

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)  
}
```

```
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)
}
```

Stack		Heap
Name	Value	
		<u>in/out</u>

- How will the objects be created?

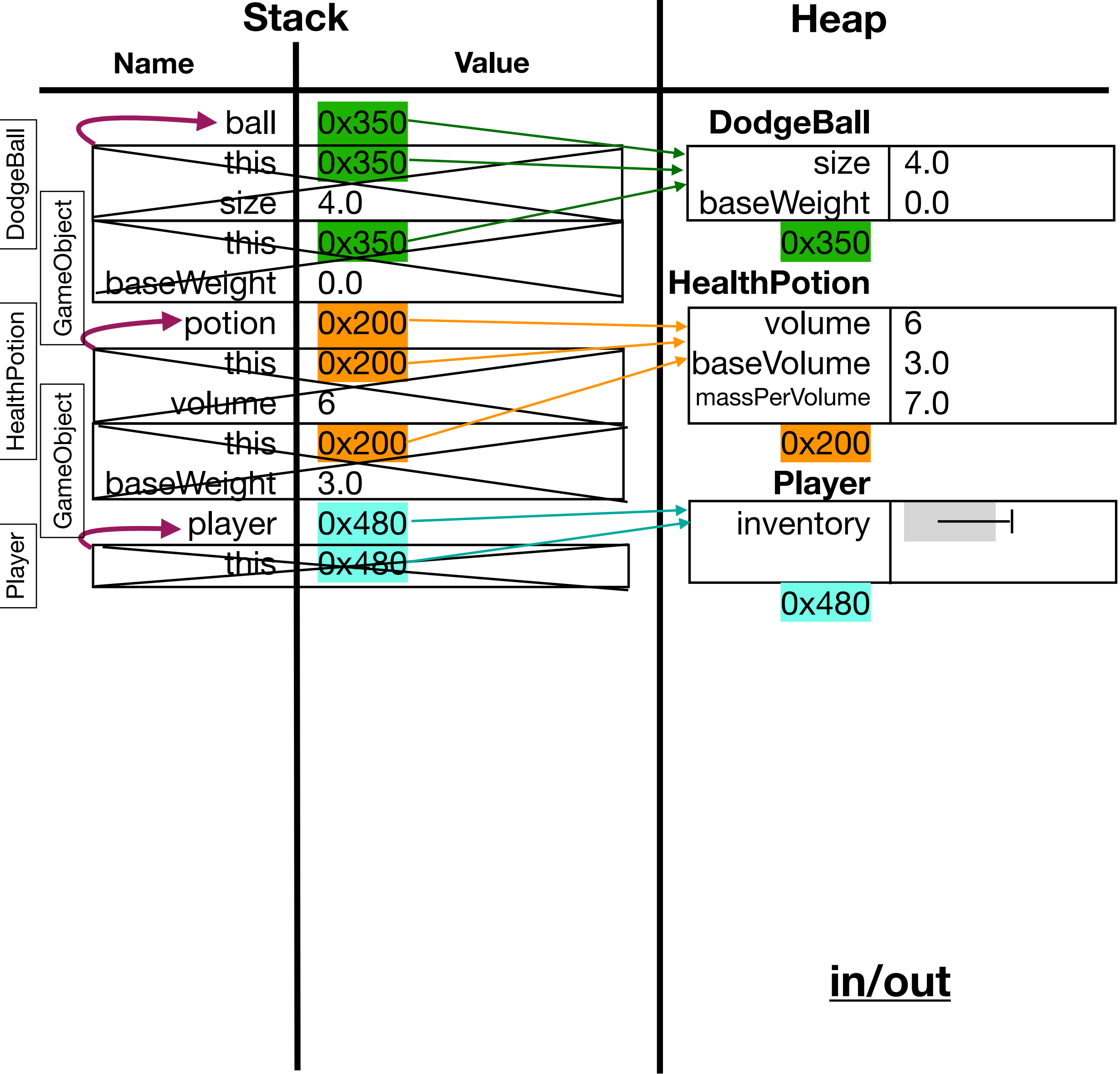
```
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}

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)
}
```



- If you said this
- You're right!

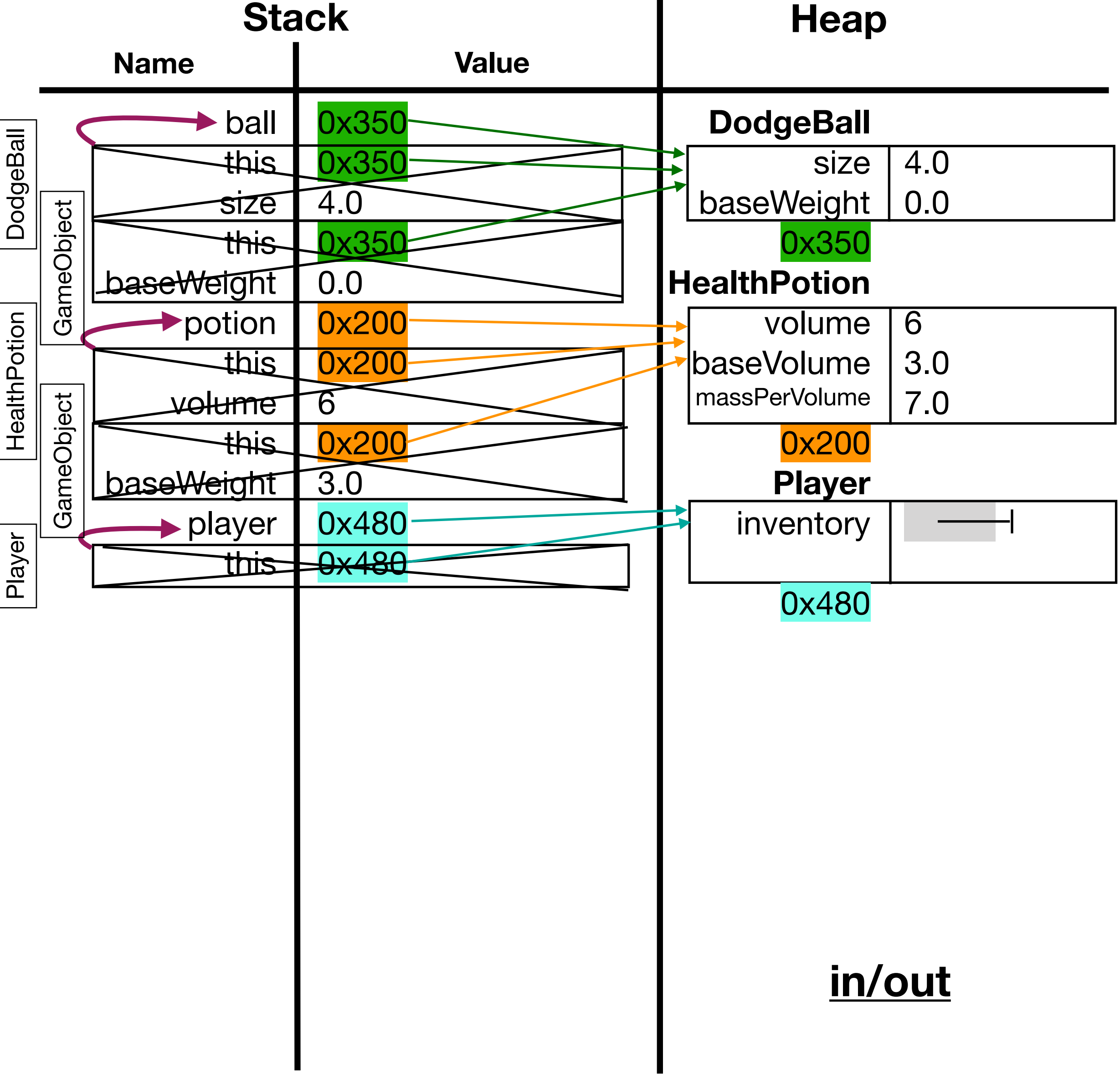
```
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}

