

Objects and Classes

Objects

Objects have State and Behavior

Objects

- State / Variables
 - Objects store their state in variables
- Behavior / Functions
 - Objects contain functions that can depend on its state
 - [Vocab] When a function is part of an object it's called a **method**

Object With State

```
object ObjectWithState {  
  
  // State of the object  
  var x: Int = 10  
  var y: Int = 7  
  
  // Behavior of the object  
  def doubleX(): Unit = {  
    this.x *= 2  
  }  
  
}
```

- Any variable outside of all methods is part of the state of the object
- Keyword **this** stores a reference to the enclosing object
- Use `this.<variable_name>` to access state from within the object

Object With State

```
object ObjectWithState {  
  
  // State of the object  
  var x: Int = 10  
  var y: Int = 7  
  
  // Behavior of the object  
  def doubleX(): Unit = {  
    this.x *= 2  
  }  
  
}
```

- The variables defining the state of an object have many different names
 - Instance variables
 - Member variables
 - Fields
 - State variables <-- I'll use this one in CSE116

Object With State

```
object ObjectWithState {  
  
  // State of the object  
  var x: Int = 10  
  var y: Int = 7  
  
  // Behavior of the object  
  def doubleX(): Unit = {  
    this.x *= 2  
  }  
  
}
```

```
object ObjectMain {  
  
  def main(args: Array[String]): Unit = {  
    ObjectWithState.doubleX()  
    println(ObjectWithState.x)  
  }  
  
}
```

- Any code with access to an object can also access its state/behavior with the dot notation
- This syntax is used to call methods in the Math object

Object With State

```
object ObjectWithState {  
  
  // State of the object  
  var x: Int = 10  
  var y: Int = 7  
  
  // Behavior of the object  
  def doubleX(): Unit = {  
    this.x *= 2  
  }  
  
}
```

```
object ObjectMain {  
  
  def main(args: Array[String]): Unit = {  
    ObjectWithState.doubleX()  
    println(ObjectWithState.x)  
  }  
  
}
```

- The state of an object can be changed
- We called a method that changed the value of a state variable

Every *value* in Scala is an **object**

Classes

- Every value in Scala is an object
 - You can use the . dot operator to access the state and behaviour of any value
 - Example: Calling methods from a String object (length, split, contains, toLowerCase)
 - Example: Accessing the PI value from the Math object

Classes

- Classes are templates used to create objects
 - Objects are **instantiated** from classes using the keyword **new**
- Classes define a type
 - Used to create many objects of the same type
 - Each object can have a different state
 - Each has its own copies of the state variables

Classes

- Let's create a Player class with
 - A location on an x/y coordinate system
 - A fixed max hit points
 - Current hit points
 - The ability to damage other players

Classes

```
class Player(var xLocation: Double, var yLocation: Double, val maxHitPoints: Int) {  
  
    var hp: Int = this.maxHitPoints  
    val damageDealt: Int = 4  
  
    def takeDamage(damage: Int): Unit = {  
        this.hp -= damage  
    }  
  
    def attack(otherPlayer: Player): Unit = {  
        otherPlayer.takeDamage(this.damageDealt)  
    }  
  
    def conscious(): Boolean = {  
        this.hp > 0  
    }  
  
    def move(dx: Double, dy: Double): Unit = {  
        this.xLocation += dx  
        this.yLocation += dy  
    }  
}
```

```
// ...  
  
val player1: Player = new Player(0.0, 0.0, 10)  
val player2: Player = new Player(7.0, -4.0, 10)  
  
player2.move(-6.5, 3.4)  
  
player2.attack(player1)  
player2.attack(player1)  
  
assert(player1.hp == 2)  
  
// ...
```

- Define a class to represent a player in a game
- We'll analyze this code piece by piece

Classes

```
class Player(var xLocation: Double, var yLocation: Double, val maxHitPoints: Int) {  
    var hp: Int = this.maxHitPoints  
    val damageDealt: Int = 4  
  
    def takeDamage(damage: Int): Unit = {  
        this.hp -= damage  
    }  
  
    def attack(otherPlayer: Player): Unit = {  
        otherPlayer.takeDamage(this.damageDealt)  
    }  
  
    def conscious(): Boolean = {  
        this.hp > 0  
    }  
  
    def move(dx: Double, dy: Double): Unit = {  
        this.xLocation += dx  
        this.yLocation += dy  
    }  
}
```

```
// ...  
  
val player1: Player = new Player(0.0, 0.0, 10)  
val player2: Player = new Player(7.0, -4.0, 10)  
  
player2.move(-6.5, 3.4)  
  
player2.attack(player1)  
player2.attack(player1)  
  
assert(player1.hp == 2)  
  
// ...
```

- This class defines several state variables
- Each object of type Player will contain its own copies of each of these variables

Classes

```
class Player(var xLocation: Double, var yLocation: Double, val maxHitPoints: Int) {  
    var hp: Int = this.maxHitPoints  
    val damageDealt: Int = 4  
  
    def takeDamage(damage: Int): Unit = {  
        this.hp -= damage  
    }  
  
    def attack(otherPlayer: Player): Unit = {  
        otherPlayer.takeDamage(this.damageDealt)  
    }  
  
    def conscious(): Boolean = {  
        this.hp > 0  
    }  
  
    def move(dx: Double, dy: Double): Unit = {  
        this.xLocation += dx  
        this.yLocation += dy  
    }  
}
```

```
// ...  
val player1: Player = new Player(0.0, 0.0, 10)  
val player2: Player = new Player(7.0, -4.0, 10)  
  
player2.move(-6.5, 3.4)  
  
player2.attack(player1)  
player2.attack(player1)  
  
assert(player1.hp == 2)  
  
// ...
```

- This class has several methods that define its behaviour
- These methods can be called on each object of type Player

Classes

```
class Player(var xLocation: Double, var yLocation: Double, val maxHitPoints: Int) {  
  
    var hp: Int = this.maxHitPoints  
    val damageDealt: Int = 4  
  
    def takeDamage(damage: Int): Unit = {  
        this.hp -= damage  
    }  
  
    def attack(otherPlayer: Player): Unit = {  
        otherPlayer.takeDamage(this.damageDealt)  
    }  
  
    def conscious(): Boolean = {  
        this.hp > 0  
    }  
  
    def move(dx: Double, dy: Double): Unit = {  
        this.xLocation += dx  
        this.yLocation += dy  
    }  
}
```

```
// ...  
  
val player1: Player = new Player(0.0, 0.0, 10)  
val player2: Player = new Player(7.0, -4.0, 10)  
  
player2.move(-6.5, 3.4)  
  
player2.attack(player1)  
player2.attack(player1)  
  
assert(player1.hp == 2)  
  
// ...
```

- Classes contain a method called a constructor
- This method is called when a new object is created using this class
- Any code calling the constructor can use its parameters to set the initial state of the created object

Classes

```
class Player(var xLocation: Double, var yLocation: Double, val maxHitPoints: Int) {  
    var hp: Int = this.maxHitPoints  
    val damageDealt: Int = 4  
  
    def takeDamage(damage: Int): Unit = {  
        this.hp -= damage  
    }  
  
    def attack(otherPlayer: Player): Unit = {  
        otherPlayer.takeDamage(this.damageDealt)  
    }  
  
    def conscious(): Boolean = {  
        this.hp > 0  
    }  
  
    def move(dx: Double, dy: Double): Unit = {  
        this.xLocation += dx  
        this.yLocation += dy  
    }  
}
```

```
// ...  
val player1: Player = new Player(0.0, 0.0, 10)  
val player2: Player = new Player(7.0, -4.0, 10)  
  
player2.move(-6.5, 3.4)  
  
player2.attack(player1)  
player2.attack(player1)  
  
assert(player1.hp == 2)  
  
// ...
```

- [In Scala] All constructor parameters become state variables
- The constructor parameters can be declared with either val or var
 - If neither val nor var is used, the parameter is a val **and** it cannot be accessed from outside the class

