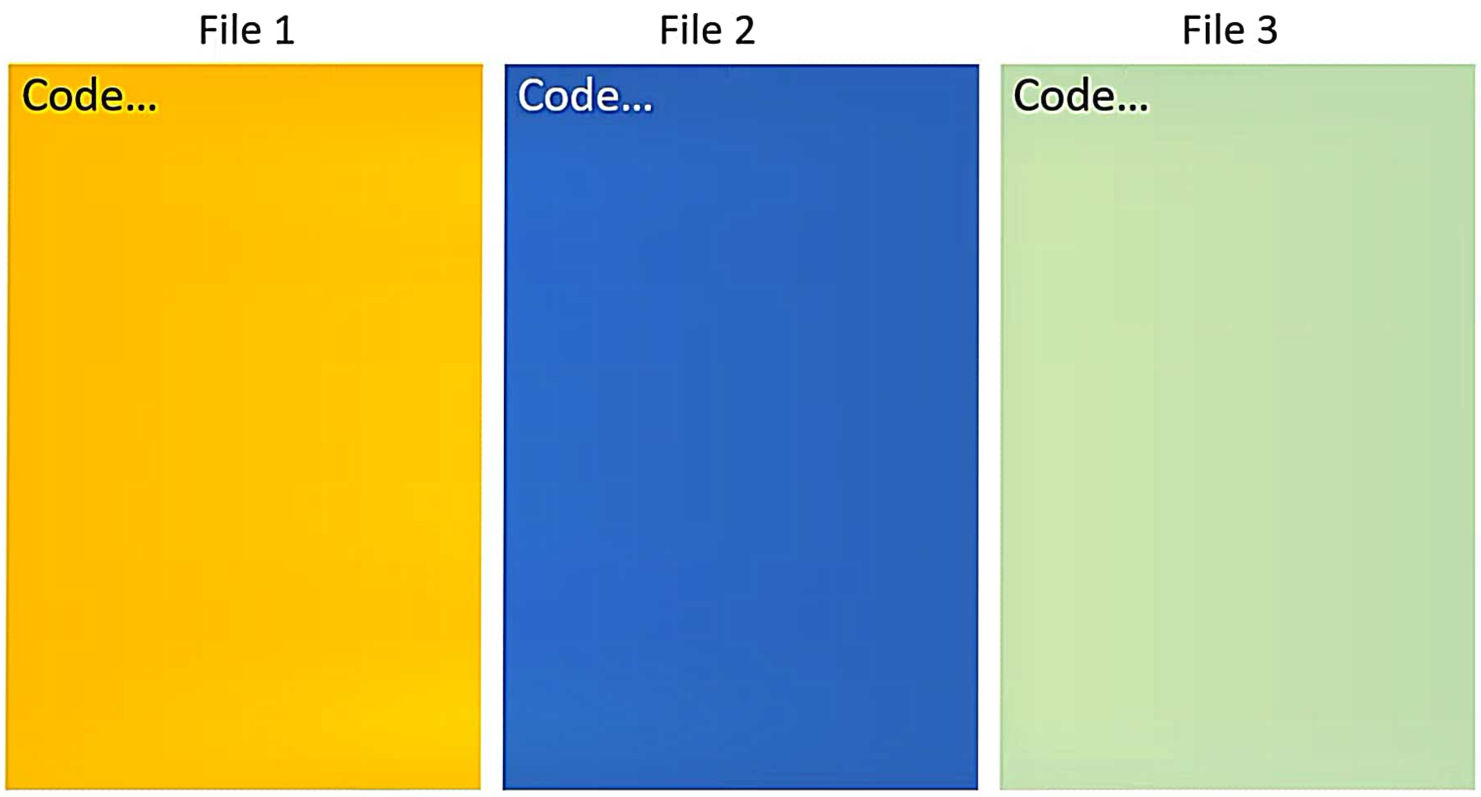
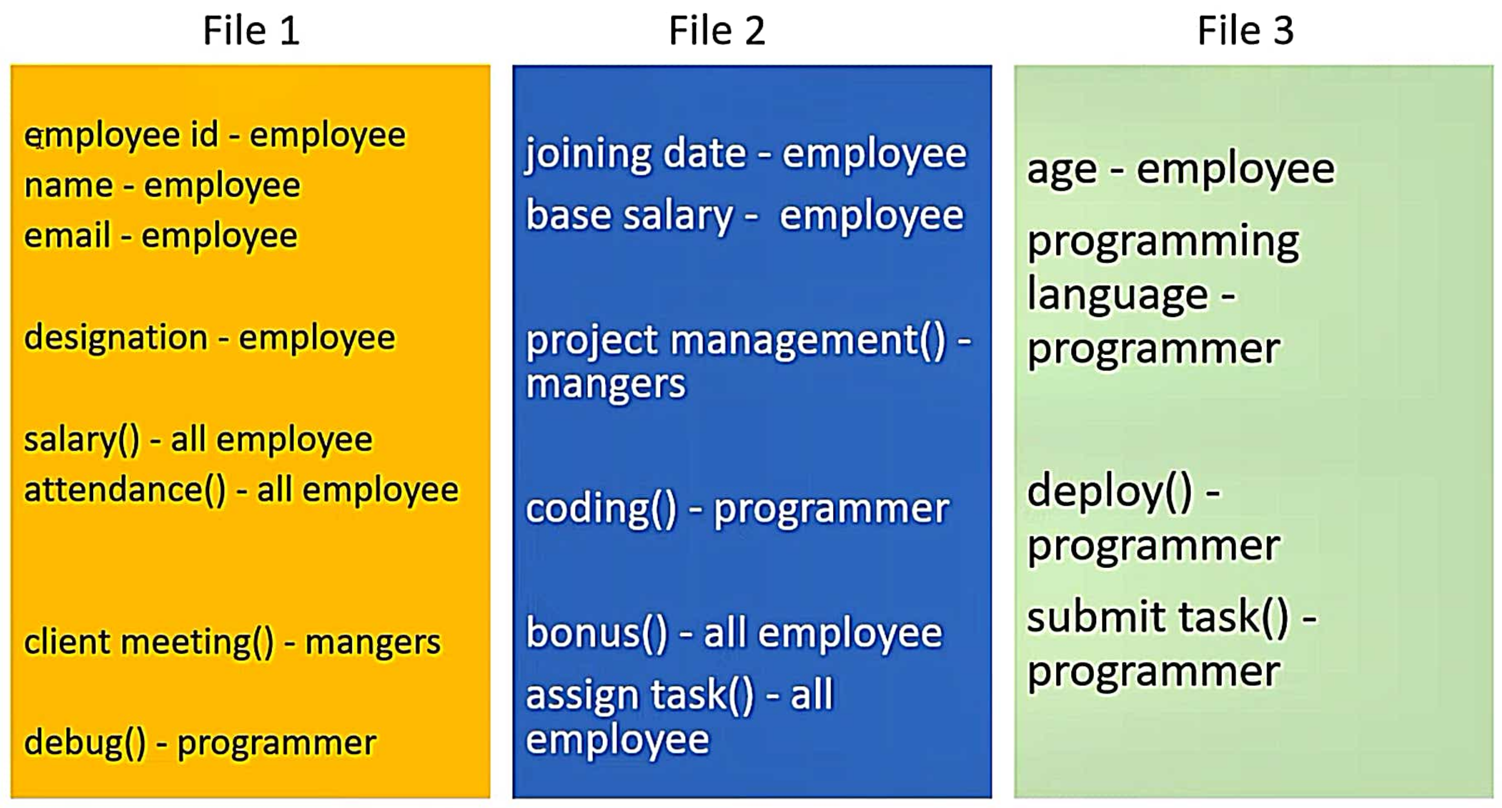
Understand Object Oriented concept in programming

