

Inheritance

Overview

- Suppose we're making a game and we want to various Items that a Player can use
 - A HealthPotion that will heal a Player when used
 - A Weapon that Players can equip
- Note: We won't build this full game, but we will build some of the game mechanics

HealthPotion Class

```
public class HealthPotion{
    private int increase;
    private double xLoc;
    private double yLoc;

    public HealthPotion(double xLoc, double yLoc, int increase){
        this.xLoc=xLoc;
        this.yLoc=yLoc;
        this.increase=increase;
    }
    public void use(){
        System.out.println("Points Healed: " + this.increase);
    }
}
```

- Let's start with the HealthPotion class
- This class has:
 - A constructor that takes three parameters
 - The (x, y) location of the potion as 2 doubles
 - An int representing amount of health restored by consuming the potion
 - A use method that will heal the Player (Simulated with a println for now)

HealthPotion Class

- Cool.. but we already know how to do all that
- What are we learning today?

```
public class HealthPotion{
    private int increase;
    private double xLoc;
    private double yLoc;

    public HealthPotion(double xLoc, double yLoc, int increase){
        this.xLoc=xLoc;
        this.yLoc=yLoc;
        this.increase=increase;
    }
    public void use(){
        System.out.println("Points Healed: " + this.increase);
    }
}
```


Inheritance!

Inheritance

```
public class GameItem {  
    private double xLoc;  
    private double yLoc;  
  
    public GameItem(double xLoc, double yLoc){  
        this.xLoc = xLoc;  
        this.yLoc = yLoc;  
    }  
}
```

```
public class HealthPotion extends GameItem{  
    private int increase;  
  
    public HealthPotion(double xLoc, double yLoc, int increase){  
        super(xLoc,yLoc);  
        this.increase=increase;  
    }  
    public void use(){  
        System.out.println("Points Healed: " + this.increase);  
    }  
}
```

- Instead of defining all of the HealthPotion functionality in its own class:
- Have it inherit from another class
- Let's breakdown how this works

Inheritance

```
public class GameItem {  
    private double xLoc;  
    private double yLoc;  
  
    public GameItem(double xLoc, double yLoc){  
        this.xLoc = xLoc;  
        this.yLoc = yLoc;  
    }  
}
```

```
public class HealthPotion extends GameItem{  
    private int increase;  
  
    public HealthPotion(double xLoc, double yLoc, int increase){  
        super(xLoc,yLoc);  
        this.increase=increase;  
    }  
    public void use(){  
        System.out.println("Points Healed: " + this.increase);  
    }  
}
```

- First, we'll write a separate class, GameItem
- GameItem will contain all the state and behavior common to every item in our game
- Every item will have a location
- Move (x, y) location to GameItem

Inheritance

```
public class GameItem {  
    private double xLoc;  
    private double yLoc;  
  
    public GameItem(double xLoc, double yLoc){  
        this.xLoc = xLoc;  
        this.yLoc = yLoc;  
    }  
}
```

```
public class HealthPotion extends GameItem{  
    private int increase;  
  
    public HealthPotion(double xLoc, double yLoc, int increase){  
        super(xLoc,yLoc);  
        this.increase=increase;  
    }  
    public void use(){  
        System.out.println("Points Healed: " + this.increase);  
    }  
}
```

- Use the **extends** keyword to inherit from another class
- HealthPotion inherits from GameItem
 - or, HealthPotion **extends** GameItem
- HealthPotion *inherits* all the **state** and **behavior** of GameItem

Inheritance

```
public class GameItem {  
    private double xLoc;  
    private double yLoc;  
  
    public GameItem(double xLoc, double yLoc){  
        this.xLoc = xLoc;  
        this.yLoc = yLoc;  
    }  
}
```

```
public class HealthPotion extends GameItem{  
    private int increase;  
  
    public HealthPotion(double xLoc, double yLoc, int increase){  
        super(xLoc,yLoc);  
        this.increase=increase;  
    }  
    public void use(){  
        System.out.println("Points Healed: " + this.increase);  
    }  
}
```

