Polymorphism

Polymorphism

If an object **is a** *type*

It can be stored in variables of that *type*

```
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}
```

```
class DodgeBall(val size: Double) extends GameObject(0.0) {
  override def weight(): Double = {size * 5.0}
}
```

```
class HealthPotion(val volume: Int) extends GameObject(3.0) {
  val massPerVolume: Double = 7.0
  override def weight(): Double = {
    val bottleWeight: Double = super.weight()
    bottleWeight + this.volume * this.massPerVolume
  }
}
```

```
class Player() {
  var inventory: List[GameObject] = List()
  def pickUp(obj: GameObject): Unit = {
    this.inventory = obj :: this.inventory
  }
  def totalWeight(): Double = {
    var total: Double = 0.0
    for(obj <- this.inventory){
      total += obj.weight()
    }
    total
  }
}
```

```
def main(args: Array[String]): Unit = {
  val ball: DodgeBall = new DodgeBall(4.0)
  val potion: HealthPotion = new HealthPotion(6)
  val player: Player = new Player()
  player.pickUp(ball)
  player.pickUp(potion)
  val totalWeight: Double = player.totalWeight()
  println(totalWeight)
}
```

```scala
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}
```

```scala
class DodgeBall(val size: Double) extends GameObject(0.0) {
  override def weight(): Double = {size * 5.0}
}
```

```scala
class HealthPotion(val volume: Int) extends GameObject(3.0) {
  val massPerVolume: Double = 7.0
  override def weight(): Double = {
    val bottleWeight: Double = super.weight()
    bottleWeight + this.volume * this.massPerVolume
  }
}
```

```scala
class Player() {
  var inventory: List[GameObject] = List()
  def pickUp(obj: GameObject): Unit = {
    this.inventory = obj :: this.inventory
  }
  def totalWeight(): Double = {
    var total: Double = 0.0
    for(obj <- this.inventory){
      total += obj.weight()
    }
    total
  }
}
```

```scala
def main(args: Array[String]): Unit = {
  val ball: DodgeBall = new DodgeBall(4.0)
  val potion: HealthPotion = new HealthPotion(6)
  val player: Player = new Player()
  player.pickUp(ball)
  player.pickUp(potion)
  val totalWeight: Double = player.totalWeight()
  println(totalWeight)
}
```

Stack		Heap
Name	Value	

in/out

- How will the 3 objects be created?

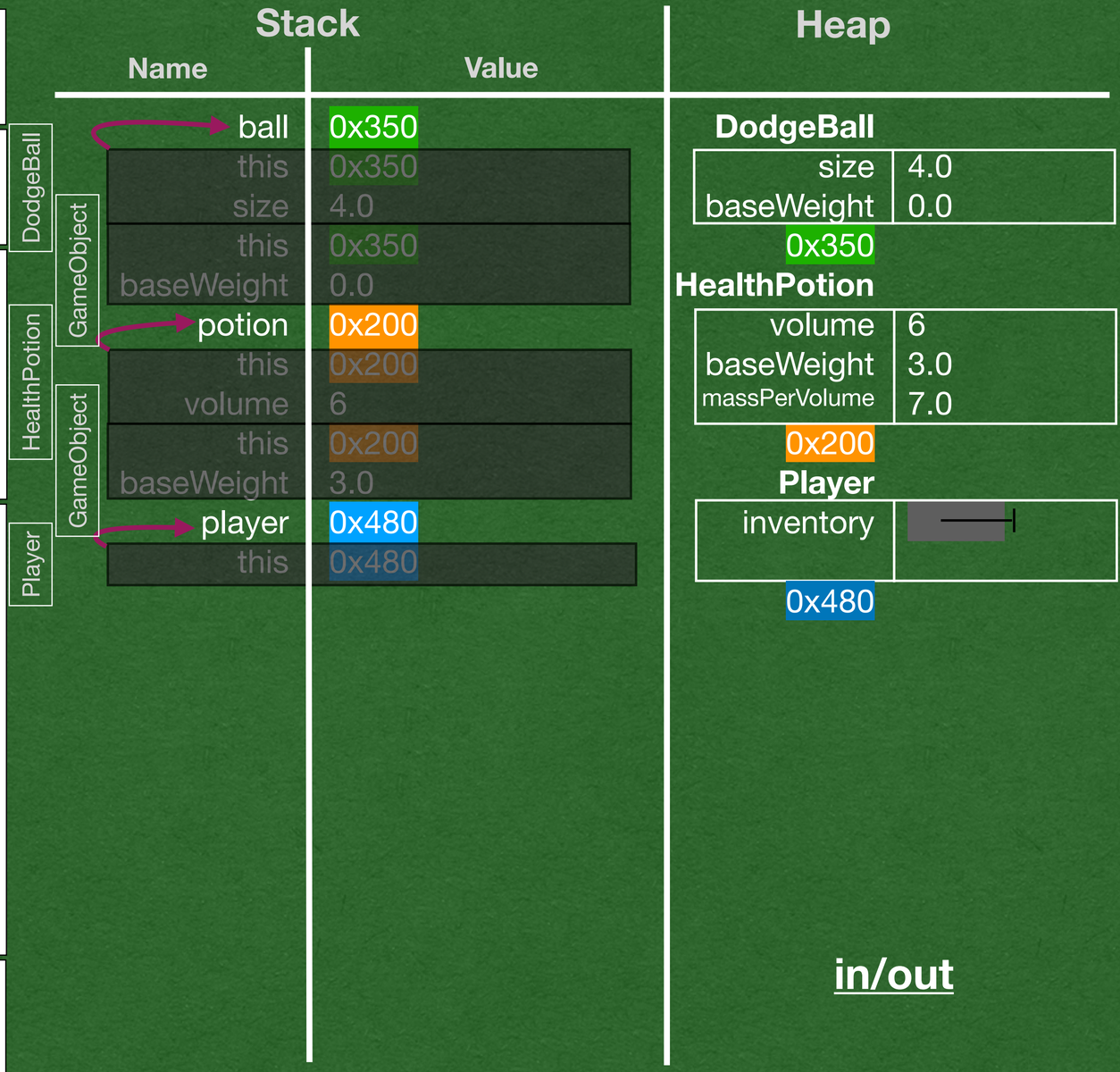
```scala
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}
```

```scala
class DodgeBall(val size: Double) extends GameObject(0.0) {
  override def weight(): Double = {size * 5.0}
}
```

```scala
class HealthPotion(val volume: Int) extends GameObject(3.0) {
  val massPerVolume: Double = 7.0
  override def weight(): Double = {
    val bottleWeight: Double = super.weight()
    bottleWeight + this.volume * this.massPerVolume
  }
}
```

```scala
class Player() {
  var inventory: List[GameObject] = List()
  def pickUp(obj: GameObject): Unit = {
    this.inventory = obj :: this.inventory
  }
  def totalWeight(): Double = {
    var total: Double = 0.0
    for(obj <- this.inventory){
      total += obj.weight()
    }
    total
  }
}
```

```scala
def main(args: Array[String]): Unit = {
  val ball: DodgeBall = new DodgeBall(4.0)
  val potion: HealthPotion = new HealthPotion(6)
  val player: Player = new Player()
  player.pickUp(ball)
  player.pickUp(potion)
  val totalWeight: Double = player.totalWeight()
  println(totalWeight)
}
```



- If you said this
  - You're right!

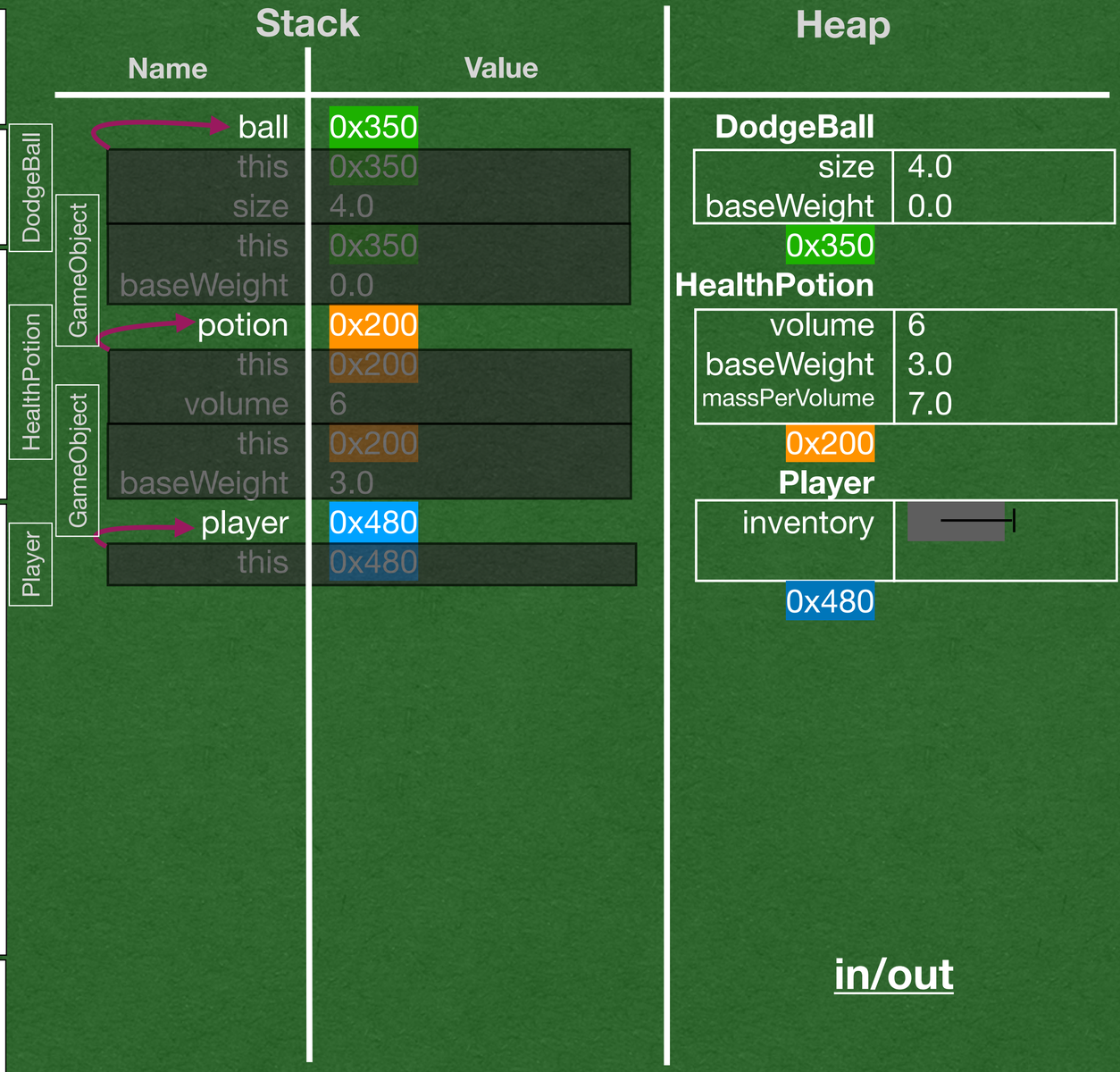
```scala
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}
```

```scala
class DodgeBall(val size: Double) extends GameObject(0.0) {
  override def weight(): Double = {size * 5.0}
}
```

```scala
class HealthPotion(val volume: Int) extends GameObject(3.0) {
  val massPerVolume: Double = 7.0
  override def weight(): Double = {
    val bottleWeight: Double = super.weight()
    bottleWeight + this.volume * this.massPerVolume
  }
}
```

```scala
class Player() {
  var inventory: List[GameObject] = List()
  def pickUp(obj: GameObject): Unit = {
    this.inventory = obj :: this.inventory
  }
  def totalWeight(): Double = {
    var total: Double = 0.0
    for(obj <- this.inventory){
      total += obj.weight()
    }
    total
  }
}
```

```scala
def main(args: Array[String]): Unit = {
  val ball: DodgeBall = new DodgeBall(4.0)
  val potion: HealthPotion = new HealthPotion(6)
  val player: Player = new Player()
  player.pickUp(ball)
  player.pickUp(potion)
  val totalWeight: Double = player.totalWeight()
  println(totalWeight)
}
```