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)
}
```



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

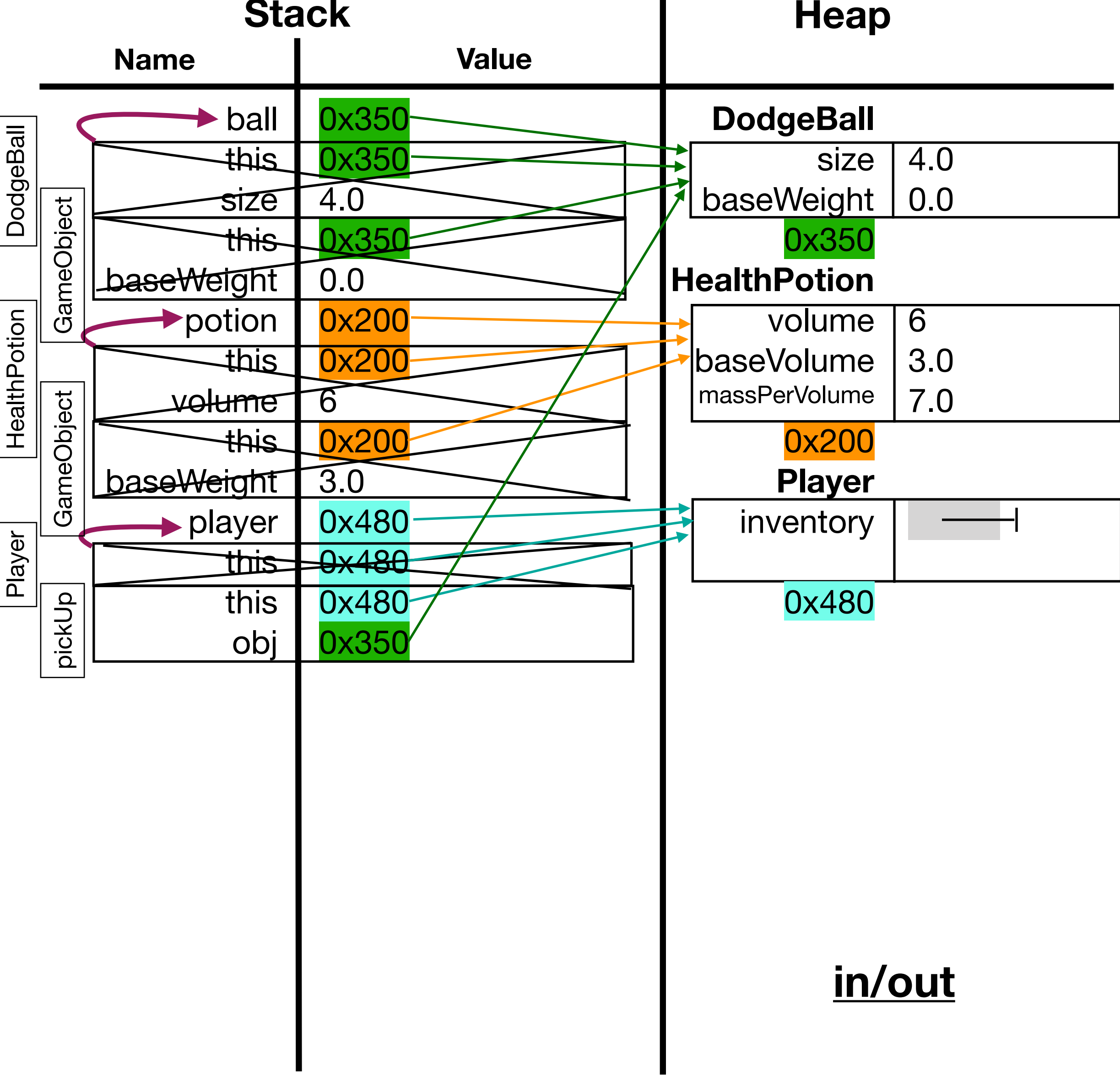

```
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}

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)
}
```



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

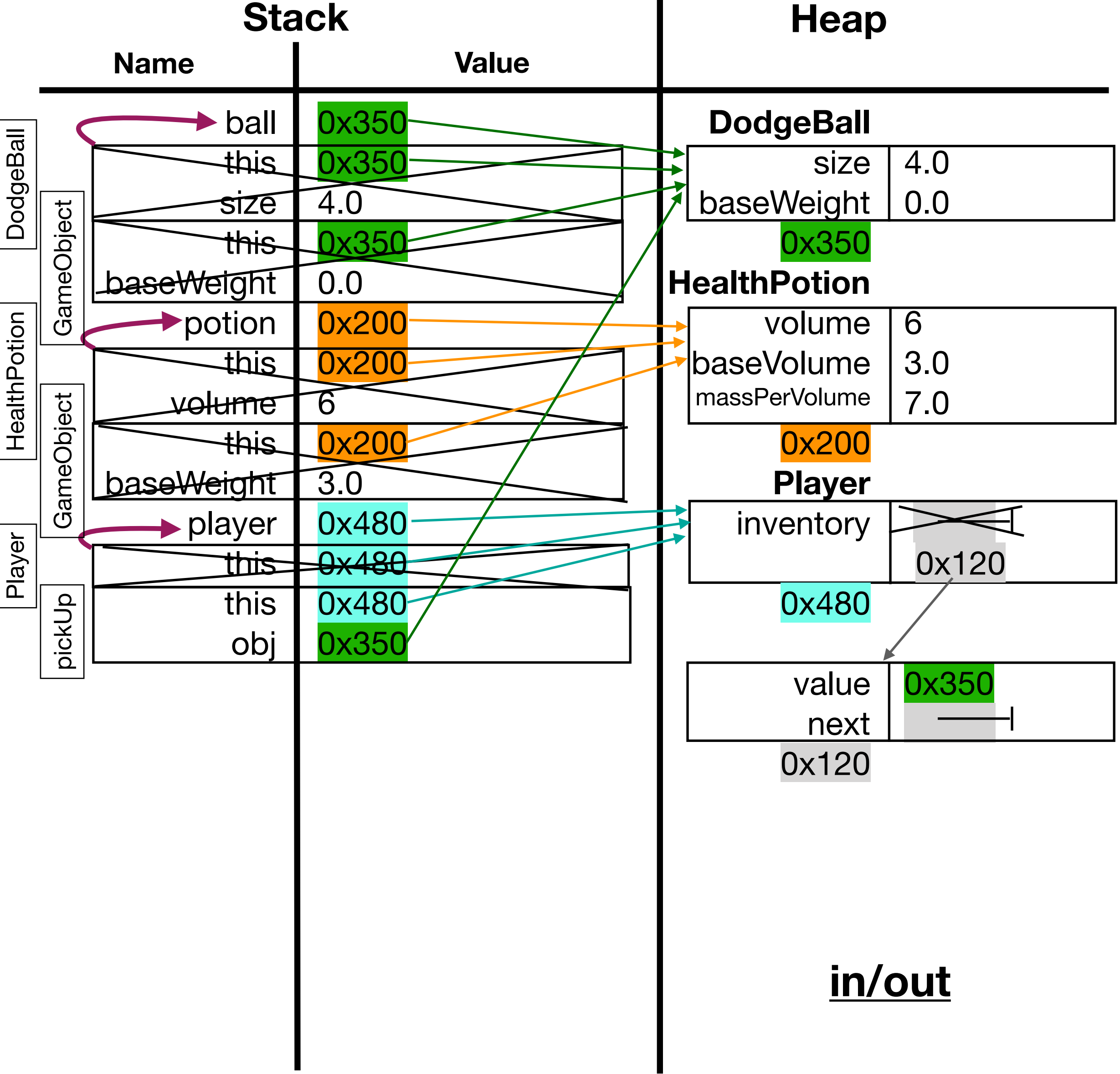
```
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}

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)
}
```



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

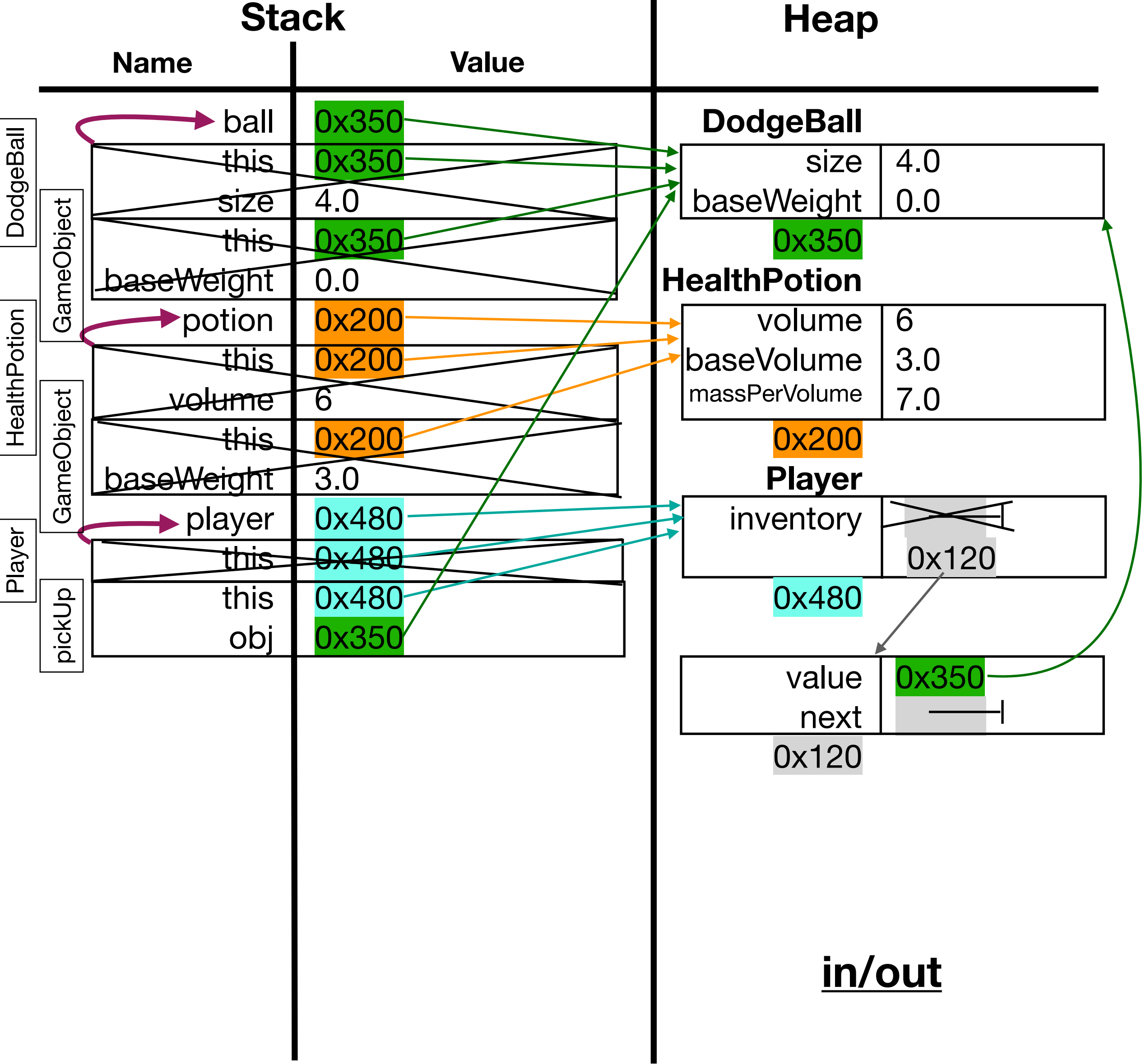