- HealthPotion *inherits* all the **state** and **behavior** of GameItem
- All instance variables declared in GameItem are also instance variables of every HealthPotion
- HealthPotions now have an (x, y) location as part of their state *without* declaring these variables directly

Inheritance

```
public class GameItem {  
    private double xLoc;  
    private double yLoc;  
  
    public GameItem(double xLoc, double yLoc){  
        this.xLoc = xLoc;  
        this.yLoc = yLoc;  
    }  
}
```

```
public class HealthPotion extends GameItem{  
    private int increase;  
  
    public HealthPotion(double xLoc, double yLoc, int increase){  
        super(xLoc,yLoc);  
        this.increase=increase;  
    }  
    public void use(){  
        System.out.println("Points Healed: " + this.increase);  
    }  
}
```

- HealthPotion *inherits* all the **state** and **behavior** of GameItem
- All methods declared in the GameItem class are methods available to every GameItem object
- Only the constructor for now

Inheritance

```
public class GameItem {  
    private double xLoc;  
    private double yLoc;  
  
    public GameItem(double xLoc, double yLoc){  
        this.xLoc = xLoc;  
        this.yLoc = yLoc;  
    }  
}
```

```
public class HealthPotion extends GameItem{  
    private int increase;  
  
    public HealthPotion(double xLoc, double yLoc, int increase){  
        super(xLoc,yLoc);  
        this.increase=increase;  
    }  
    public void use(){  
        System.out.println("Points Healed: " + this.increase);  
    }  
}
```

- Vocab:
- GameItem is the **super** class of HealthPotion
- -or- GameItem is the **parent** class of HealthPotion
- HealthPotion is a **child** class of GameItem

Inheritance

```
public class GameItem {  
    private double xLoc;  
    private double yLoc;  
  
    public GameItem(double xLoc, double yLoc){  
        this.xLoc = xLoc;  
        this.yLoc = yLoc;  
    }  
}
```

```
public class HealthPotion extends GameItem{  
    private int increase;  
  
    public HealthPotion(double xLoc, double yLoc, int increase){  
        super(xLoc,yLoc);  
        this.increase=increase;  
    }  
    public void use(){  
        System.out.println("Points Healed: " + this.increase);  
    }  
}
```

- When calling a classes constructor
 - You must call its super class constructor
 - Use the keyword **super** to access the super class
 - Calling super as a method will call the super class constructor
- If the super class constructor takes parameters, this call must be explicit

Inheritance

```
public class GameItem {  
    private double xLoc;  
    private double yLoc;  
  
    public GameItem(){  
        this.xLoc = 0.0;  
        this.yLoc = 0.0;  
    }  
}
```

```
public class HealthPotion extends GameItem{  
    private int increase;  
  
    public HealthPotion(double xLoc, double yLoc, int increase){  
        this.increase=increase;  
    }  
    public void use(){  
        System.out.println("Points Healed: " + this.increase);  
    }  
}
```

- A special case:
 - *If* the super class has a constructor that takes no parameters, it will be called implicitly
 - The constructor is **still called**, you just don't have to type `super()`

Inheritance

- BUT Y THO?

```
public class GameItem {  
    private double xLoc;  
    private double yLoc;  
  
    public GameItem(double xLoc, double yLoc){  
        this.xLoc = xLoc;  
        this.yLoc = yLoc;  
    }  
}
```

```
public class HealthPotion extends GameItem{  
    private int increase;  
  
    public HealthPotion(double xLoc, double yLoc, int increase){  
        super(xLoc,yLoc);  
        this.increase=increase;  
    }  
    public void use(){  
        System.out.println("Points Healed: " + this.increase);  
    }  
}
```

- Isn't this more work to get the same result?