in/out

- Linked-Lists:
  - inventory is empty; shown as referring to nothing

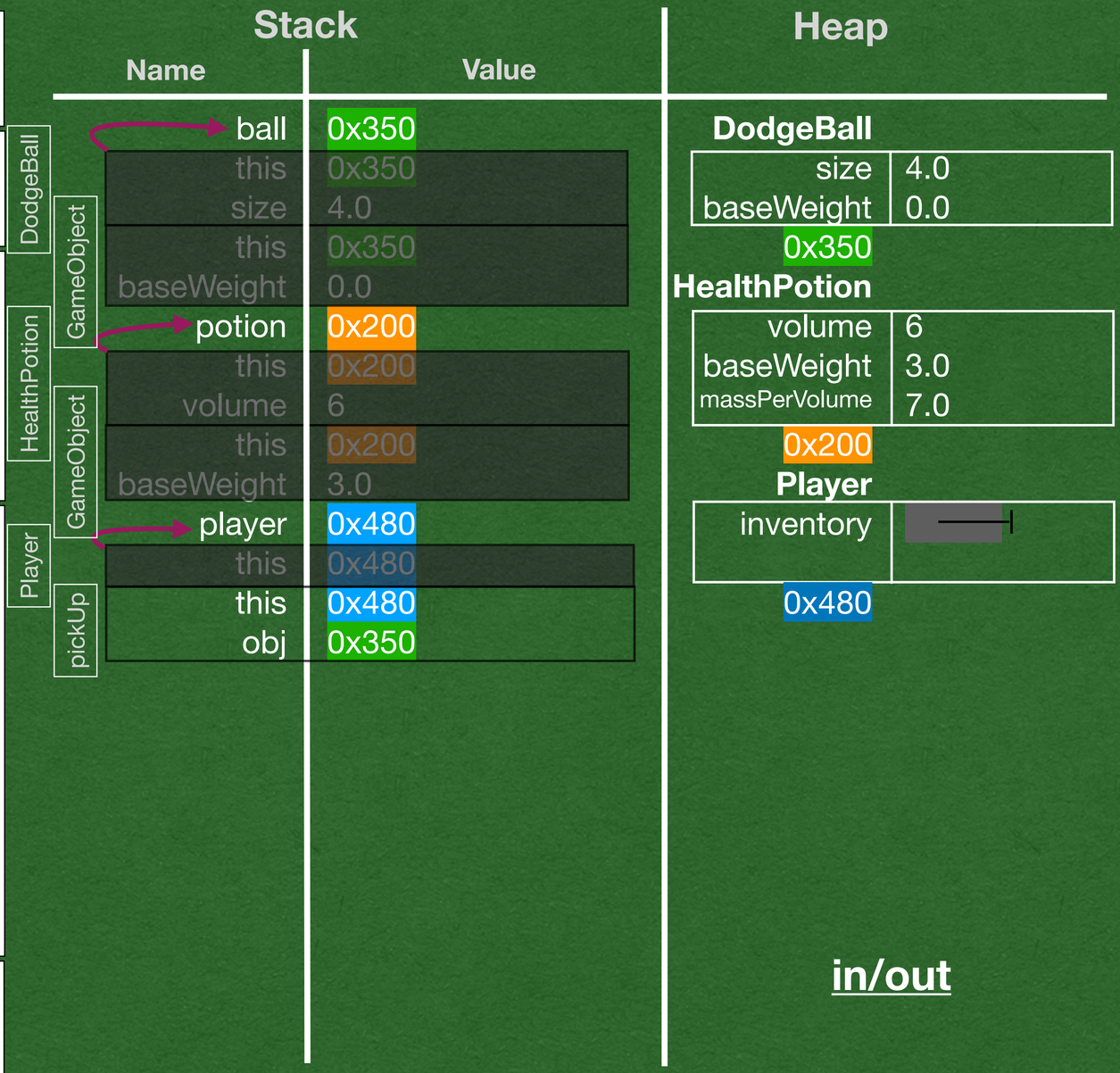
```scala
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}
```

```scala
class DodgeBall(val size: Double) extends GameObject(0.0) {
  override def weight(): Double = {size * 5.0}
}
```

```scala
class HealthPotion(val volume: Int) extends GameObject(3.0) {
  val massPerVolume: Double = 7.0
  override def weight(): Double = {
    val bottleWeight: Double = super.weight()
    bottleWeight + this.volume * this.massPerVolume
  }
}
```

```scala
class Player() {
  var inventory: List[GameObject] = List()
  def pickUp(obj: GameObject): Unit = {
    this.inventory = obj :: this.inventory
  }
  def totalWeight(): Double = {
    var total: Double = 0.0
    for(obj <- this.inventory){
      total += obj.weight()
    }
    total
  }
}
```

```scala
def main(args: Array[String]): Unit = {
  val ball: DodgeBall = new DodgeBall(4.0)
  val potion: HealthPotion = new HealthPotion(6)
  val player: Player = new Player()
  player.pickUp(ball)
  player.pickUp(potion)
  val totalWeight: Double = player.totalWeight()
  println(totalWeight)
}
```



in/out

- The pickUp method takes a GameObject
- DodgeBall is a GameObject; Polymorphism let's us pick up DodgeBalls

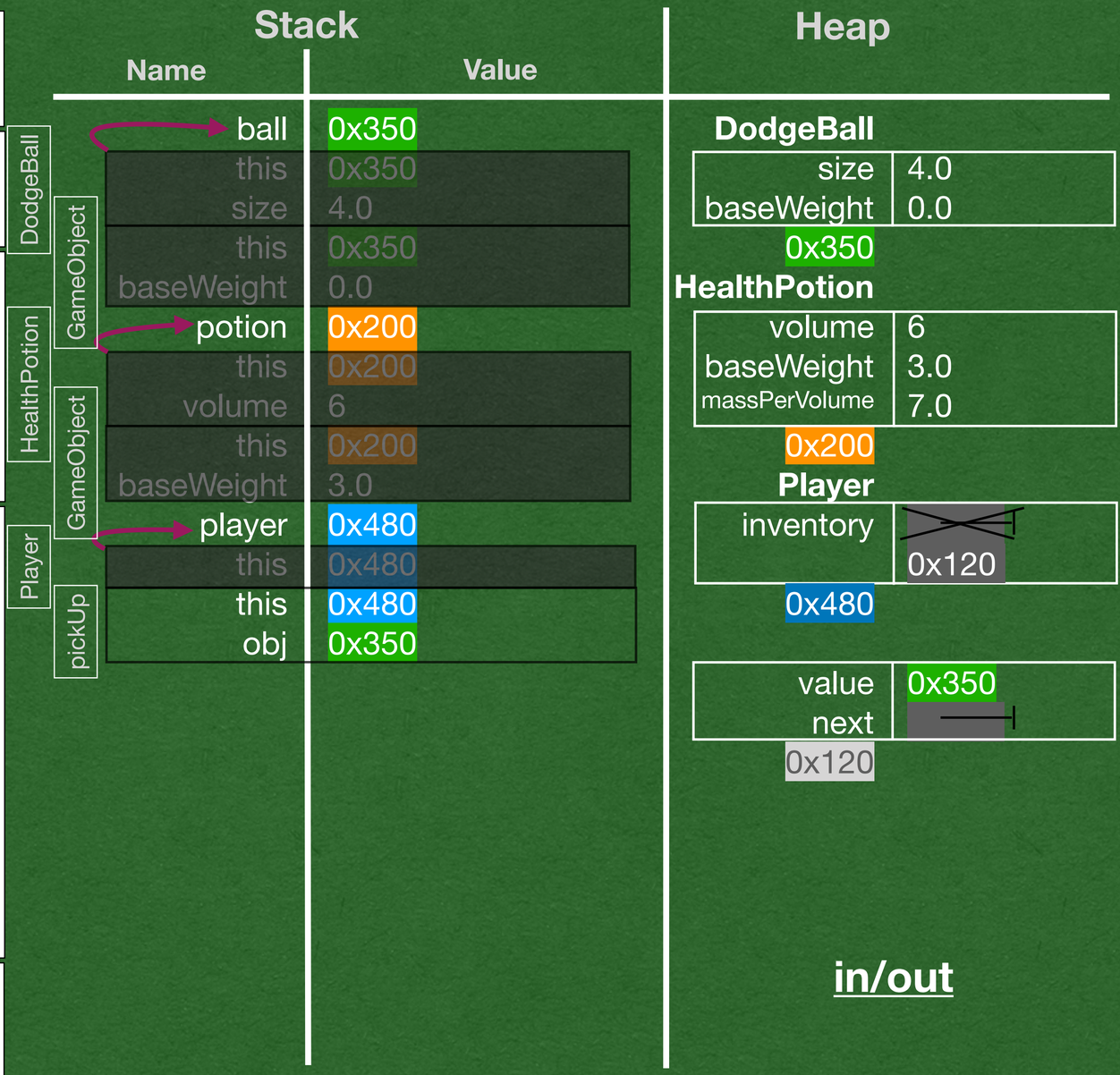
```scala
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}
```

```scala
class DodgeBall(val size: Double) extends GameObject(0.0) {
  override def weight(): Double = {size * 5.0}
}
```

```scala
class HealthPotion(val volume: Int) extends GameObject(3.0) {
  val massPerVolume: Double = 7.0
  override def weight(): Double = {
    val bottleWeight: Double = super.weight()
    bottleWeight + this.volume * this.massPerVolume
  }
}
```

```scala
class Player() {
  var inventory: List[GameObject] = List()
  def pickUp(obj: GameObject): Unit = {
    this.inventory = obj :: this.inventory
  }
  def totalWeight(): Double = {
    var total: Double = 0.0
    for(obj <- this.inventory){
      total += obj.weight()
    }
    total
  }
}
```

```scala
def main(args: Array[String]): Unit = {
  val ball: DodgeBall = new DodgeBall(4.0)
  val potion: HealthPotion = new HealthPotion(6)
  val player: Player = new Player()
  player.pickUp(ball)
  player.pickUp(potion)
  val totalWeight: Double = player.totalWeight()
  println(totalWeight)
}
```



- Create a new linked-list object with the added value
- Reassign inventory to refer to the new object