```
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}

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)
}
```



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

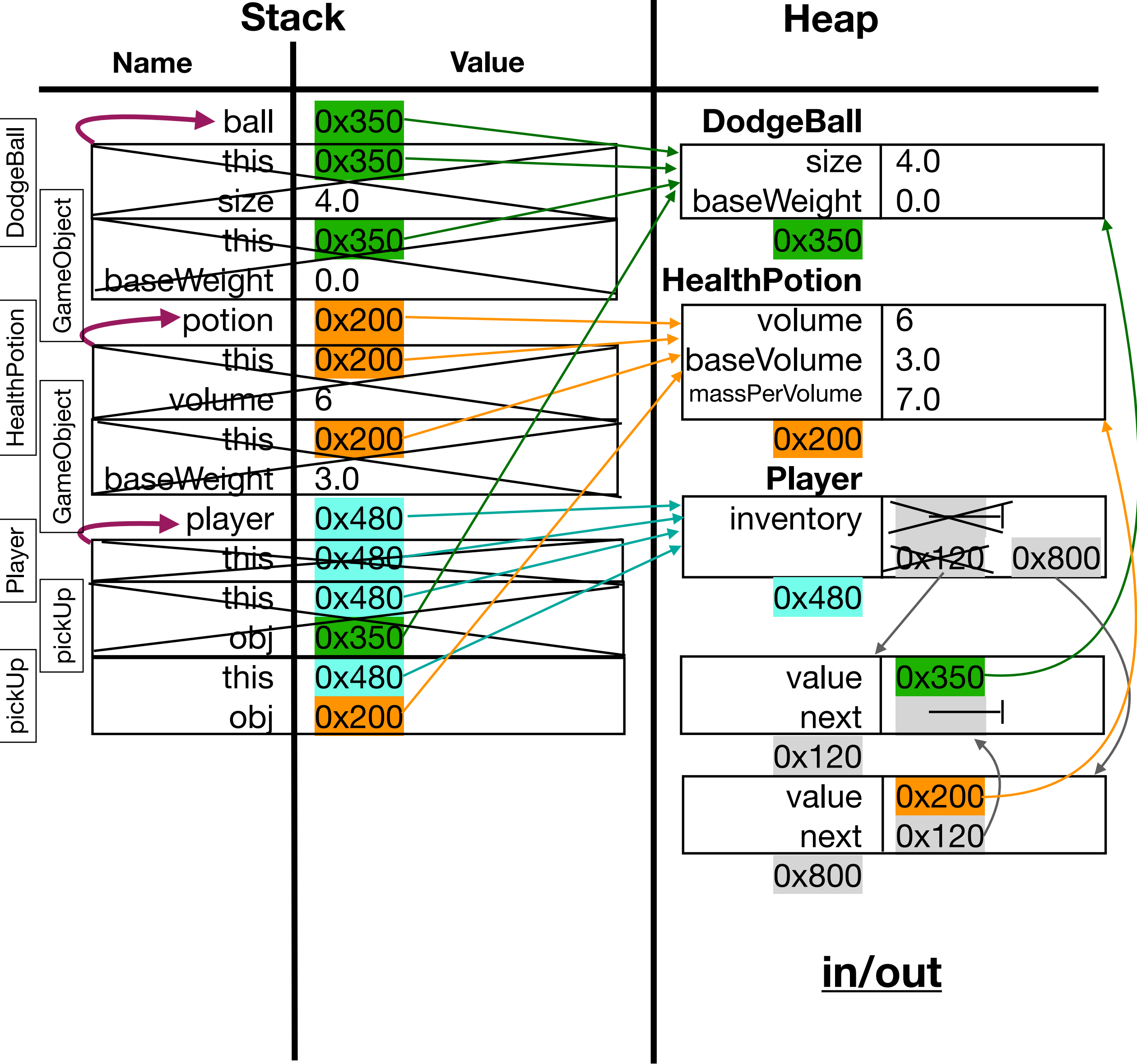
```
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}

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)
}
```



• Repeat to prepend the HealthPotion to the list

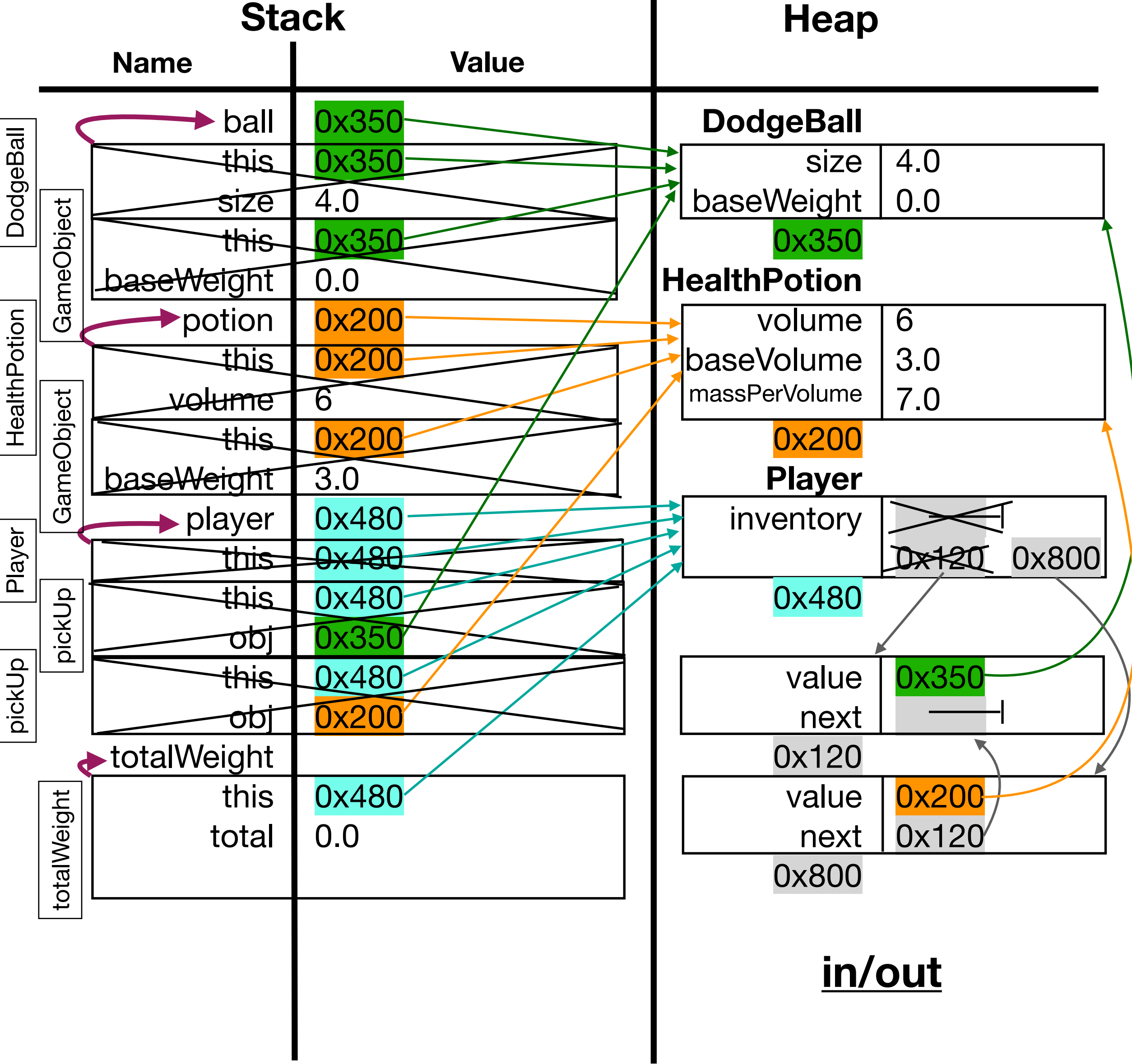
```
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}

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)
}
```



• Add a stack frame for totalWeight

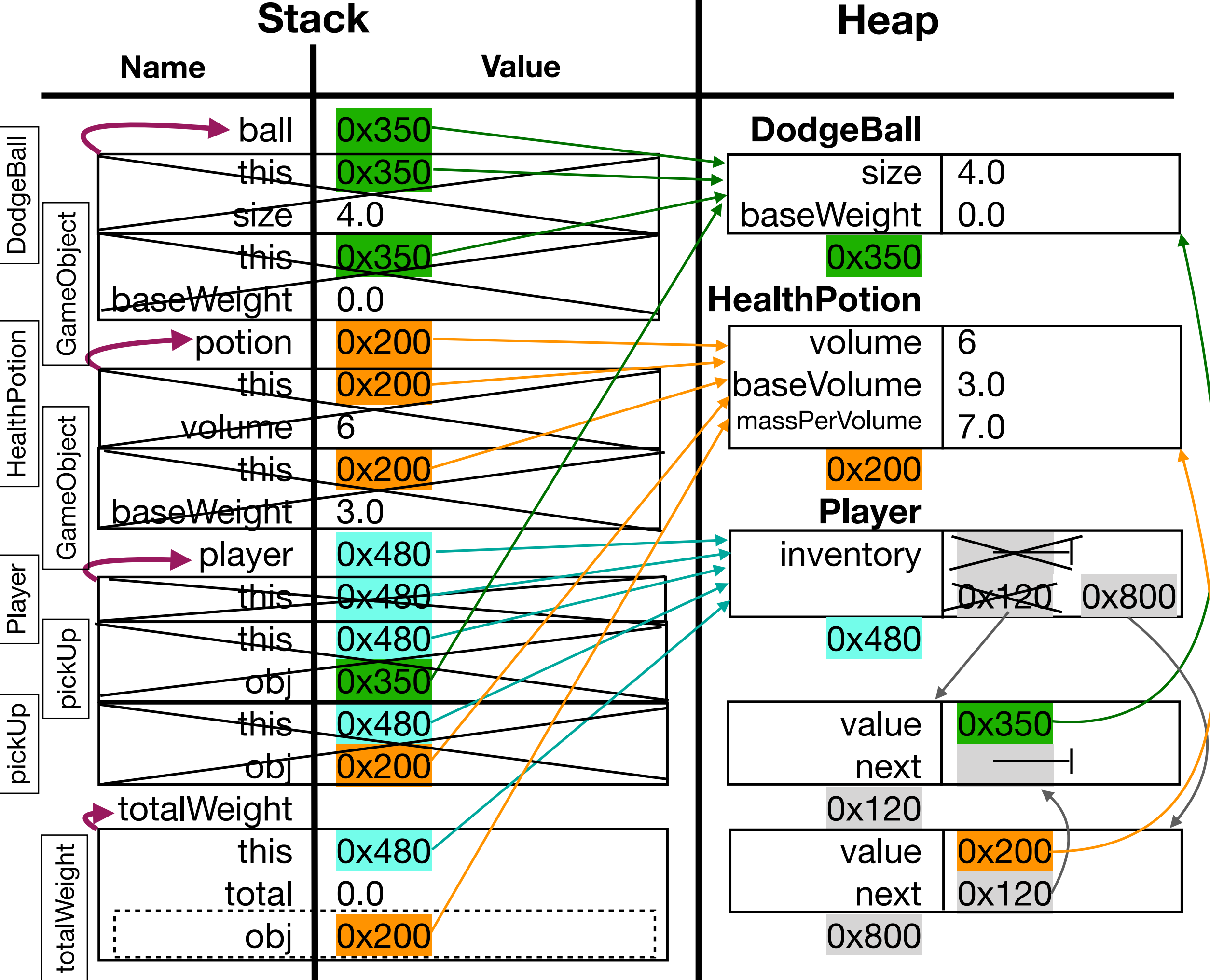