Classes

```
class Player(var xLocation: Double, var yLocation: Double, val maxHitPoints: Int) {  
  
    var hp: Int = this.maxHitPoints  
    val damageDealt: Int = 4  
  
    def takeDamage(damage: Int): Unit = {  
        this.hp -= damage  
    }  
  
    def attack(otherPlayer: Player): Unit = {  
        otherPlayer.takeDamage(this.damageDealt)  
    }  
  
    def conscious(): Boolean = {  
        this.hp > 0  
    }  
  
    def move(dx: Double, dy: Double): Unit = {  
        this.xLocation += dx  
        this.yLocation += dy  
    }  
}
```

```
// ...  
  
val player1: Player = new Player(0.0, 0.0, 10)  
val player2: Player = new Player(7.0, -4.0, 10)  
  
player2.move(-6.5, 3.4)  
  
player2.attack(player1)  
player2.attack(player1)  
  
assert(player1.hp == 2)  
  
// ...
```

- The keyword "this" is a reference to the calling object
- It is used to access the state and behavior of the object through which the method was called

Classes

```
class Player(var xLocation: Double, var yLocation: Double, val maxHitPoints: Int) {  
  
    var hp: Int = this.maxHitPoints  
    val damageDealt: Int = 4  
  
    def takeDamage(damage: Int): Unit = {  
        this.hp -= damage  
    }  
  
    def attack(otherPlayer: Player): Unit = {  
        otherPlayer.takeDamage(this.damageDealt)  
    }  
  
    def conscious(): Boolean = {  
        this.hp > 0  
    }  
  
    def move(dx: Double, dy: Double): Unit = {  
        this.xLocation += dx  
        this.yLocation += dy  
    }  
}
```

```
// ...  
  
val player1: Player = new Player(0.0, 0.0, 10)  
val player2: Player = new Player(7.0, -4.0, 10)  
  
player2.move(-6.5, 3.4)  
  
player2.attack(player1)  
player2.attack(player1)  
  
assert(player1.hp == 2)  
  
// ...
```

- When you write a class, you define a new type
- This type can be used like any other type
 - Variable of this type, methods that take this type as a parameter, etc.

Classes

```
class Player(var xLocation: Double, var yLocation: Double, val maxHitPoints: Int) {  
  
    var hp: Int = this.maxHitPoints  
    val damageDealt: Int = 4  
  
    def takeDamage(damage: Int): Unit = {  
        this.hp -= damage  
    }  
  
    def attack(otherPlayer: Player): Unit = {  
        otherPlayer.takeDamage(this.damageDealt)  
    }  
  
    def conscious(): Boolean = {  
        this.hp > 0  
    }  
  
    def move(dx: Double, dy: Double): Unit = {  
        this.xLocation += dx  
        this.yLocation += dy  
    }  
}
```

```
// ...  
val player1: Player = new Player(0.0, 0.0, 10)  
val player2: Player = new Player(7.0, -4.0, 10)  
  
player2.move(-6.5, 3.4)  
  
player2.attack(player1)  
player2.attack(player1)  
  
assert(player1.hp == 2)  
  
// ...
```

- Use the keyword **new** to call the constructor method
- The constructor creates a new object of this type
- The constructor returns a reference to the new object

Classes

```
class Player(var xLocation: Double, var yLocation: Double, val maxHitPoints: Int) {  
  
    var hp: Int = this.maxHitPoints  
    val damageDealt: Int = 4  
  
    def takeDamage(damage: Int): Unit = {  
        this.hp -= damage  
    }  
  
    def attack(otherPlayer: Player): Unit = {  
        otherPlayer.takeDamage(this.damageDealt)  
    }  
  
    def conscious(): Boolean = {  
        this.hp > 0  
    }  
  
    def move(dx: Double, dy: Double): Unit = {  
        this.xLocation += dx  
        this.yLocation += dy  
    }  
}
```

```
// ...  
val player1: Player = new Player(0.0, 0.0, 10)  
val player2: Player = new Player(7.0, -4.0, 10)  
  
player2.move(-6.5, 3.4)  
player2.attack(player1)  
player2.attack(player1)  
assert(player1.hp == 2)  
  
// ...
```


- Use the references to these objects to access their state and behaviour
- Each object has its own copy of all the state variables
 - Allows player1 and player2 to move independently and have different hp

Classes

- Int, Double, Boolean, List, Array, Map
 - Are all classes
 - We use these classes to create objects

```
var list: List[Int] = List(2, 3, 4)
```

- Create objects by calling the constructor for that class
- List is setup in a way that we don't use **new**
- For our classes we will use the **new** keyword



```
def main(args: Array[String]): Unit = {

  val player1: Player = new Player(0.0, 0.0, 10)
  val player2: Player = new Player(7.0, -4.0, 10)
  val player3: Player = player1

  player2.attack(player1)
}
```

[illegible]

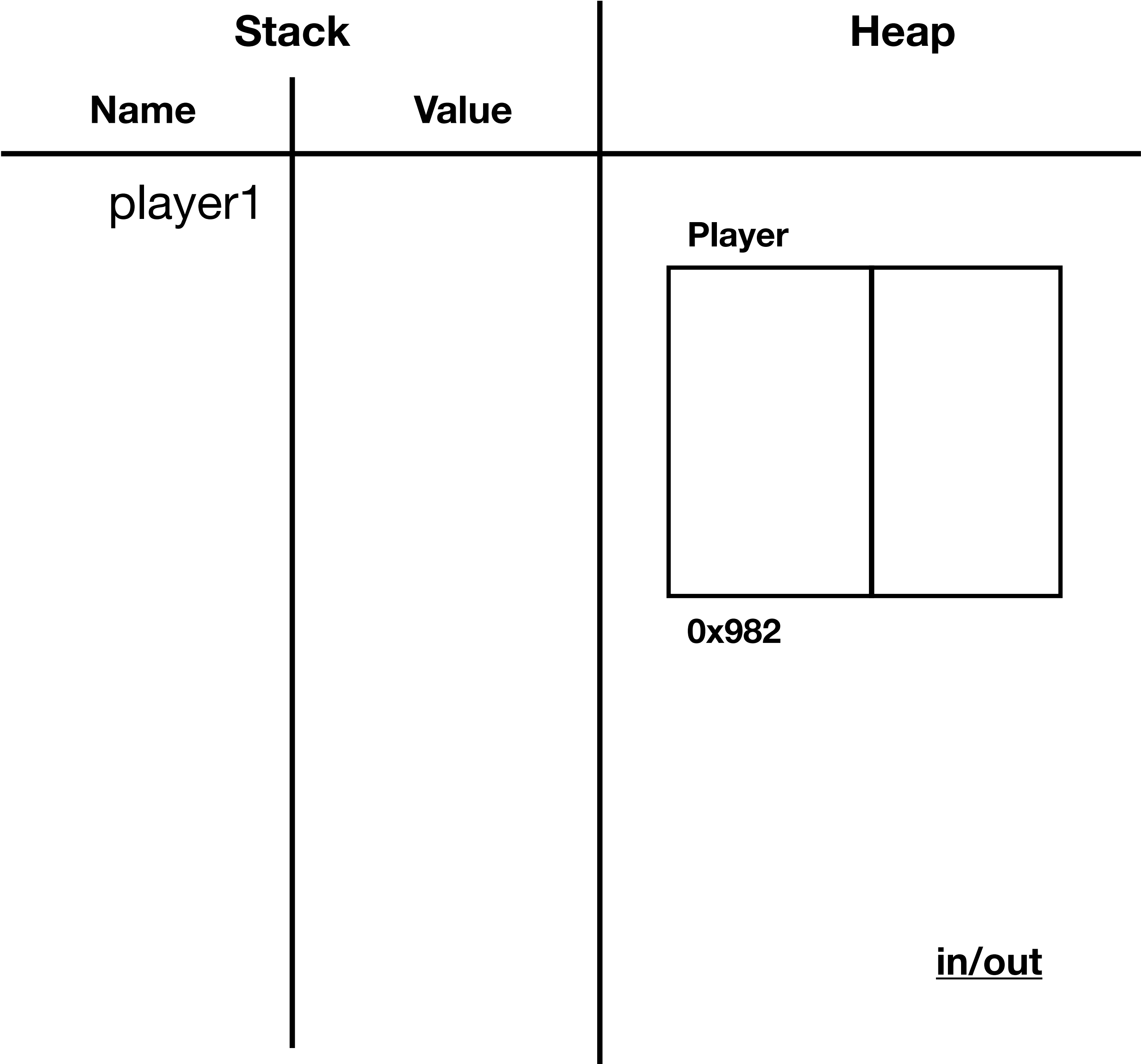
- Let's trace this program in a memory diagram
- [Slight changes to the code]

```
class Player(var x: Double, var y: Double, val maxHP: Int) {  
  
    var hp: Int = this.maxHP  
    val damageDealt: Int = 4  
  
    def takeDamage(damage: Int): Unit = {  
        this.hp -= damage  
    }  
  
    def attack(otherPlayer: Player): Unit = {  
        otherPlayer.takeDamage(this.damageDealt)  
    }  
  
    def move(dx: Double, dy: Double): Unit = {  
        this.x += dx  
        this.y += dy  
    }  
}
```



```
def main(args: Array[String]): Unit = {  
  
    val player1: Player = new Player(0.0, 0.0, 10)  
    val player2: Player = new Player(7.0, -4.0, 10)  
    val player3: Player = player1  
  
    player2.attack(player1)  
}
```

