Open Closed Principle

- In good application design and the code writing part, you should avoid change in the existing code when requirements change
- Instead, you should extend the existing functionality by adding new code to meet the new requirements. You can achieve this by following the **Open Closed Principle**.
- The Open Closed Principle represents the “O” of the five SOLID software engineering principles to write well-designed code that are more readable, maintainable, and easier to upgrade and modify.
- This principle states: “software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification “.
  1. Open for extension “: This means that the behavior of a software module, say a class can be extended to make it behave in new and different ways.
     - note here that the term “extended ” is not limited to inheritance using the Java extend keyword. What it means here is that a module should provide extension points to alter its behavior. One way is to make use of polymorphism to invoke extended behaviors of an object at run time.
  2. “Closed for modification “: This means that the source code of such a module remains unchanged.

Violation (Bad) Example

HealthInsuranceSurveyor.java

```
package guru.springframework.blog.openclosedprinciple;
    public class HealthInsuranceSurveyor{
        public boolean isValidClaim(){
            System.out.println("HealthInsuranceSurveyor: Validating health insurance claim...");
            /*Logic to validate health insurance claims*/
            return true;
        }
    }
```

ClaimApprovalManager.java

```
package guru.springframework.blog.openclosedprinciple;
    public class ClaimApprovalManager {
        public void processHealthClaim (HealthInsuranceSurveyor surveyor)    {
            if(surveyor.isValidClaim()){
                System.out.println("ClaimApprovalManager: Valid claim. Currently processing claim for approval....");
            }
        }
    }
```

The above follows the single responsibillity principle, but when a new type of insurance claim in introduced the code has to be modified: : Modified ClaimApprovalManager.java

```
    package guru.springframework.blog.openclosedprinciple;
    public class ClaimApprovalManager {
        public void processHealthClaim (HealthInsuranceSurveyor surveyor)    {
            if(surveyor.isValidClaim()){
                System.out.println("ClaimApprovalManager: Valid claim. Currently processing claim for approval....");
            }
        }
        public void processVehicleClaim (VehicleInsuranceSurveyor surveyor)    {
            if(surveyor.isValidClaim()){
                System.out.println("ClaimApprovalManager: Valid claim. Currently processing claim for approval....");
            }
        }
    }
```

Coding to the Open Closed Principal

- Open to support more types of claims
- Closed for any modifications when support for a new type of claim is added

InsuranceSurveyor.java

```
    package guru.springframework.blog.openclosedprinciple;
    public abstract class InsuranceSurveyor {
        public abstract boolean isValidClaim();
    }
```

```
HealthInsuranceSurveyor.java
```

```
package guru.springframework.blog.openclosedprinciple;
public class HealthInsuranceSurveyor extends InsuranceSurveyor{
    public boolean isValidClaim(){
        System.out.println("HealthInsuranceSurveyor: Validating health insurance claim...");
        /*Logic to validate health insurance claims*/
        return true;
    }
}
```

VehicleInsuranceSurveyor.java

```
package guru.springframework.blog.openclosedprinciple;
public class VehicleInsuranceSurveyor extends InsuranceSurveyor{
    public boolean isValidClaim(){
       System.out.println("VehicleInsuranceSurveyor: Validating vehicle insurance claim...");
        /*Logic to validate vehicle insurance claims*/
        return true;
    }
}
```

ClaimApprovalManager.java

```
package guru.springframework.blog.openclosedprinciple;
public class ClaimApprovalManager {
    public void processClaim(InsuranceSurveyor surveyor){
        if(surveyor.isValidClaim()){
            System.out.println("ClaimApprovalManager: Valid claim. Currently processing claim for approval....");
        }
    }
}
```

Our insurance system is now open to support more types of insurance claims, and closed for any modifications whenever a new claim type is added.

Summary

Most of the times real closure of a software entity is practically not possible because there is always a chance that a change will violate the closure. For example, in our insurance example a change in the business rule to process a specific type of claim will require modifying the ClaimApprovalManager class. So, during enterprise application development, even if you might not

always manage to write code that satisfies the Open Closed Principle in every aspect, taking the steps towards it will be beneficial as the application evolves.

Revision #2
Created 17 April 2022 00:52:38 by Elkip
Updated 17 April 2022 01:37:44 by Elkip