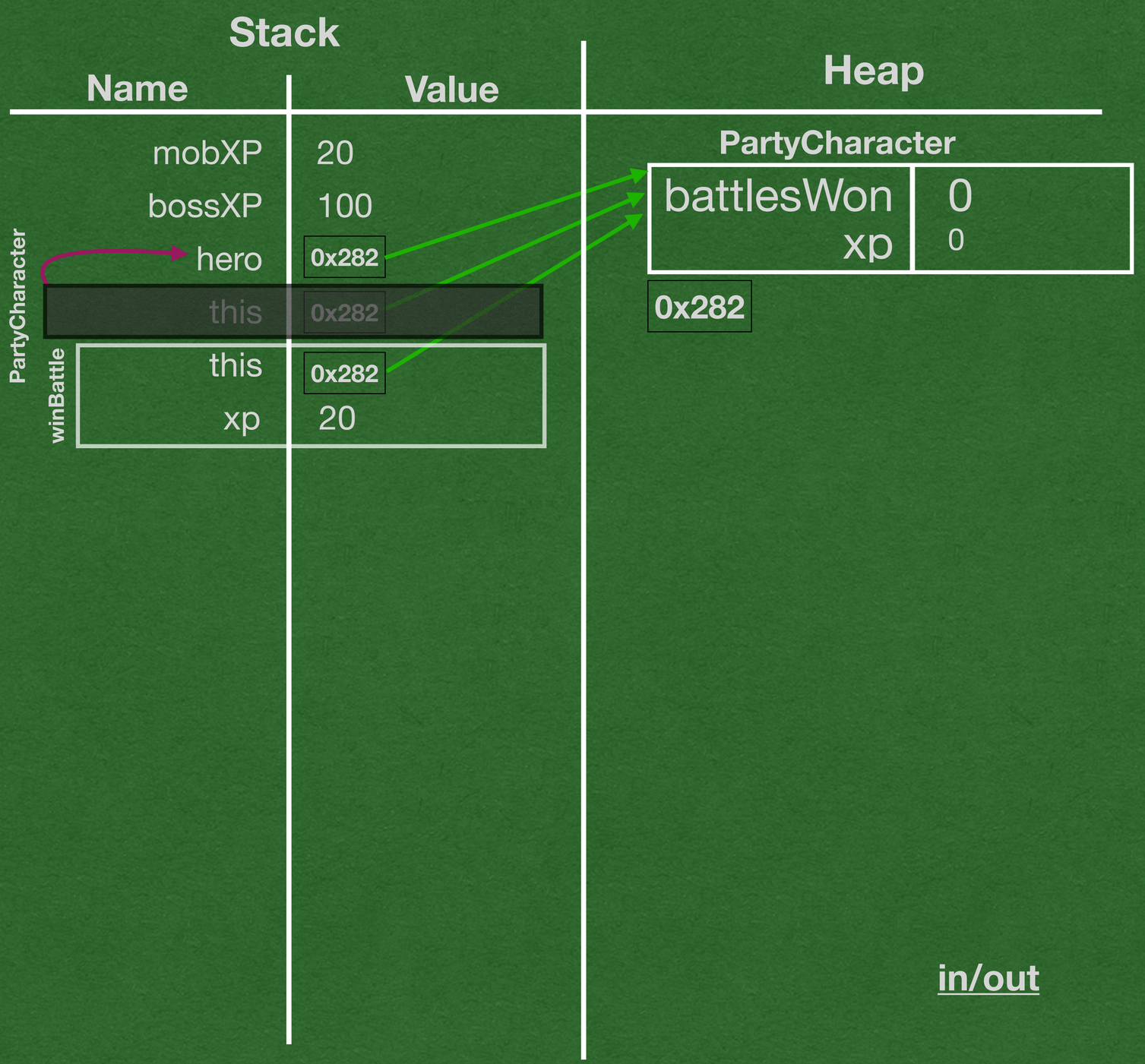
- Constructor stack frame ends and is removed from the stack
- Return a reference to the newly constructed object to hero

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.character2.xp)
}
```



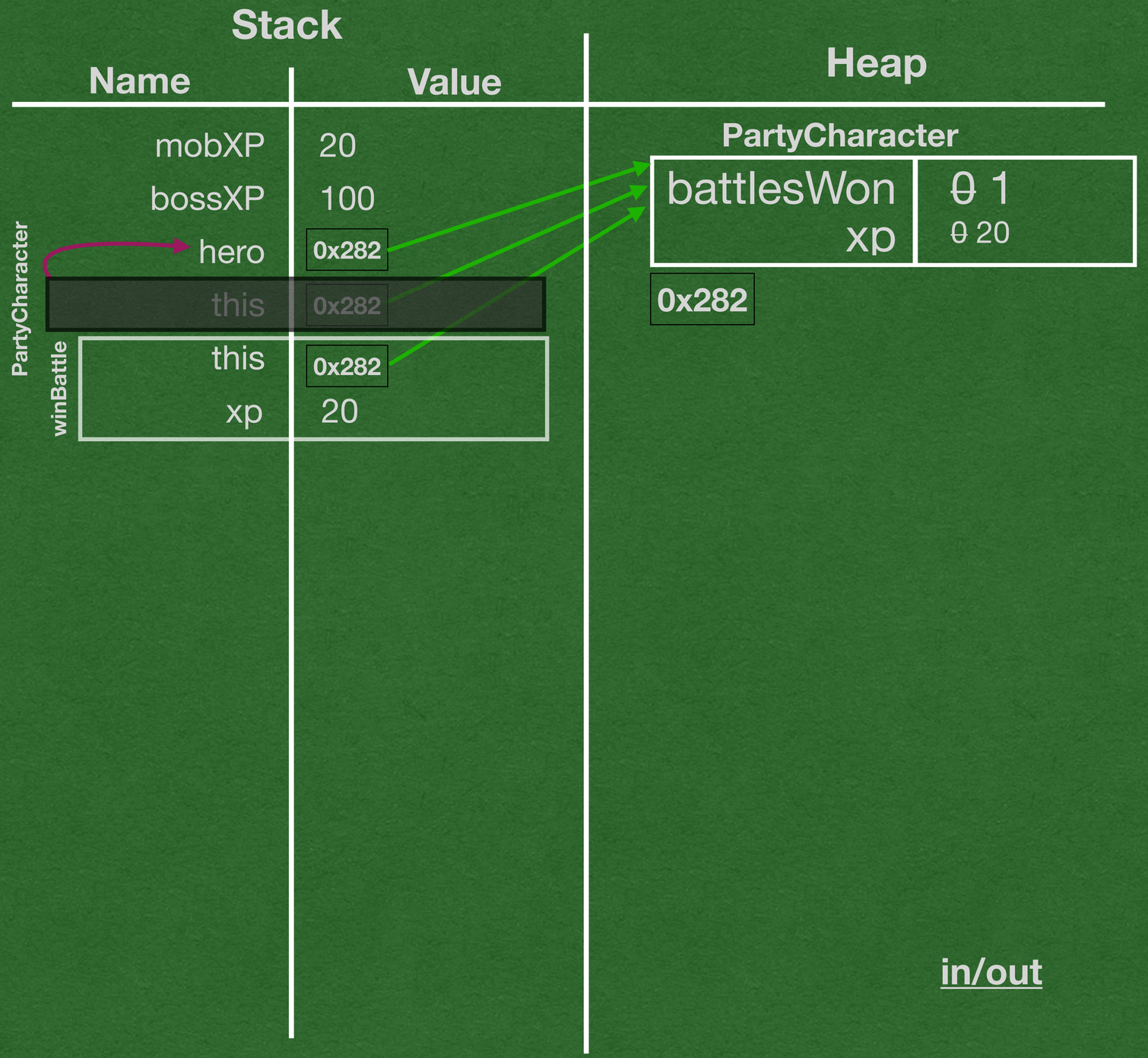
- Create a stack frame for the winBattle method call
- "this" stores a reference to the calling object

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.character2.xp)
}
```