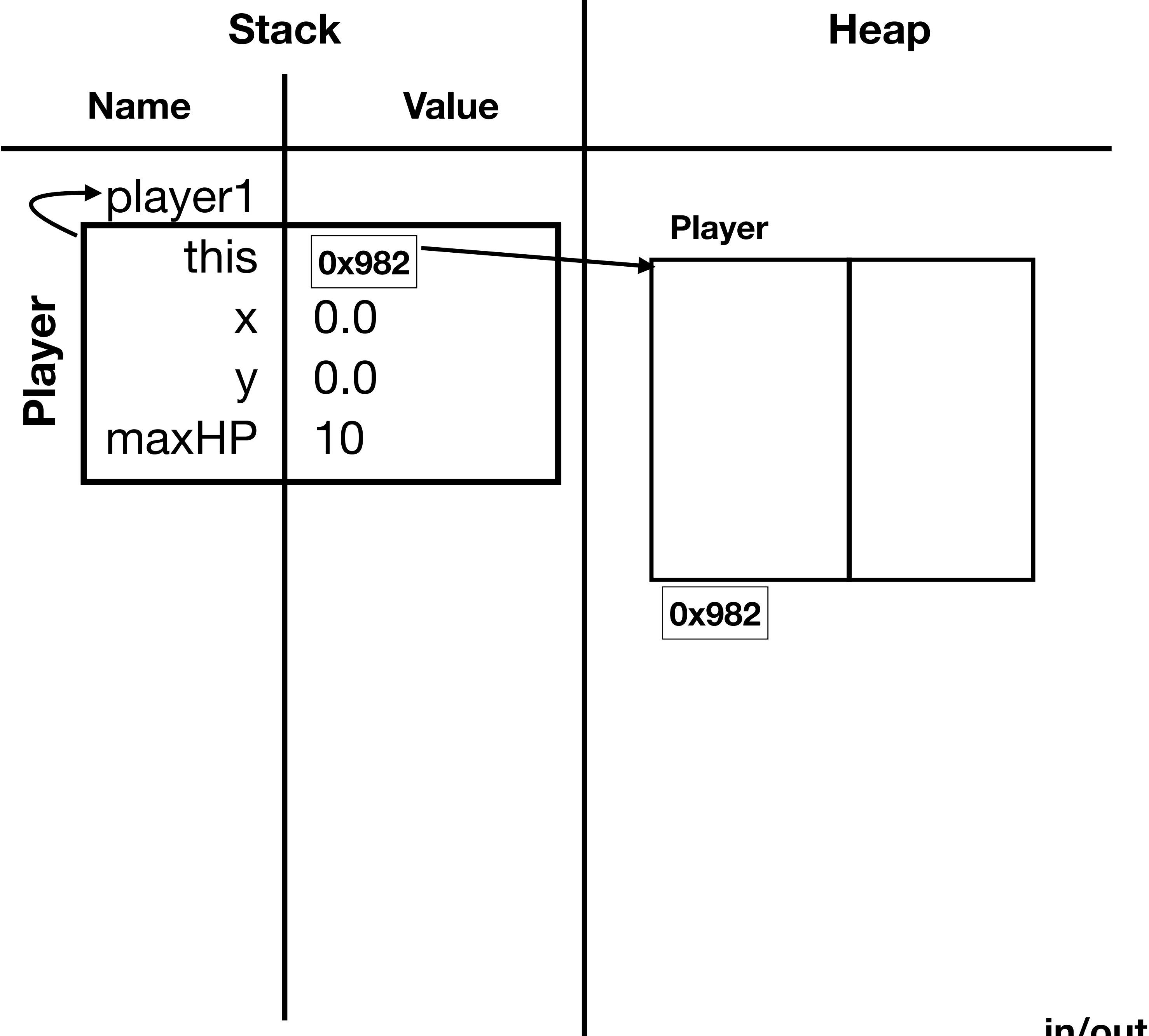
- Every time you see the keyword "new" something is being added to the heap
- Create a new object of type Player




```
class Player(var x: Double, var y: Double, val maxHP: Int) {  
  
    var hp: Int = this.maxHP  
    val damage: Int = 4  
  
    def takeDamage(damage: Int): Unit = {  
        this.hp -= damage  
    }  
  
    def attack(otherPlayer: Player): Unit = {  
        otherPlayer.takeDamage(this.damage)  
    }  
  
    def move(dx: Double, dy: Double): Unit = {  
        this.x += dx  
        this.y += dy  
    }  
}
```

➔

```
def main(args: Array[String]): Unit = {  
  
    val player1: Player = new Player(0.0, 0.0, 10)  
    val player2: Player = new Player(7.0, -4.0, 10)  
    val player3: Player = player1  
  
    player2.attack(player1)  
}
```