```
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}
```

```
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)
}
```



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

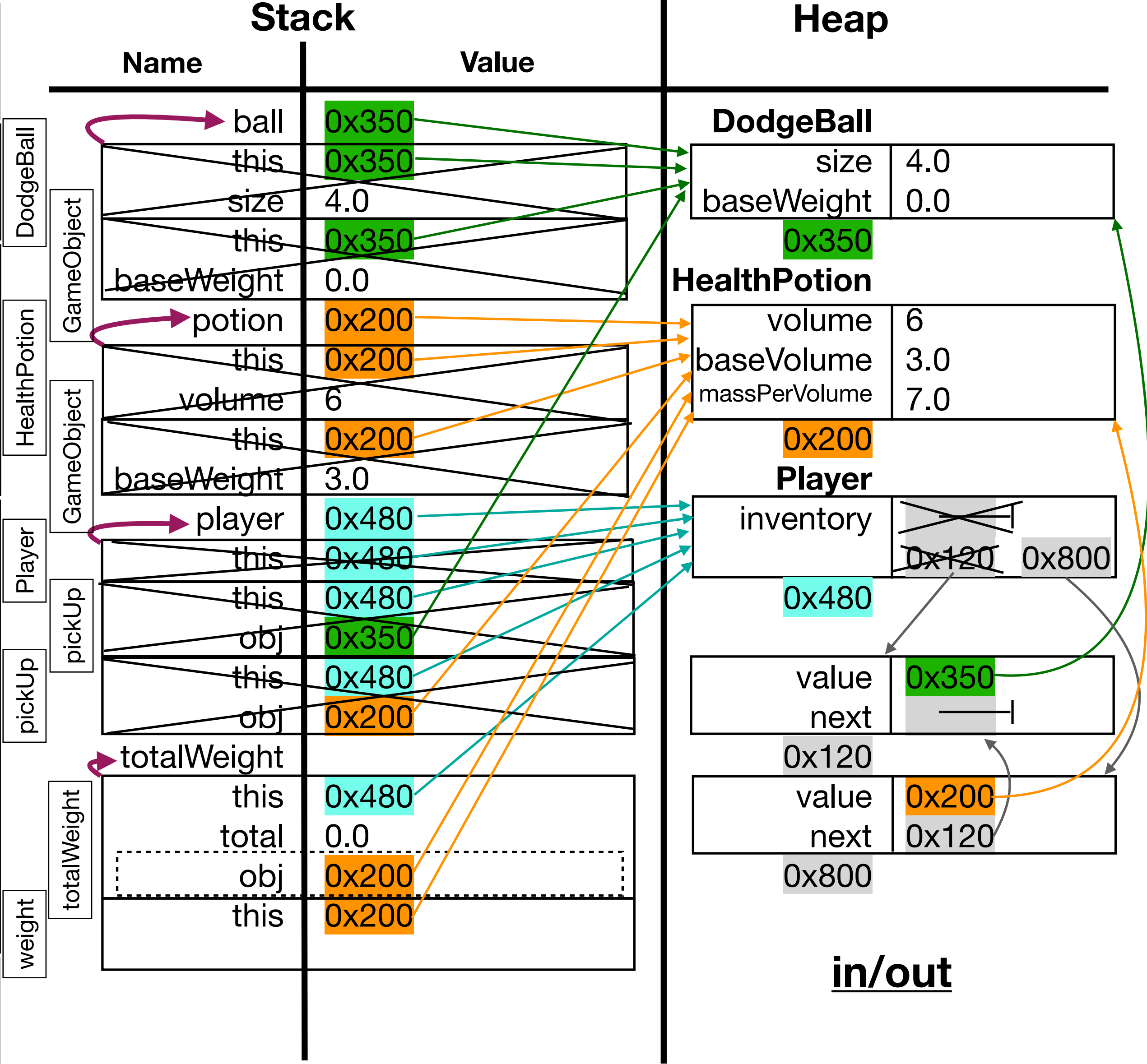
```
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}

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)
}
```



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

→

```
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}
}
```

→

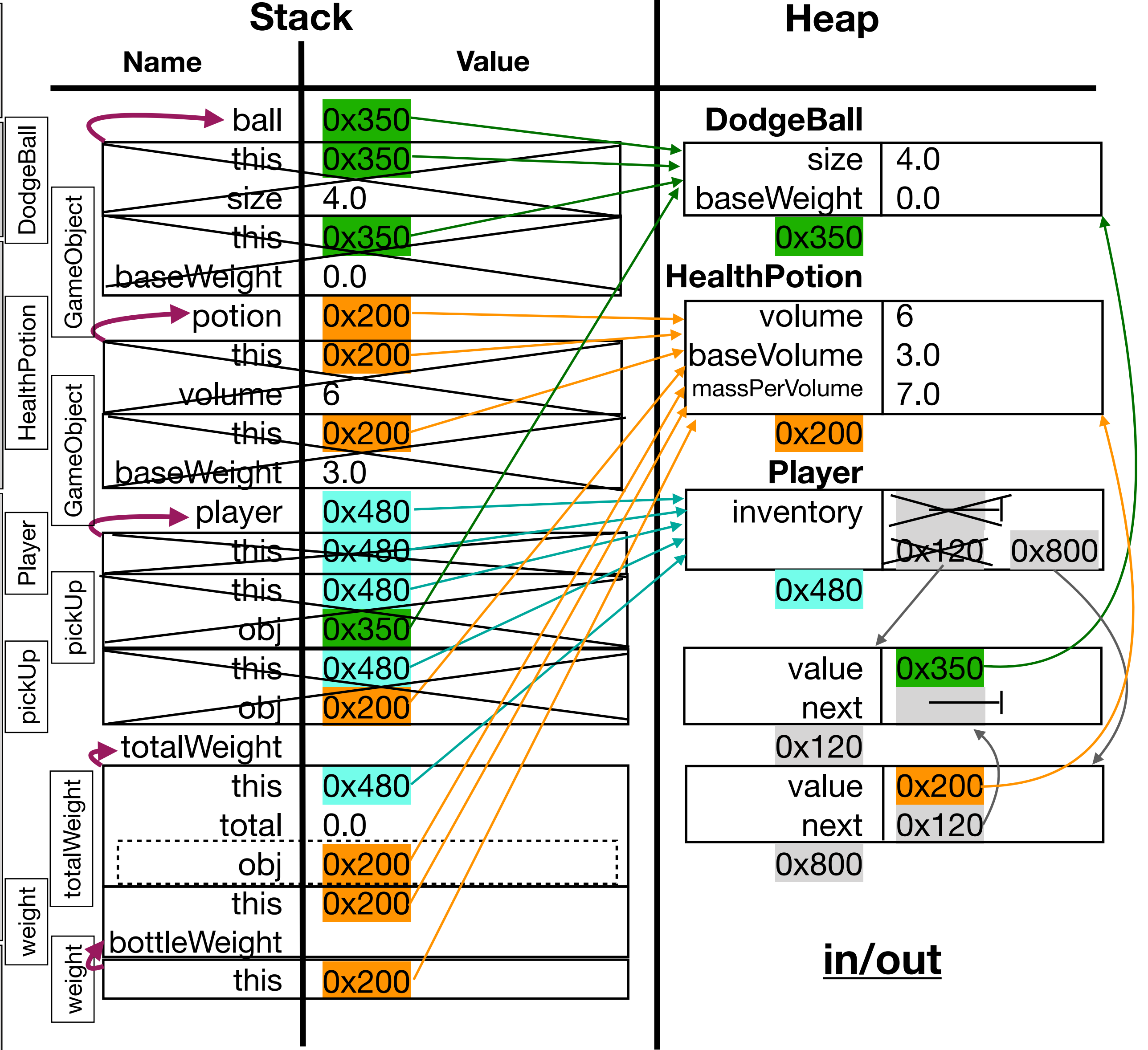
```
class HealthPotion(val volume: Int) extends GameObject(3.0) {
  val massPerVolume: Double = 7.0
  override def weight(): Double = {
    val bottleWeight: Double = super.weight()
    bottleWeight + this.volume * this.massPerVolume
  }
}
```

→