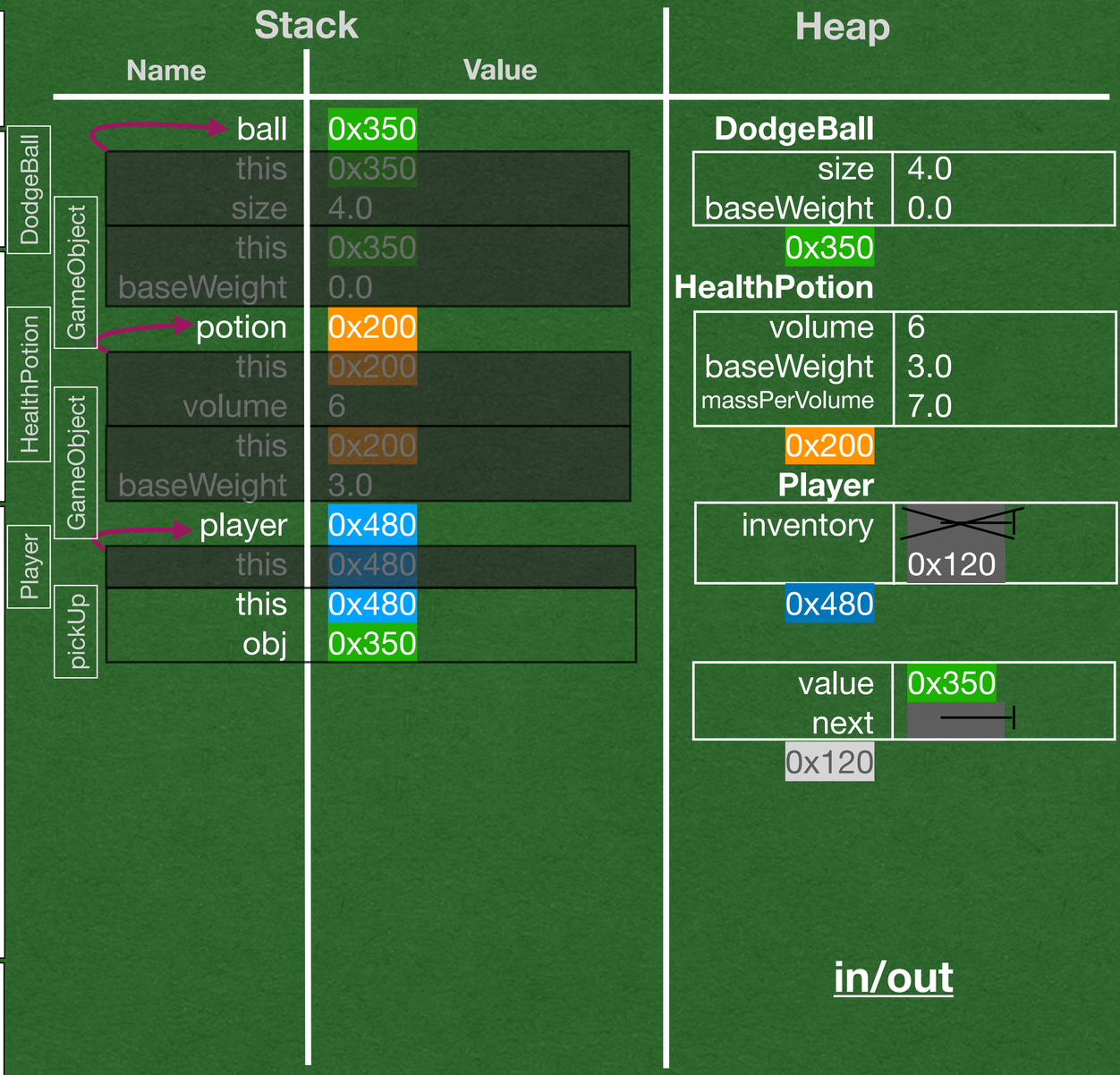
```scala
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}
```

```scala
class DodgeBall(val size: Double) extends GameObject(0.0) {
  override def weight(): Double = {size * 5.0}
}
```

```scala
class HealthPotion(val volume: Int) extends GameObject(3.0) {
  val massPerVolume: Double = 7.0
  override def weight(): Double = {
    val bottleWeight: Double = super.weight()
    bottleWeight + this.volume * this.massPerVolume
  }
}
```

```scala
class Player() {
  var inventory: List[GameObject] = List()
  def pickUp(obj: GameObject): Unit = {
    this.inventory = obj :: this.inventory
  }
  def totalWeight(): Double = {
    var total: Double = 0.0
    for(obj <- this.inventory){
      total += obj.weight()
    }
    total
  }
}
```

```scala
def main(args: Array[String]): Unit = {
  val ball: DodgeBall = new DodgeBall(4.0)
  val potion: HealthPotion = new HealthPotion(6)
  val player: Player = new Player()
  player.pickUp(ball)
  player.pickUp(potion)
  val totalWeight: Double = player.totalWeight()
  println(totalWeight)
}
```



- Each linked-list object refers to the next "link" in the list
- The last link refers to nothing (or null)

```scala
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}
```

```scala
class DodgeBall(val size: Double) extends GameObject(0.0) {
  override def weight(): Double = {size * 5.0}
}
```

```scala
class HealthPotion(val volume: Int) extends GameObject(3.0) {
  val massPerVolume: Double = 7.0
  override def weight(): Double = {
    val bottleWeight: Double = super.weight()
    bottleWeight + this.volume * this.massPerVolume
  }
}
```

```scala
class Player() {
  var inventory: List[GameObject] = List()
  def pickUp(obj: GameObject): Unit = {
    this.inventory = obj :: this.inventory
  }
  def totalWeight(): Double = {
    var total: Double = 0.0
    for(obj <- this.inventory){
      total += obj.weight()
    }
    total
  }
}
```

```scala
def main(args: Array[String]): Unit = {
  val ball: DodgeBall = new DodgeBall(4.0)
  val potion: HealthPotion = new HealthPotion(6)
  val player: Player = new Player()
  player.pickUp(ball)
  player.pickUp(potion)
  val totalWeight: Double = player.totalWeight()
  println(totalWeight)
}
```

- Repeat to prepend the HealthPotion to the list

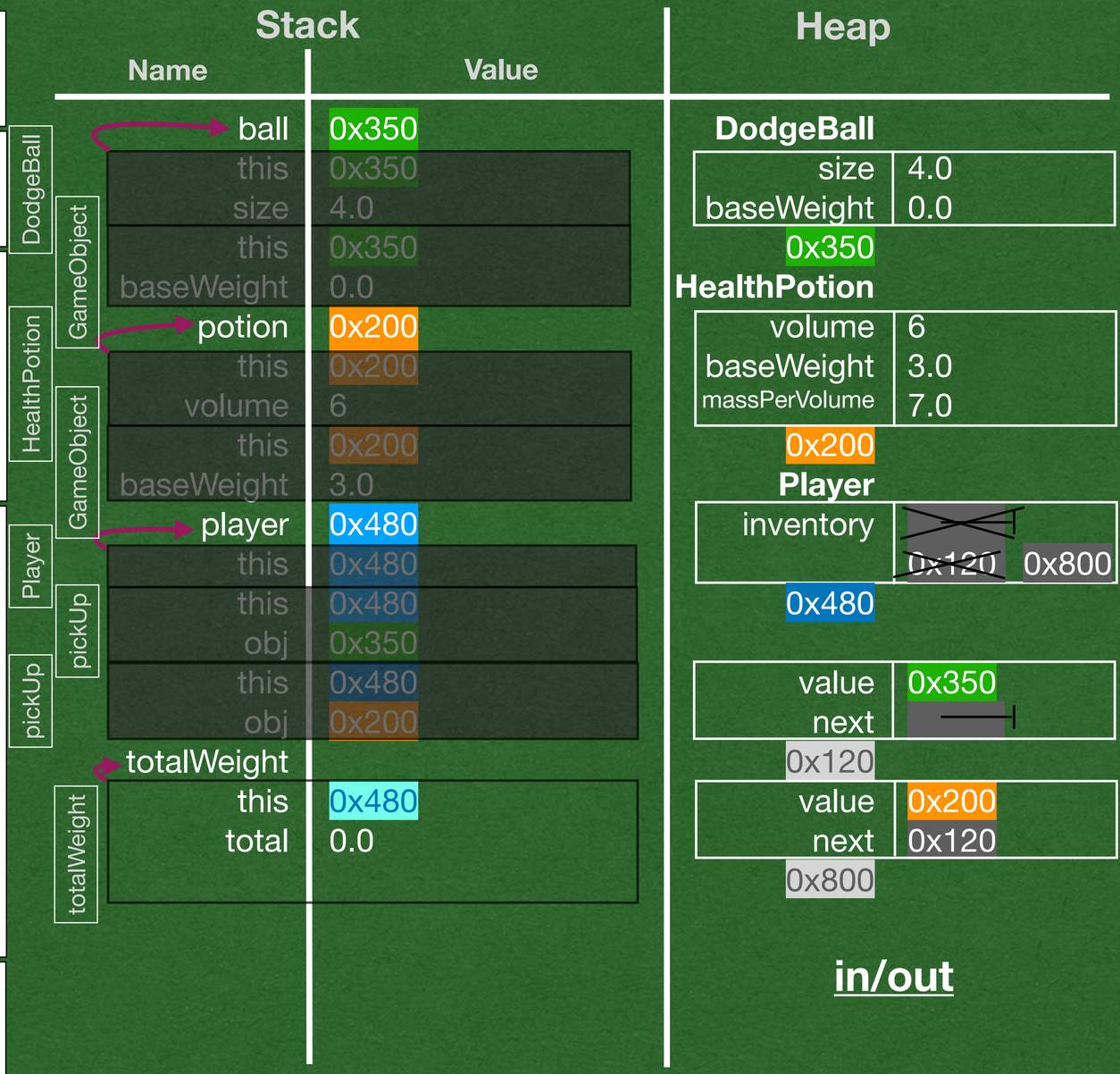
```scala
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}
```

```scala
class DodgeBall(val size: Double) extends GameObject(0.0) {
  override def weight(): Double = {size * 5.0}
}
```

```scala
class HealthPotion(val volume: Int) extends GameObject(3.0) {
  val massPerVolume: Double = 7.0
  override def weight(): Double = {
    val bottleWeight: Double = super.weight()
    bottleWeight + this.volume * this.massPerVolume
  }
}
```

```scala
class Player() {
  var inventory: List[GameObject] = List()
  def pickUp(obj: GameObject): Unit = {
    this.inventory = obj :: this.inventory
  }
  def totalWeight(): Double = {
    var total: Double = 0.0
    for(obj <- this.inventory){
      total += obj.weight()
    }
    total
  }
}
```

```scala
def main(args: Array[String]): Unit = {
  val ball: DodgeBall = new DodgeBall(4.0)
  val potion: HealthPotion = new HealthPotion(6)
  val player: Player = new Player()
  player.pickUp(ball)
  player.pickUp(potion)
  val totalWeight: Double = player.totalWeight()
  println(totalWeight)
}
```



