**ЛР 15-16 ООП**

**Контрольные вопросы**

Тема "Объекты в JavaScript"

1. \*\*Что понимается под объектом в JavaScript? Приведите примеры известных вам объектов в JavaScript.\*\*

В JavaScript объект — это коллекция данных и функциональности, объединённая в виде набора пар "ключ-значение" (свойства и методы). Объекты позволяют структурировать данные и поведение в одном месте.

\*\*Примеры объектов в JavaScript:\*\*

- `Object` — базовый объект для всех других объектов.

- `Array` — объект для хранения упорядоченных коллекций.

- `Date` — объект для работы с датами и временем.

- `Math` — объект, содержащий математические функции и константы.

- `Function` — объект, представляющий функции.

2. \*\*Приведите пример описания объекта в JS.\*\*

```javascript

const car = {

brand: "Toyota",

model: "Corolla",

year: 2020,

start: function() {

console.log("Engine started");

}

};

```

3. \*\*Приведите пример доступа к свойствам объекта в JS.\*\*

```javascript

console.log(car.brand); // Точечная нотация

console.log(car["model"]); // Квадратные скобки

```

4. \*\*Какая команда в JS позволяет выполнить удаление свойства? Приведите пример.\*\*

```javascript

delete car.year;

console.log(car.year); // undefined

```

5. \*\*Какая команда в JS позволяет выполнить проверку существования свойства? Приведите пример.\*\*

```javascript

console.log("brand" in car); // true

console.log(car.hasOwnProperty("model")); // true

```

6. \*\*Какая команда в JS позволяет выполнить перебор свойств объекта? Приведите пример.\*\*

```javascript

for (let key in car) {

console.log(key + ": " + car[key]);

}

```

**Практическая часть**

**Задание 1**

### 1. \*\*Инкапсуляция\*\*

Инкапсуляция заключается в объединении данных и методов, работающих с ними, в одном объекте. В коде это реализовано через методы внутри объекта `Hero`, которые манипулируют его внутренними состояниями, например, `this.variant`, `this.direction`, и другие.

```javascript

Hero.prototype.changeColor = function(newColor) {

this.variant = newColor;

};

```

### 2. \*\*Наследование\*\*

Наследование позволяет классу расширять функциональность другого класса. В данном случае, `Hero` наследует от `Phaser.Sprite`, используя `Object.create` для расширения функционала базового класса.

```javascript

Hero.prototype = Object.create(Phaser.Sprite.prototype);

Hero.prototype.constructor = Hero;

```

### 3. \*\*Полиморфизм\*\*

Полиморфизм позволяет объектам разных классов обрабатывать одинаковые методы по-разному. В данном коде это видно в методе `move`, который зависит от направления движения (`xDir`, `yDir`), и вызывает различные анимации в зависимости от направления.

```javascript

this.animations.play(this.getAnimName());

```

Полиморфизм также проявляется в `Hero.prototype.getAnimName`, где функция генерирует разные имена анимаций в зависимости от направления и цвета героя.

### 4. \*\*Конструкторы\*\*

Конструктор используется для инициализации объекта. В данном примере это функция `Hero`, которая инициализирует объект `Hero`, устанавливает начальные параметры и вызывает методы для настройки анимаций и звуков.

```javascript

var Hero = function(x, y, variant, facing, map) {

Phaser.Sprite.call(this, game, x, y, 'hero', 0);

// Инициализация других свойств

};

```

### 5. \*\*Геттеры и Сеттеры\*\*

Геттеры и сеттеры позволяют контролировать доступ к свойствам объекта. В данном коде используется метод `changeColor`, который действует как сеттер для изменения цвета героя.

```javascript

Hero.prototype.changeColor = function(newColor) {

this.variant = newColor;

};