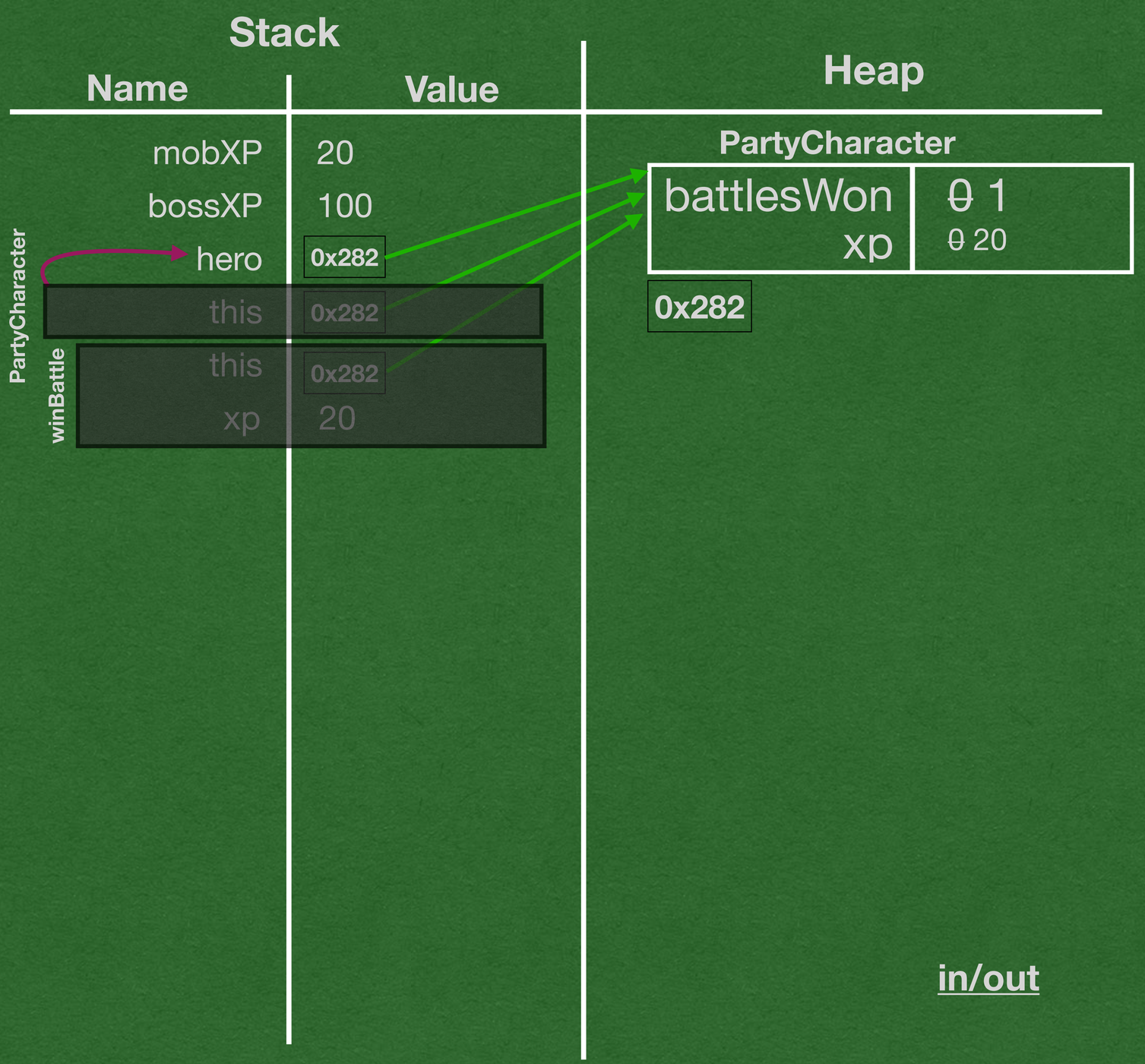
- Update the battlesWon and xp of "this"

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.character2.xp)
}
```



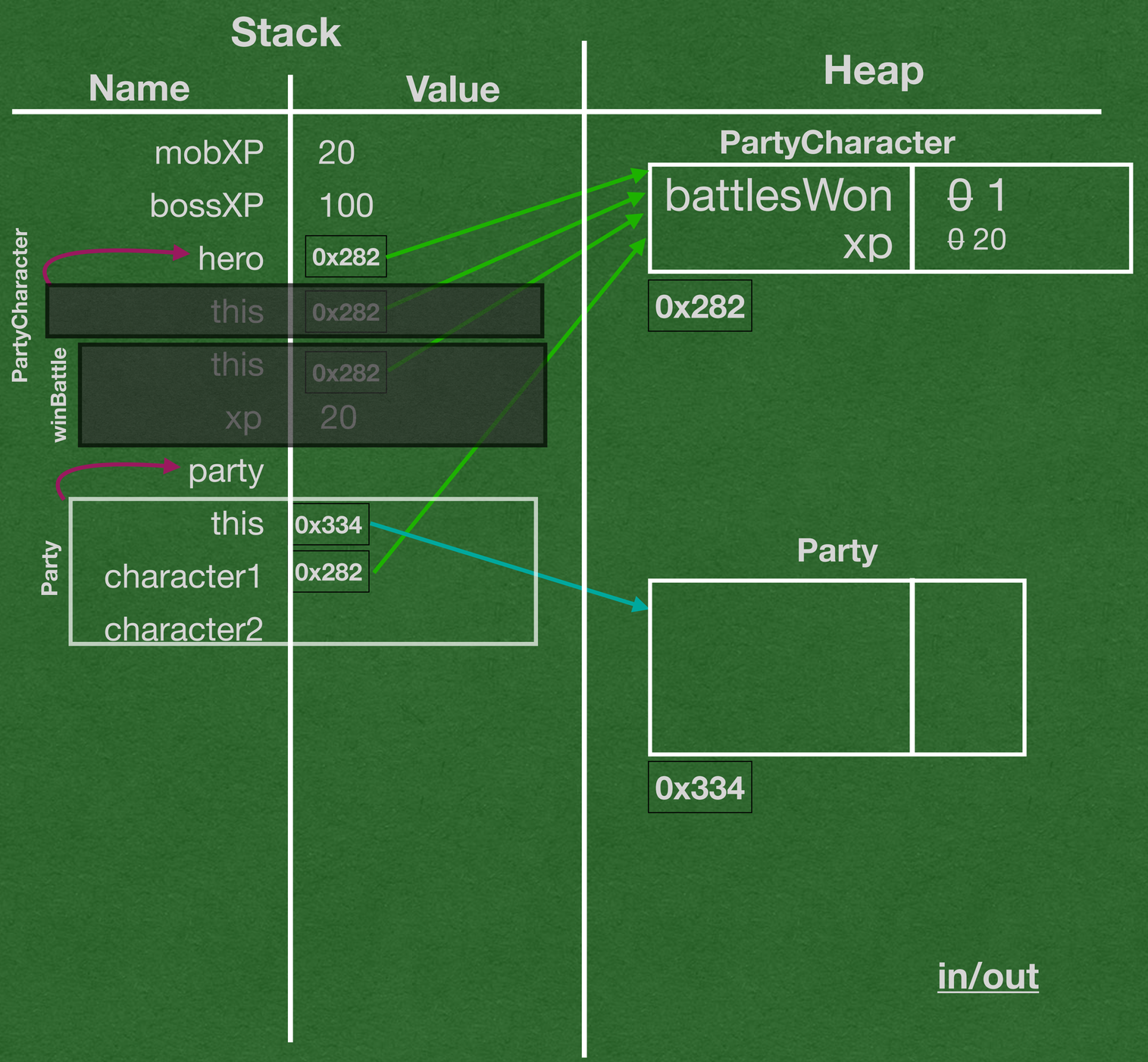
- The stack frame ends and is removed from the stack
- The changes made to the heap persist!

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.character2.xp)
}
```



