✓ **Congratulations! You passed!**
TO PASS 75% or higher

Keep Learning

GRADE
83.33%

Module 3 Review

LATEST SUBMISSION GRADE

83.33%

1. What does MVC Stand for? Use spaces between each word, no upper case letters, and no punctuation.

1 / 1 point

model view controller

✓ **Correct**
Correct! The model view controller pattern is important for user-interface applications, and it previews some software architectures that we will talk about in the next course.

2. Select the **two** elements of the open/closed principle:

1 / 1 point

☐ Open for modification

☐ Closed for maintenance.

☑ Closed for modification

✓ **Correct**
Correct! Good software strives to close parts off for modification, which means that they should not need to be opened up again when extending functionality.

☐ Open for maintenance

☑ Open for extension

✓ **Correct**
Correct! Well-designed software should strive to be open for extension, implying that the code can be extended without having to change existing parts.

☐ Closed for extension.

3. What is the best description of the Dependency Inversion principle?

0 / 1 point

◉ Client objects depend on an Adaptor Pattern to interface with the rest of the system.

○ Client objects depend on generalizations instead of concrete objects.

○ Service objects subscribe to their prospective client objects as Observers, watching for a request.

○ Client objects are dependent on a service interface that directs their requests.

! **Incorrect**
Incorrect. This is not what the Dependency Inversion principle is!

4. Which of these statements is true about the Composing Objects principle?
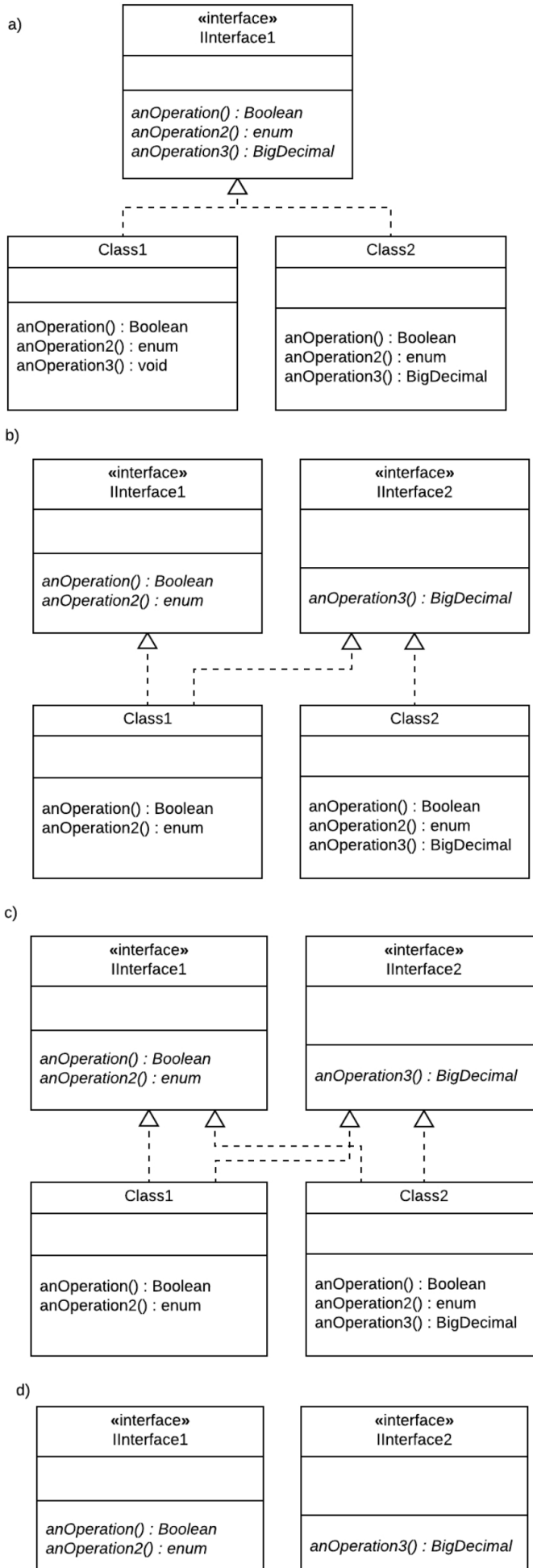
1 / 1 point
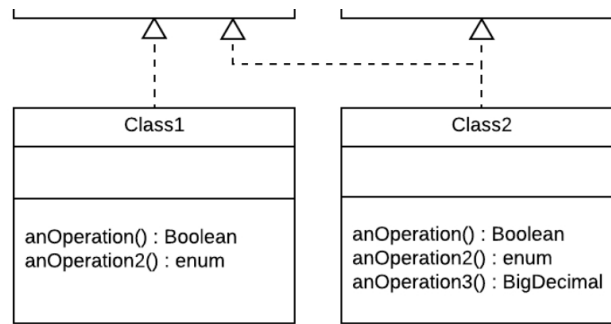
1. it provides behaviour with aggregation instead of inheritance

2. it leads to tighter coupling

◉ The first statement is true

○ The second statement is true

○ Neither statement is true

○ Both statements are true

✓ **Correct**
Correct! Behaviour can be built by aggregating objects instead of using inheritance. This is an an inherently more flexible approach.