- Add a stack frame for totalWeight

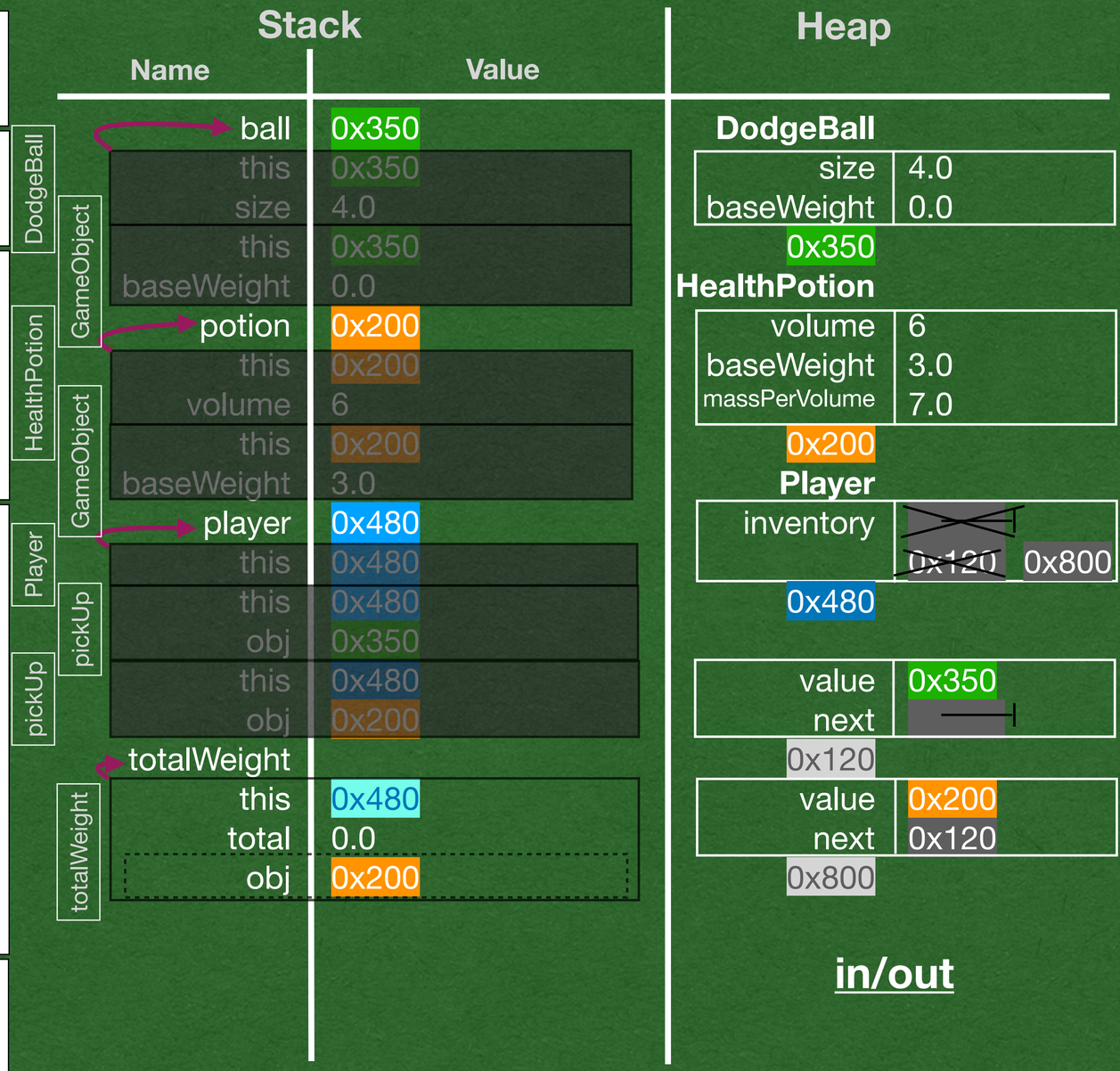
```scala
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}
```

```scala
class DodgeBall(val size: Double) extends GameObject(0.0) {
  override def weight(): Double = {size * 5.0}
}
```

```scala
class HealthPotion(val volume: Int) extends GameObject(3.0) {
  val massPerVolume: Double = 7.0
  override def weight(): Double = {
    val bottleWeight: Double = super.weight()
    bottleWeight + this.volume * this.massPerVolume
  }
}
```

```scala
class Player() {
  var inventory: List[GameObject] = List()
  def pickUp(obj: GameObject): Unit = {
    this.inventory = obj :: this.inventory
  }
  def totalWeight(): Double = {
    var total: Double = 0.0
    for(obj <- this.inventory){
      total += obj.weight()
    }
    total
  }
}
```

```scala
def main(args: Array[String]): Unit = {
  val ball: DodgeBall = new DodgeBall(4.0)
  val potion: HealthPotion = new HealthPotion(6)
  val player: Player = new Player()
  player.pickUp(ball)
  player.pickUp(potion)
  val totalWeight: Double = player.totalWeight()
  println(totalWeight)
}
```



- Enter to loop block
- Follow the references to iterate through the List

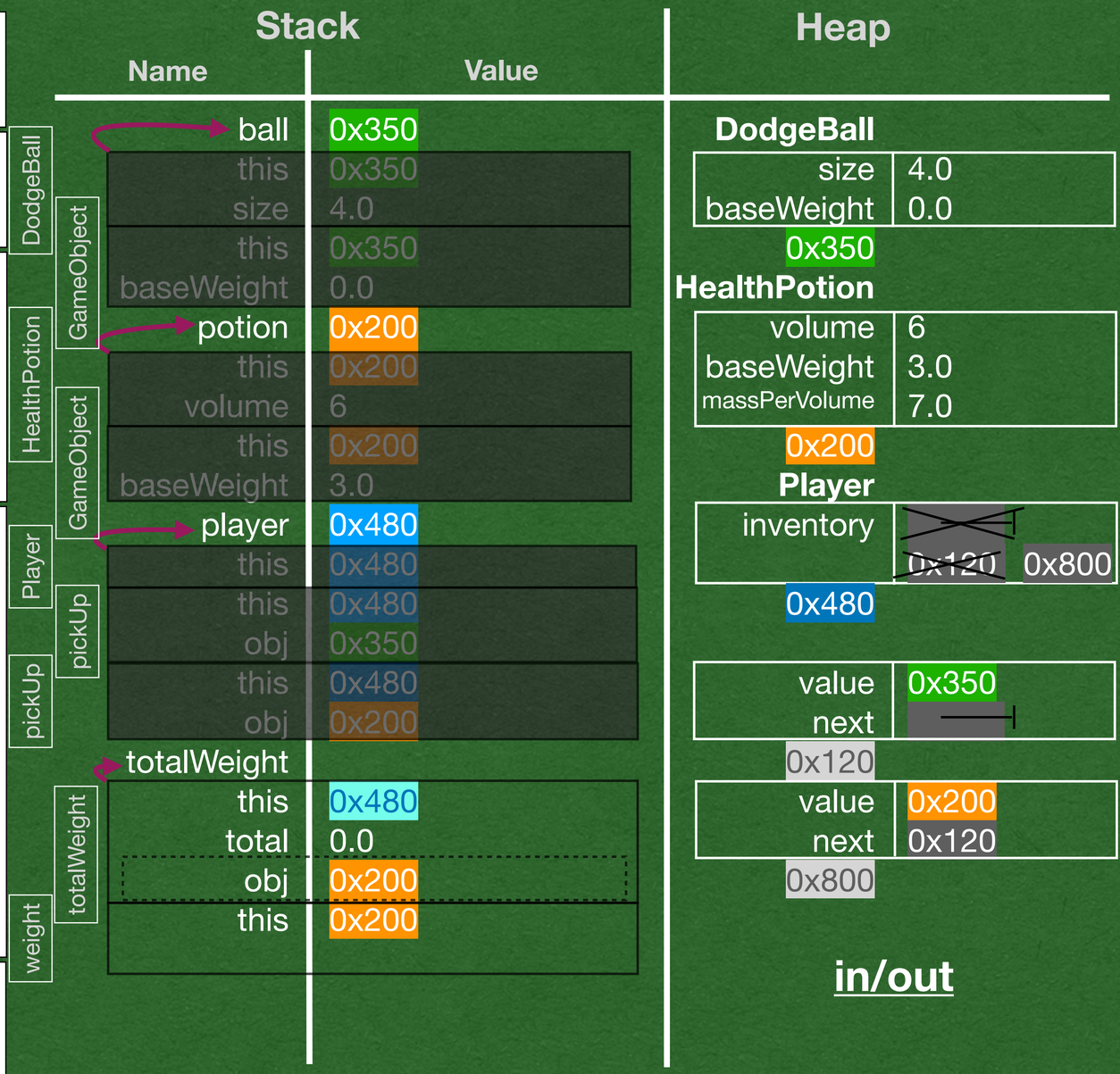
```scala
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}
```

```scala
class DodgeBall(val size: Double) extends GameObject(0.0) {
  override def weight(): Double = {size * 5.0}
}
```

```scala
class HealthPotion(val volume: Int) extends GameObject(3.0) {
  val massPerVolume: Double = 7.0
  override def weight(): Double = {
    val bottleWeight: Double = super.weight()
    bottleWeight + this.volume * this.massPerVolume
  }
}
```

```scala
class Player() {
  var inventory: List[GameObject] = List()
  def pickUp(obj: GameObject): Unit = {
    this.inventory = obj :: this.inventory
  }
  def totalWeight(): Double = {
    var total: Double = 0.0
    for(obj <- this.inventory){
      total += obj.weight()
    }
    total
  }
}
```

```scala
def main(args: Array[String]): Unit = {
  val ball: DodgeBall = new DodgeBall(4.0)
  val potion: HealthPotion = new HealthPotion(6)
  val player: Player = new Player()
  player.pickUp(ball)
  player.pickUp(potion)
  val totalWeight: Double = player.totalWeight()
  println(totalWeight)
}
```



- Call obj.weight() on a reference to a HealthPotion
- Use the HeathPotion weight() method