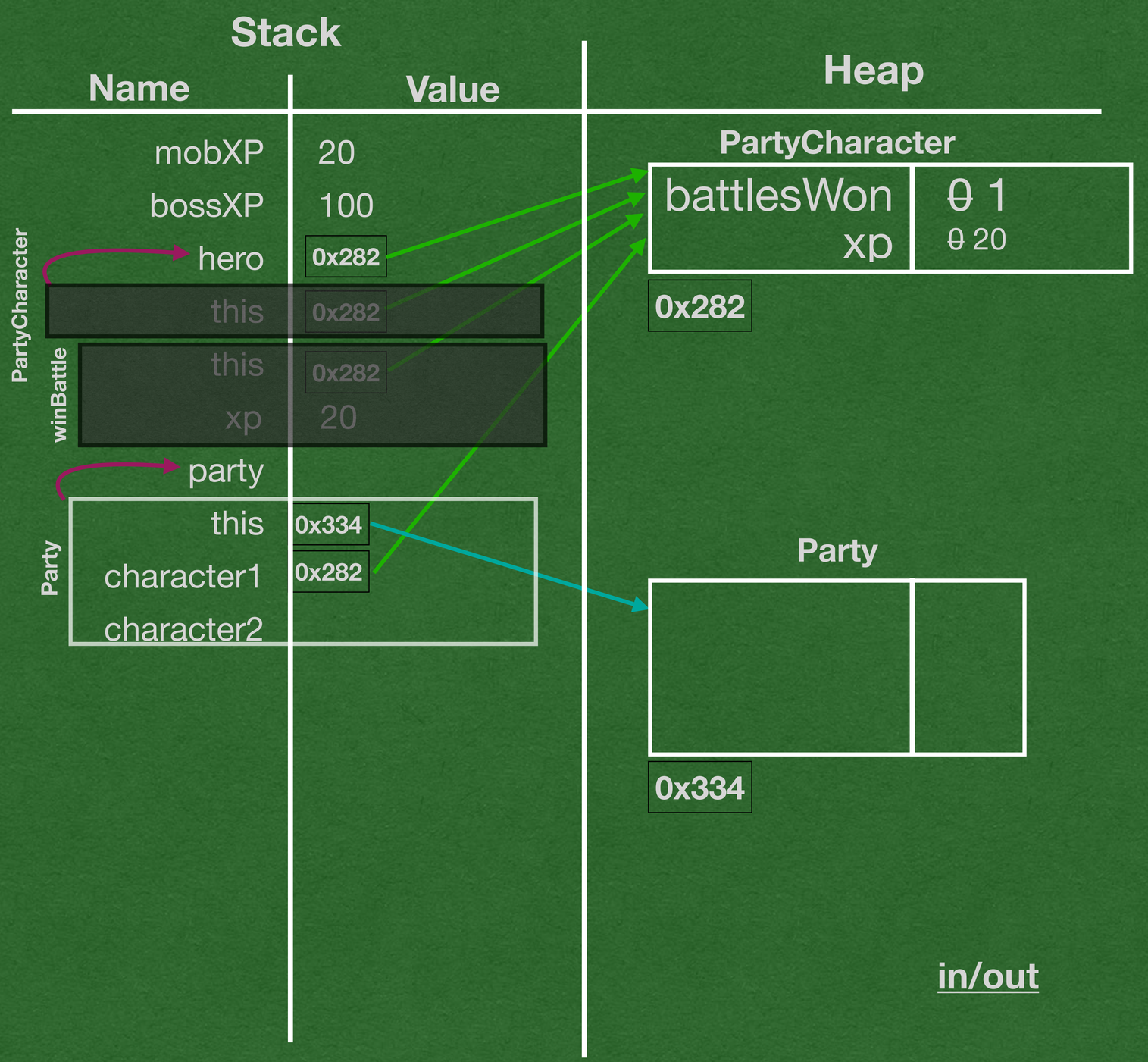
- Create a variable to store a new party
- Call the Party constructor and draw a stack frame

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.character2.xp)
}
```



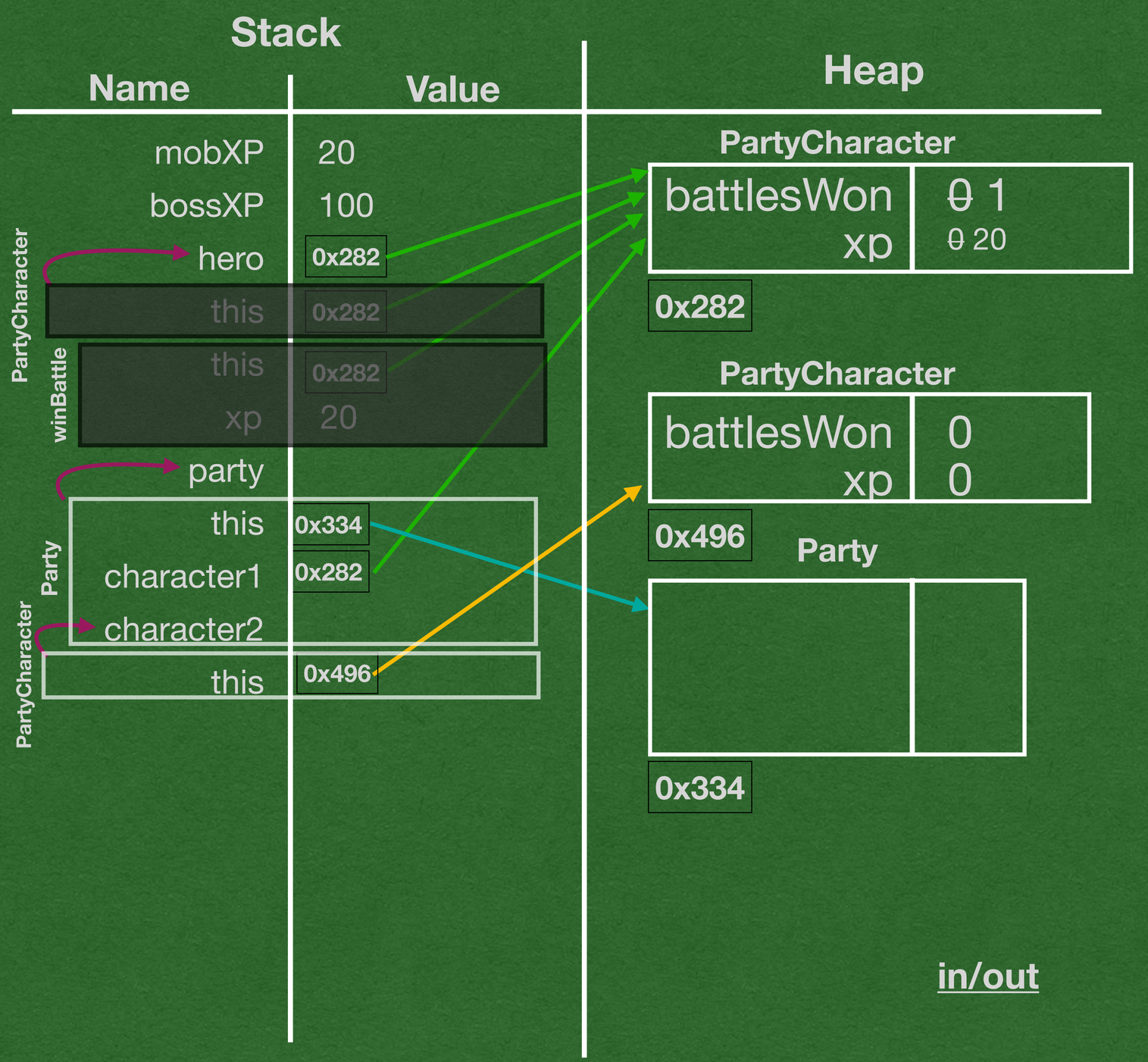
- **But what's the value of character2??**
- We need to create another stack frame for a PartyCharacter constructor

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.character2.xp)
}
```