(procedural vs object oriented)

- Consider Employee Management System
- To Manage Mangers and Programmers
- For now you will adapt procedural way to code (variable, statement and function)
- Huge code will be distributed among many files.
- Files contains variable and function and we write as per requirement.


File 1
File 2
File 3
Code...
Code...
Code...

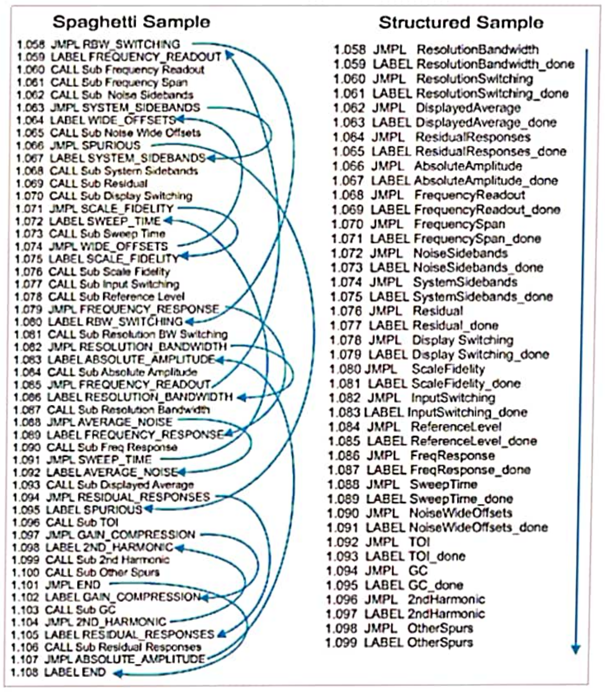


File 1	File 2	File 3
employee id - employee name - employee email - employee designation - employee salary() - all employee attendance() - all employee client meeting() - mangers debug() - programmer	joining date - employee base salary - employee project management() - mangers coding() - programmer bonus() - all employee assign task() - all employee	age - employee programming language - programmer deploy() - programmer submit task() - programmer

- Now look at the problem

1. Don’t repeat yourself failed
2. Hard to debug and manage

Spaghetti-code



Spaghetti Sample

```
1.058 JMPL RBW_SWITCHING
1.059 LABEL FREQUENCY_READOUT
1.060 CALL Sub Frequency Readout
1.061 CALL Sub Frequency Span
1.062 CALL Sub  Noise Sidebands
1.063 JMPL SYSTEM_SIDEBANDS
1.064 LABEL WIDE_OFFSETS
1.065 CALL Sub Noise Wide Offsets
1.066 JMPL SPURIOUS
1.067 LABEL SYSTEM_SIDEBANDS
1.068 CALL Sub System Sidebands
1.069 CALL Sub Residual
1.070 CALL Sub Display Switching
1.071 JMPL SCALE_FIDELITY
1.072 LABEL SWEEP_TIME
1.073 CALL Sub Sweep Time
1.074 JMPL WIDE_OFFSETS
1.075 LABEL SCALE_FIDELITY
1.076 CALL Sub Scale Fidelity
1.077 CALL Sub Input Switching
1.078 CALL Sub Reference Level
1.079 JMPL FREQUENCY_RESPONSE
1.080 LABEL RBW_SWITCHING
1.081 CALL Sub Resolution BW Switching
1.082 JMPL RESOLUTION_BANDWIDTH
1.083 LABEL ABSOLUTE_AMPLITUDE
1.084 CALL Sub Absolute Amplitude
1.085 JMPL FREQUENCY_READOUT
1.086 LABEL RESOLUTION_BANDWIDTH
1.087 CALL Sub Resolution Bandwidth
1.088 JMPL AVERAGE_NOISE
1.089 LABEL FREQUENCY_RESPONSE
1.090 CALL Sub Freq Response
1.091 JMPL SWEEP_TIME
1.092 LABEL AVERAGE_NOISE
1.093 CALL Sub Displayed Average
1.094 JMPL RESIDUAL_RESPONSES
1.095 LABEL SPURIOUS
1.096 CALL Sub TOI
1.097 JMPL GAIN_COMPRESSION
1.098 LABEL 2ND_HARMONIC
1.099 CALL Sub 2nd Harmonic
1.100 CALL Sub Other Spurs
1.101 JMPL END
1.102 LABEL GAIN_COMPRESSION
1.103 CALL Sub GC
1.104 JMPL 2ND_HARMONIC
1.105 LABEL RESIDUAL_RESPONSES
1.106 CALL Sub Residual Responses
1.107 JMPL ABSOLUTE_AMPLITUDE
1.108 LABEL END
```