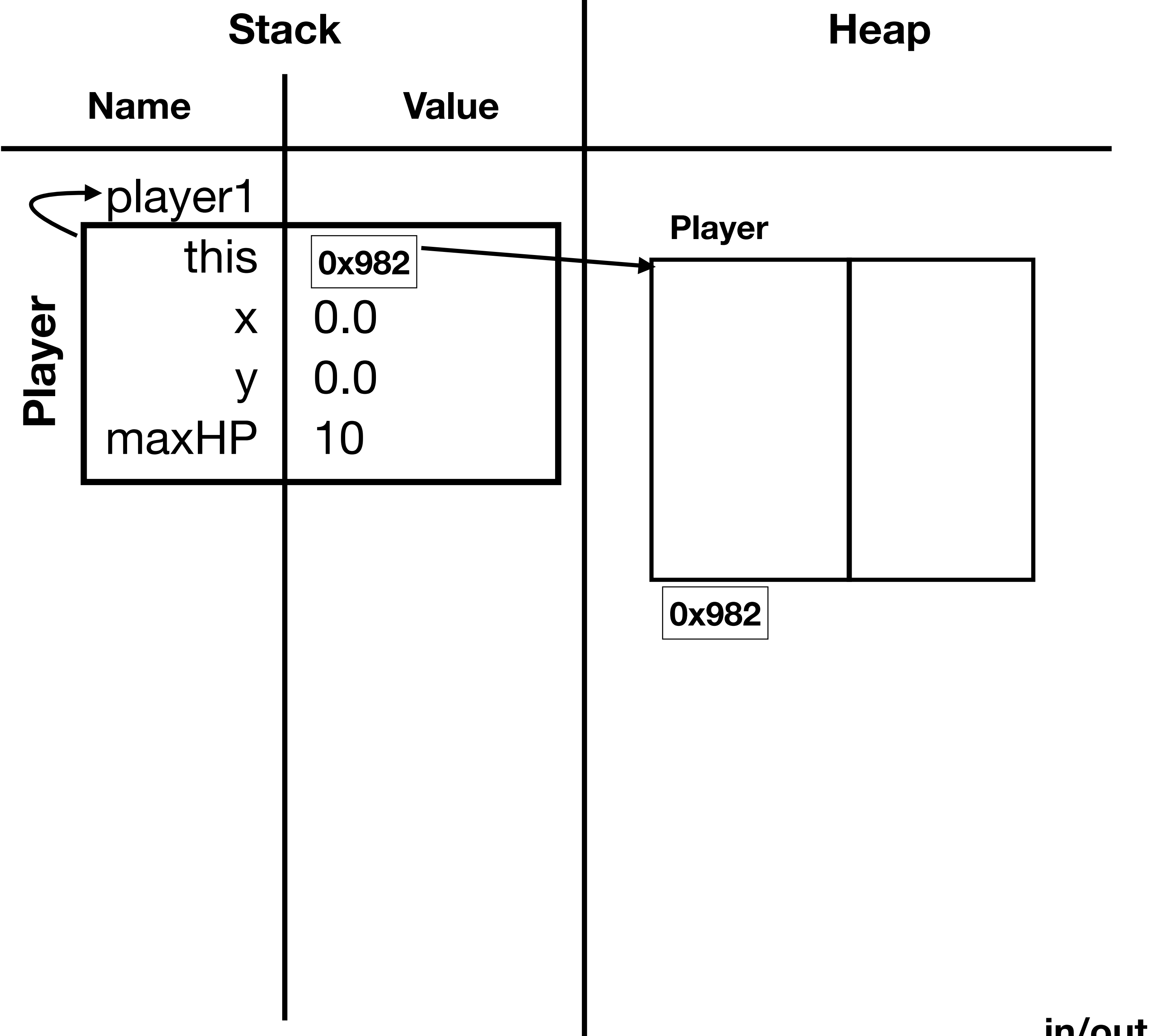


- Constructors are methods!
- Add a stack frame for the constructor call


```
class Player(var x: Double, var y: Double, val maxHP: Int) {  
  
    var hp: Int = this.maxHP  
    val damage: Int = 4  
  
    def takeDamage(damage: Int): Unit = {  
        this.hp -= damage  
    }  
  
    def attack(otherPlayer: Player): Unit = {  
        otherPlayer.takeDamage(this.damage)  
    }  
  
    def move(dx: Double, dy: Double): Unit = {  
        this.x += dx  
        this.y += dy  
    }  
}
```

➔

```
def main(args: Array[String]): Unit = {  
  
    val player1: Player = new Player(0.0, 0.0, 10)  
    val player2: Player = new Player(7.0, -4.0, 10)  
    val player3: Player = player1  
  
    player2.attack(player1)  
}
```



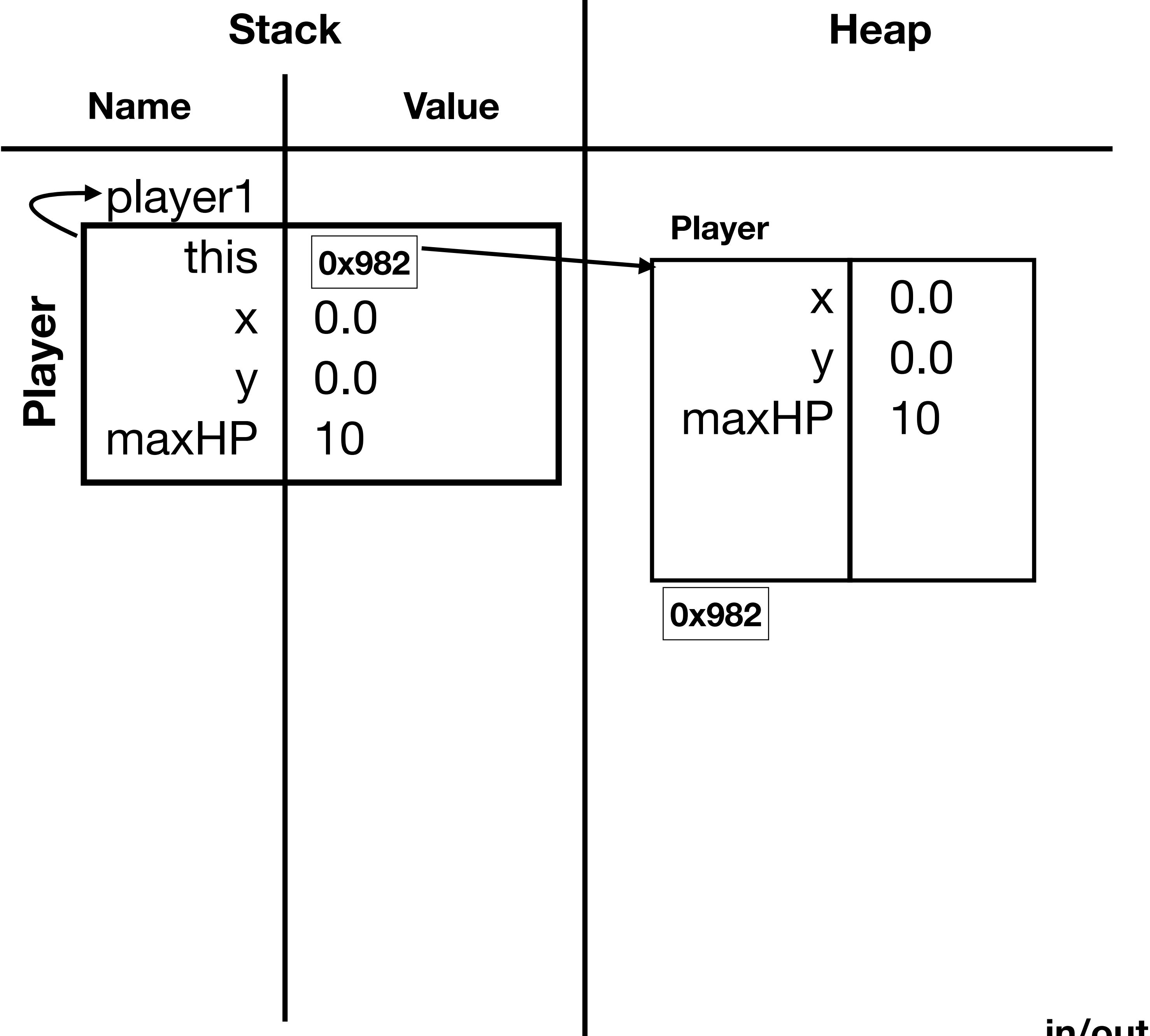
- When calling a method we have an implicit part of "this"
- "this" stores a reference to the calling object (Object being created when the method is a constructor)




```
class Player(var x: Double, var y: Double, val maxHP: Int) {  
    var hp: Int = this.maxHP  
    val damage: Int = 4  
  
    def takeDamage(damage: Int): Unit = {  
        this.hp -= damage  
    }  
  
    def attack(otherPlayer: Player): Unit = {  
        otherPlayer.takeDamage(this.damage)  
    }  
  
    def move(dx: Double, dy: Double): Unit = {  
        this.x += dx  
        this.y += dy  
    }  
}
```

```
def main(args: Array[String]): Unit = {  
    val player1: Player = new Player(0.0, 0.0, 10)  
    val player2: Player = new Player(7.0, -4.0, 10)  
    val player3: Player = player1  
  
    player2.attack(player1)  
}
```