```
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)
}
```



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

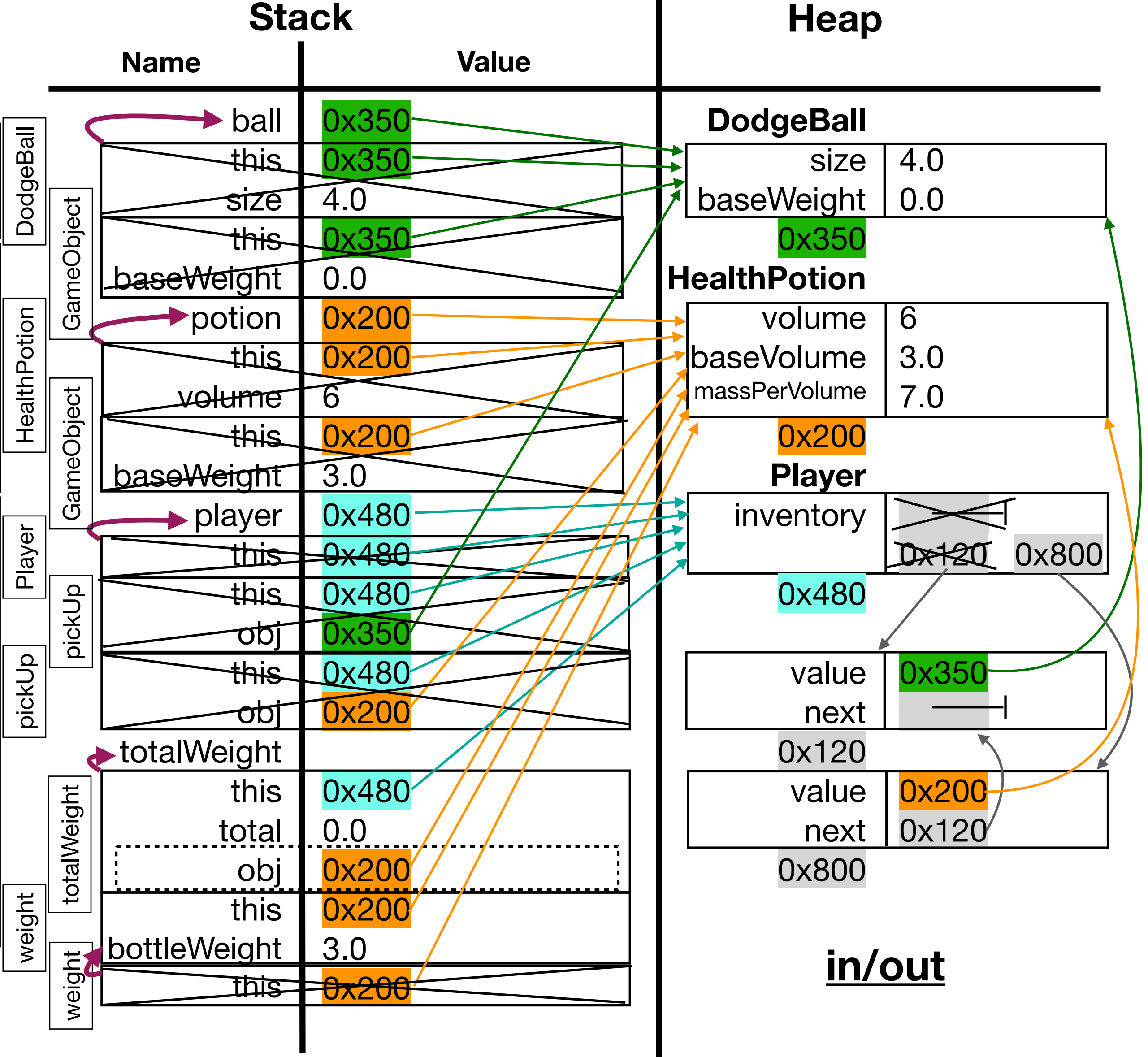
```
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}

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)
}
```



- The super class method returns
- Continue with the method

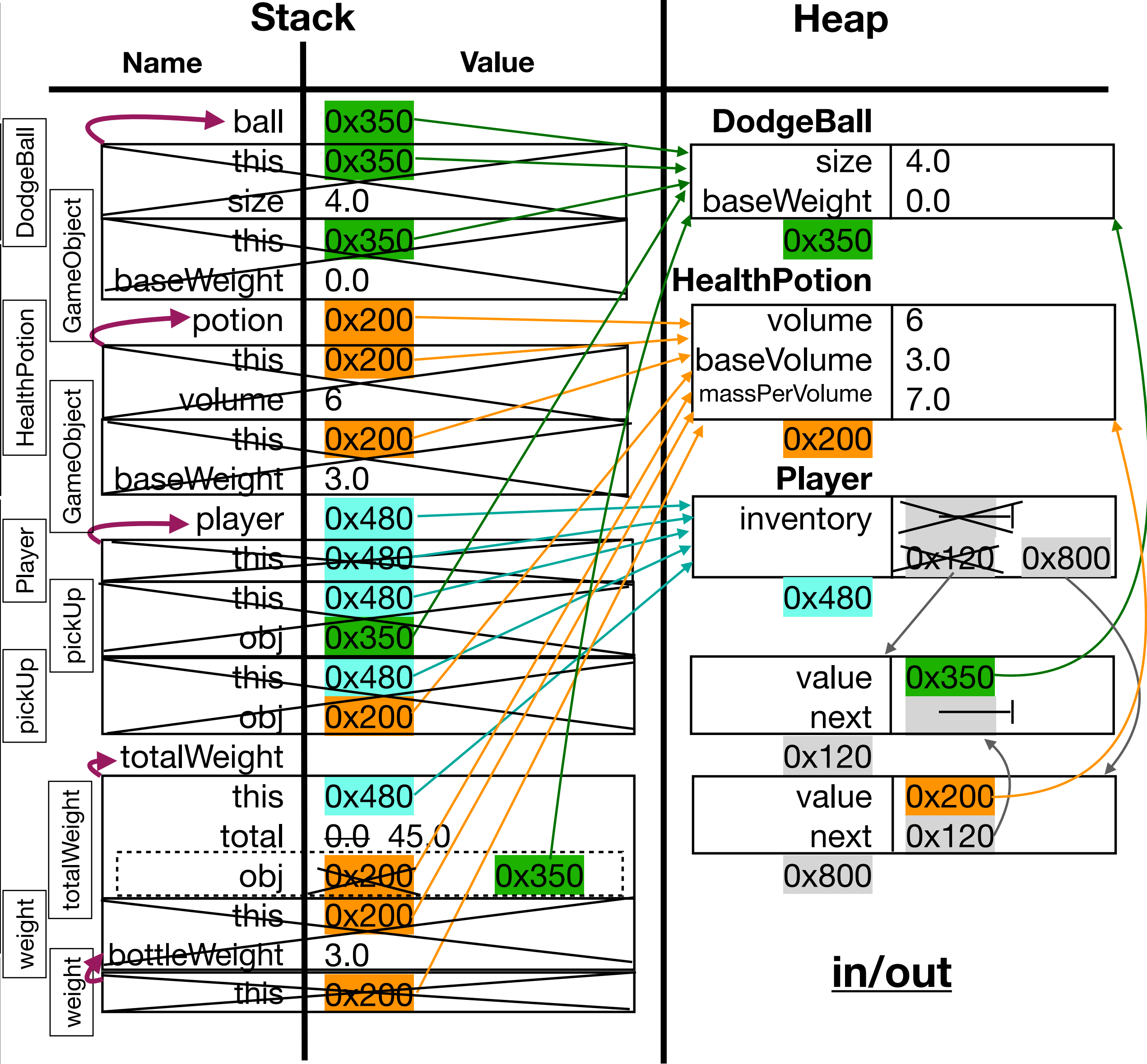