5. Which of these UML diagrams demonstrates the Interface Segregation principle?

1 / 1 point

a)

«interface»
IInterface1

anOperation() : Boolean
anOperation2() : enum
anOperation3() : BigDecimal

Class1

anOperation() : Boolean
anOperation2() : enum
anOperation3() : void

Class2

anOperation() : Boolean
anOperation2() : enum
anOperation3() : BigDecimal

b)

«interface»
IInterface1

anOperation() : Boolean
anOperation2() : enum

«interface»
IInterface2

anOperation3() : BigDecimal

Class1

anOperation() : Boolean
anOperation2() : enum

Class2

anOperation() : Boolean
anOperation2() : enum
anOperation3() : BigDecimal

c)

«interface»
IInterface1

anOperation() : Boolean
anOperation2() : enum

«interface»
IInterface2

anOperation3() : BigDecimal

Class1

anOperation() : Boolean
anOperation2() : enum

Class2

anOperation() : Boolean
anOperation2() : enum
anOperation3() : BigDecimal

d)

«interface»
IInterface1

anOperation() : Boolean
anOperation2() : enum

«interface»
IInterface2

anOperation3() : BigDecimal

Class1

anOperation() : Boolean
anOperation2() : enum

Class2

anOperation() : Boolean
anOperation2() : enum
anOperation3() : BigDecimal

○ a)

○ b)

○ c)

◉ d)

✓ **Correct**

Correct! Class1 does not need all of the methods, so it makes sense to have two different interfaces.

---

6. Which of these code examples **violates** the Principle of Least Knowledge, or Law of Demeter?   `1 / 1 point`

◉
```
1 ▾ public class O {
2    M I = new M();
3
4 ▾   public void anOperation2() {
5        this.I.N.anOperation();
6    }
7  }
```

○
```
1 ▾ public class Class1 {
2 ▾   public void N() {
3        System.out.println("Method N invoked");
4    }
5  }
6
7 ▾ public class Class2 {
8 ▾   public void M(Class1 P) {
9        P.N();
10       System.out.println("Method M invoked");
11   }
12
```

○
```
1 ▾ public class O {
2 ▾   public void M() {
3        this.N();
4        System.out.println("Method M invoked");
5    }
6
7 ▾   public void N() {
8        System.out.println("Method N invoked");
9    }
10 }
```

○
```
1 ▾ public class P {
2 ▾   public void N() {
3        System.out.println("Method N invoked");
4    }
5  }
6
7 ▾ public class O {
8 ▾   public void M() {
9    P I = new P();
10   I.N();
11       System.out.println("Method M invoked");
12   }
13 }
```

✓ **Correct**

Correct! In this example, the method call in the class (O) reaches through the object (I) to a method in another object (N). This is not local and therefore the Principle is violated.

---

7. How can Comments be considered a code smell?   `1 / 1 point`

○ They can't! Comments help clarify code.

○ When a comment is used to explain the rationale behind a design decision

○ Too many comments make the files too large to compile.

◉ Excessive commenting can be a coverup for bad code

✓ **Correct**

Correct! Sometimes, developers use excessive comments like a "deodorant" for bad code, instead of fixing the code.

8. What is the primitive obsession code smell about?   **1 / 1 point**

    ○ Code that contains many low-level objects, without using OO principles like aggregation or inheritance.

    ⦿ Overuse of primitive data types like int, long, float

    ○ Using many different primitive types instead of settling on a few that together capture that appropriate level of detail for your system.

    ○ Using key-value pairs instead of abstract data types.

    > ✓ **Correct**
    > Correct! Excessive use of primitives may mean that you are not identifying appropriate abstractions.

9. You have a class that you keep adding to. Whenever you add new functionality, it just seems like the most natural place to put it, but it is starting to become a problem! Which code smell is this?   **0 / 1 point**

    ○ Long Method

    ○ Large Class

    ⦿ Divergent Change

    ○ Speculative generality

    > ! **Incorrect**
    > Incorrect. Divergent change is a related problem, since more responsibilities are added to a class, but tends to be more associated with lowering cohesion.

10. Why is it important to avoid message chains whenever possible?   **1 / 1 point**

    ○ It's a workaround to get to private methods, which are important for encapsulation.

    ○ They lower cohesion in your class.

    ⦿ The resulting code is usually rigid and complex.

    ○ If an unexpected object is returned, this could easily lead to runtime errors.

    > ✓ **Correct**
    > Correct! Code with message chains is more difficult to not only maintain, but also to read. They will require Shotgun Surgery when changes need to be made.

11. Look at the code snippet. Which code smell do you detect?   **1 / 1 point**

    ```
    1  public class Class1 {
    2
    3   ...
    4
    5   public void M(Class2 C) {
    6     C.doSomething(x);
    7     C.foo(y);
    8     C.foo2(z, i);
    9   }
    10 }
    ```

    ○ Long Parameter List

    ⦿ Feature Envy

    ○ Inappropriate Intimacy

    ○ Divergent Change

    > ✓ **Correct**
    > Correct! The method M calls lots of methods in the object C. Perhaps it would be better to have this method in that object.

12. Joseph was developing a class for his smartphone poker game, and decided that one day he would like to be able to change the picture on the backs of the cards, so he created a Deck superclass. Since his app does not have that feature yet, Deck has only one subclass, RegularDeck. What code smell is this?   **1 / 1 point**

    ○ Divergent Change

    ○ Refused Bequest

    ○ Primitive Obsession

    ⦿ Speculative Generality

    > ✓ **Correct**
    > Correct! Coding for anticipated needs instead of the current ones is not good Agile Development.