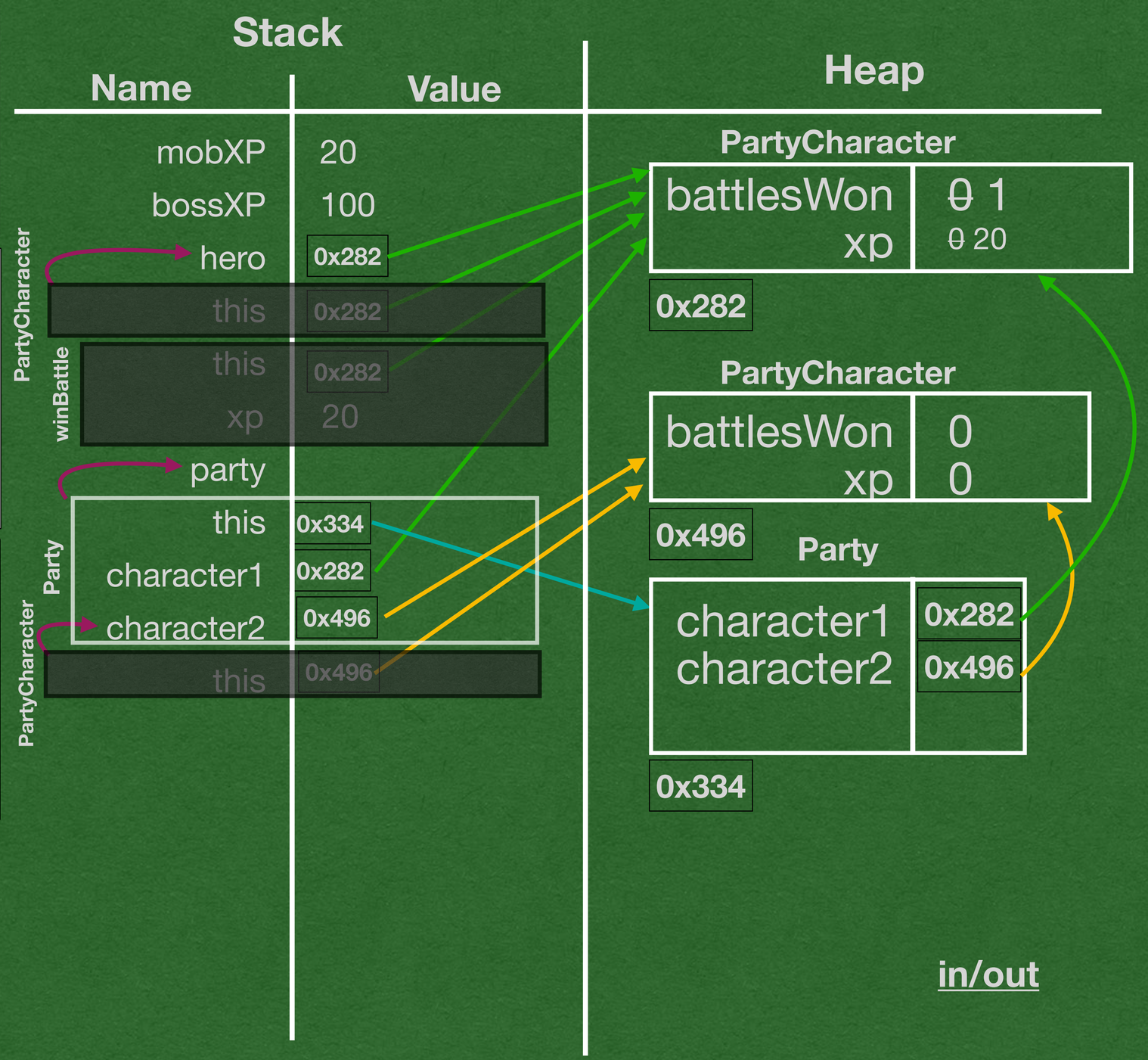
- The PartyCharacter constructor will return directly to the character2 parameter of the Party constructor

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.character2.xp)
}
```



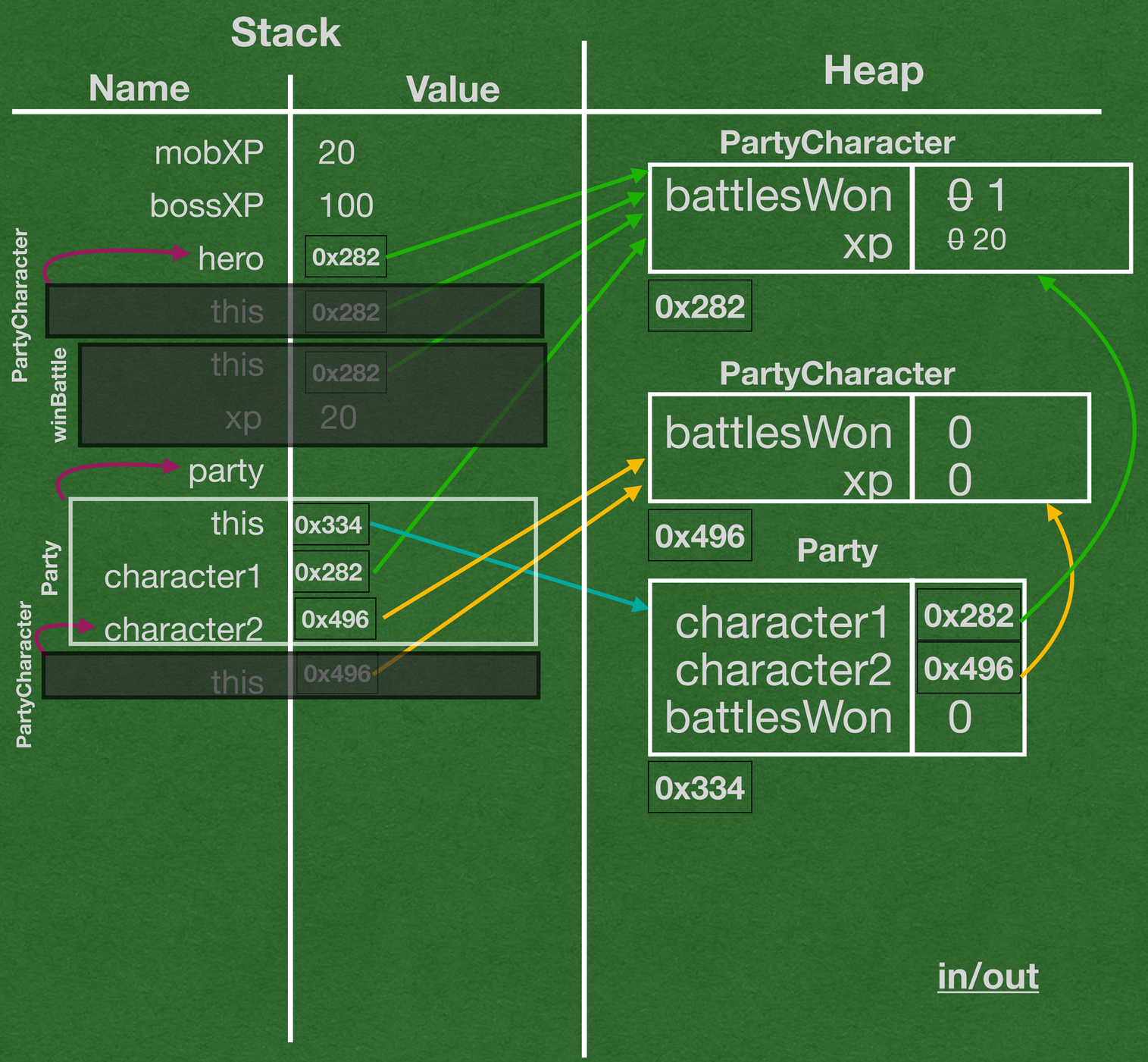
- Now that we have all the parameters
  - We can run the Party constructor

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.character2.xp)
}
```



