Object Oriented Programming

Lasse Haverinen 2024

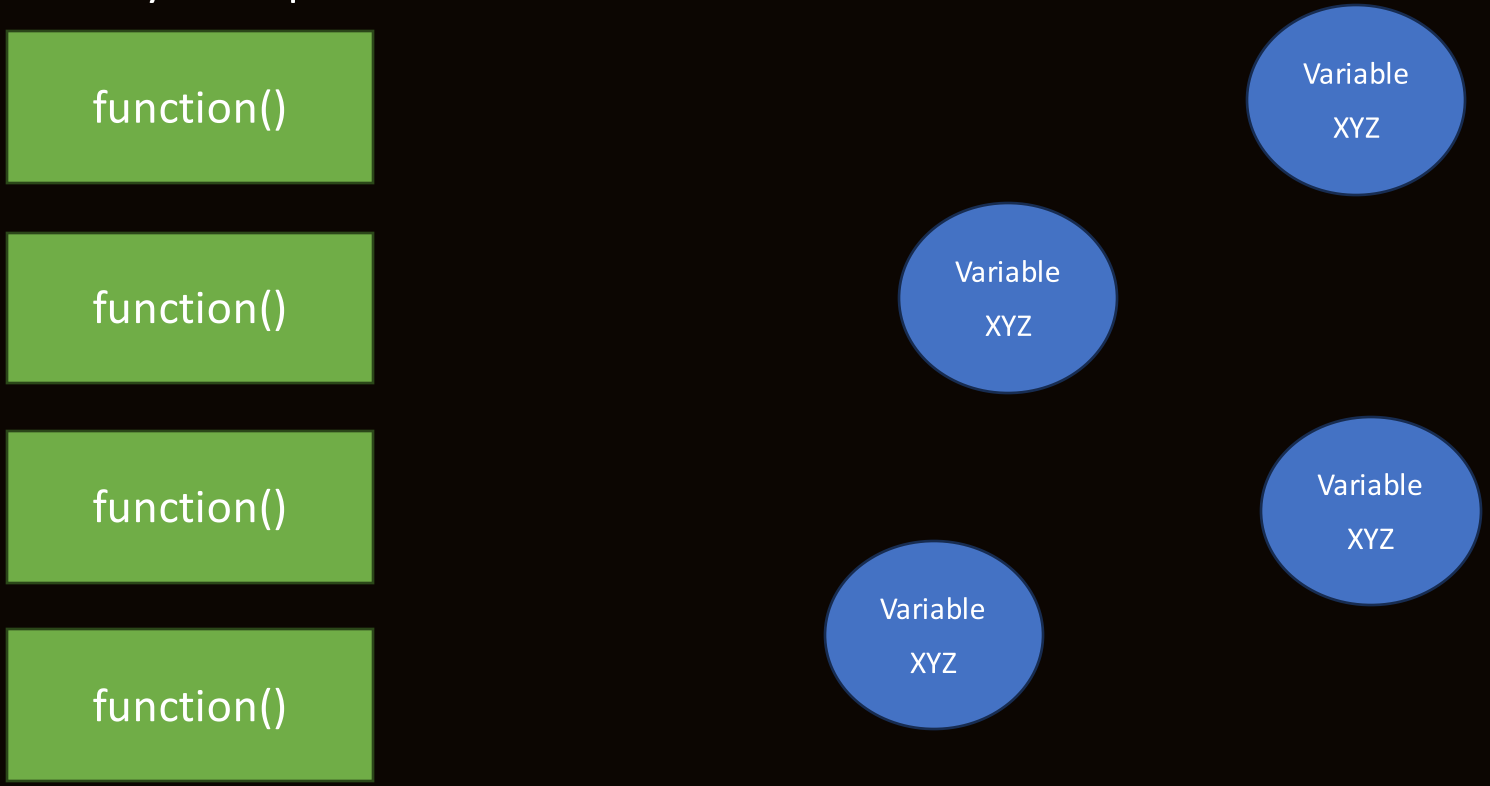
Programming paradigms

- Procedural programming
- Object oriented programming

Procedural programming

Application is built with functions and variables holding data

Data and functionality are separated



Procedural programming example

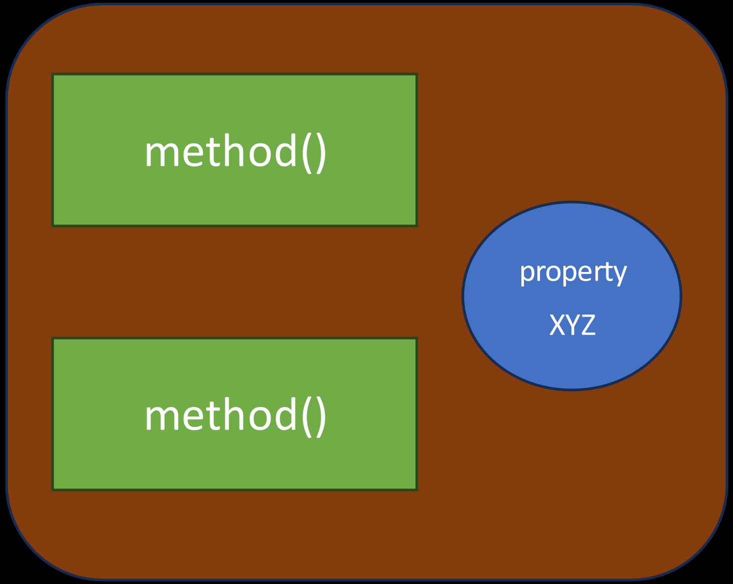
```
// Function to calculate the area of a rectangle
function calculateRectangleArea(width, height) {
  return width * height;
}

// Function to display the area of a rectangle
function displayRectangleArea(width, height) {
  var area = calculateRectangleArea(width, height);
  console.log('The area of the rectangle is: ' + area);
}

// Function to check if the area is greater than a threshold
function checkAreaThreshold(width, height, threshold) {
  var area = calculateRectangleArea(width, height);
  return area > threshold;
}

// Example usage
var rectangleWidth = 5;
var rectangleHeight = 10;