```
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}

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)
}
```



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

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



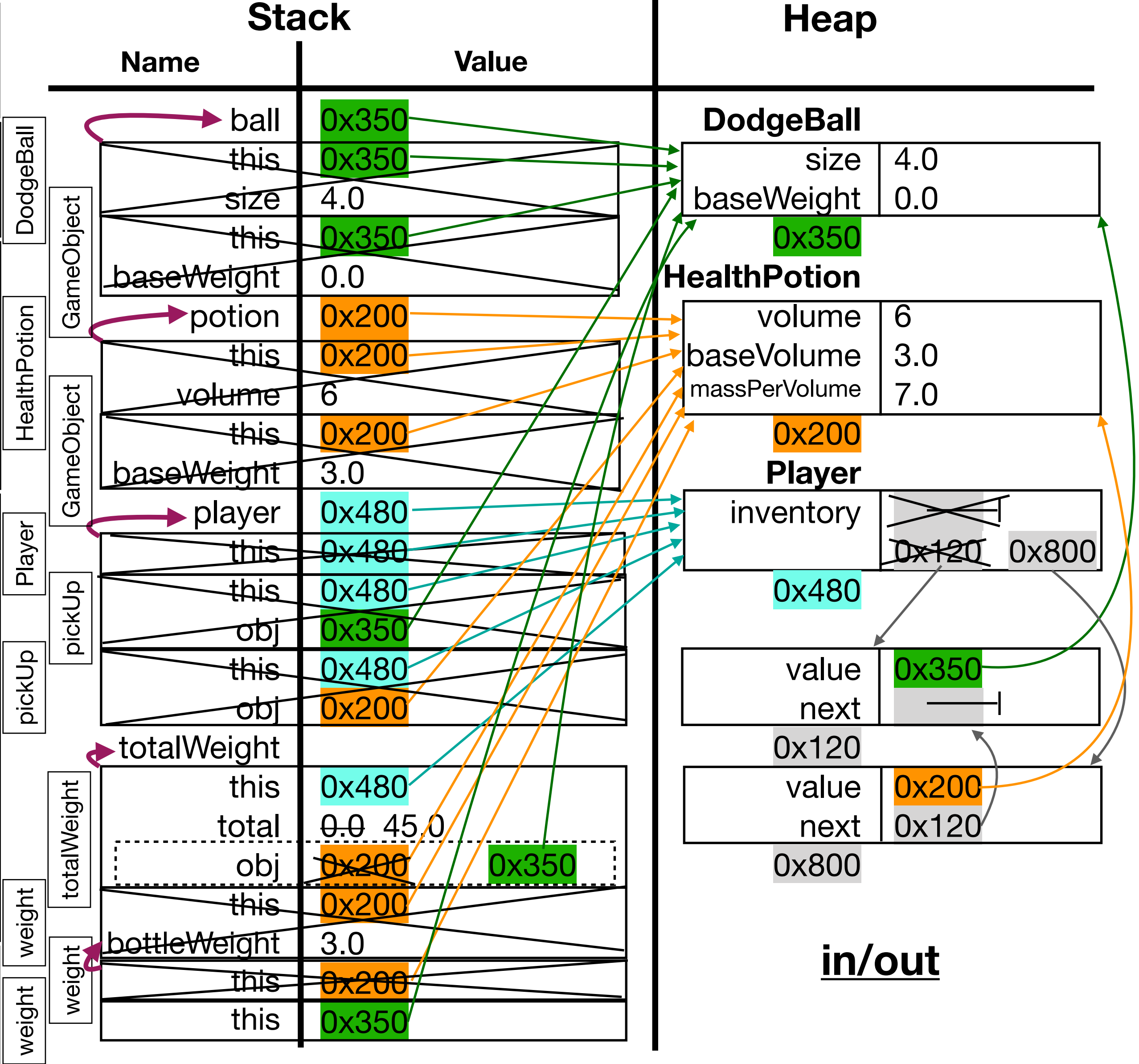
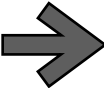
```
class DodgeBall(val size: Double) extends GameObject(0.0) {
  override def weight(): Double = {size * 5.0}
}
```

```
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)
}
```



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

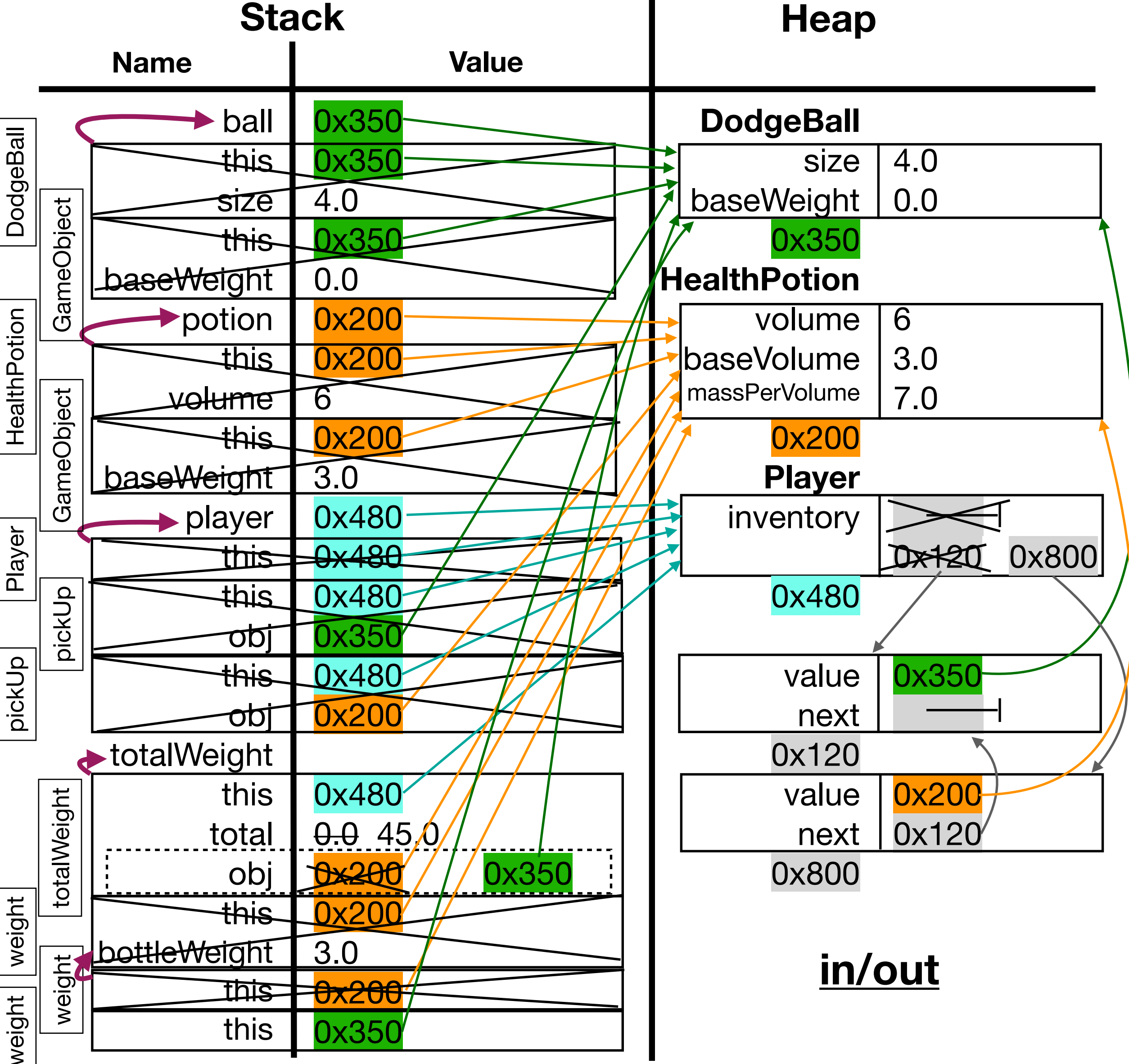

```
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}

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)
}
```



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

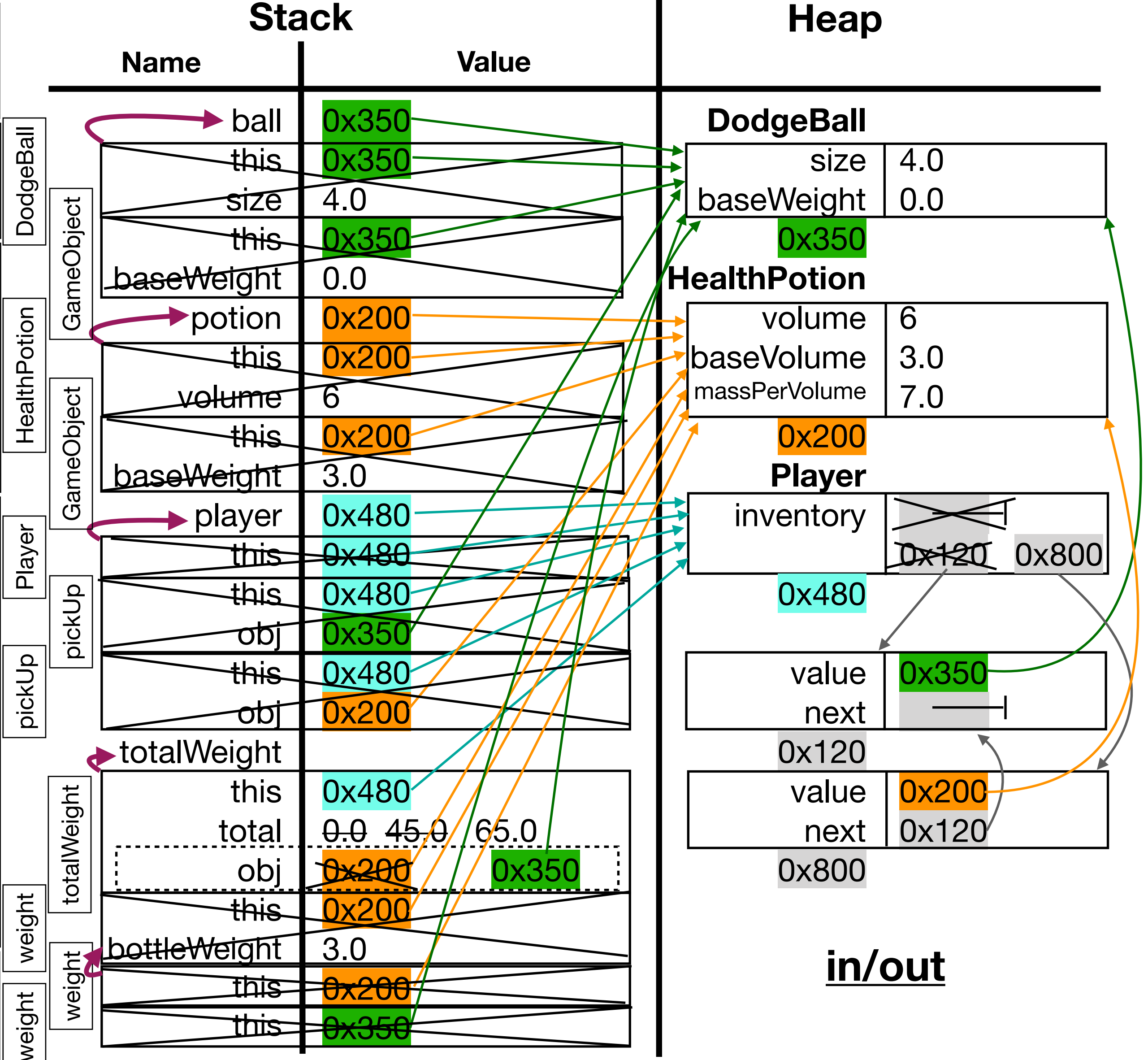