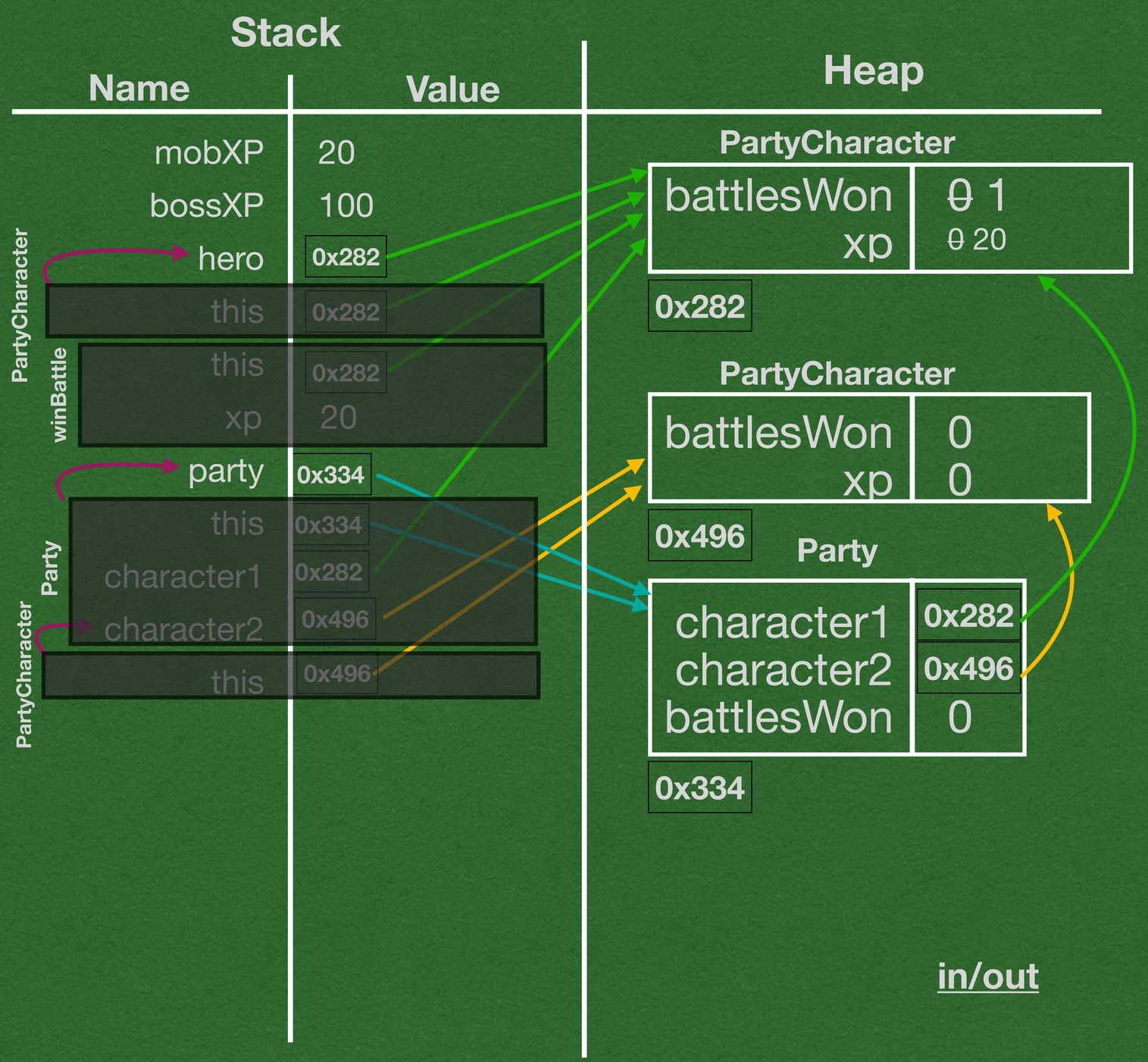
- Add battlesWon to the Party object

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.character2.xp)
}
```

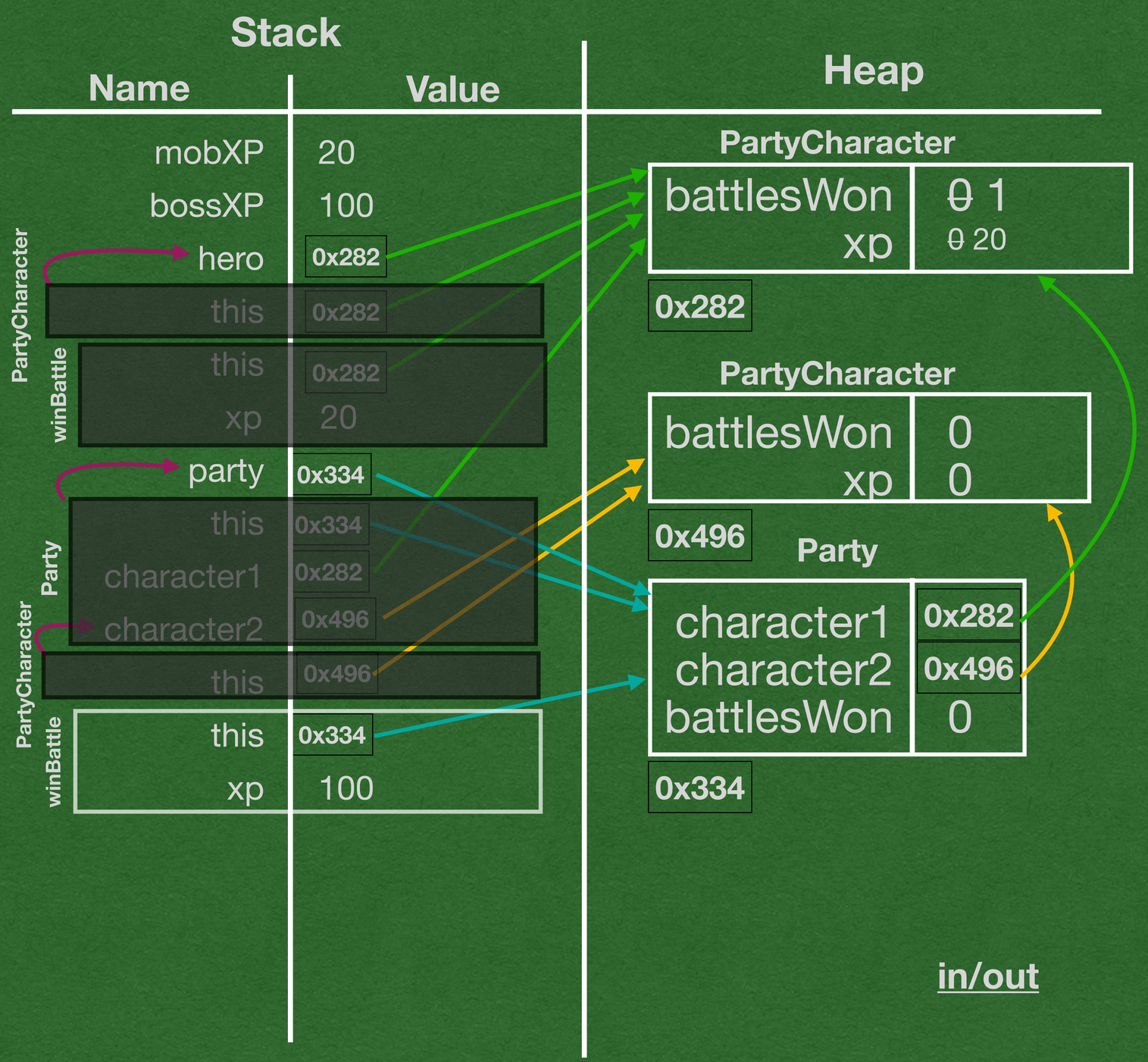


- Constructor call ends and returns a reference to the new Party