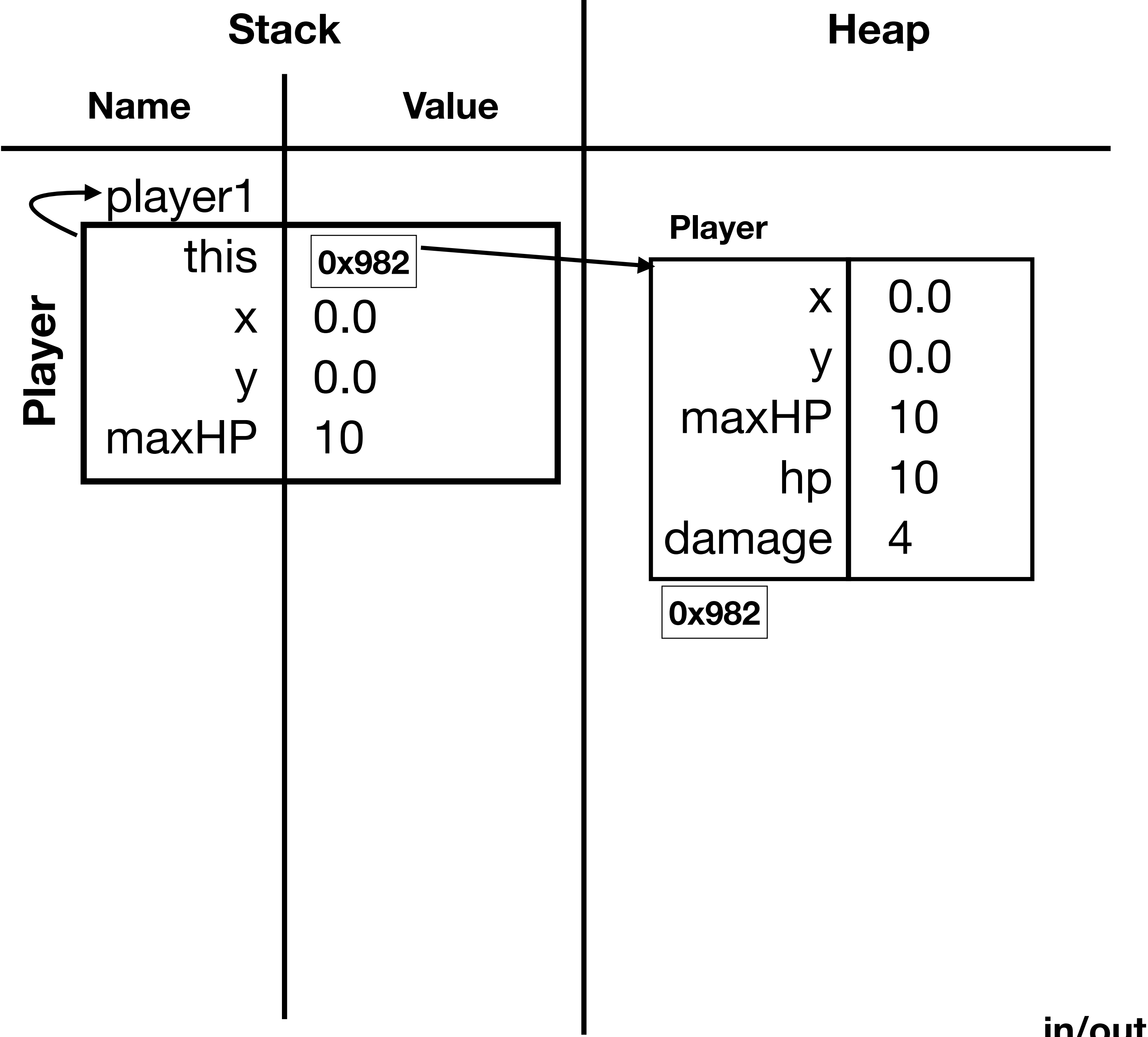
- Constructor parameters become state variables in the object





```
class Player(var x: Double, var y: Double, val maxHP: Int) {  
  
    var hp: Int = this.maxHP  
    val damage: Int = 4  
  
    def takeDamage(damage: Int): Unit = {  
        this.hp -= damage  
    }  
  
    def attack(otherPlayer: Player): Unit = {  
        otherPlayer.takeDamage(this.damage)  
    }  
  
    def move(dx: Double, dy: Double): Unit = {  
        this.x += dx  
        this.y += dy  
    }  
}
```

```
def main(args: Array[String]): Unit = {  
  
    val player1: Player = new Player(0.0, 0.0, 10)  
    val player2: Player = new Player(7.0, -4.0, 10)  
    val player3: Player = player1  
  
    player2.attack(player1)  
}
```



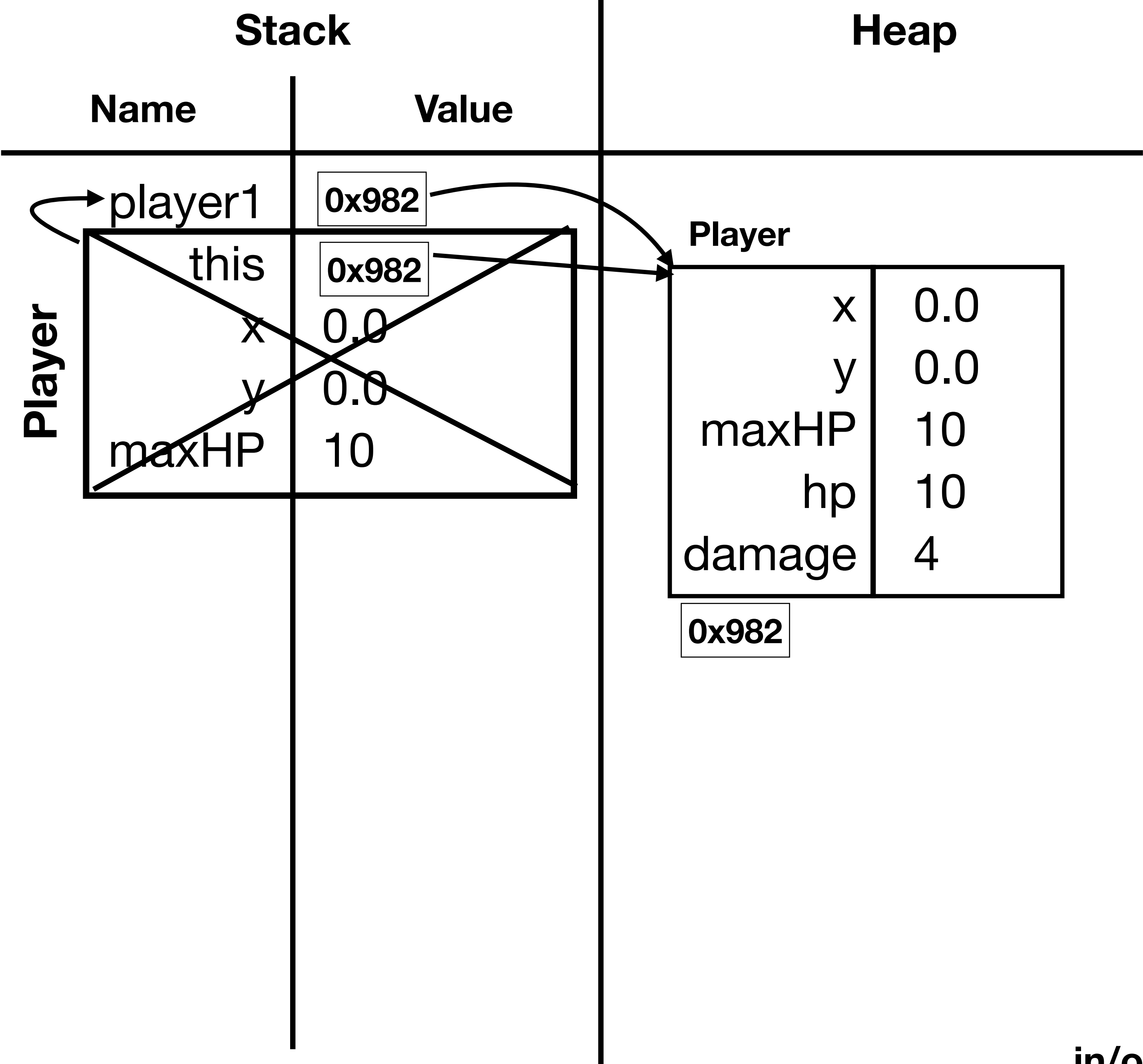
- Run all code that's outside of methods when the constructor is called
- Any variable declared become state variables in the object

```
class Player(var x: Double, var y: Double, val maxHP: Int) {  
  
    var hp: Int = this.maxHP  
    val damage: Int = 4  
  
    def takeDamage(damage: Int): Unit = {  
        this.hp -= damage  
    }  
  
    def attack(otherPlayer: Player): Unit = {  
        otherPlayer.takeDamage(this.damage)  
    }  
  
    def move(dx: Double, dy: Double): Unit = {  
        this.x += dx  
        this.y += dy  
    }  
}
```



```
def main(args: Array[String]): Unit = {  
  
    val player1: Player = new Player(0.0, 0.0, 10)  
    val player2: Player = new Player(7.0, -4.0, 10)  
    val player3: Player = player1  
  
    player2.attack(player1)  
}
```