Structured Sample

```
1.058 JMPL   ResolutionBandwidth
1.059 LABEL ResolutionBandwidth_done
1.060 JMPL   ResolutionSwitching
1.061 LABEL ResolutionSwitching_done
1.062 JMPL   DisplayedAverage
1.063 LABEL DisplayedAverage_done
1.064 JMPL   ResidualResponses
1.065 LABEL ResidualResponses_done
1.066 JMPL   AbsoluteAmplitude
1.067 LABEL AbsoluteAmplitude_done
1.068 JMPL   FrequencyReadout
1.069 LABEL FrequencyReadout_done
1.070 JMPL   FrequencySpan
1.071 LABEL FrequencySpan_done
1.072 JMPL   NoiseSidebands
1.073 LABEL NoiseSidebands_done
1.074 JMPL   SystemSidebands
1.075 LABEL SystemSidebands_done
1.076 JMPL   Residual
1.077 LABEL Residual_done
1.078 JMPL   Display Switching
1.079 LABEL Display Switching_done
1.080 JMPL   ScaleFidelity
1.081 LABEL ScaleFidelity_done
1.082 JMPL   InputSwitching
1.083 LABEL InputSwitching_done
1.084 JMPL   ReferenceLevel
1.085 LABEL ReferenceLevel_done
1.086 JMPL   FreqResponse
1.087 LABEL FreqResponse_done
1.088 JMPL   SweepTime
1.089 LABEL SweepTime_done
1.090 JMPL   NoiseWideOffsets
1.091 LABEL NoiseWideOffsets_done
1.092 JMPL   TOI
1.093 LABEL TOI_done
1.094 JMPL   GC
1.095 LABEL GC_done
1.096 JMPL   2ndHarmonic
1.097 LABEL 2ndHarmonic
1.098 JMPL   OtherSpurs
1.099 LABEL OtherSpurs
```