```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.character2.xp)
}
```



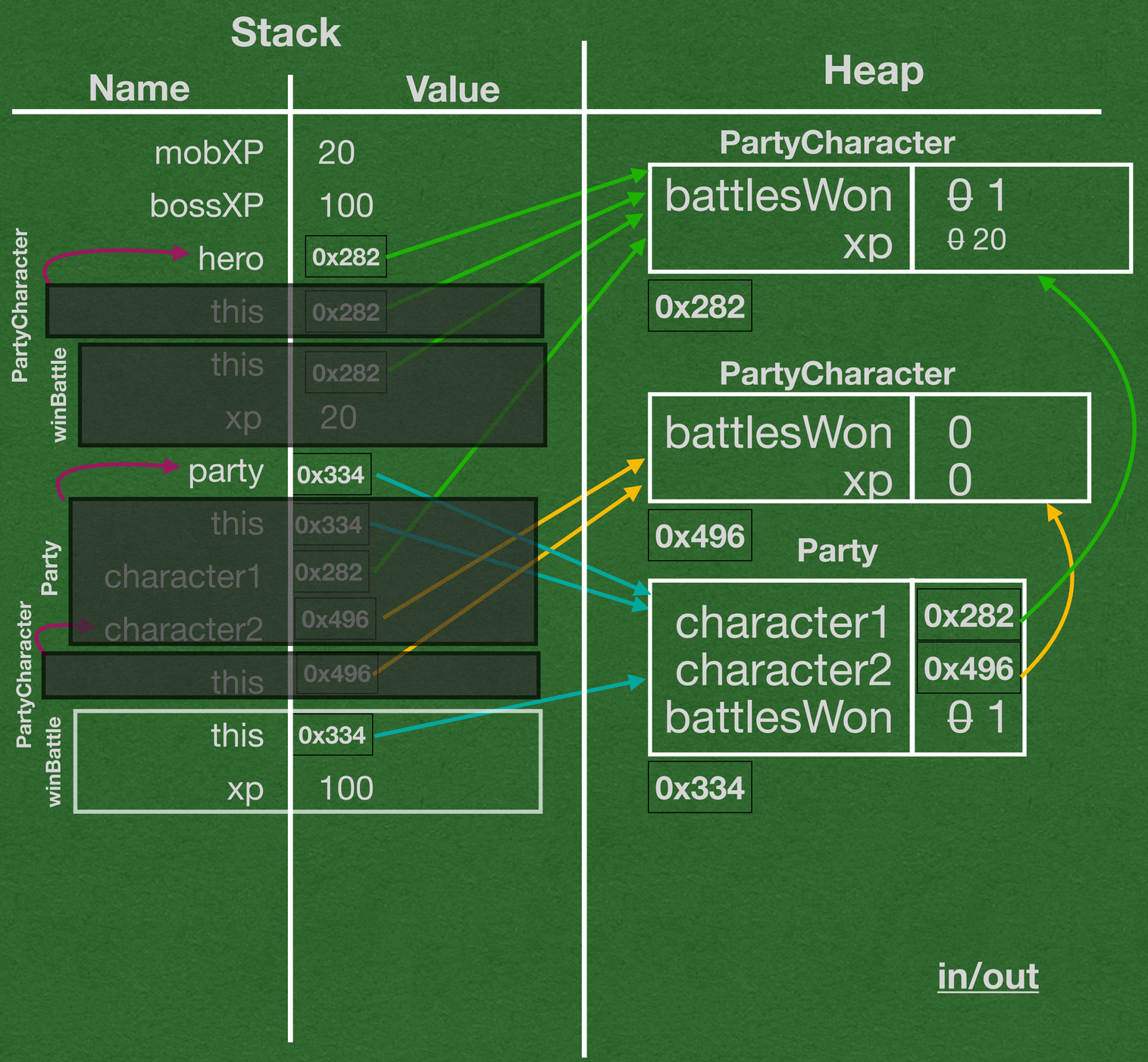
- Call winBattle and add a stack frame

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.character2.xp)
}
```



- Increment battlesWon

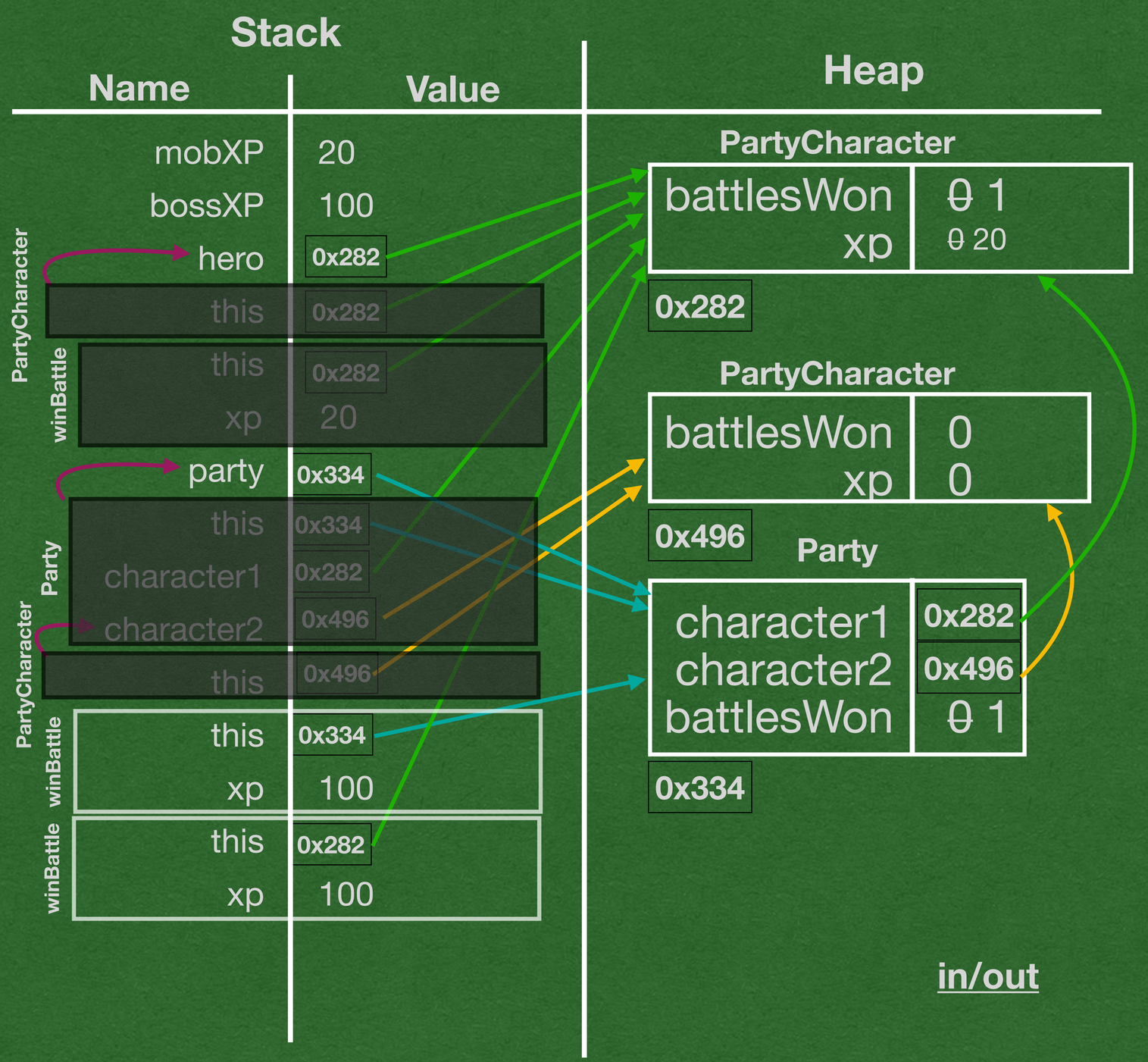
in/out

```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.character2.xp)
}
```