Inheritance

- BUT Y THO?

```
public class GameItem {  
    private double xLoc;  
    private double yLoc;  
  
    public GameItem(double xLoc, double yLoc){  
        this.xLoc = xLoc;  
        this.yLoc = yLoc;  
    }  
}
```

```
public class Weapon extends GameItem {  
    private int damage;  
  
    public Weapon(double xloc, double yLoc, int damage) {  
        super(xloc, yLoc);  
        this.damage = damage;  
    }  
    public void use() {  
        System.out.println("Damage dealt: " + this.damage);  
    }  
}
```

```
public class HealthPotion extends GameItem {  
    private int increase;  
  
    public HealthPotion(double xLoc, double yLoc, int increase){  
        super(xLoc, yLoc);  
        this.increase=increase;  
    }  
    public void use(){  
        System.out.println("Points Healed: " + this.increase);  
    }  
}
```

- Inheritance is useful when multiple classes extend the same super class
- Every item in our game will extend GameItem
- Every item in our game has an (x, y) location without having to implement extra code (Or cut n' paste code)

Inheritance

- **BUT Y THO?**

- Since the child classes also inherit behavior

- Write a method in GameItem and **ALL** child classes inherit that method

- Convenient with our 2 game items

- Very powerful when there are 100's of game items!

```
public class GameItem {
    private double xLoc;
    private double yLoc;

    public GameItem(double xLoc, double yLoc){
        this.xLoc = xLoc;
        this.yLoc = yLoc;
    }
    public void move(double dx, double dy){
        this.xLoc+=dx;
        this.yLoc+=dy;
    }
}
```

```
public class Weapon extends GameItem {
    private int damage;

    public Weapon(double xloc, double yLoc, int damage) {
        super(xloc, yLoc);
        this.damage = damage;
    }
    public void use() {
        System.out.println("Damage dealt: " + this.damage);
    }
}
```

```
public class HealthPotion extends GameItem{
    private int increase;

    public HealthPotion(double xLoc, double yLoc, int increase){
        super(xLoc,yLoc);
        this.increase=increase;
    }
    public void use(){
        System.out.println("Points Healed: " + this.increase);
    }
}
```


Memory Diagram


```
public class GameItem {
    private double xLoc;
    private double yLoc;

    public GameItem(double xLoc, double yLoc){
        this.xLoc = xLoc;
        this.yLoc = yLoc;
    }
    public void move(double dx, double dy){
        this.xLoc += dx;
        this.yLoc += dy;
    }
}
```

```
public class Weapon extends GameItem {
    private int damage;

    public Weapon(double xloc, double yLoc, int damage) {
        super(xloc, yLoc);
        this.damage = damage;
    }
    public void use() {
        System.out.println("Damage dealt: " + this.damage);
    }
}
```

```
public static void main(String[] args) {
    ➡ Weapon weapon = new Weapon(3.9, -0.5, 10);
    weapon.use();
    weapon.move(-0.5, 1.3);
}
```

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

- As we go through the trace, note:
 - The super class constructor is called when creating an object
 - The method called can be a super class method


```
public class GameItem {
    private double xLoc;
    private double yLoc;

    public GameItem(double xLoc, double yLoc){
        this.xLoc = xLoc;
        this.yLoc = yLoc;
    }
    public void move(double dx, double dy){
        this.xLoc += dx;
        this.yLoc += dy;
    }
}
```

```
public class Weapon extends GameItem {
    private int damage;

    ➡ public Weapon(double xloc, double yLoc, int damage) {
        super(xloc, yLoc);
        this.damage = damage;
    }
    public void use() {
        System.out.println("Damage dealt: " + this.damage);
    }
}
```

```
public static void main(String[] args) {
    ➡ Weapon weapon = new Weapon(3.9, -0.5, 10);
    weapon.use();
    weapon.move(-0.5, 1.3);
}
```