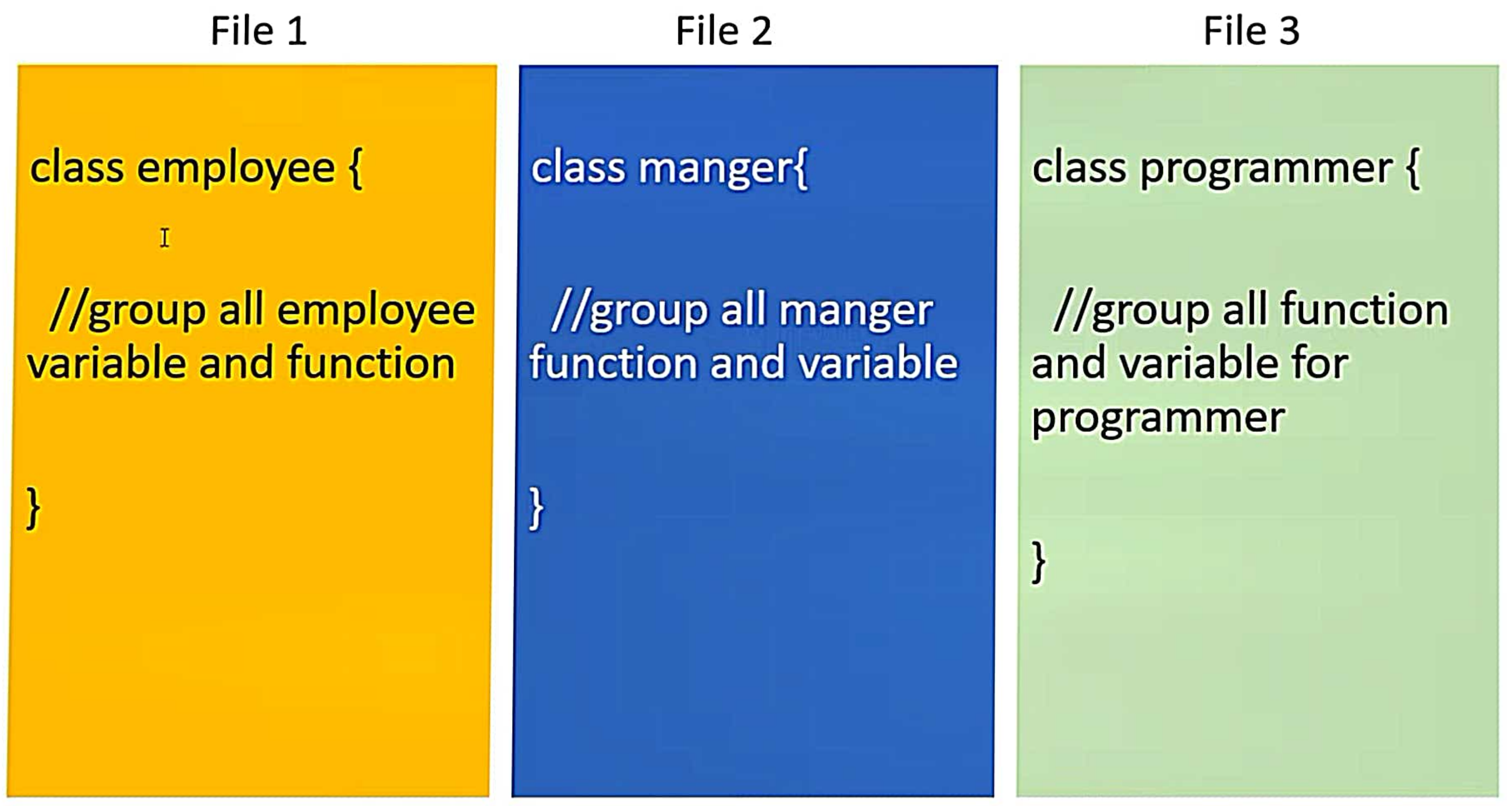
object oriented programming (OOP)

- In OOP Concept we group function and variable in a Block Called Class

- In OOP Concept we group function and variable in a Block Called Class

This is not the exact syntax.

This is not the exact syntax.