- Create another stack frame



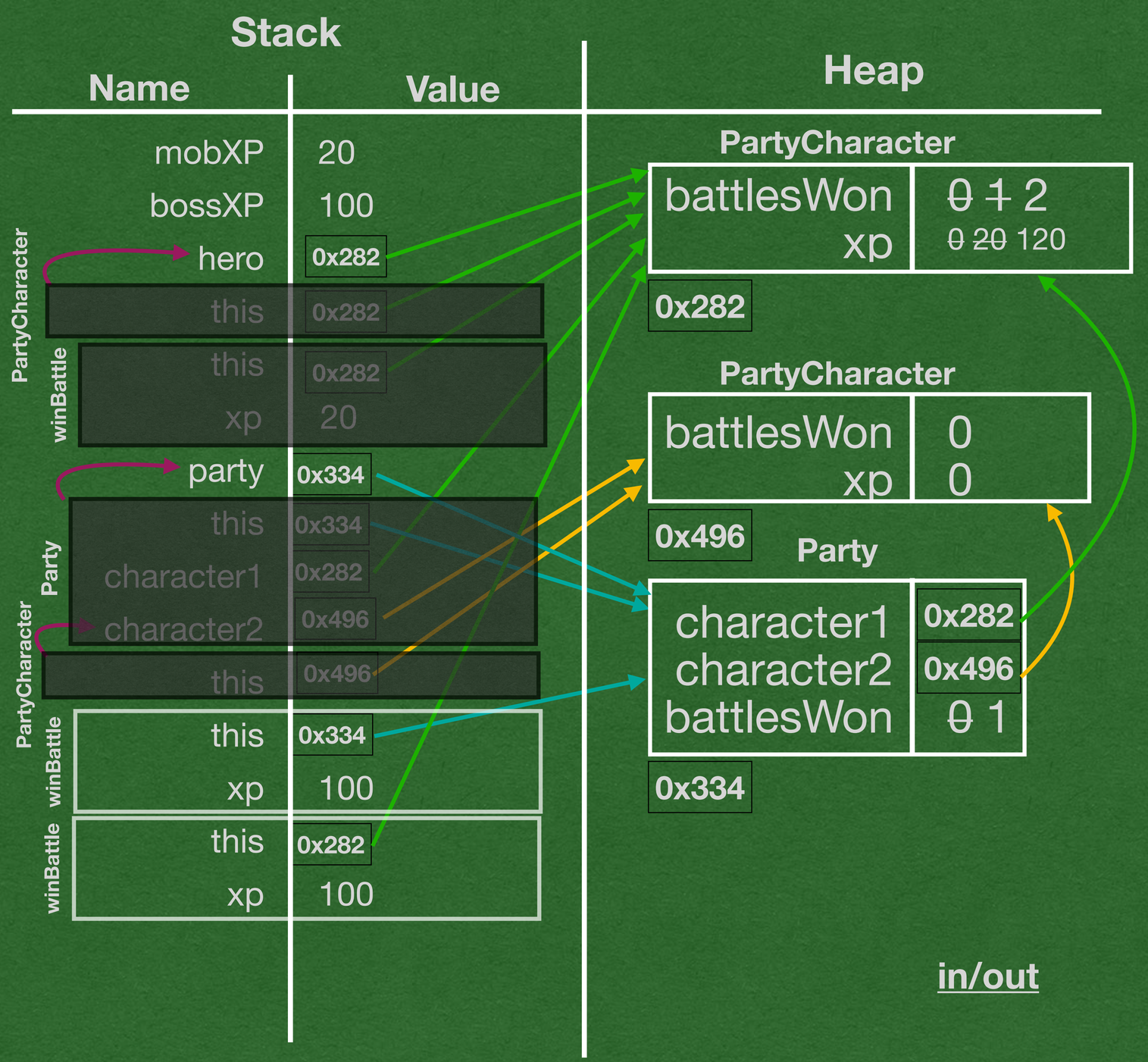
in/out

```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.character2.xp)
}
```

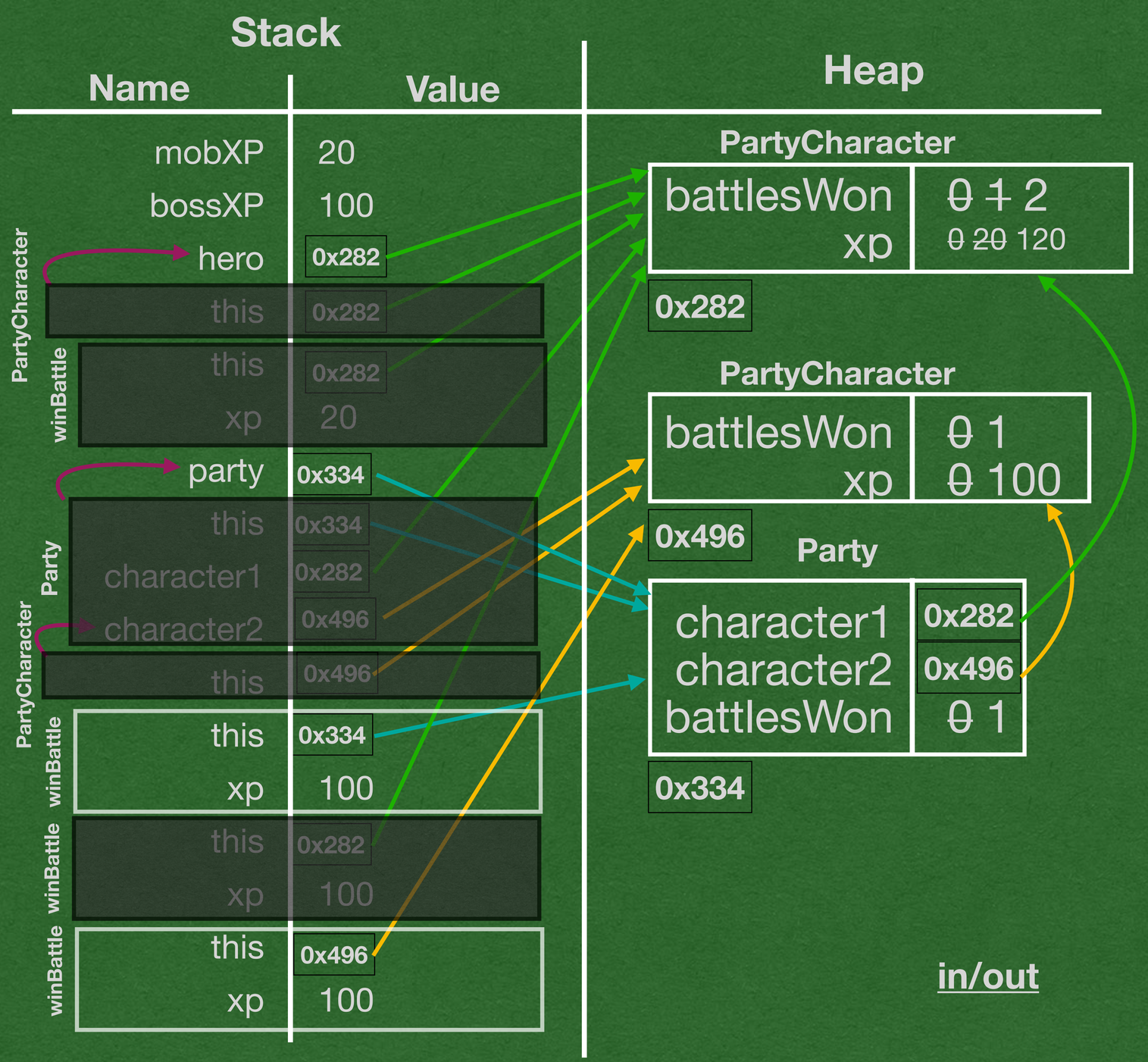
- Update values



```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.character2.xp)
}
```