- Constructor ends and returns a reference to the object that was created

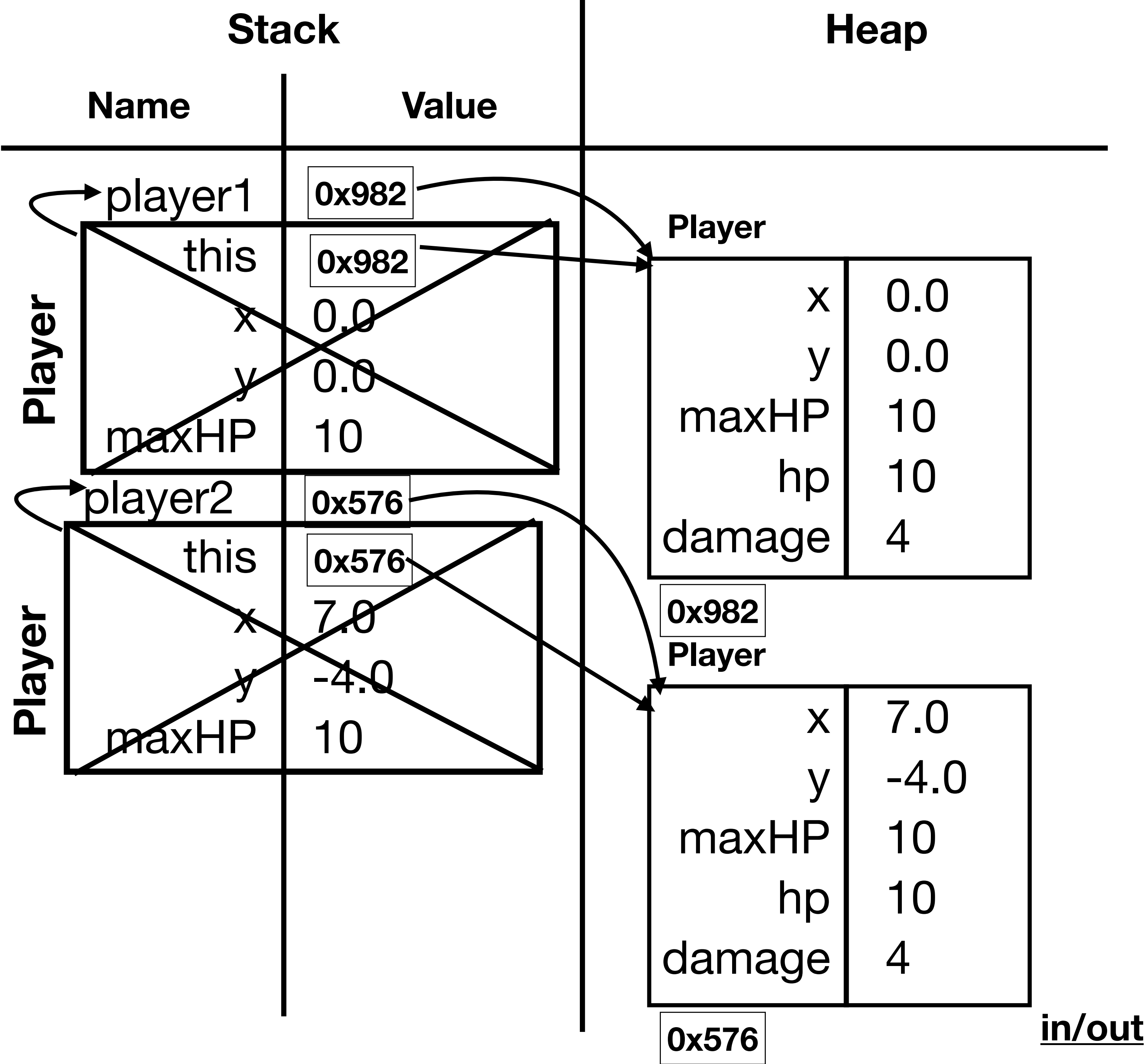


```
class Player(var x: Double, var y: Double, val maxHP: Int) {  
  
  var hp: Int = this.maxHP  
  val damage: Int = 4  
  
  def takeDamage(damage: Int): Unit = {  
    this.hp -= damage  
  }  
  
  def attack(otherPlayer: Player): Unit = {  
    otherPlayer.takeDamage(this.damage)  
  }  
  
  def move(dx: Double, dy: Double): Unit = {  
    this.x += dx  
    this.y += dy  
  }  
}
```



```
def main(args: Array[String]): Unit = {  
  
  val player1: Player = new Player(0.0, 0.0, 10)  
  val player2: Player = new Player(7.0, -4.0, 10)  
  val player3: Player = player1  
  
  player2.attack(player1)  
}
```

- Repeat the process for player2

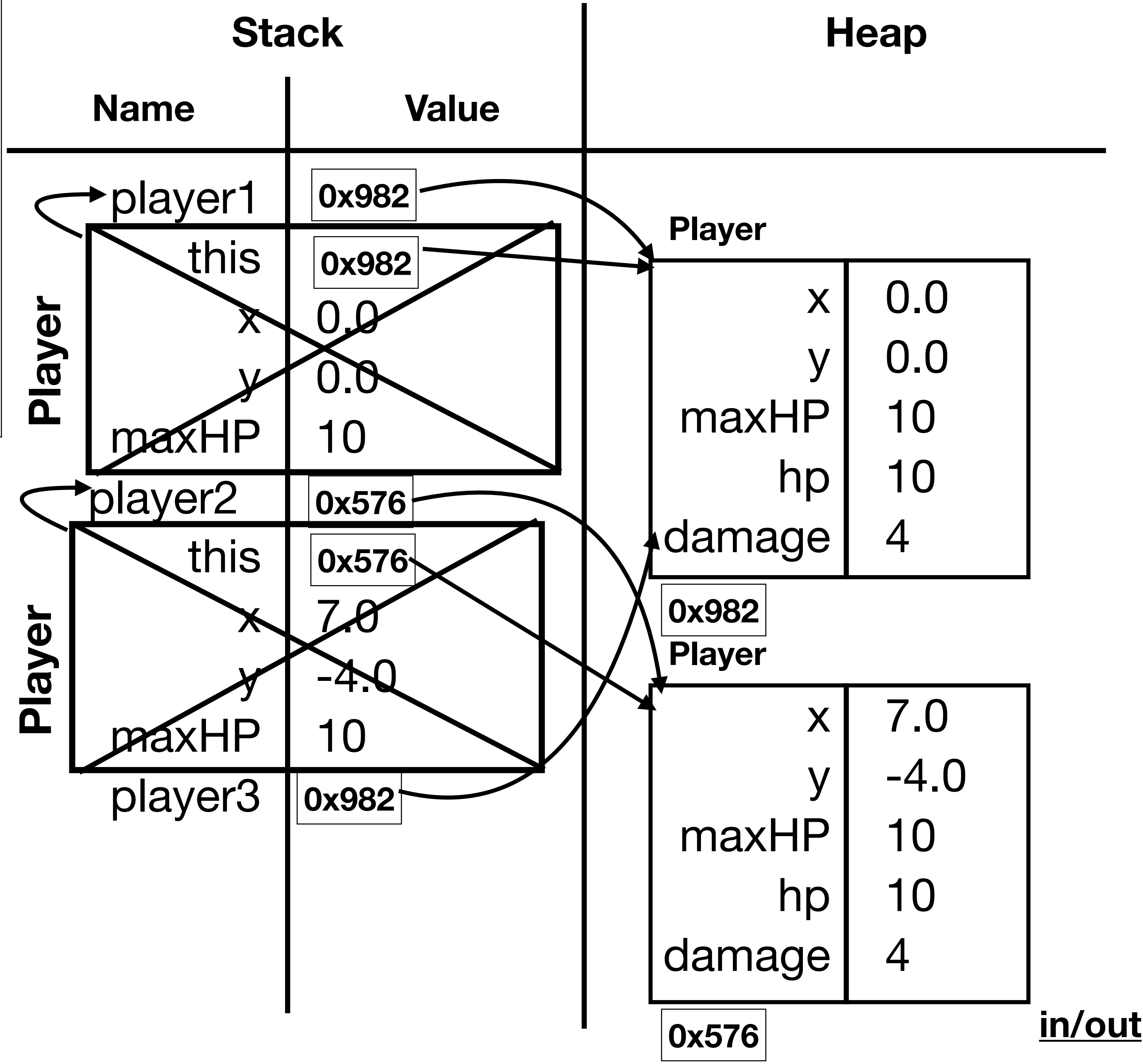


```
class Player(var x: Double, var y: Double, val maxHP: Int) {  
  
    var hp: Int = this.maxHP  
    val damage: Int = 4  
  
    def takeDamage(damage: Int): Unit = {  
        this.hp -= damage  
    }  
  
    def attack(otherPlayer: Player): Unit = {  
        otherPlayer.takeDamage(this.damage)  
    }  
  
    def move(dx: Double, dy: Double): Unit = {  
        this.x += dx  
        this.y += dy  
    }  
}
```

