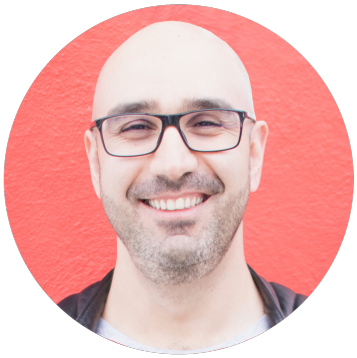
Interfaces

Quiz

Code with Mosh (codewithmosh.com)

1st Edition

About the Author



Hi! My name is Mosh Hamedani. I'm a software engineer with two decades of experience and I've taught over three million people how to code or how to become a professional software engineer. It's my mission to make software engineering simple and accessible to everyone.

https://codewithmosh.com

https://youtube.com/user/programmingwithmosh

https://twitter.com/moshhamedani

https://facebook.com/programmingwithmosh/

Questions

1- Why do we use interfaces?

2- What is tightly-coupled code?

3- Is this code loosely or tightly coupled and why?

```
public interface TaxCalculator
{}

public class TaxCalculator2018 implements TaxCalculator
{}

public class TaxReport {
    private TaxCalculator calculator;

    public TaxReport() {
        calculator = new TaxCalculator2018();
    }

}
```

4- What is dependency injection?

5- What are the various types of dependency injection?

6- What is the Interface Segregation Principle (ISP)?

7- Why shouldn’t we declare fields, static or private methods in interfaces?

8- What are the similarities and differences between interfaces and abstract classes?

9- Should we extract an interface from every class? Why?

Answers

1- We use interfaces to build loosely-coupled, extensible and testable applications.

2- Tightly-coupled code is code that is hard to change because there is a strong dependency between the entities (eg classes) in the code. Changing one class may result in several cascading, breaking changes in the code.

3- Even though the type of the **calculator** field in **TaxReport** is an interface, we're initializing this field to an instance of **TaxCalculator2018** in the constructor. So, **TaxReport** is tightly coupled to **TaxCalculator2018**, which is an implementation, not an interface.

4- Dependency injection refers to passing or injecting dependencies of a class.

5- We can inject dependencies via constructors, setters and regular methods.

6- The Interface Segregation Principle (ISP) suggests that we should segregate or divide big, fat interfaces into smaller ones, each focusing on a single responsibility or capability. Smaller interfaces are less likely to change. Changes to one capability, will only affect a single interface and fewer classes that depend on that interface.

7- Fields, static and private methods are all about implementation. Interfaces are contracts and should not have any implementation.

8- Both are abstract concepts and we cannot instantiate them. Interfaces are contracts and should only have method declarations. Abstract classes are partially-implemented classes. We use them to share some common code across their derivates. The new features in Java allow writing code and logic in interfaces but this is a bad practice and should be avoided.

9- Blindly extracting interfaces doesn't solve any problems nor is it considered a best practice. If you extract an interface from every single class, you'll end up with an explosion of interfaces that don't necessarily add any values. You should use interfaces in situations where you want to decouple a class from its dependencies so you can swap these dependencies. This allows building applications that are extensible and testable.