

Java-Success.com

Industrial strength Java/JEE Career Companion for those who want to go places

[Home](#)
[Java FAQs](#)
[600+ Java Q&As](#)
[Career](#)
[Tutorials](#)
[Member](#)
[Why?](#)
[Can u Debug?](#)
[Java 8 ready?](#)
[Top X](#)
[Productivity Tools](#)
[Judging Experience?](#)

[Home](#) › [Tech Key Areas](#) › [13 Technical Key Areas Interview Q&A](#) › [Design](#)

[Concepts](#) › Understanding Open/Closed Principle (OCP) from the SOLID OO principles with a Java example

Understanding Open/Closed Principle (OCP) from the SOLID OO principles with a Java example

Posted on [November 22, 2014](#) by [Arulkumaran Kumaraswamipillai](#) — No

[Comments](#) ↓

0

Like

0

G+1

Share

Share

Q. Is there anything wrong with the following class design? If yes, can the design be improved?

```
1 package com.ocp;
2
3 import javax.management.RuntimeErrorException;
```

[9 tips to earn more](#) | [What can u do to go places?](#) | **945+** members. [LinkedIn Group](#). [Reviews](#)

600+ Full Stack Java/JEE Interview Q&As ♥Free ♦FAQs

[open all](#) | [close all](#)

- ☰ [Ice Breaker Interview](#)
- ☰ [Core Java Interview C](#)
- ☰ [Java Overview \(4\)](#)
- ☰ [Data types \(6\)](#)
- ☰ [constructors-methc](#)
- ☰ [Reserved Key Wor](#)
- ☰ [Classes \(3\)](#)
- ☰ [Objects \(8\)](#)
- ☰ [OOP \(10\)](#)
- ☰ [♥ Design princip](#)
- ☰ [♦ 30+ FAQ Java](#)

```

4
5 import org.apache.commons.lang.StringUtils;
6
7 public class MathOperation {
8
9     public int operate(int input1, int input2, String operator) {
10
11         if(StringUtils.isEmpty(operator)){
12             throw new IllegalArgumentException("Invalid operator");
13         }
14
15         if(operator.equalsIgnoreCase("+")){
16             return input1 + input2;
17         }
18         else if(operator.equalsIgnoreCase("*")){
19             return input1 * input2;
20         } else {
21             throw new RuntimeException("unsupported operator");
22         }
23     }
24 }
25 }
26

```

JUnit test class.

```

1 package com.ocp;
2
3 import junit.framework.Assert;
4
5 import org.junit.Before;
6 import org.junit.Test;
7
8 public class MathOperationTest {
9
10     MathOperation operation;
11
12     @Before
13     public void init(){
14         operation = new MathOperation();
15     }
16
17     @Test
18     public void testAddition() {
19         Assert.assertEquals(8, operation.operate(5, 3, "+"));
20     }
21
22     @Test
23     public void testMultiplication() {
24         Assert.assertEquals(15, operation.operate(5, 3, "*"));
25     }
26 }
27 }
28

```

A. It's not a good idea to try to anticipate changes in requirements ahead of time, but you should focus on writing

[Why favor composition over inheritance?](#)
[08: Write code that is easy to test](#)
[Explain abstract classes and interfaces](#)
[How to create a generic class](#)
[Top 5 OOPs tips](#)
[Top 6 tips to go from Junior to Senior Java Developer](#)
[Understanding Closures](#)
[What are good reasons to use a design pattern?](#)

[GC \(2\)](#)
[Generics \(5\)](#)
[FP \(8\)](#)
[IO \(7\)](#)
[Multithreading \(12\)](#)
[Algorithms \(5\)](#)
[Annotations \(2\)](#)
[Collection and Data Structures \(1\)](#)
[Differences Between Java and JavaScript](#)
[Event Driven Programming](#)
[Exceptions \(2\)](#)
[Java 7 \(2\)](#)
[Java 8 \(24\)](#)
[JVM \(6\)](#)
[Reactive Programming](#)
[Swing & AWT \(2\)](#)
[JEE Interview Q&A \(3\)](#)
[Pressed for time? Java Interview Questions](#)
[SQL, XML, UML, JSC](#)
[Hadoop & BigData Interview Questions](#)
[Java Architecture Interview Questions](#)
[Scala Interview Q&As](#)
[Spring, Hibernate, & JPA Interview Questions](#)
[Testing & Profiling/Static Analysis](#)
[Other Interview Q&A \(1\)](#)
[Free Java Interview Questions](#)

As a Java Architect

[Java architecture & design concepts](#)

code that is well written enough so that it's easy to change. This means, you should strive to **write code that doesn't have to be changed every time the requirements change**. This is what the **Open/Closed principle** is. According to **GoF design pattern** authors "software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification". Spring framework promotes this principle.

In the above example, you can anticipate more operators like "-" (subtraction) and division (/) to be supported in the future and the class "**MathOperation**" is not closed for modification. When you need to support operators "-" and "%" you need to add 2 more "else if" statements. Whenever you see large if/else or switch statements, you need to think if "**Open/Closed**" design principle is more suited.

Let's open for extension and close for modifications

In the rewritten example below, the classes **AddOperation** and **MultiplyOperation** are closed for modification, but open for extension by allowing you to add new classes like **SubtractOperation** and **DivisionOperation** by implementing the **Operation** interface.