- We start with the main method
- Create a stack frame for the Weapon constructor
- Create the Weapon object on the heap


```
public class GameItem {
    private double xLoc;
    private double yLoc;

    public GameItem(double xLoc, double yLoc){
        this.xLoc = xLoc;
        this.yLoc = yLoc;
    }
    public void move(double dx, double dy){
        this.xLoc += dx;
        this.yLoc += dy;
    }
}
```

```
public class Weapon extends GameItem {
    private int damage;

    ➡ public Weapon(double xloc, double yLoc, int damage) {
        super(xloc, yLoc);
        this.damage = damage;
    }
    public void use() {
        System.out.println("Damage dealt: " + this.damage);
    }
}
```

```
public static void main(String[] args) {
    ➡ Weapon weapon = new Weapon(3.9, -0.5, 10);
    weapon.use();
    weapon.move(-0.5, 1.3);
}
```



- Since `Weapon` extends `GameItem`
- `Weapon` *inherits* all of `GameItem`'s instance variables
- `xLoc` and `yLoc` are instance variables of every `Weapon` object


```
public class GameItem {
    private double xLoc;
    private double yLoc;

    ➡ public GameItem(double xLoc, double yLoc){
        this.xLoc = xLoc;
        this.yLoc = yLoc;
    }
    public void move(double dx, double dy){
        this.xLoc += dx;
        this.yLoc += dy;
    }
}
```

```
public class Weapon extends GameItem {
    private int damage;

    ➡ public Weapon(double xloc, double yLoc, int damage) {
        super(xloc, yLoc);
        this.damage = damage;
    }
    public void use() {
        System.out.println("Damage dealt: " + this.damage);
    }
}
```

```
public static void main(String[] args) {
    ➡ Weapon weapon = new Weapon(3.9, -0.5, 10);
    weapon.use();
    weapon.move(-0.5, 1.3);
}
```



- The super class constructor is called
- This creates another stack frame
- this is still a reference to the object that's being created


```
public class GameItem {
    private double xLoc;
    private double yLoc;

    public GameItem(double xLoc, double yLoc){
        this.xLoc = xLoc;
        this.yLoc = yLoc;
    }
    public void move(double dx, double dy){
        this.xLoc += dx;
        this.yLoc += dy;
    }
}
```

```
public class Weapon extends GameItem {
    private int damage;

    public Weapon(double xloc, double yLoc, int damage) {
        super(xloc, yLoc);
        this.damage = damage;
    }
    public void use() {
        System.out.println("Damage dealt: " + this.damage);
    }
}
```

```
public static void main(String[] args) {
    Weapon weapon = new Weapon(3.9, -0.5, 10);
    weapon.use();
    weapon.move(-0.5, 1.3);
}
```



- Execute all the super class constructor code


```
public class GameItem {
    private double xLoc;
    private double yLoc;

    public GameItem(double xLoc, double yLoc){
        this.xLoc = xLoc;
        this.yLoc = yLoc;
    }
    public void move(double dx, double dy){
        this.xLoc += dx;
        this.yLoc += dy;
    }
}
```

```
public class Weapon extends GameItem {
    private int damage;

    public Weapon(double xloc, double yLoc, int damage) {
        super(xloc, yLoc);
        this.damage = damage;
    }
    public void use() {
        System.out.println("Damage dealt: " + this.damage);
    }
}
```

```
public static void main(String[] args) {
    Weapon weapon = new Weapon(3.9, -0.5, 10);
    weapon.use();
    weapon.move(-0.5, 1.3);
}
```



- Super class constructor frame is removed from memory
- Run the code in the child class constructor


```
public class GameItem {
    private double xLoc;
    private double yLoc;

    public GameItem(double xLoc, double yLoc){
        this.xLoc = xLoc;
        this.yLoc = yLoc;
    }
    public void move(double dx, double dy){
        this.xLoc += dx;
        this.yLoc += dy;
    }
}
```

```
public class Weapon extends GameItem {
    private int damage;

    public Weapon(double xloc, double yLoc, int damage) {
        super(xloc, yLoc);
        this.damage = damage;
    }
    public void use() {
        System.out.println("Damage dealt: " + this.damage);
    }
}
```

```
public static void main(String[] args) {
    ➡ Weapon weapon = new Weapon(3.9, -0.5, 10);
    weapon.use();
    weapon.move(-0.5, 1.3);
}
```