```
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}

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)
}
```



• Return 20.0 and add it to total

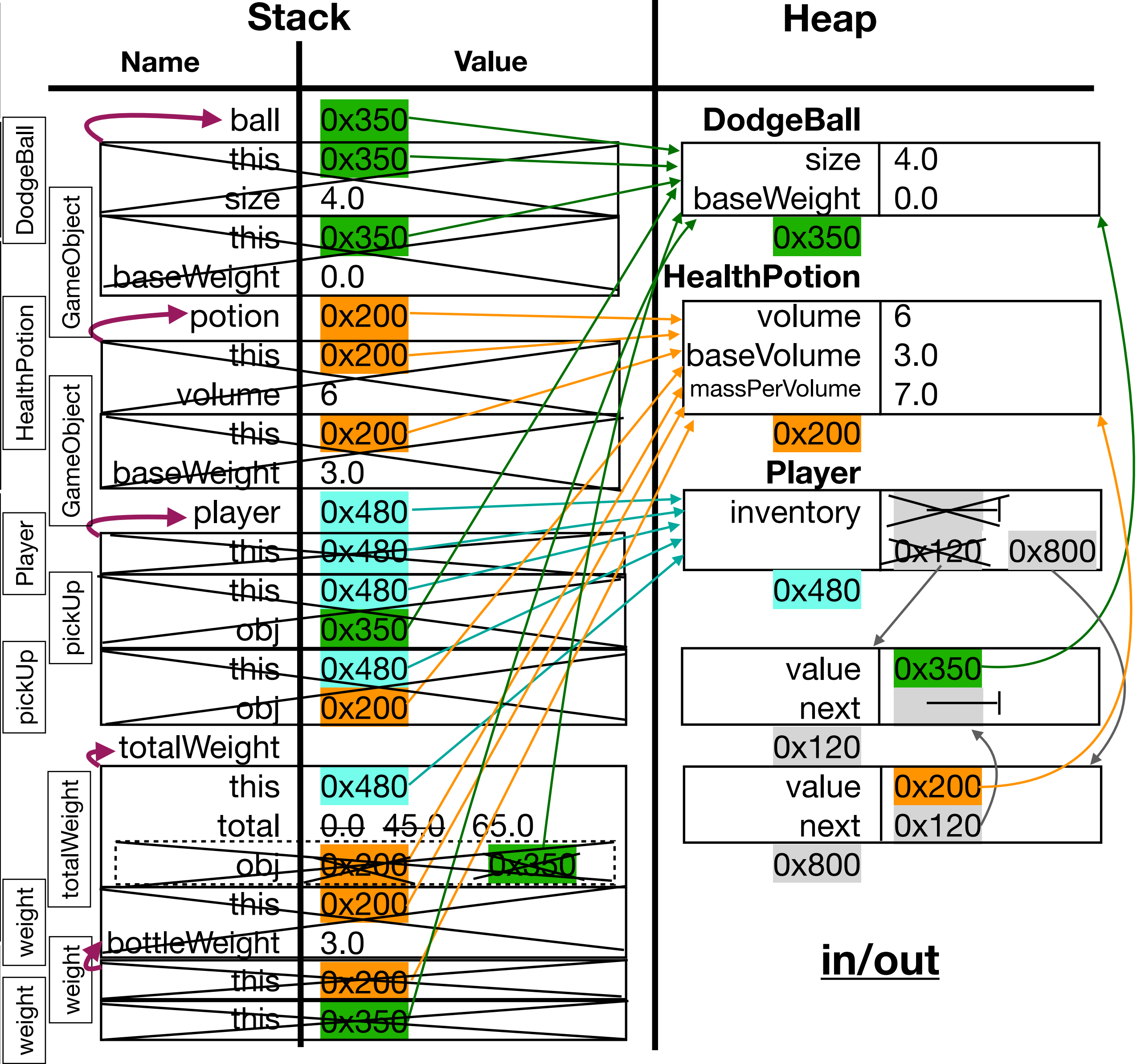
```
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}

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)
}
```



• End of loop

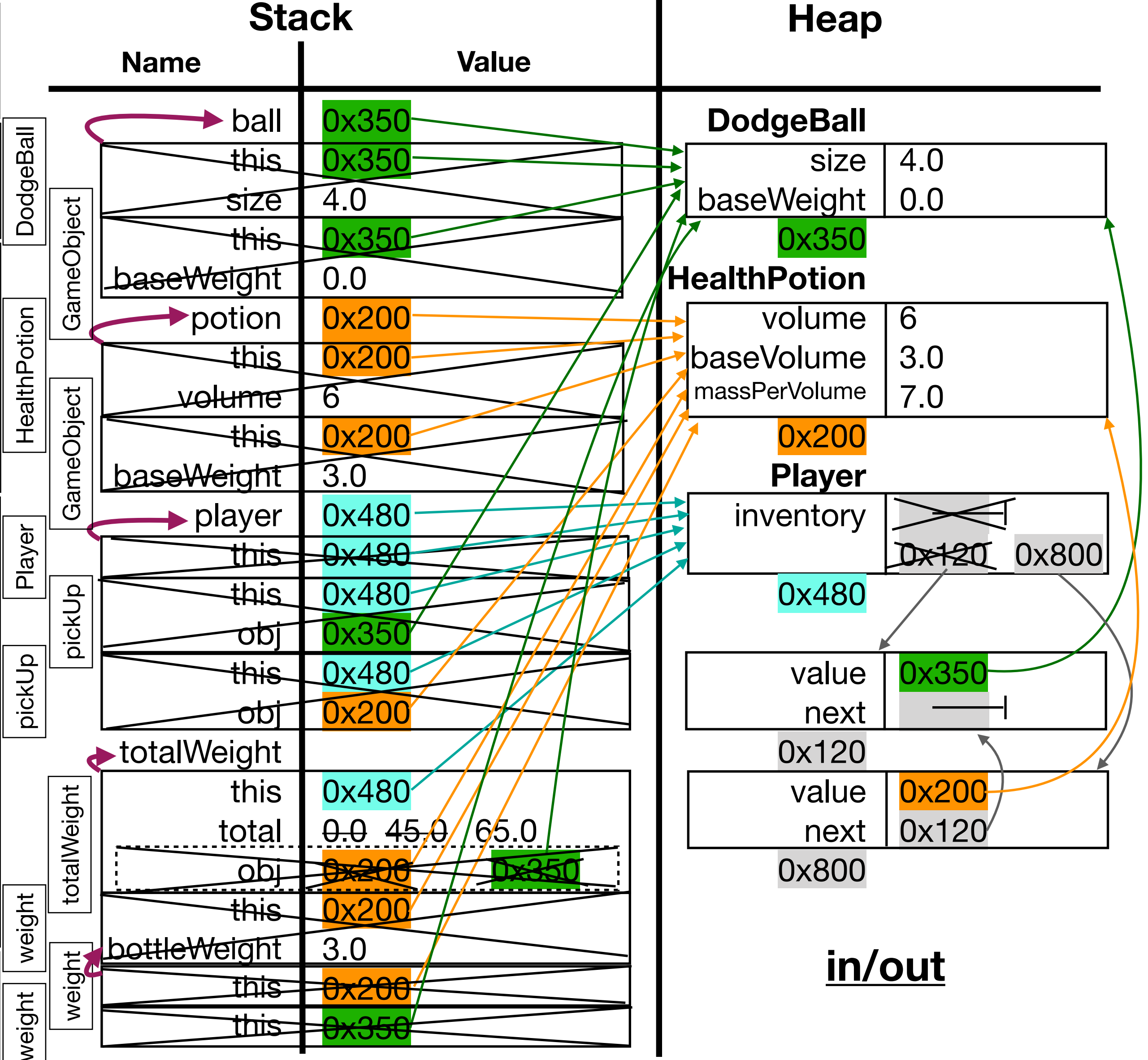

```
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}

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)
}
```



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

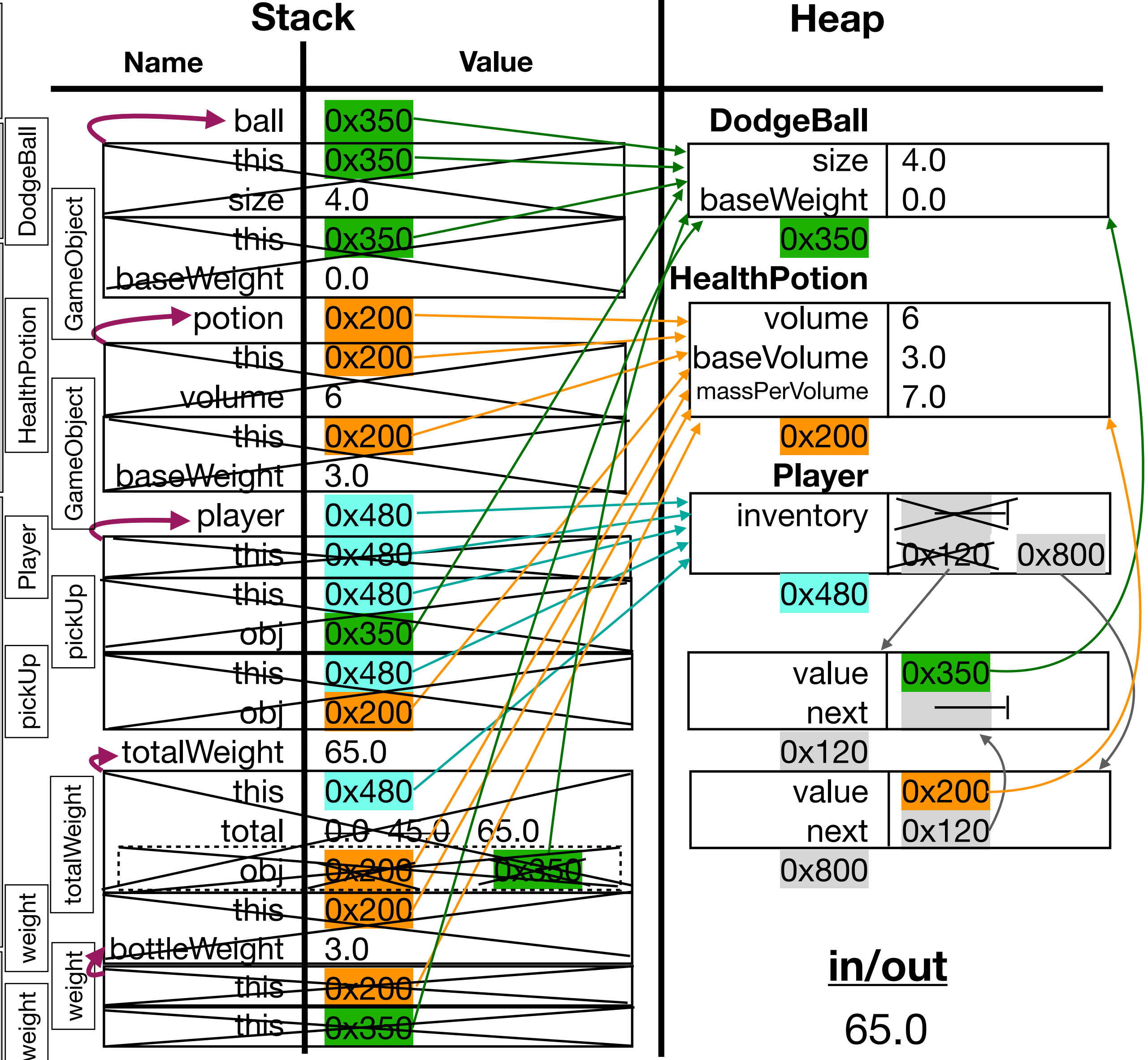
```
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}