File 1

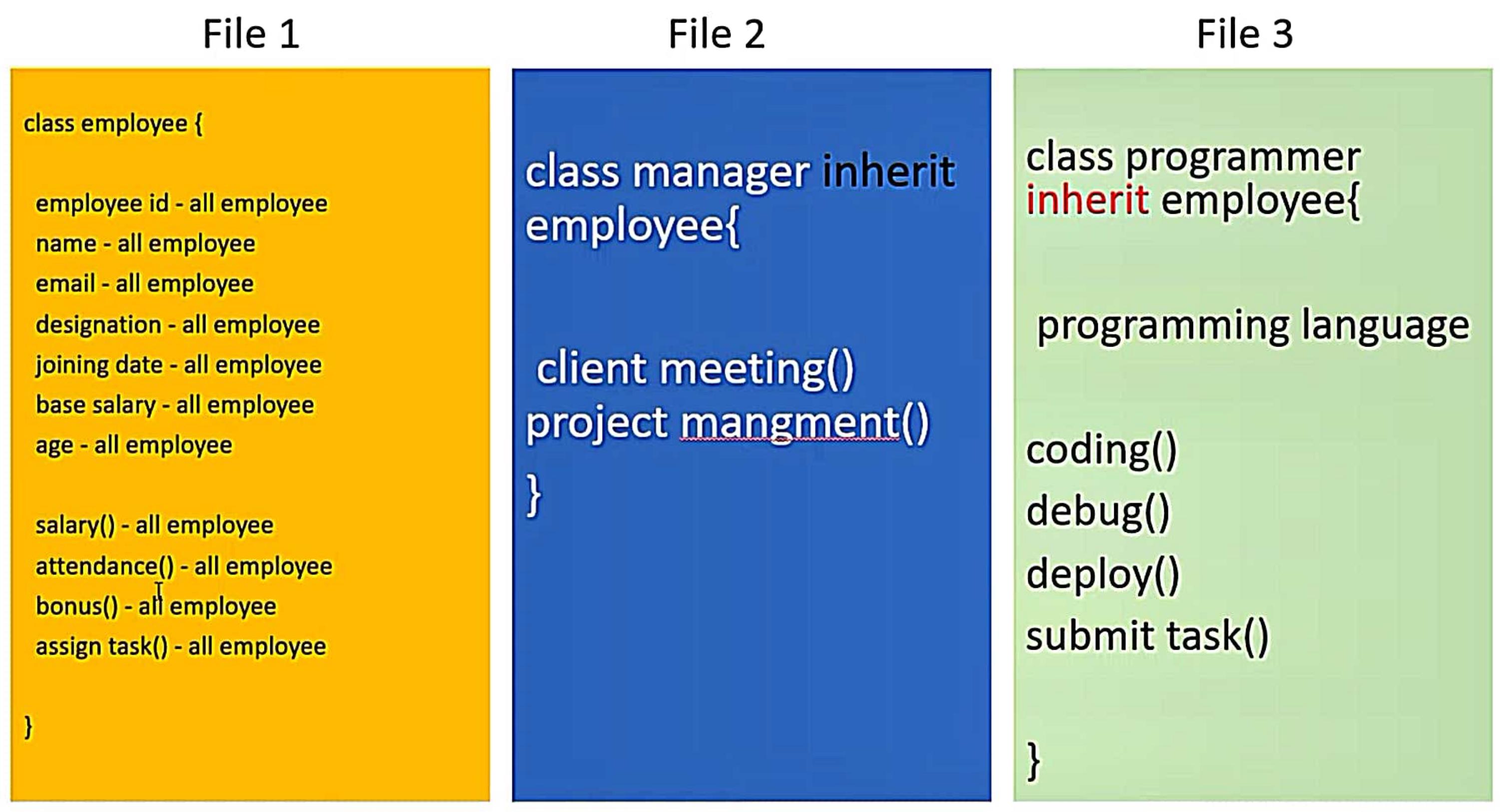
```
class employee {

  //group all employee
variable and function

}
```

File 2

```
class manger{

  //group all manger
function and variable

}
```

File 3

```
class programmer {

  //group all function
and variable for
programmer

}
```