- Child class constructor returns a reference to the object that was created
- Do not forget to write the super class constructor stack frame on your memory diagrams! <-- Very common mistake

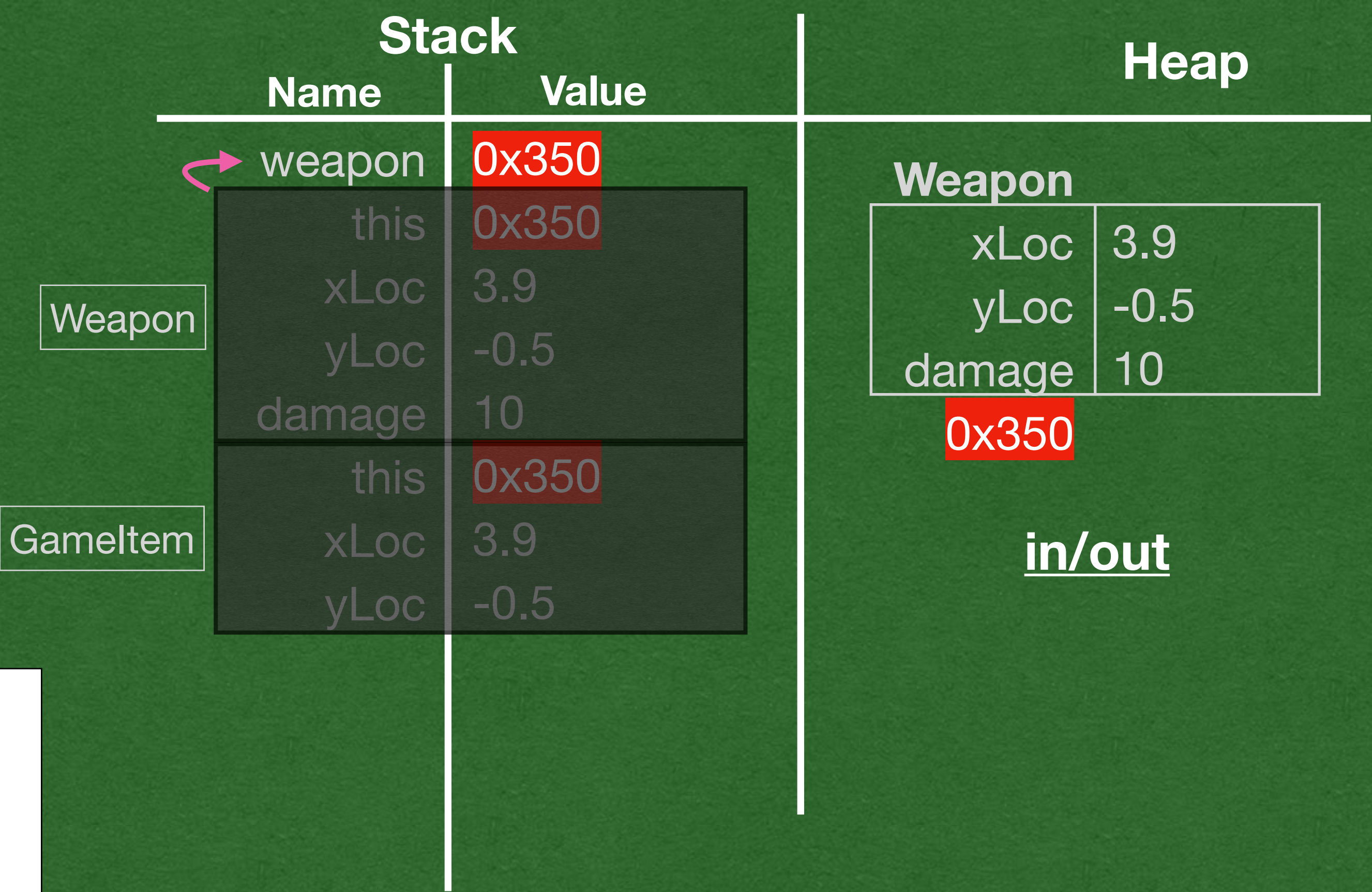

```
public class GameItem {
    private double xLoc;
    private double yLoc;

    public GameItem(double xLoc, double yLoc){
        this.xLoc = xLoc;
        this.yLoc = yLoc;
    }
    public void move(double dx, double dy){
        this.xLoc += dx;
        this.yLoc += dy;
    }
}
```

```
public class Weapon extends GameItem {
    private int damage;

    public Weapon(double xloc, double yLoc, int damage) {
        super(xloc, yLoc);
        this.damage = damage;
    }
    public void use() {
        System.out.println("Damage dealt: " + this.damage);
    }
}
```

```
public static void main(String[] args) {
    Weapon weapon = new Weapon(3.9, -0.5, 10);
    ➡ weapon.use();
    weapon.move(-0.5, 1.3);
}
```