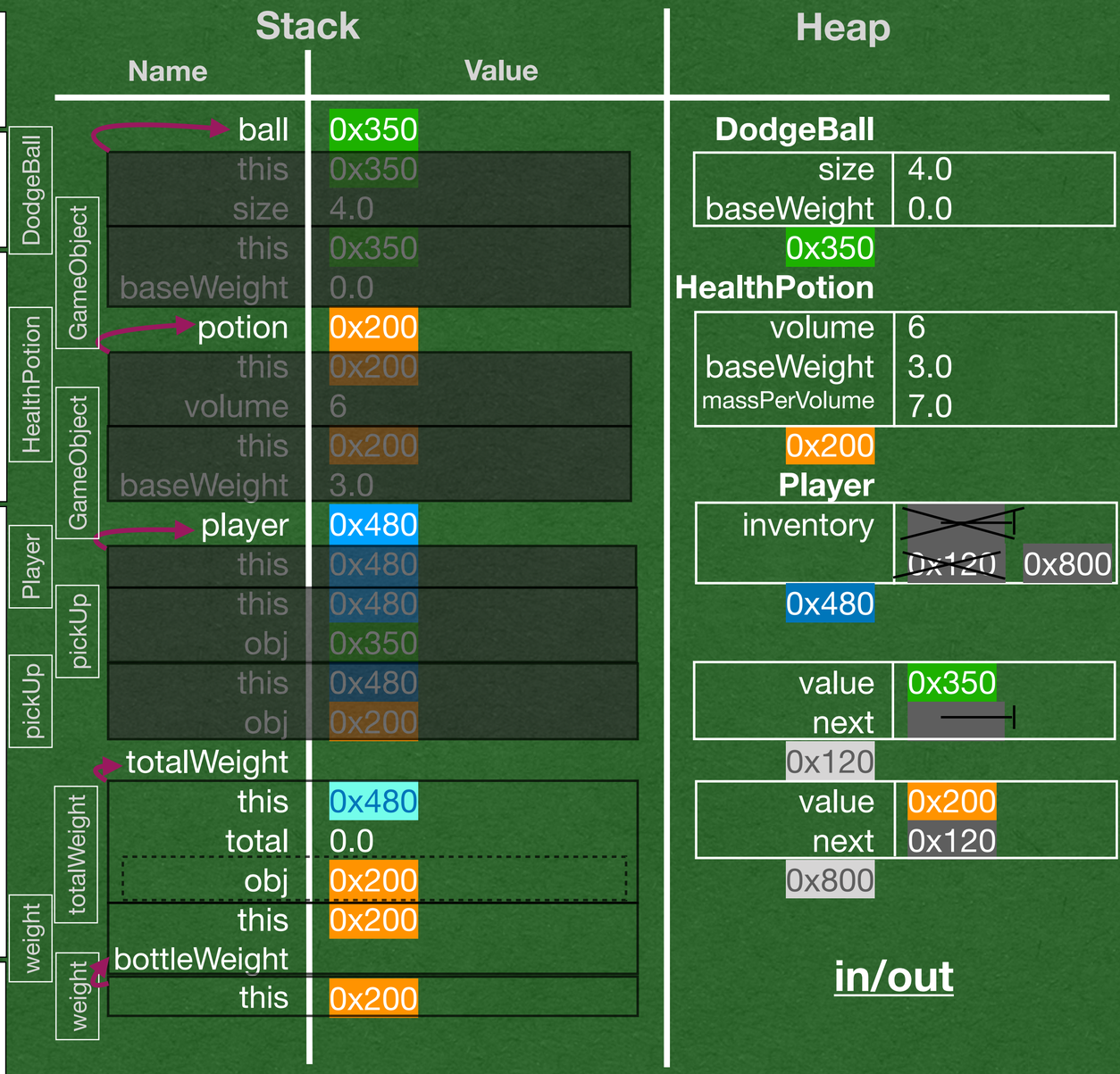
```scala
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}
```

```scala
class DodgeBall(val size: Double) extends GameObject(0.0) {
  override def weight(): Double = {size * 5.0}
}
```

```scala
class HealthPotion(val volume: Int) extends GameObject(3.0) {
  val massPerVolume: Double = 7.0
  override def weight(): Double = {
    val bottleWeight: Double = super.weight()
    bottleWeight + this.volume * this.massPerVolume
  }
}
```

```scala
class Player() {
  var inventory: List[GameObject] = List()
  def pickUp(obj: GameObject): Unit = {
    this.inventory = obj :: this.inventory
  }
  def totalWeight(): Double = {
    var total: Double = 0.0
    for(obj <- this.inventory){
      total += obj.weight()
    }
    total
  }
}
```

```scala
def main(args: Array[String]): Unit = {
  val ball: DodgeBall = new DodgeBall(4.0)
  val potion: HealthPotion = new HealthPotion(6)
  val player: Player = new Player()
  player.pickUp(ball)
  player.pickUp(potion)
  val totalWeight: Double = player.totalWeight()
  println(totalWeight)
}
```



- Use "super" to access behavior that has been overridden
- super.weight calls the overridden GameObject method

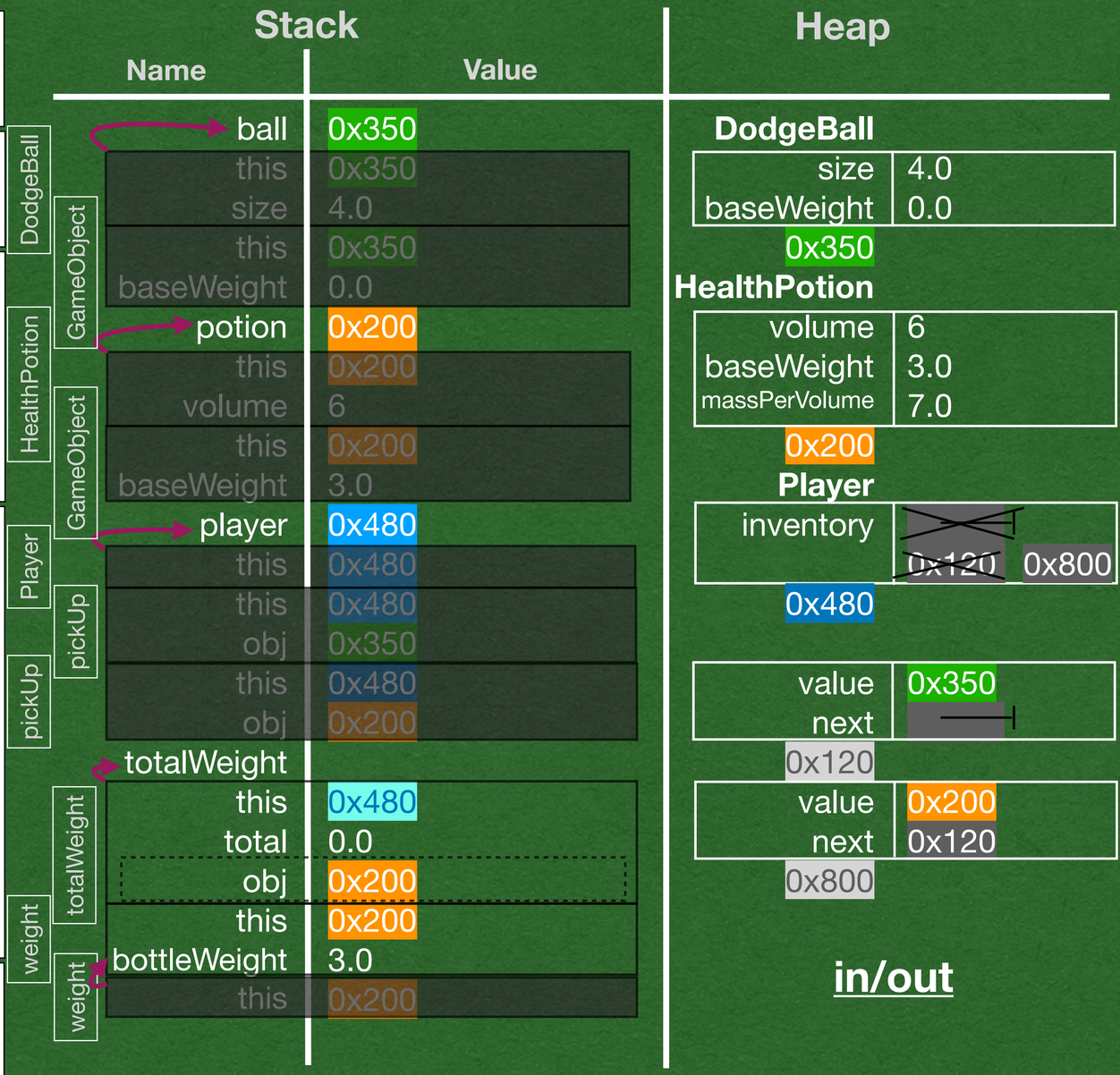
```scala
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}
```

```scala
class DodgeBall(val size: Double) extends GameObject(0.0) {
  override def weight(): Double = {size * 5.0}
}
```

```scala
class HealthPotion(val volume: Int) extends GameObject(3.0) {
  val massPerVolume: Double = 7.0
  override def weight(): Double = {
    val bottleWeight: Double = super.weight()
    bottleWeight + this.volume * this.massPerVolume
  }
}
```

```scala
class Player() {
  var inventory: List[GameObject] = List()
  def pickUp(obj: GameObject): Unit = {
    this.inventory = obj :: this.inventory
  }
  def totalWeight(): Double = {
    var total: Double = 0.0
    for(obj <- this.inventory){
      total += obj.weight()
    }
    total
  }
}
```

```scala
def main(args: Array[String]): Unit = {
  val ball: DodgeBall = new DodgeBall(4.0)
  val potion: HealthPotion = new HealthPotion(6)
  val player: Player = new Player()
  player.pickUp(ball)
  player.pickUp(potion)
  val totalWeight: Double = player.totalWeight()
  println(totalWeight)
}
```



- The super class method returns
- Continue with the method

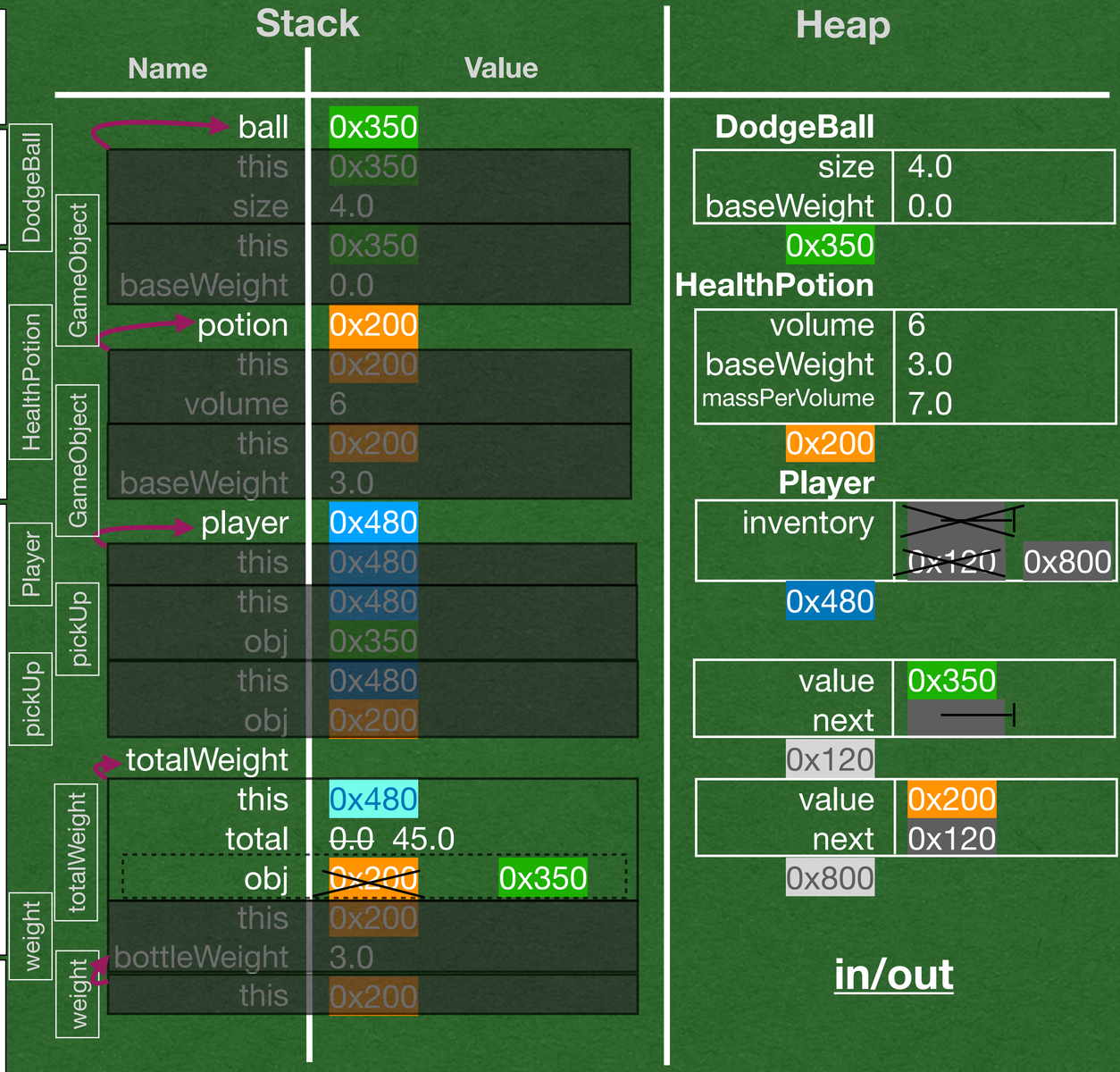
```scala
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}
```

```scala
class DodgeBall(val size: Double) extends GameObject(0.0) {
  override def weight(): Double = {size * 5.0}
}
```

```scala
class HealthPotion(val volume: Int) extends GameObject(3.0) {
  val massPerVolume: Double = 7.0
  override def weight(): Double = {
    val bottleWeight: Double = super.weight()
    bottleWeight + this.volume * this.massPerVolume
  }
}
```

```scala
class Player() {
  var inventory: List[GameObject] = List()
  def pickUp(obj: GameObject): Unit = {
    this.inventory = obj :: this.inventory
  }
  def totalWeight(): Double = {
    var total: Double = 0.0
    for(obj <- this.inventory){
      total += obj.weight()
    }
    total
  }
}
```

```scala
def main(args: Array[String]): Unit = {
  val ball: DodgeBall = new DodgeBall(4.0)
  val potion: HealthPotion = new HealthPotion(6)
  val player: Player = new Player()
  player.pickUp(ball)
  player.pickUp(potion)
  val totalWeight: Double = player.totalWeight()
  println(totalWeight)
}
```