displayRectangleArea(rectangleWidth, rectangleHeight);

var thresholdValue = 40;
if (checkAreaThreshold(rectangleWidth, rectangleHeight, thresholdValue))
{
  console.log('The area is greater than the threshold.');
} else {
  console.log('The area is not greater than the threshold.');
}
```

Object Oriented Programming Principles

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

OOP Principles - Encapsulation

- Data and functionality (called methods!) are bundled together as an OBJECT
- Restricting access to some (or all) of the data inside the OBJECT
- The STATE of the object is hidden or not visible to the outside world directly

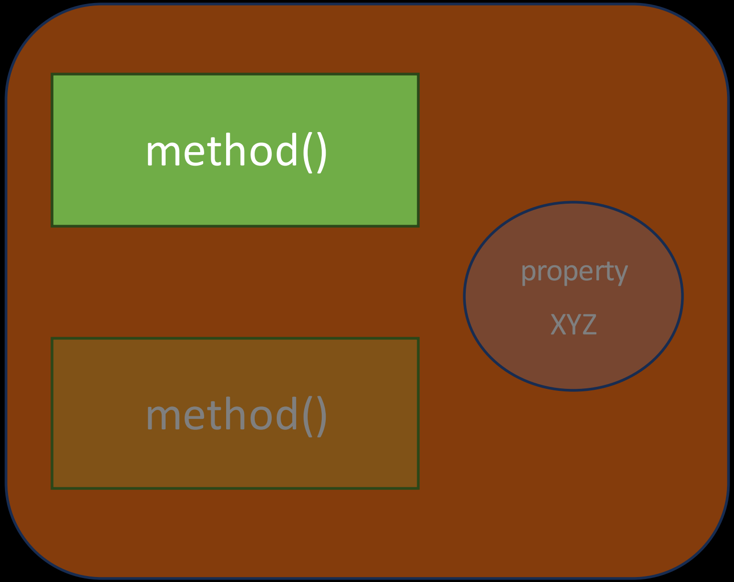


OOP Principles – Encapsulation example

```
function createRectangle(width, height) {
  return {
    width: width,
    height: height,
    calculateArea: function () {
      return this.width * this.height;
    }
  };
}

// Example usage
var rectangle = createRectangle(5, 10);

console.log(rectangle.calculateArea());
```

