**Урок №14. Введение в ООП**

**Вводный блок (5 минут)**

**Цели и задачи урока:**

1. Рассмотреть принципы построения объектов с использованием ООП.
2. Создать класс «Главный персонаж».
3. Разработать методы класса «Главный персонаж».

**Познание нового (10 минут)**

Объектно-ориентированное программирование - подход в разработке, где все построение программ строится на объектах. До этого мы пытались создавать объекты и брать оттуда данные при необходимости.

В ООП же все построение идет внутри классов и для экземпляров класса происходит работа с методами и свойствами внутри этого класса.

Класс - набор свойств и методов. Это может быть монетка, может быть персонаж и так далее.

На примере любого персонажа можно понять, что является для него свойствами, а что методами.

У каждого персонажа есть его размеры, координаты и какие-то особые черты (изображения, специфические свойства и тд). Это все является свойствами. Что может делать персонаж? Передвигаться, прыгать и т.д. Это является методами класса.

При этом, когда мы работаем с экземпляром класса, то мы вызываем только методы работы с ним, а как они устроены, зачастую, мы не знаем, если пользуемся сторонними библиотеками.

Так же и тут, могут быть однотипные объекты и чтобы для каждого не прописывать несколько методов работы с ними, мы можем создать один класс и потом вызывать методы. Это называется инкапсуляцией, когда все реализация идет внутри класса.

**Работа над проектом (30 минут)**

Тема: Создание нового проекта и написание программы, в которой создается игровой персонаж с использованием парадигмы ООП.

Создадим персонажа у которого есть следующие свойства:

Координаты

Размеры

Спрайт

При создании класса сначала пишется ключевое слово class а затем наименование этого класса

class mainHero {

}

Мы определились со свойствами, которые есть у этого класса, а их сначала нужно инициализировать. Для этого создается конструктор класса:

class mainHero {

**constructor(x, y, w, h, image){**

**}**

}

Внутри конструктора необходимо инициализировать все передаваемые переменные в конструктор:

class mainHero {

constructor(x, y, w, h, image) {

this.x = x;

this.y = y;

this.w = w;

this.h = h;

this.image = new Image();

this.image.src = image;

}

}

Для инициализации переменных внутри класса используется ключевое слово this, которое ссылается на созданный объект и переменные внутри него.

Создадим экземпляр этого класса и попробуем туда передать данные для генерации

const canvas = document.querySelector('canvas');

const context = canvas.getContext('2d');

class mainHero {

constructor(x, y, w, h, image) {

this.x = x;

this.y = y;

this.w = w;

this.h = h;

this.image = new Image();

this.image.src = image;

}

}

**const hero = new mainHero(100, 100, 100, 100, "player.png");**

В последней строке происходит создание персонажа и делается это с помощью конструкции new (новый объект). Таких объектов на карте может быть сколько угодно.

Далее, обычно, мы отображали на экране созданного персонажа. По идее, это тоже метод класса и обычно для отображения прописывают метод с названием draw. Название может быть и другим, просто именно это наименование прижилось и используется обычно для отображения. Плюс заключается в том, что если у всех классов будет одно и тоже наименование метода, то запутаться будет куда сложнее.

Создадим метод draw внутри класса mainHero:

class mainHero {

constructor(x, y, w, h, image) {

this.x = x;

this.y = y;

this.w = w;

this.h = h;

this.image = new Image();

this.image.src = image;

}

**draw(context) {**

**context.drawImage(this.image, this.x, this.y, this.w, this.h)**

**}**

}

Важный момент для метода draw, в него нужно передавать контекст холста, с которым мы работаем, тогда в функции move для покадрового отображения все будет выглядеть следующим образом:

const hero = new mainHero(100, 100, 100, 100, "player.png");

**function move() {**

**context.clearRect(0, 0, canvas.width, canvas.height)**

**hero.draw(context)**

**requestAnimationFrame(move);**

**}**

**move();**

Для закрепления работы с ООП создадим метод для перемещения объекта, который будет называться move. Для этого, внутри класса mainHero напишем следующее:

class mainHero {

constructor(x, y, w, h, image) {

this.x = x;

this.y = y;

this.w = w;

this.h = h;

this.image = new Image();

this.image.src = image;

}

draw(context) {

context.drawImage(this.image, this.x, this.y, this.w, this.h)

}

**move(dx, dy) {**

**this.x += dx;**

**this.y += dy;**

**}**

}

Далее, вызовем метод move в основной функции смены кадров и создадим две переменные для передвижения.

Помимо этого, с прошлых проектов возьмем функции для перемещения:

**let move\_x = 0;**

**let move\_y = 0;**

const hero = new mainHero(100, 100, 100, 100, "player.png");

function move() {

context.clearRect(0, 0, canvas.width, canvas.height)

hero.draw(context)

**hero.move(move\_x, move\_y);**

requestAnimationFrame(move);

}

move();

**window.addEventListener("keydown", (event) => {**

**if (event.key.toLowerCase() == "w") {move\_y = -2;}**

**if (event.key.toLowerCase() == "s") {move\_y = 2;}**

**if (event.key.toLowerCase() == "a") {move\_x = -2;}**

**if (event.key.toLowerCase() == "d") {move\_x = 2;}**

**})**

**window.addEventListener("keyup", (event) => {**

**if (event.key.toLowerCase() == "w") {move\_y = 0;}**

**if (event.key.toLowerCase() == "s") {move\_y = 0;}**

**if (event.key.toLowerCase() == "a") {move\_x = 0;}**

**if (event.key.toLowerCase() == "d") {move\_x = 0; }**

**})**

Таким образом, у персонажа есть два метода: отрисовка и перемещение.

Подобным способом добавляются и другие методы, здесь самое важное это понять что требуется создать для персонажа.

**Некомпьютерная активность (10 минут)**

Игра с коробками

1. Раздайте участникам коробки с различными предметами внутри.
2. Попросите участников не открывать коробки и только описать, что, по их мнению, находится внутри.
3. Объясните, что коробки представляют собой "объекты", а их содержимое - "внутренние данные", которые скрыты от внешнего мира.
4. Затем разрешите участникам открыть коробки и сравнить свои предположения с реальным содержимым.
5. Обсудите, как принцип инкапсуляции позволяет скрывать внутреннюю реализацию объекта.

**Работа над проектом (30 минут)**

**Задание:** Добавить следующие методы для созданного класса:

1. Увеличение и уменьшение скорости на кнопки **p** и **o** соответственно. Минимальная скорость - 1, максимальная скорость - 10 (14.1)
2. При нажатии на кнопку **c** в верхнем правом углу будут выводиться координаты игрока (14.1)

**Рефлексия (5 минут)**

"Что нового вы узнали сегодня?"

"Что такое класс? Что такое экземпляр класса? Что такое конструктор класса? Как мы использовали все эти понятия для создания персонажа?"