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)
}
```



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

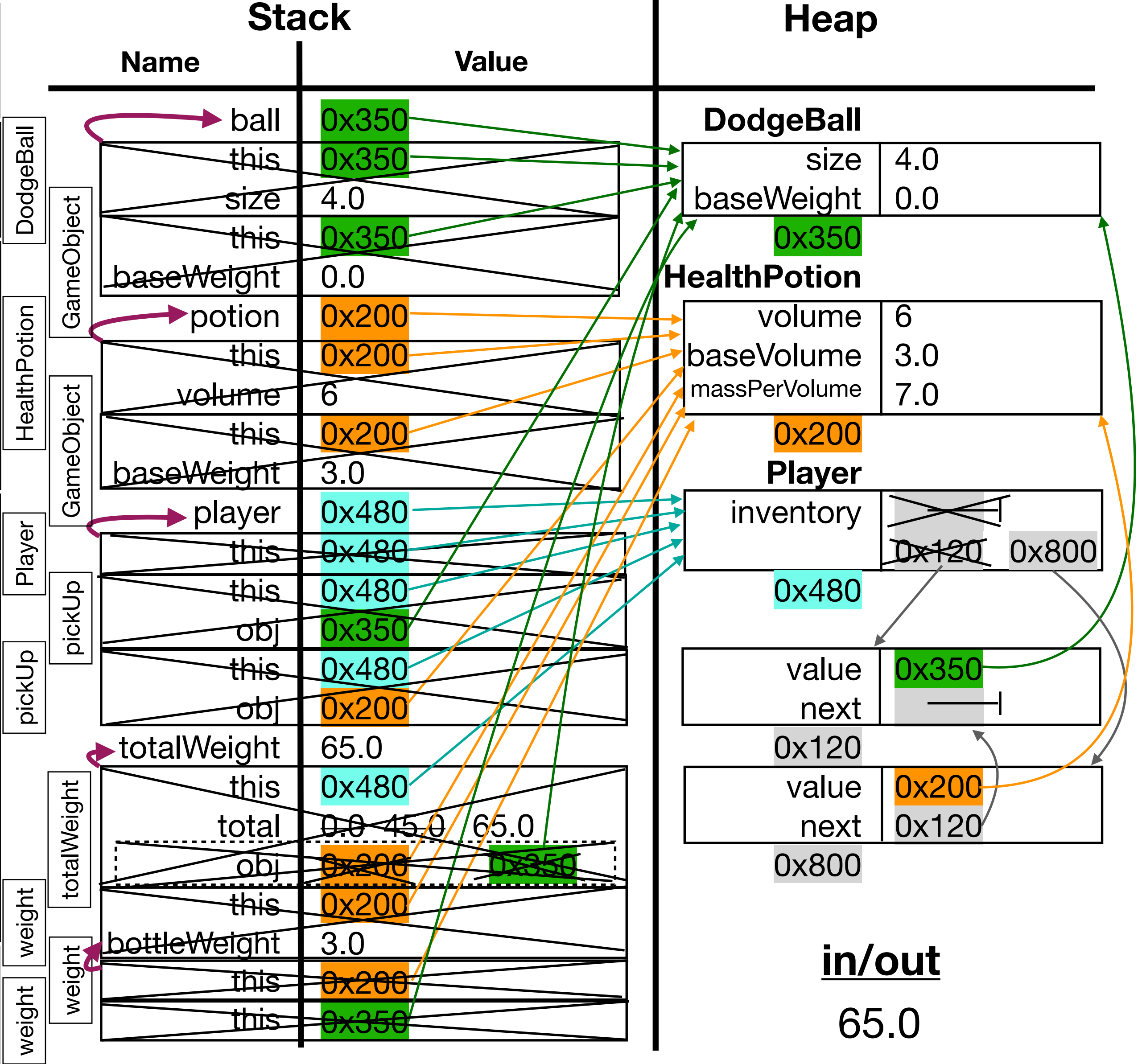

```
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}

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)
}
```



• Program ends

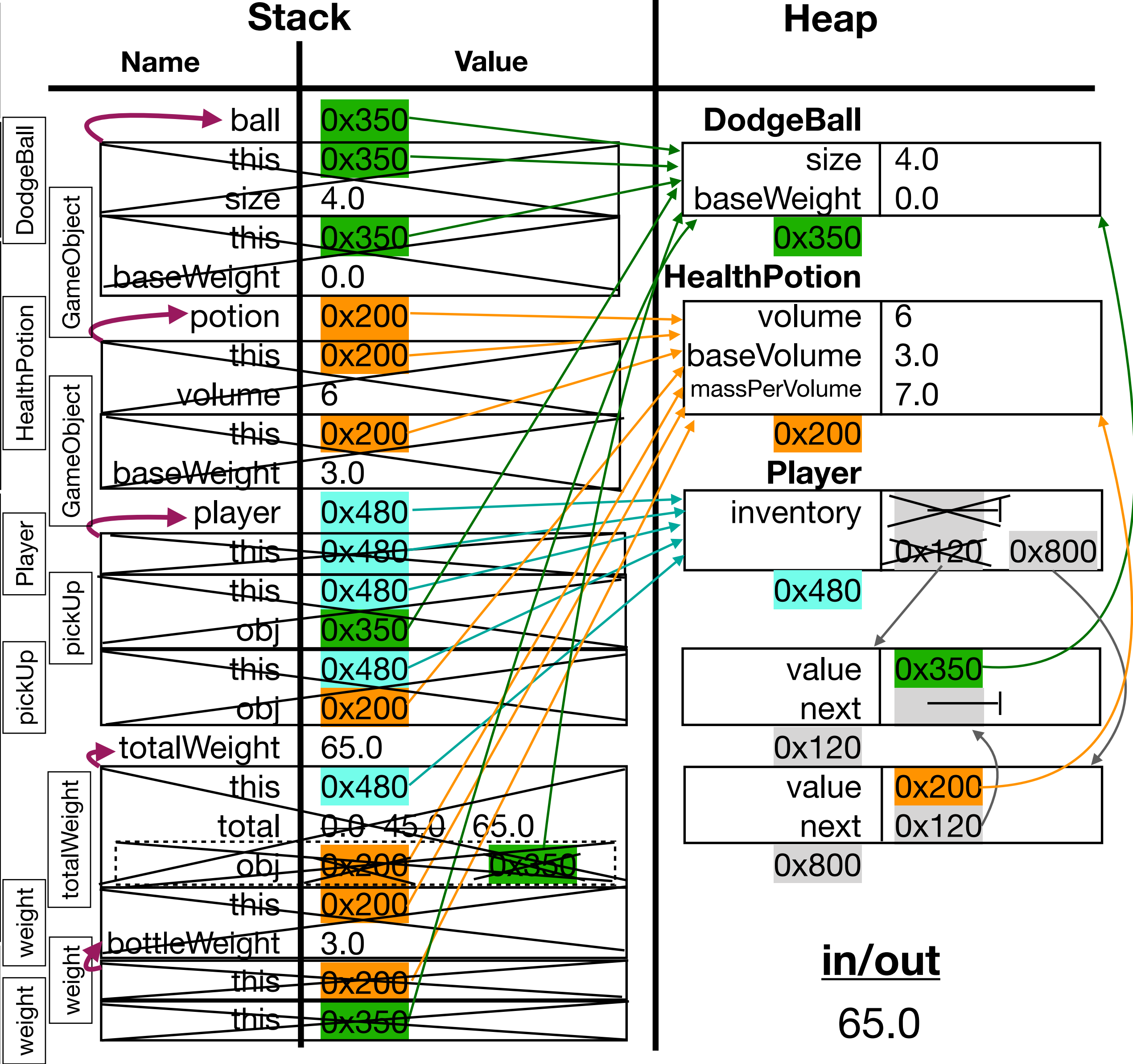

```
abstract class GameObject(val baseWeight: Double) {
  def weight(): Double = {baseWeight}
}

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)
}
```



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