- When a method is called:
 - Look in the class matching the type of the **object**
 - Find a method with the name of the method being called


```
public class GameItem {
    private double xLoc;
    private double yLoc;

    public GameItem(double xLoc, double yLoc){
        this.xLoc = xLoc;
        this.yLoc = yLoc;
    }
    public void move(double dx, double dy){
        this.xLoc += dx;
        this.yLoc += dy;
    }
}
```

```
public class Weapon extends GameItem {
    private int damage;

    public Weapon(double xloc, double yLoc, int damage) {
        super(xloc, yLoc);
        this.damage = damage;
    }
    ➡ public void use() {
        System.out.println("Damage dealt: " + this.damage);
    }
}
```

```
public static void main(String[] args) {
    ➡ Weapon weapon = new Weapon(3.9, -0.5, 10);
    weapon.use();
    weapon.move(-0.5, 1.3);
}
```



- "use" is called through an object of type Weapon
- Look in the Weapon class
- Find a method named use and call it


```
public class GameItem {
    private double xLoc;
    private double yLoc;

    public GameItem(double xLoc, double yLoc){
        this.xLoc = xLoc;
        this.yLoc = yLoc;
    }
    public void move(double dx, double dy){
        this.xLoc += dx;
        this.yLoc += dy;
    }
}
```

```
public class Weapon extends GameItem {
    private int damage;

    public Weapon(double xloc, double yLoc, int damage) {
        super(xloc, yLoc);
        this.damage = damage;
    }
    public void use() {
        System.out.println("Damage dealt: " + this.damage);
    }
}
```

```
public static void main(String[] args) {
    Weapon weapon = new Weapon(3.9, -0.5, 10);
    ➡ weapon.use();
    weapon.move(-0.5, 1.3);
}
```



- The method prints to the screen and returns


```
public class GameItem {
    private double xLoc;
    private double yLoc;

    public GameItem(double xLoc, double yLoc){
        this.xLoc = xLoc;
        this.yLoc = yLoc;
    }
    public void move(double dx, double dy){
        this.xLoc += dx;
        this.yLoc += dy;
    }
}
```

```
public class Weapon extends GameItem {
    private int damage;

    public Weapon(double xloc, double yLoc, int damage) {
        super(xloc, yLoc);
        this.damage = damage;
    }
    public void use() {
        System.out.println("Damage dealt: " + this.damage);
    }
}
```

```
public static void main(String[] args) {
    Weapon weapon = new Weapon(3.9, -0.5, 10);
    weapon.use();
    ➡ weapon.move(-0.5, 1.3);
}
```



- When move is called:
- Look for a method named move in the Weapon class
- .. but we don't find one




```
public class GameItem {
    private double xLoc;
    private double yLoc;

    public GameItem(double xLoc, double yLoc){
        this.xLoc = xLoc;
        this.yLoc = yLoc;
    }
    public void move(double dx, double dy){
        this.xLoc += dx;
        this.yLoc += dy;
    }
}
```

```
public class Weapon extends GameItem {
    private int damage;

    public Weapon(double xloc, double yLoc, int damage) {
        super(xloc, yLoc);
        this.damage = damage;
    }
    public void use() {
        System.out.println("Damage dealt: " + this.damage);
    }
}
```

```
public static void main(String[] args) {
    Weapon weapon = new Weapon(3.9, -0.5, 10);
    weapon.use();
    weapon.move(-0.5, 1.3);
}
```