➔

```
def main(args: Array[String]): Unit = {  
    val player1: Player = new Player(0.0, 0.0, 10)  
    val player2: Player = new Player(7.0, -4.0, 10)  
    val player3: Player = player1  
  
    player2.attack(player1)  
}
```

- player3 is assigned the value of player1
- player1's value is a **reference!!**

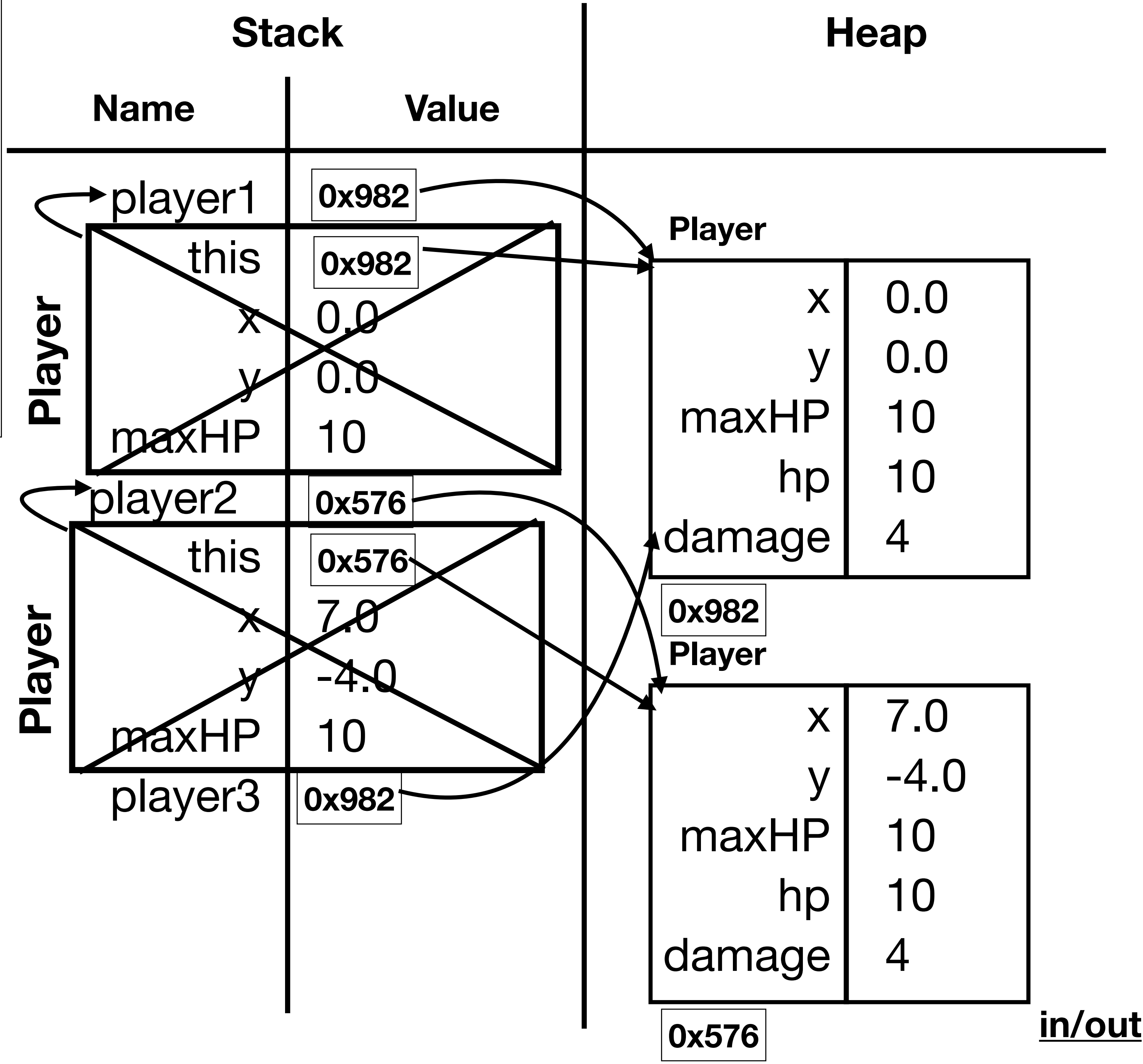



```
class Player(var x: Double, var y: Double, val maxHP: Int) {  
  
  var hp: Int = this.maxHP  
  val damage: Int = 4  
  
  def takeDamage(damage: Int): Unit = {  
    this.hp -= damage  
  }  
  
  def attack(otherPlayer: Player): Unit = {  
    otherPlayer.takeDamage(this.damage)  
  }  
  
  def move(dx: Double, dy: Double): Unit = {  
    this.x += dx  
    this.y += dy  
  }  
}
```

➔

```
def main(args: Array[String]): Unit = {  
  
  val player1: Player = new Player(0.0, 0.0, 10)  
  val player2: Player = new Player(7.0, -4.0, 10)  
  val player3: Player = player1  
  
  player2.attack(player1)  
}
```

