JavaScript Class Inheritance

Learning Objectives

- Understand what class inheritance is
- Understand how to implement inheritance in JavaScript

Material

Inheritance

Often, we'll find that our system contains classes which share a lot of the same functionality. Let's imagine we're writing the code for a video game. Our video game might have two types of characters - knights and dragons:

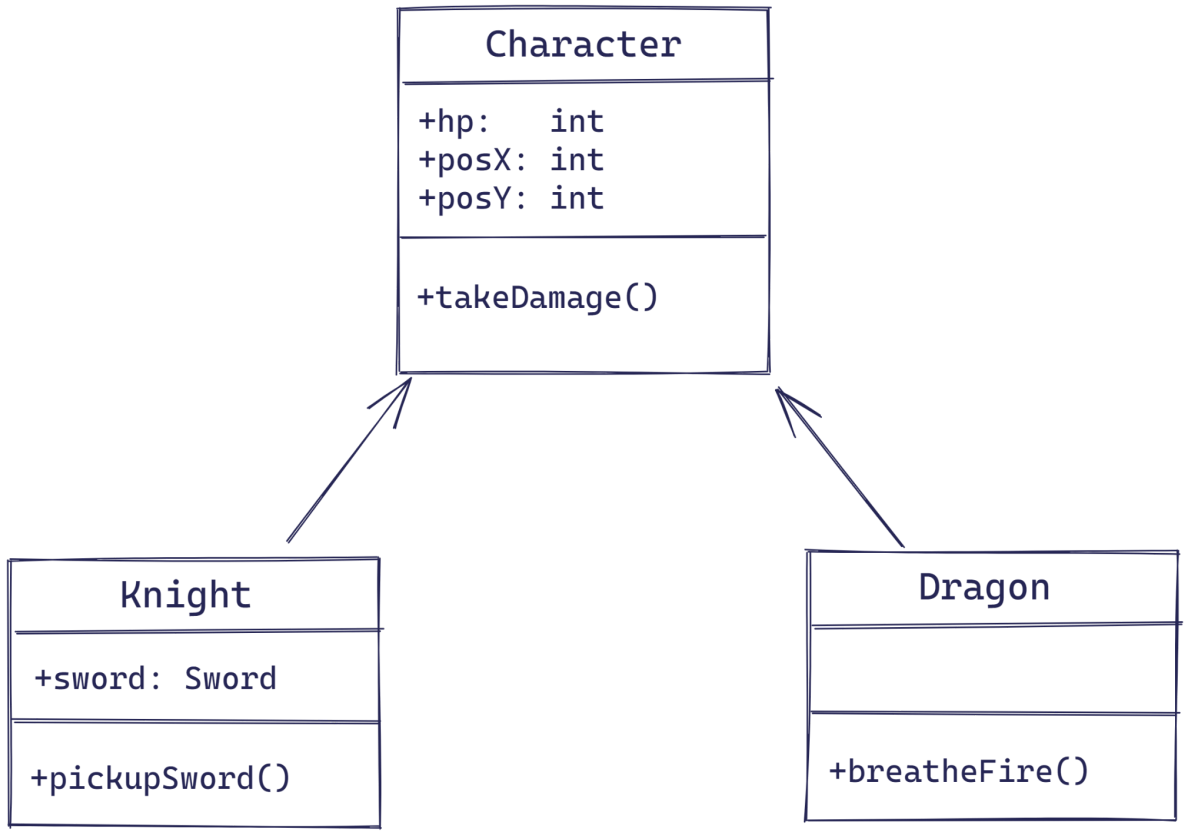
```
// knight.js
class Knight {
    constructor(x, y, sword) {
        this.hp = 100;
        this.posX = x;
        this.posY = y;
        this.sword = sword;
    }
    takeDamage(damage) {
        this.hp -= damage;
    }
    pickupSword(sword) {
        this.sword = sword;
    }
}
```

```
// dragon.js
class Dragon {
    constructor(x, y) {
        this.hp = 100;
        this.posX = x;
        this.posY = y;
    }
    takeDamage(damage) {
        this.hp -= damage;
    }
```

```
    breatheFire(enemy) {
        enemy.burn();
    }
}
```

`Dragon` and `Knight` are not exactly the same but they do share a lot of the same code. This is bad: if we decided we wanted to change our game's system for hp or x-y coordinates, we'd have to change it in both `Dragon`, `Knight` and any other characters we create.

Fortunately, we can use inheritance to allow these two classes to share code. We can create a base class called `Character` which contains only the properties which `Knight` and `Dragon` share. In our sub-classes, we then only need to add what they have on top of the base class.




```
// character.js
class Character {
    constructor(x, y) {
        this.hp = 100;
        this.posX = x;
        this.posY = y;
```

```
    }
    takeDamage(damage) {
        this.hp -= damage;
    }
}

module.exports = Character;
```

We can make a class inherit from another class using the `extends` keyword. `Dragon` is exactly the same as `Character` but with an added `breatheFire` method so we only need to add that.

```
// dragon.js
const Character = require("./character");

class Dragon extends Character {
    breatheFire(enemy) {
        enemy.burn();
    }
}
```

`Knight` also adds a new `sword` property so we need to overwrite the `Character`'s constructor. We create a new constructor and, before we add `sword`, we call the base class' constructor. We do this using the super() function, passing in the two values it uses.

```
// knight.js
const Character = require("./character");

class Knight extends Character {
    constructor(x, y, sword) {
        super(x, y);
        this.sword = sword;
    }
    pickupSword(sword) {
        this.sword = sword;
    }
}
```

Inheritance vs Composition

Inheritance is when one class is a specialised version of a more general class:

- A dragon **is** a character

- A house **is** a building
- A novel **is** a book

Composition is where a class is composed of other classes:

- A dragon **has** wings
- A house **has** rooms
- A novel **has** pages

If we made a `Novel` class, it might have a `this.pages` array of `Page` objects; however, it would **not** make sense for Novel to inherit from Page or vice versa.

Core Assignments

Common Ancestor

Pick your two favourite animals. You're going to create classes for them. Decide on some properties and methods they should share, and some they should not. Create a base `Animal` class with the shared properties and methods and create another two classes for the other animals that extend it.

Write tests for your code using jest.