- If a method is not found, check in the super class
- Weapon inherited the methods defined in GameItem
 - "move" is part of the Weapon class and can be called from objects of type Weapon


```
public class GameItem {
    private double xLoc;
    private double yLoc;

    public GameItem(double xLoc, double yLoc){
        this.xLoc = xLoc;
        this.yLoc = yLoc;
    }
    public void move(double dx, double dy){
        this.xLoc += dx;
        this.yLoc += dy;
    }
}
```

```
public class Weapon extends GameItem {
    private int damage;

    public Weapon(double xloc, double yLoc, int damage) {
        super(xloc, yLoc);
        this.damage = damage;
    }
    public void use() {
        System.out.println("Damage dealt: " + this.damage);
    }
}
```

```
public static void main(String[] args) {
    Weapon weapon = new Weapon(3.9, -0.5, 10);
    weapon.use();
    weapon.move(-0.5, 1.3);
}
```



- this is still a reference to the Weapon object
- We call a GameItem method where this refers to a Weapon!


```
public class GameItem {
    private double xLoc;
    private double yLoc;

    public GameItem(double xLoc, double yLoc){
        this.xLoc = xLoc;
        this.yLoc = yLoc;
    }
    public void move(double dx, double dy){
        this.xLoc += dx;
        this.yLoc += dy;
    }
}
```

```
public class Weapon extends GameItem {
    private int damage;

    public Weapon(double xloc, double yLoc, int damage) {
        super(xloc, yLoc);
        this.damage = damage;
    }
    public void use() {
        System.out.println("Damage dealt: " + this.damage);
    }
}
```

```
public static void main(String[] args) {
    Weapon weapon = new Weapon(3.9, -0.5, 10);
    weapon.use();
    weapon.move(-0.5, 1.3);
}
```



- End program

Object

The Object Class

```
public class GameItem {  
    private double xLoc;  
    private double yLoc;  
  
    public GameItem(double xLoc, double yLoc){  
        this.xLoc=xLoc;  
        this.yLoc=yLoc;  
    }  
    public void move(double dx, double dy){  
        this.xLoc+=dx;  
        this.yLoc+=dy;  
    }  
}
```

```
public class GameItem extends Object {  
    private double xLoc;  
    private double yLoc;  
  
    public GameItem(double xLoc, double yLoc) {  
        this.xLoc = xLoc;  
        this.yLoc = yLoc;  
    }  
    public void move(double dx, double dy) {  
        this.xLoc += dx;  
        this.yLoc += dy;  
    }  
}
```

- If your class does not explicitly extend any super class
- It will implicitly extend the Object class
- These two classes are identical

The Object Class

```
public class GameItem {  
    private double xLoc;  
    private double yLoc;  
  
    public GameItem(double xLoc, double yLoc){  
        this.xLoc=xLoc;  
        this.yLoc=yLoc;  
    }  
    public void move(double dx, double dy){  
        this.xLoc+=dx;  
        this.yLoc+=dy;  
    }  
}
```

```
public class GameItem extends Object {  
    private double xLoc;  
    private double yLoc;  
  
    public GameItem(double xLoc, double yLoc) {  
        this.xLoc = xLoc;  
        this.yLoc = yLoc;  
    }  
    public void move(double dx, double dy) {  
        this.xLoc += dx;  
        this.yLoc += dy;  
    }  
}
```

- The Object class contains several useful methods
 - toString
 - equals
- Every class in Java extends Objects either directly or indirectly
 - Weapon extends GameItem which extends Object
- Every object in Java has a toString and equals method that it inherited from Object