- Add to the total
- Follow the references to the next element in the List

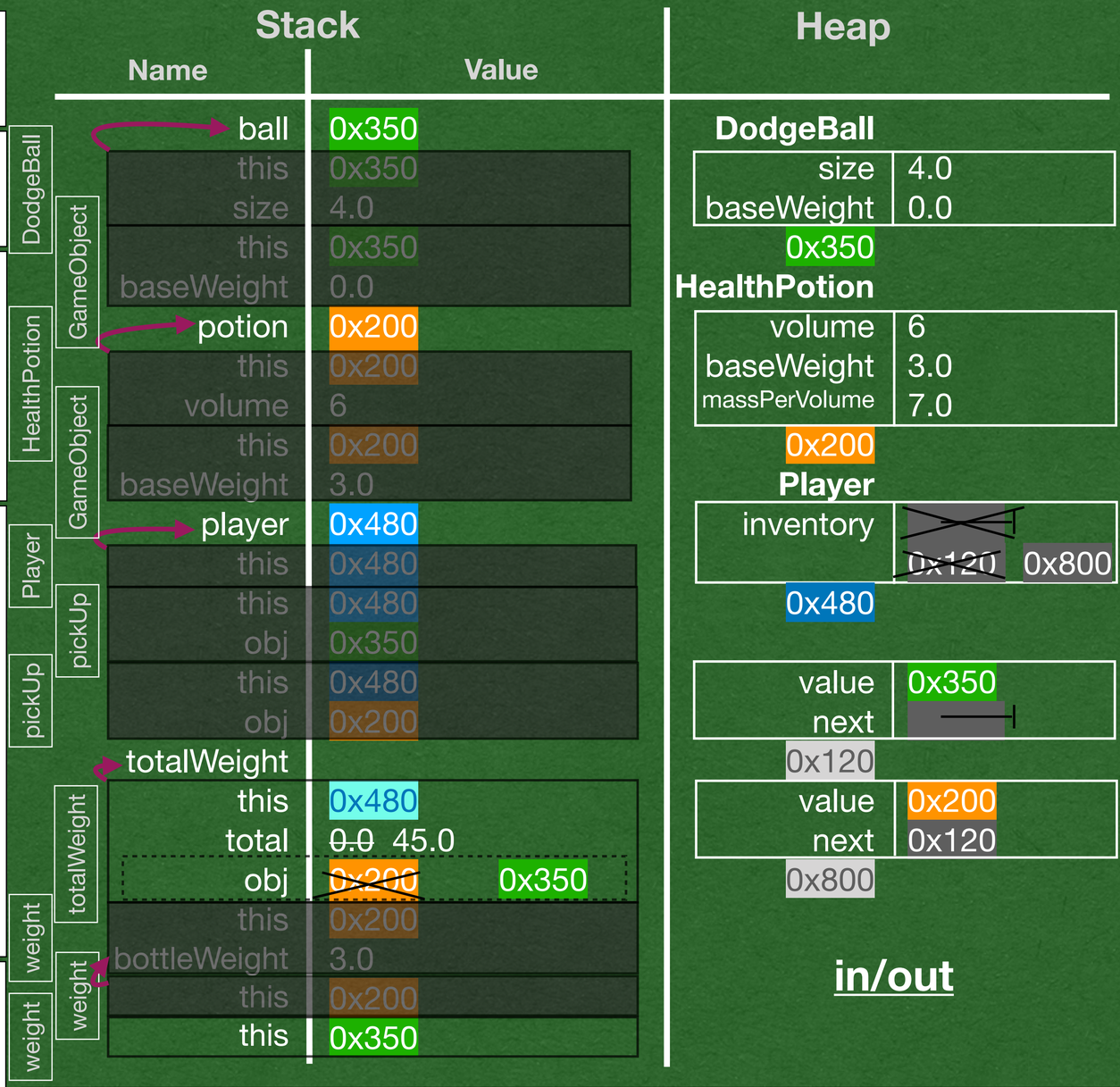
```scala
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}
```

```scala
class DodgeBall(val size: Double) extends GameObject(0.0) {
  override def weight(): Double = {size * 5.0}
}
```

```scala
class HealthPotion(val volume: Int) extends GameObject(3.0) {
  val massPerVolume: Double = 7.0
  override def weight(): Double = {
    val bottleWeight: Double = super.weight()
    bottleWeight + this.volume * this.massPerVolume
  }
}
```

```scala
class Player() {
  var inventory: List[GameObject] = List()
  def pickUp(obj: GameObject): Unit = {
    this.inventory = obj :: this.inventory
  }
  def totalWeight(): Double = {
    var total: Double = 0.0
    for(obj <- this.inventory){
      total += obj.weight()
    }
    total
  }
}
```

```scala
def main(args: Array[String]): Unit = {
  val ball: DodgeBall = new DodgeBall(4.0)
  val potion: HealthPotion = new HealthPotion(6)
  val player: Player = new Player()
  player.pickUp(ball)
  player.pickUp(potion)
  val totalWeight: Double = player.totalWeight()
  println(totalWeight)
}
```



- This time, weight is called from an object of type DodgeBall
- Use the DodgeBall weight method

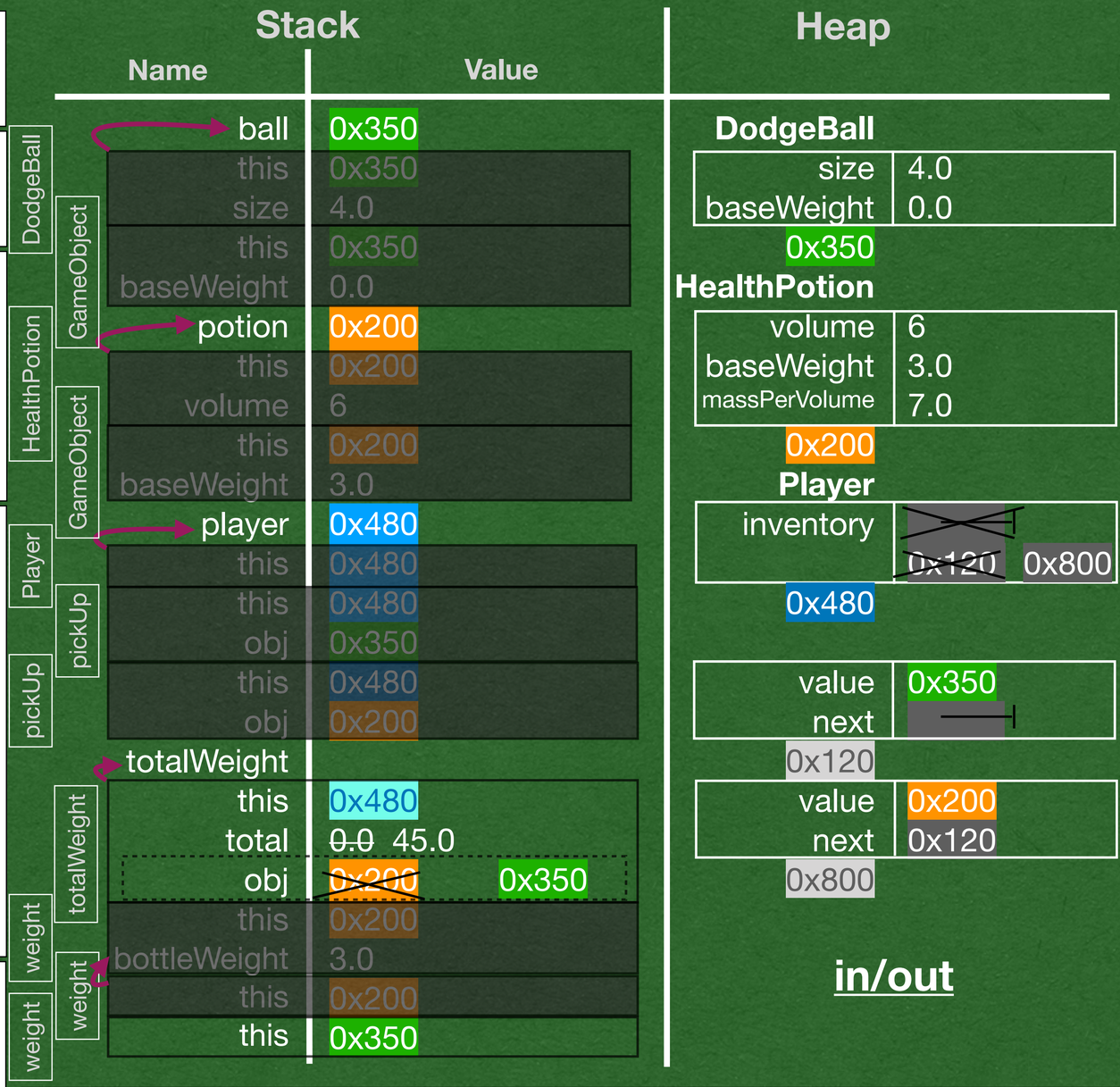
```scala
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}
```

```scala
class DodgeBall(val size: Double) extends GameObject(0.0) {
  override def weight(): Double = {size * 5.0}
}
```

```scala
class HealthPotion(val volume: Int) extends GameObject(3.0) {
  val massPerVolume: Double = 7.0
  override def weight(): Double = {
    val bottleWeight: Double = super.weight()
    bottleWeight + this.volume * this.massPerVolume
  }
}
```

```scala
class Player() {
  var inventory: List[GameObject] = List()
  def pickUp(obj: GameObject): Unit = {
    this.inventory = obj :: this.inventory
  }
  def totalWeight(): Double = {
    var total: Double = 0.0
    for(obj <- this.inventory){
      total += obj.weight()
    }
    total
  }
}
```

```scala
def main(args: Array[String]): Unit = {
  val ball: DodgeBall = new DodgeBall(4.0)
  val potion: HealthPotion = new HealthPotion(6)
  val player: Player = new Player()
  player.pickUp(ball)
  player.pickUp(potion)
  val totalWeight: Double = player.totalWeight()
  println(totalWeight)
}
```