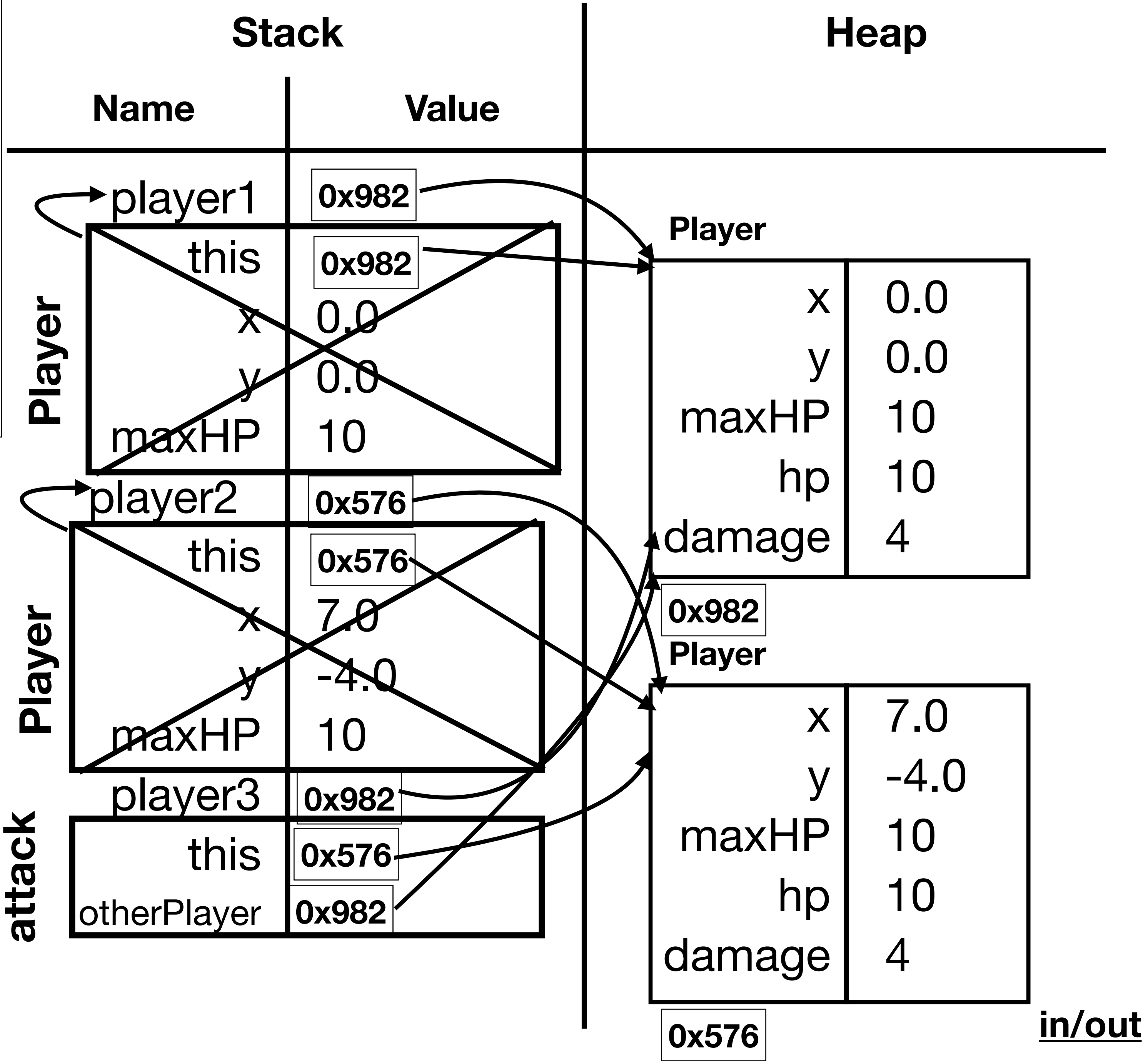
- player3 and player1 refer to the same object!
- No new object was created for player3



```
class Player(var x: Double, var y: Double, val maxHP: Int) {  
  
  var hp: Int = this.maxHP  
  val damage: Int = 4  
  
  def takeDamage(damage: Int): Unit = {  
    this.hp -= damage  
  }  
  
  def attack(otherPlayer: Player): Unit = {  
    otherPlayer.takeDamage(this.damage)  
  }  
  
  def move(dx: Double, dy: Double): Unit = {  
    this.x += dx  
    this.y += dy  
  }  
}
```

```
def main(args: Array[String]): Unit = {  
  
  val player1: Player = new Player(0.0, 0.0, 10)  
  val player2: Player = new Player(7.0, -4.0, 10)  
  val player3: Player = player1  
  
  player2.attack(player1)  
}
```

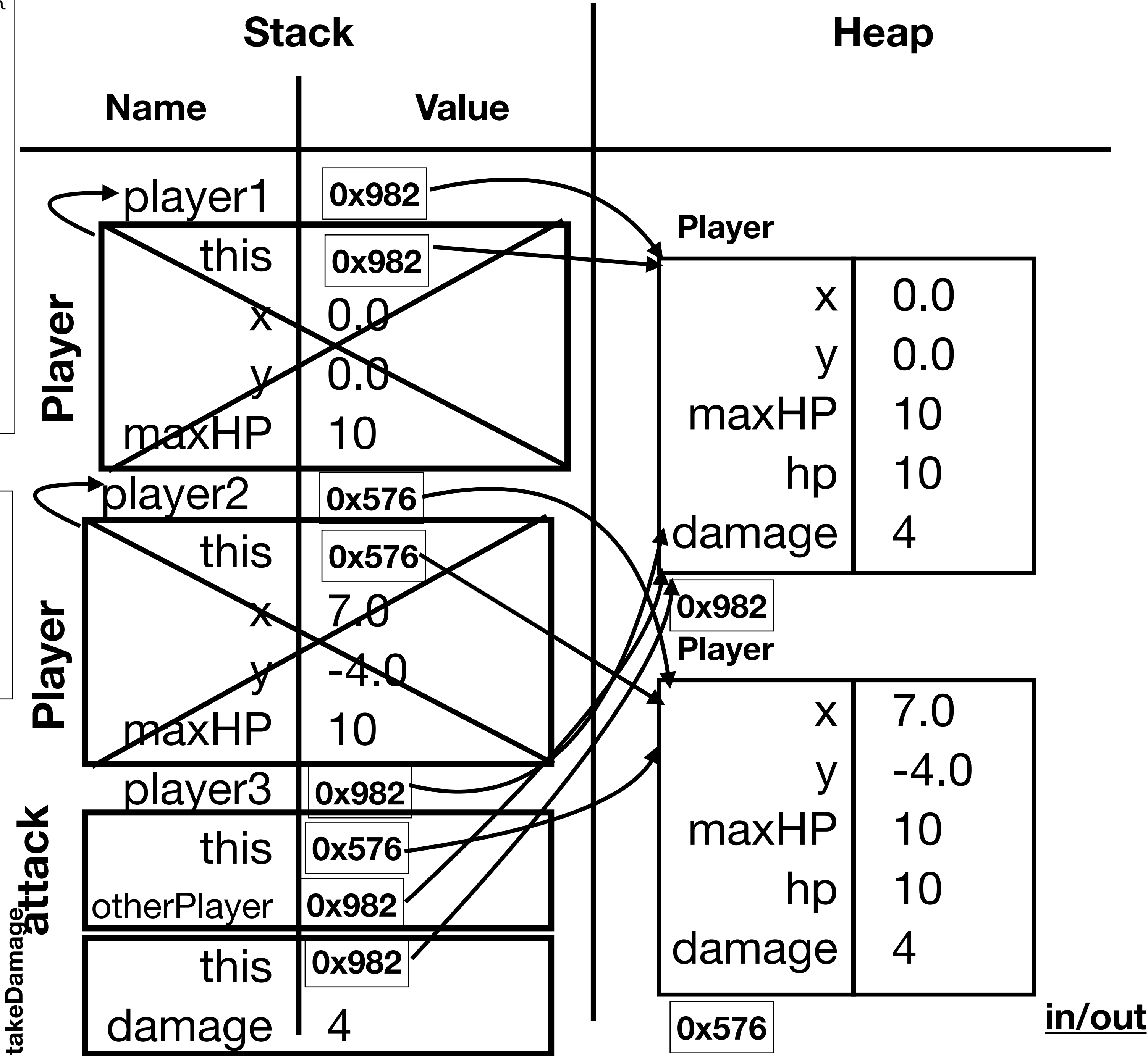
- player2 attacks player1!
- New stack frame
- implicit parameter of this with the reference stored in player2



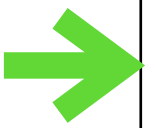

```
class Player(var x: Double, var y: Double, val maxHP: Int) {  
  
    var hp: Int = this.maxHP  
    val damage: Int = 4  
  
    def takeDamage(damage: Int): Unit = {  
        this.hp -= damage  
    }  
  
    def attack(otherPlayer: Player): Unit = {  
        otherPlayer.takeDamage(this.damage)  
    }  
  
    def move(dx: Double, dy: Double): Unit = {  
        this.x += dx  
        this.y += dy  
    }  
}
```

```
def main(args: Array[String]): Unit = {  
  
    val player1: Player = new Player(0.0, 0.0, 10)  
    val player2: Player = new Player(7.0, -4.0, 10)  
    val player3: Player = player1  
  
    player2.attack(player1)  
}
```

- Call takeDamage



```
class Player(var x: Double, var y: Double, val maxHP: Int) {  
    var hp: Int = this.maxHP  
    val damage: Int = 4  
  
    def takeDamage(damage: Int): Unit = {  
        this.hp -= damage  
    }  
  
    def attack(otherPlayer: Player): Unit = {  
        otherPlayer.takeDamage(this.damage)  
    }  
  
    def move(dx: Double, dy: Double): Unit = {  
        this.x += dx  
        this.y += dy  
    }  
}
```



```
def main(args: Array[String]): Unit = {  
    val player1: Player = new Player(0.0, 0.0, 10)  
    val player2: Player = new Player(7.0, -4.0, 10)  
    val player3: Player = player1  
  
    player2.attack(player1)  
}
```

- Decrement hp and return
- player1 and player3 both see the damage