Define the **interface** Operation.

```
1 package com.ocp;
2
3 public interface Operation {
4     abstract int operate(int input1, int input2);
5 }
6
```

Define the **implementations**

```
1 package com.ocp;
2
3
4 public class AddOperation implements Operation {
5
```

[interview Q&As with diagrams](#) | [What should be a typical Java EE architecture?](#)

Senior Java developers must have a good handle on

[open all](#) | [close all](#)

- [Best Practice \(6\)](#)
- [Coding \(26\)](#)
- [Concurrency \(6\)](#)
- [Design Concepts \(7\)](#)
- [Design Patterns \(11\)](#)
- [Exception Handling \(3\)](#)
- [Java Debugging \(21\)](#)
- [Judging Experience Level \(1\)](#)
- [Low Latency \(7\)](#)
- [Memory Management \(1\)](#)
- [Performance \(13\)](#)
- [QoS \(8\)](#)
- [Scalability \(4\)](#)
- [SDLC \(6\)](#)
- [Security \(13\)](#)
- [Transaction Management \(1\)](#)

80+ step by step Java Tutorials

[open all](#) | [close all](#)

- [Setting up Tutorial \(6\)](#)
- [Tutorial - Diagnosis \(2\)](#)
- [Akka Tutorial \(9\)](#)
- [Core Java Tutorials \(2\)](#)
- [Hadoop & Spark Tutorial \(1\)](#)

```

6  @Override
7  public int operate(int input1, int input2) {
8      return input1 + input2;
9  }
10
11 }
12

```

```

1  package com.ocp;
2
3  public class MultiplyOperation implements Operat
4
5  @Override
6  public int operate(int input1, int input2) {
7      return input1 * input2;
8  }
9
10 }
11

```

Finally, the **JUnit** test class

```

1  package com.ocp;
2
3  import junit.framework.Assert;
4
5  import org.junit.Before;
6  import org.junit.Test;
7
8  public class MathOperation2Test {
9
10     Operation operation;
11
12     @Test
13     public void testAddition() {
14         operation = new AddOperation();
15         Assert.assertEquals(8, operation.operate(5, 3)
16     }
17
18     @Test
19     public void testMultiplication() {
20         operation = new MultiplyOperation();
21         Assert.assertEquals(15, operation.operate(5, 3)
22     }
23
24 }
25

```

This is only a trivial example, but in real life applications, **wherever you have large if/else statements, you need to think if OCP can be applied.** Spring framework promotes this principle.

Popular Posts

- [JEE Tutorials \(19\)](#)
- [Scala Tutorials \(1\)](#)
- [Spring & Hibernate Tutorials \(1\)](#)
- [Tools Tutorials \(19\)](#)
- [Other Tutorials \(45\)](#)

Preparing for Java written & coding tests

[open all](#) | [close all](#)

- [◆ Complete the given code](#)
- [Can you write code? | Part 1](#)
- [Converting from A to B](#)
- [Designing your classes](#)
- [Java Data Structures](#)
- [Passing the unit tests](#)
- [What is wrong with this code?](#)
- [Writing Code Home Assignment](#)
- [Written Test Core Java](#)
- [Written Test JEE \(1\)](#)

How good are your...to go places?

[open all](#) | [close all](#)

- [Career Making Knowledge](#)
- [Job Hunting & Resumes](#)

[♦ 11 Spring boot interview questions & answers](#)

861 views

[♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers](#)

829 views

[18 Java scenarios based interview Questions and Answers](#)

448 views

[001A: ♦ 7+ Java integration styles & patterns interview questions & answers](#)

407 views

[♦ 7 Java debugging interview questions & answers](#)

311 views

[♦ 10 ERD \(Entity-Relationship Diagrams\) Interview Questions and Answers](#)

303 views

[01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers](#)

294 views

[01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints](#)

288 views

[♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers](#)

263 views

[8 Git Source control system interview questions & answers](#)

215 views

Bio

Latest Posts

**Arulkumaran
Kumaraswamipillai**

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via [Amazon.com](#) in 2005, and sold 35,000+ copies. Books are



outdated and replaced with this subscription based site.



About [Arulkumaran Kumaraswamipillai](#)

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via [Amazon.com](#) in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.

◀ 3 scenarios to get handle on Java generics

◆ 17 Java Coding Tips for job interviews and pre-interview coding

tests ▶

Posted in Design Concepts, member-paid, OOP

Leave a Reply

[Logged in as geethika.](#) [Log out?](#)

Comment

Post Comment

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ [Java generics in no time](#) ☀ [Top 6 tips to transforming your thinking from OOP to FP](#) ☀ [How does a HashMap internally work? What is a hashing function?](#)
☀ [10+ Java String class interview Q&As](#) ☀ [Java auto un/boxing benefits & caveats](#) ☀ [Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect](#)

Non-Technical Know Hows

☀ [6 Aspects that can motivate you to fast-track your career & go places](#) ☀ [Are you reinventing yourself as a Java developer?](#) ☀ [8 tips to safeguard your Java career against offshoring](#) ☀ [My top 5 career mistakes](#)

Prepare to succeed

☀ [Turn readers of your Java CV go from “Blah blah” to “Wow”?](#) ☀ [How to prepare for Java job interviews?](#) ☀ [16 Technical Key Areas](#) ☀ [How to choose from multiple Java job offers?](#)

Select Category ▼

© Disclaimer

The contents in this Java-Success are copy righted. The author has the right to correct or enhance the current content without any prior notice.

These are general advice only, and one needs to take his/her own circumstances into consideration. The author will not be held liable for any damages caused or alleged to be caused either directly or indirectly by these materials and resources. Any trademarked names or labels used in this blog remain the property of their respective trademark owners. No guarantees are made regarding the accuracy or usefulness of content, though I do make an effort to be accurate. Links to external sites do not imply endorsement of the linked-to sites.