- The method that is called depends on the type of the **object** (DodgeBall)
  - Not the type of the **variable** (GameObject)

```scala
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}
```

```scala
class DodgeBall(val size: Double) extends GameObject(0.0) {
  override def weight(): Double = {size * 5.0}
}
```

```scala
class HealthPotion(val volume: Int) extends GameObject(3.0) {
  val massPerVolume: Double = 7.0
  override def weight(): Double = {
    val bottleWeight: Double = super.weight()
    bottleWeight + this.volume * this.massPerVolume
  }
}
```

```scala
class Player() {
  var inventory: List[GameObject] = List()
  def pickUp(obj: GameObject): Unit = {
    this.inventory = obj :: this.inventory
  }
  def totalWeight(): Double = {
    var total: Double = 0.0
    for(obj <- this.inventory){
      total += obj.weight()
    }
    total
  }
}
```

```scala
def main(args: Array[String]): Unit = {
  val ball: DodgeBall = new DodgeBall(4.0)
  val potion: HealthPotion = new HealthPotion(6)
  val player: Player = new Player()
  player.pickUp(ball)
  player.pickUp(potion)
  val totalWeight: Double = player.totalWeight()
  println(totalWeight)
}
```

Stack

Name	Value
DodgeBall: ball	0x350
DodgeBall: this	0x350
DodgeBall: size	4.0
GameObject: this	0x350
GameObject: baseWeight	0.0
HealthPotion: potion	0x200
HealthPotion: this	0x200
HealthPotion: volume	6
GameObject: this	0x200
GameObject: baseWeight	3.0
Player: player	0x480
Player: this	0x480
pickUp: this	0x480
pickUp: obj	0x350
pickUp: this	0x480
pickUp: obj	0x200
totalWeight	
totalWeight: this	0x480
totalWeight: total	~~0.0~~ ~~45.0~~ 65.0
totalWeight: obj	~~0x200~~ 0x350
weight: this	0x200
weight: bottleWeight	3.0
weight: this	0x200
weight: this	0x350

Heap

DodgeBall

size	4.0
baseWeight	0.0

0x350

HealthPotion

volume	6
baseWeight	3.0
massPerVolume	7.0

0x200

Player

inventory	~~0x120~~ 0x800

0x480

value	0x350
next	

0x120

value	0x200
next	0x120

0x800

in/out

- Return 20.0 and add it to total

```scala
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}
```

```scala
class DodgeBall(val size: Double) extends GameObject(0.0) {
  override def weight(): Double = {size * 5.0}
}
```

```scala
class HealthPotion(val volume: Int) extends GameObject(3.0) {
  val massPerVolume: Double = 7.0
  override def weight(): Double = {
    val bottleWeight: Double = super.weight()
    bottleWeight + this.volume * this.massPerVolume
  }
}
```

```scala
class Player() {
  var inventory: List[GameObject] = List()
  def pickUp(obj: GameObject): Unit = {
    this.inventory = obj :: this.inventory
  }
  def totalWeight(): Double = {
    var total: Double = 0.0
    for(obj <- this.inventory){
      total += obj.weight()
    }
    total
  }
}
```

```scala
def main(args: Array[String]): Unit = {
  val ball: DodgeBall = new DodgeBall(4.0)
  val potion: HealthPotion = new HealthPotion(6)
  val player: Player = new Player()
  player.pickUp(ball)
  player.pickUp(potion)
  val totalWeight: Double = player.totalWeight()
  println(totalWeight)
}
```

Stack

Name	Value
DodgeBall: ball	0x350
DodgeBall: this	0x350
DodgeBall: size	4.0
GameObject: this	0x350
GameObject: baseWeight	0.0
HealthPotion: potion	0x200
HealthPotion: this	0x200
HealthPotion: volume	6
GameObject: this	0x200
GameObject: baseWeight	3.0
Player: player	0x480
Player: this	0x480
pickUp: this	0x480
pickUp: obj	0x350
pickUp: this	0x480
pickUp: obj	0x200
totalWeight	
totalWeight: this	0x480
totalWeight: total	~~0.0~~ ~~45.0~~ 65.0
totalWeight: obj	~~0x200~~ ~~0x350~~
weight: this	0x200
weight: bottleWeight	3.0
weight: this	0x200
weight: this	0x350

Heap

DodgeBall

size	4.0
baseWeight	0.0

0x350

HealthPotion

volume	6
baseWeight	3.0
massPerVolume	7.0

0x200

Player

inventory	~~0x120~~ 0x800

0x480

value	0x350
next	

0x120

value	0x200
next	0x120

0x800

in/out

- End of loop

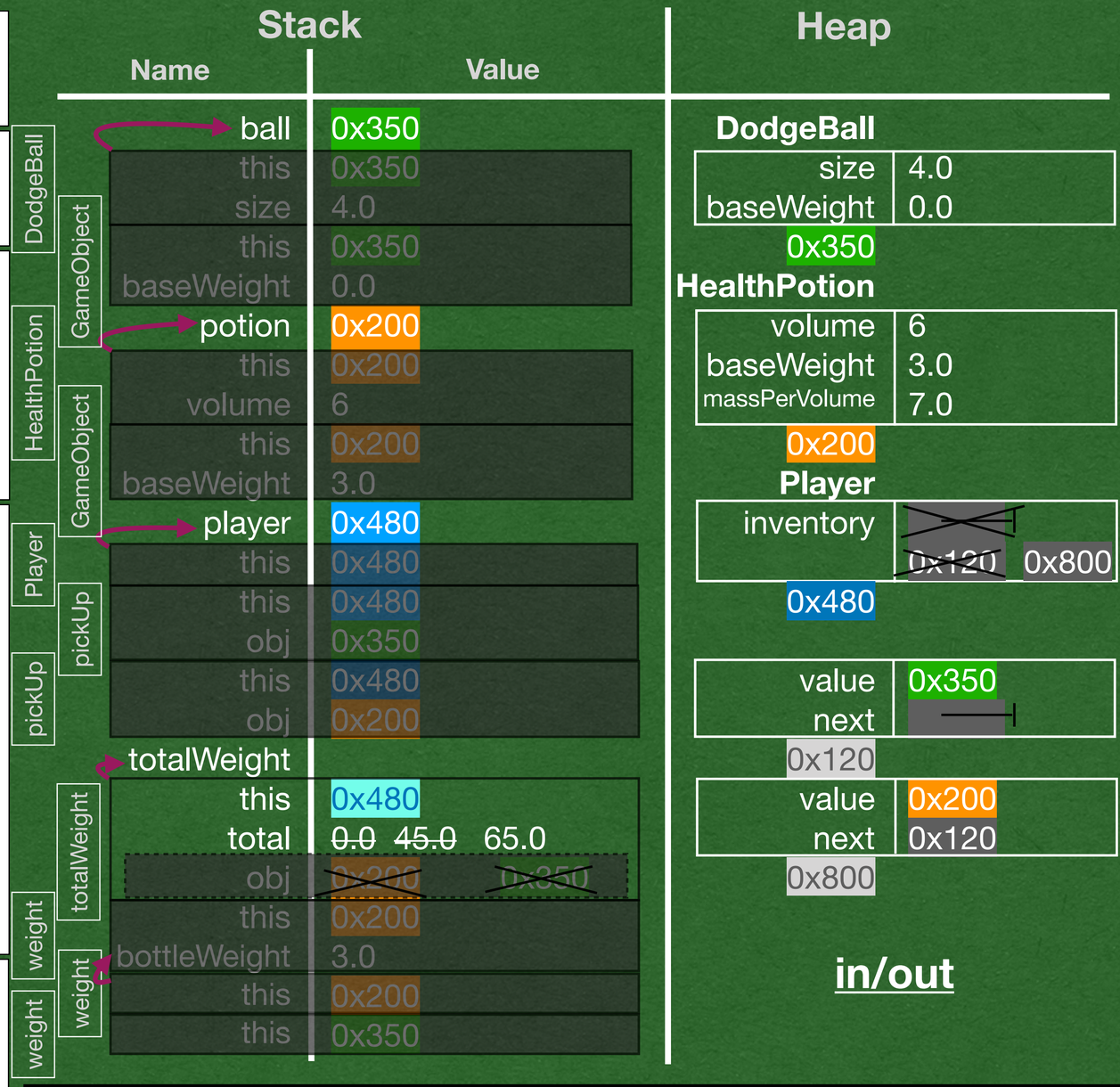
```scala
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}
```

```scala
class DodgeBall(val size: Double) extends GameObject(0.0) {
  override def weight(): Double = {size * 5.0}
}
```

```scala
class HealthPotion(val volume: Int) extends GameObject(3.0) {
  val massPerVolume: Double = 7.0
  override def weight(): Double = {
    val bottleWeight: Double = super.weight()
    bottleWeight + this.volume * this.massPerVolume
  }
}
```

```scala
class Player() {
  var inventory: List[GameObject] = List()
  def pickUp(obj: GameObject): Unit = {
    this.inventory = obj :: this.inventory
  }
  def totalWeight(): Double = {
    var total: Double = 0.0
    for(obj <- this.inventory){
      total += obj.weight()
    }
    total
  }
}
```

```scala
def main(args: Array[String]): Unit = {
  val ball: DodgeBall = new DodgeBall(4.0)
  val potion: HealthPotion = new HealthPotion(6)
  val player: Player = new Player()
  player.pickUp(ball)
  player.pickUp(potion)
  val totalWeight: Double = player.totalWeight()
  println(totalWeight)
}
```



- Last expression is "total"
- Return the value of total

```scala
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}
```

```scala
class DodgeBall(val size: Double) extends GameObject(0.0) {
  override def weight(): Double = {size * 5.0}
}
```

```scala
class HealthPotion(val volume: Int) extends GameObject(3.0) {
  val massPerVolume: Double = 7.0
  override def weight(): Double = {
    val bottleWeight: Double = super.weight()
    bottleWeight + this.volume * this.massPerVolume
  }
}
```

```scala
class Player() {
  var inventory: List[GameObject] = List()
  def pickUp(obj: GameObject): Unit = {
    this.inventory = obj :: this.inventory
  }
  def totalWeight(): Double = {
    var total: Double = 0.0
    for(obj <- this.inventory){
      total += obj.weight()
    }
    total
  }
}
```

```scala
def main(args: Array[String]): Unit = {
  val ball: DodgeBall = new DodgeBall(4.0)
  val potion: HealthPotion = new HealthPotion(6)
  val player: Player = new Player()
  player.pickUp(ball)
  player.pickUp(potion)
  val totalWeight: Double = player.totalWeight()
  println(totalWeight)
}
```