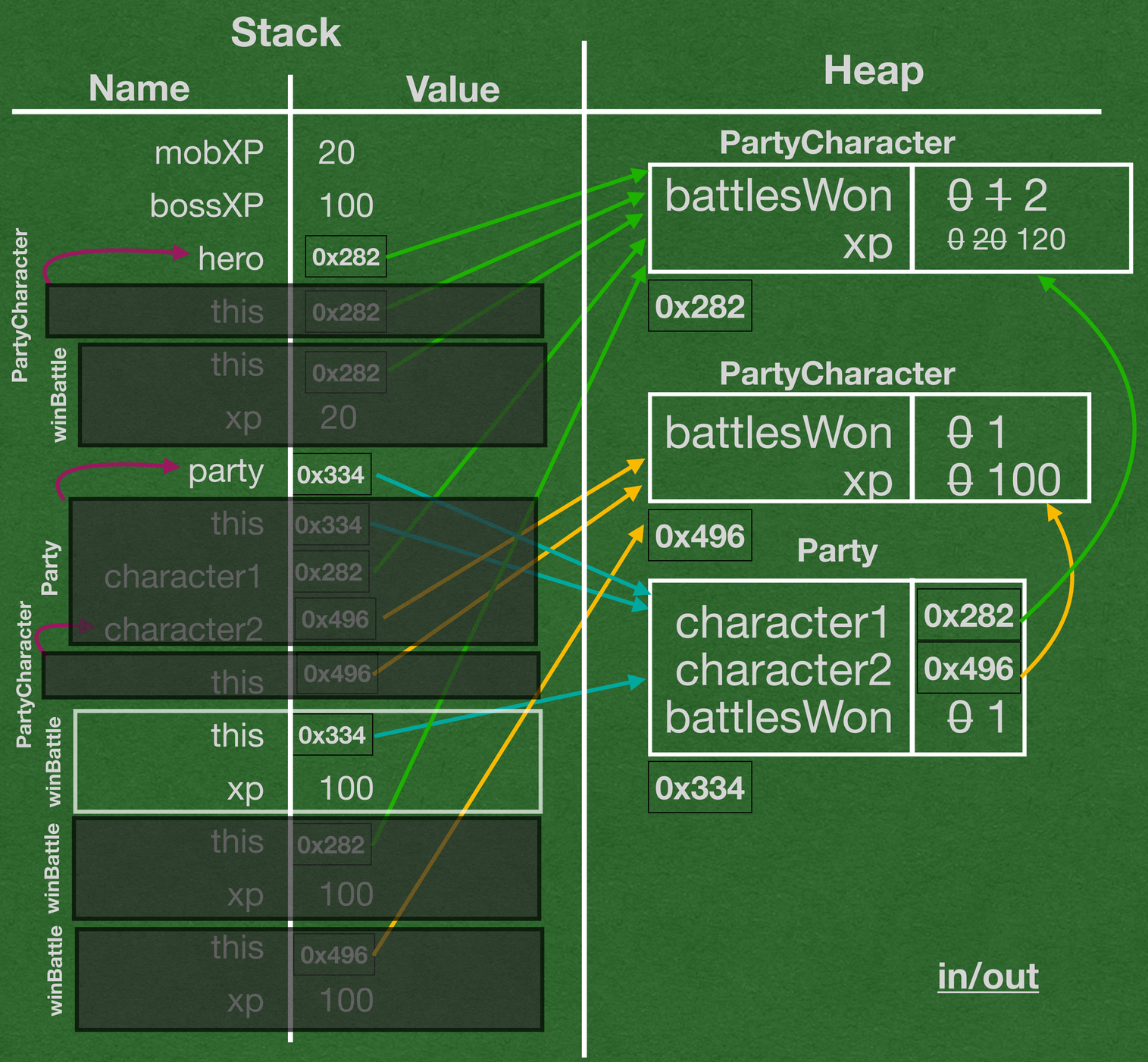
- Stack frame ends
- Repeat the process for character2

```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.character2.xp)
}
```

- Top stack frame ends

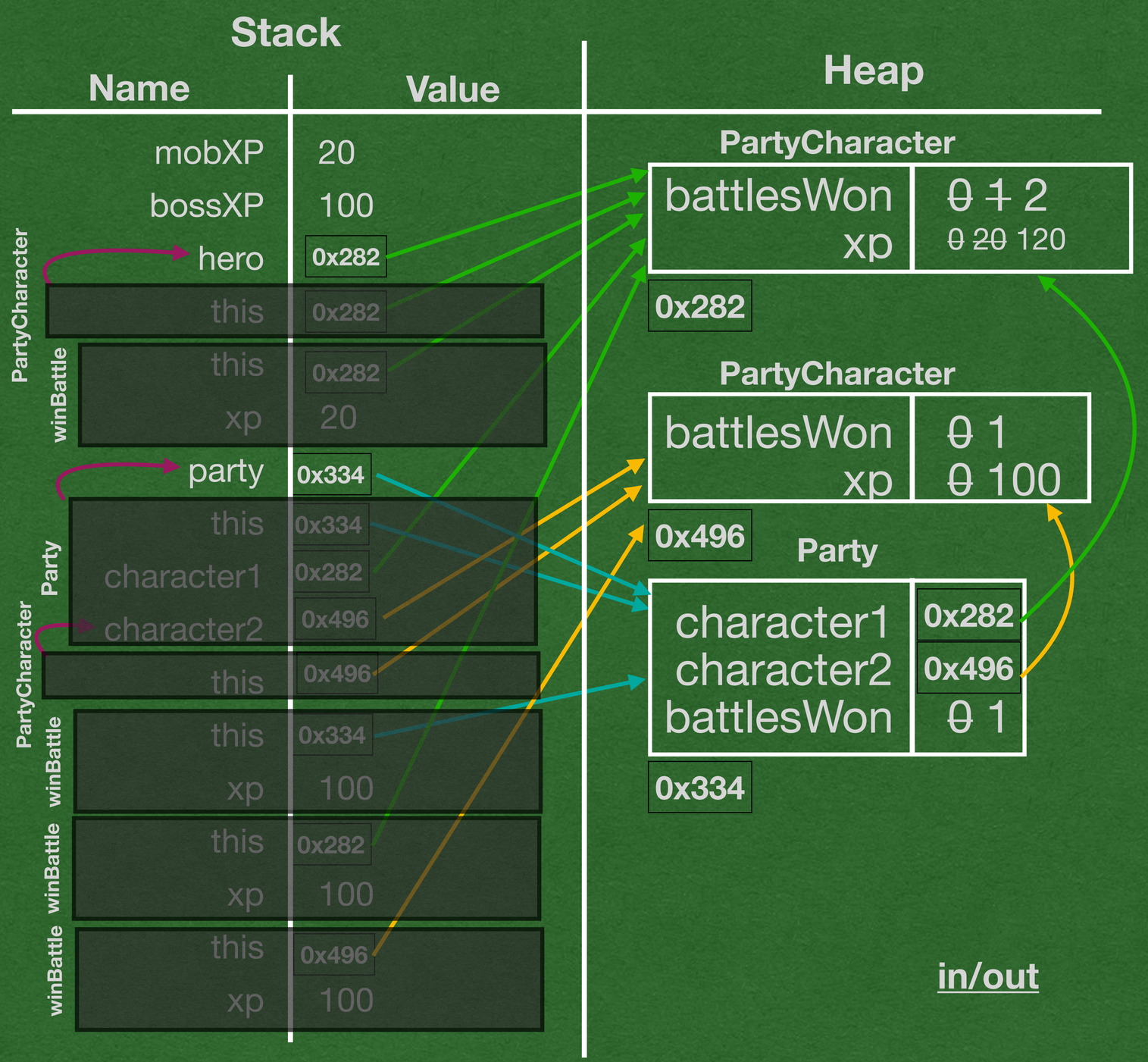


```scala
class PartyCharacter() {
  var battlesWon: Int = 0
  var xp: Int = 0

  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.character2.xp)
}
```

- Party stack frame ends



```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.character2.xp)
}
```

- Print values to the screen
- end the program