**Encapsulation**

We group related variables and fx that opérate on them into obj

**Procedural implementation:**

* variables one side, and fx on the other side

|  |
| --- |
| let baseSalary = 30\_000;  let overtime = 10;  let rate = 20;  function getWage(baseSalary, overtime, rate)  {  return baseSalary + (overTime \* rate);  } |

**Object Oriented way to solve the same:**

|  |  |
| --- | --- |
| let employee = {  baseSalary: 30\_000;  overtime: 10;  rate: 20;  getWage: function() {  return this.baseSalary + (this.overtime \* this.rate);  }  };  employee.getWage(); | Name of the object  Properties(variables)  Methods(fx)🡪 sin parametros  Para llamar a la función del objeto |

The fewer the number of parameters, the easier it is to use and maintain that fx

**Abstraction**

* We can hice some of the properties and methods from the outside
* For:
  + Simpler interface
  + Reduce the impact of change

**Inheritance**

* Properties and methods in common

**Polimorfism**

|  |  |
| --- | --- |
| *You can replace this*  switch(…){  case ‘select’: renderSelect();  case ‘text’: renderTextBox();  case ‘checkbox’: renderCheckBox();  case …  } | *We can implement a render method in each of these objects and the render method will behave differently depending on the type of the object viewer referencing*  element.render(); |