Stack

Name	Value
DodgeBall: ball	0x350
DodgeBall: this	0x350
DodgeBall: size	4.0
GameObject: this	0x350
GameObject: baseWeight	0.0
HealthPotion: potion	0x200
HealthPotion: this	0x200
HealthPotion: volume	6
GameObject: this	0x200
GameObject: baseWeight	3.0
Player: player	0x480
Player: this	0x480
pickUp: this	0x480
pickUp: obj	0x350
pickUp: this	0x480
pickUp: obj	0x200
totalWeight	65.0
totalWeight: this	0x480
totalWeight: total	0.0 45.0 65.0
totalWeight: obj	~~0x200~~ ~~0x350~~
weight: this	0x200
weight: bottleWeight	3.0
weight: this	0x200
weight: this	0x350

Heap

DodgeBall

size	4.0
baseWeight	0.0

0x350

HealthPotion

volume	6
baseWeight	3.0
massPerVolume	7.0

0x200

Player

inventory	~~0x120~~ 0x800

0x480

value	0x350
next	

0x120

value	0x200
next	0x120

0x800

in/out

65.0

- The totalWeight variable gets the returned value
- Print it to the screen

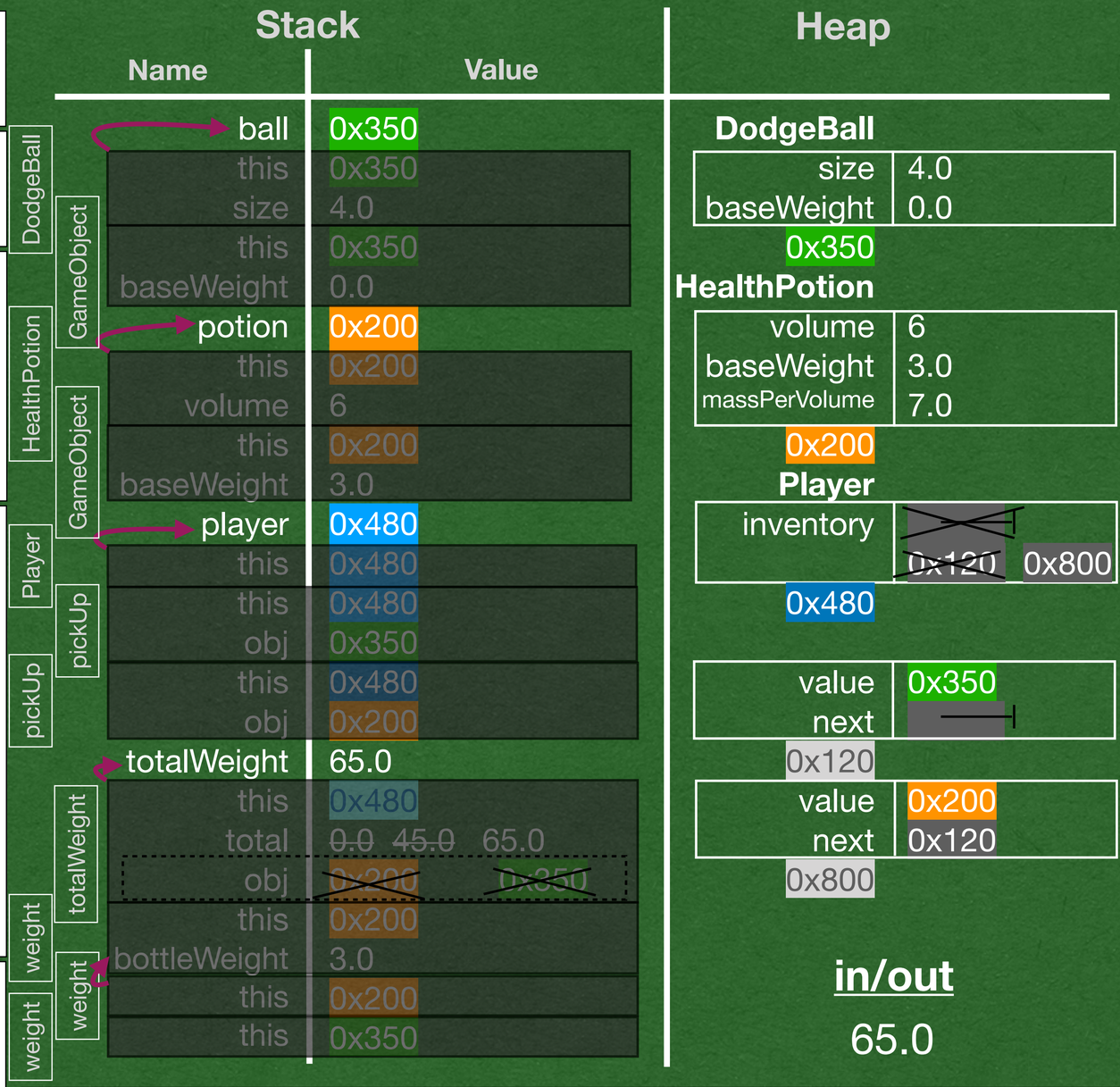
```scala
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}
```

```scala
class DodgeBall(val size: Double) extends GameObject(0.0) {
  override def weight(): Double = {size * 5.0}
}
```

```scala
class HealthPotion(val volume: Int) extends GameObject(3.0) {
  val massPerVolume: Double = 7.0
  override def weight(): Double = {
    val bottleWeight: Double = super.weight()
    bottleWeight + this.volume * this.massPerVolume
  }
}
```

```scala
class Player() {
  var inventory: List[GameObject] = List()
  def pickUp(obj: GameObject): Unit = {
    this.inventory = obj :: this.inventory
  }
  def totalWeight(): Double = {
    var total: Double = 0.0
    for(obj <- this.inventory){
      total += obj.weight()
    }
    total
  }
}
```

```scala
def main(args: Array[String]): Unit = {
  val ball: DodgeBall = new DodgeBall(4.0)
  val potion: HealthPotion = new HealthPotion(6)
  val player: Player = new Player()
  player.pickUp(ball)
  player.pickUp(potion)
  val totalWeight: Double = player.totalWeight()
  println(totalWeight)
}
```



- Program ends

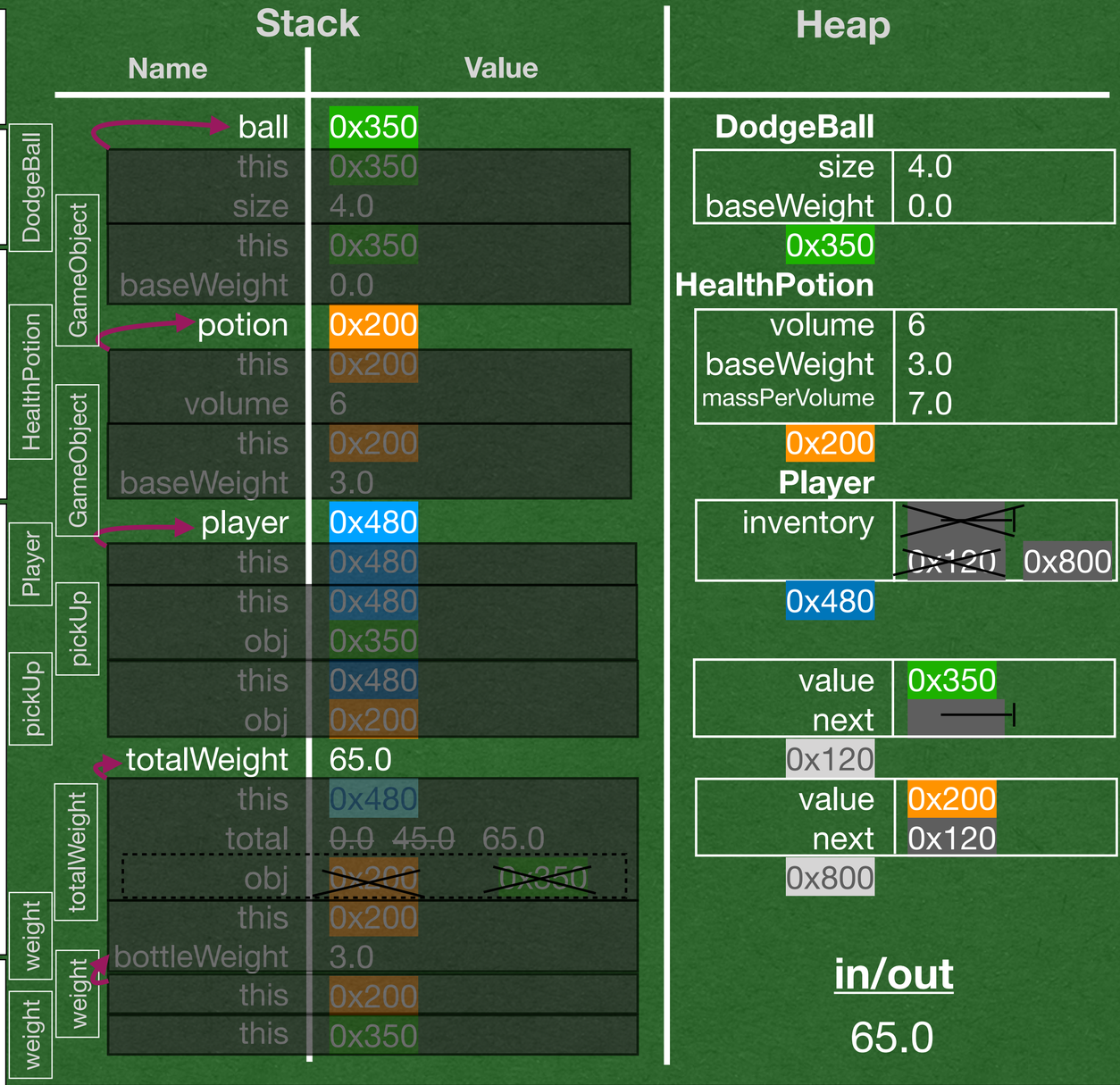
```scala
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}
```

```scala
class DodgeBall(val size: Double) extends GameObject(0.0) {
  override def weight(): Double = {size * 5.0}
}
```

```scala
class HealthPotion(val volume: Int) extends GameObject(3.0) {
  val massPerVolume: Double = 7.0
  override def weight(): Double = {
    val bottleWeight: Double = super.weight()
    bottleWeight + this.volume * this.massPerVolume
  }
}
```

```scala
class Player() {
  var inventory: List[GameObject] = List()
  def pickUp(obj: GameObject): Unit = {
    this.inventory = obj :: this.inventory
  }
  def totalWeight(): Double = {
    var total: Double = 0.0
    for(obj <- this.inventory){
      total += obj.weight()
    }
    total
  }
}
```

```scala
def main(args: Array[String]): Unit = {
  val ball: DodgeBall = new DodgeBall(4.0)
  val potion: HealthPotion = new HealthPotion(6)
  val player: Player = new Player()
  player.pickUp(ball)
  player.pickUp(potion)
  val totalWeight: Double = player.totalWeight()
  println(totalWeight)
}
```



- If you can follow this entire example, you have a great understanding of inheritance and polymorphism!