OOP Principles - Abstraction

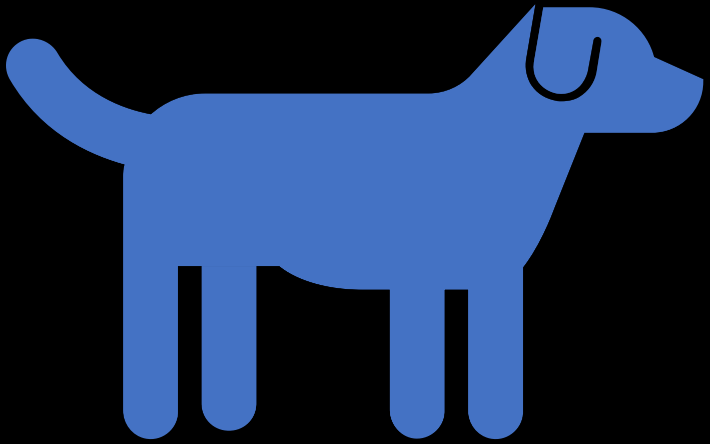
- Objects can hide the internal implementation details and provide a simpler interface
- Consider for example your mobile phone – user interface vs internal implementation, which is abstracted or hidden from the user
- Benefits?
  - The internal implementation can be updated
  - Simple interface for user



OOP Principles – Abstraction Example

```
// Object-oriented programming example in JavaScript using class
with private method
class Rectangle {
  constructor(width, height) {
    this.width = width;
    this.height = height;
  }

  #calculateArea() {
    return this.width * this.height;
  }

  displayArea() {
    var area = this.#calculateArea();
    console.log("The area of the rectangle is: " + area);
  }
}

// Example usage
var rectangle = new Rectangle(5, 10);

rectangle.displayArea();

console.log(rectangle.calculateArea()); // This will cause error
```

OOP Principles - Inheritance

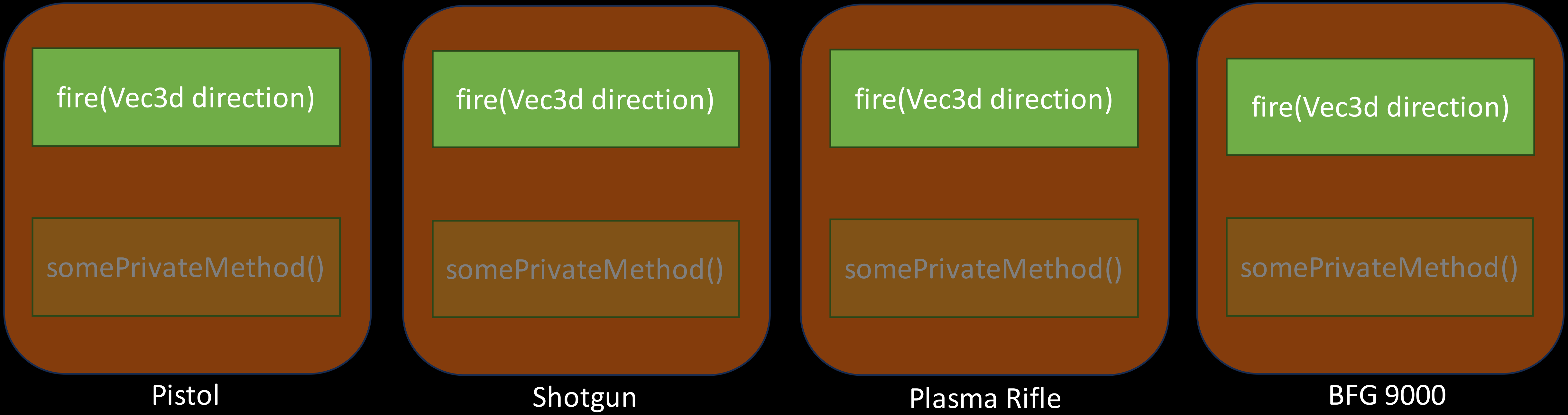


- Allows you to derive an object from an another
- X can inherit Y and X will have all the methods and properties of Y
- Consider implementing cat, dog, lion and a horse to an application?

- Reduces redundant code!

OOP Principles - Polymorphism

- Several forms
- Access/use different objects via same interface
- Consider a game where you have different weapons


fire(Vec3d direction)
somePrivateMethod()
Pistol
fire(Vec3d direction)
somePrivateMethod()
Shotgun
fire(Vec3d direction)
somePrivateMethod()
Plasma Rifle
fire(Vec3d direction)
somePrivateMethod()
BFG 9000