File 1

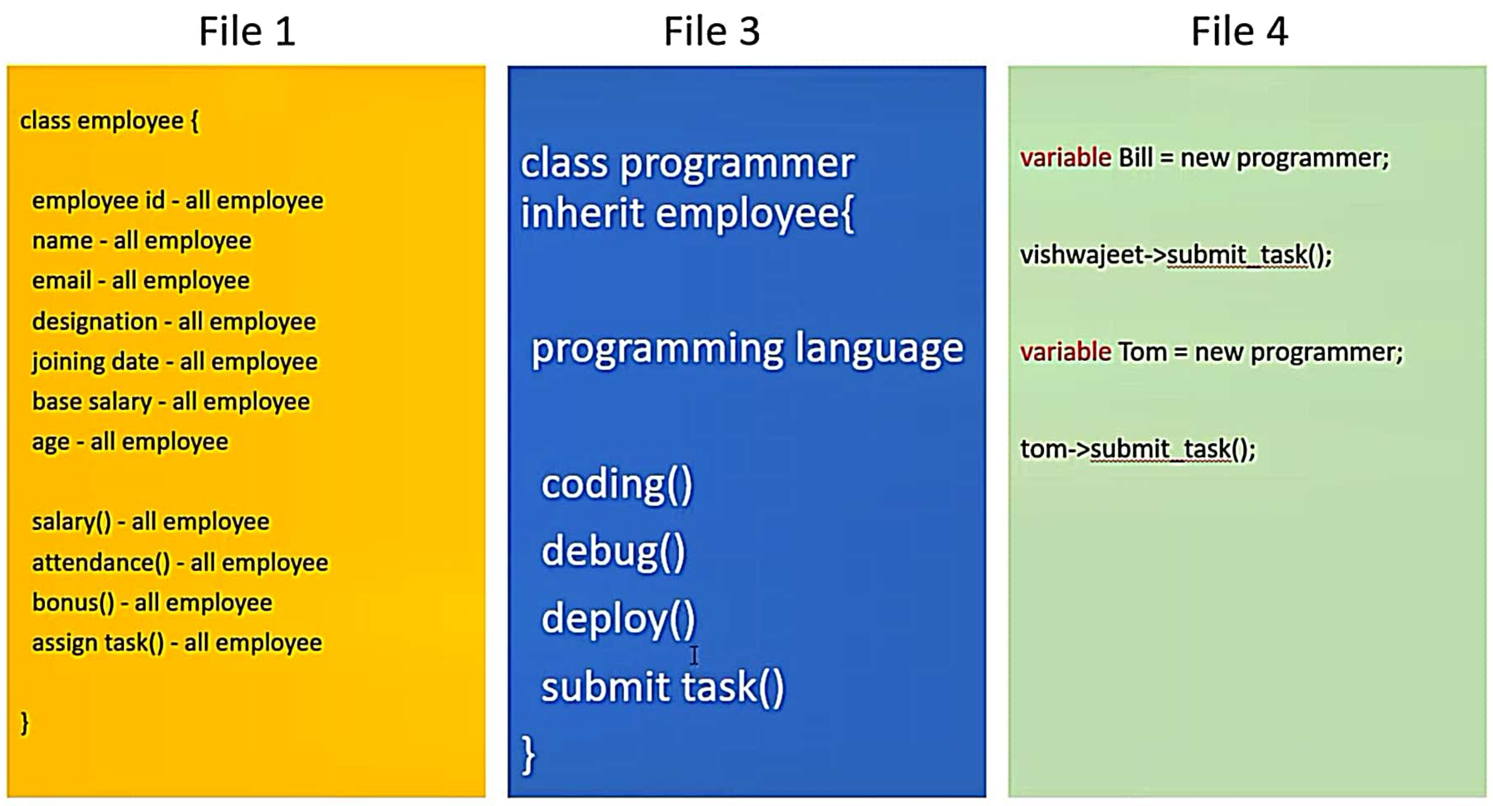
```
class employee {

  employee id - all employee
  name - all employee
  email - all employee
  designation - all employee
  joining date - all employee
  base salary - all employee
  age - all employee

  salary() - all employee
  attendance() - all employee
  bonus() - all employee
  assign task() - all employee

}
```

File 2

```
class manager inherit
employee{


 client meeting()
project mangment()
}
```

File 3

```
class programmer
inherit employee{


 programming language


coding()
debug()
deploy()
submit task()


}
```



File 1

```
class employee {

  employee id - all employee
  name - all employee
  email - all employee
  designation - all employee
  joining date - all employee
  base salary - all employee
  age - all employee

  salary() - all employee
  attendance() - all employee
  bonus() - all employee
  assign task() - all employee

}
```

File 3

```
class programmer
inherit employee{


 programming language


  coding()
  debug()
  deploy()
  submit task()
}
```

File 4

```
variable Bill = new programmer;

vishwajeet->submit_task();

variable Tom = new programmer;

tom->submit_task();
```

Four pillars of Object Oriented Programming
(important for interview)

1. Encapsulation: -

Encapsulation means wrapping up data and member function (Method) together into a single unit i.e. class.

2. Abstraction: -

Abstraction is the process of showing only essential/necessary features of an entity/object to the outside world and hide the other irrelevant information. For example to open your TV we only have a power button, It is not required to understand how infra-red waves are getting generated in TV remote control.

3. Inheritance: -

Inheritance allows a class (subclass) to acquire the properties and behavior of another class (super-class). It helps to reuse, customize and enhance the existing code. So it helps to write a code accurately and reduce the development time.

4. Polymorphism: -

So polymorphism means "many forms". A subclass can define its own unique behavior and still share the same functionalities or behavior of its parent/base class.

```
class square(){
  area()
}

Var S1 =  new square();
S1->area();
```

```
class circle(){
  area()
}

Var C1 =  new circle();
C1->area();
```