Single Responsibility principle

Every class should have 1 job to do

Class Task() {  *// this class is not reusable, it has multiple jobs*

  downloadFile()

  parseFile()

}

Open-Closed principle

Objects and entries should be open for extension, but closed for modification

Class AreaCalculator() { // closed for modification

  calculateArea(shape)

}

Class Rectangle extends AreaCalculator() { // open for extension

}

Class Triangle extends AreaCalculator() {

}

Liskov substitution principle

Subclasses should be replaceable for their base or parent class

Class Square() {

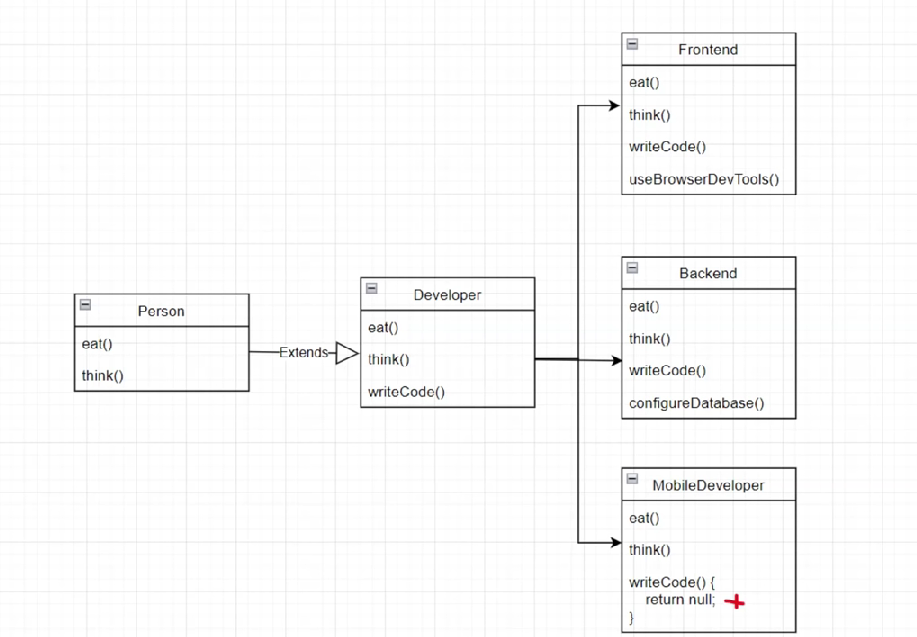
  calculateArea(a) {return Math.Pow(a,2)}

}

Class Rectangle extends Square { //not replaceable for its base

// wrong because rectangles have different sides

}



MobileDeveloper is breaking Liskov Substitution principle, because it must use expected Behavior (writeCode) from Developer Class.

Interface Segregation principle

Clients should not be forced to implement interfaces they don’t use

Class Animal() {  *// this class is not reusable, it has multiple jobs*

  feed()

  grooom()

}

Class Dog extends Animal {}

Class Tiger extends Animal {} // groom could be redundant

Dependency inversion principle

High level modules should not depend on lower level ones, but instead they both rely on abstractions(interfaces)

Class Task() {  *// this class is not reusable, it has multiple jobs*

  downloadFile()

  parseFile()

}