Register | Login | Logout | Contact Us

Java-Success.com

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

search here … Go

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 › Written Tests › Can you write code? › 08: ♦ Write code to add, subtract, multiply, and divide given numbers?

08: ♦ Write code to add, subtract, multiply, and divide given numbers?

Posted on March 10, 2015 by Arulkumaran Kumaraswamipillai

A trivial coding example (i.e. a Calculator) tackled using the following **programming paradigms** in Java not only to perform well in coding interviews, but also to learn these programming paradigms.

Approach 1: Procedural Programming
Approaches 2 – 4: Object Oriented Programming
Approach 5: Functional Programming (Java 8)

Approach 1: Procedural

```
public interface Calculate {
```

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 (
  - Java Overview (4)
  - Data types (6)
  - constructors-methc
  - Reserved Key Wor
  - Classes (3)
  - Objects (8)
  - OOP (10)
    - ♥ Design princip
    - ♦ 30+ FAQ Java

```
    abstract int calculate(int operand1, int oeran
}
```

```
public enum Operator {
    ADD, SUBTRACT, DIVIDE, MULTIPLY;
}
```

```
public class CalculateImpl implements Calculate

    @Override
    public int calculate(int operand1, int opera

        switch (operator) {
        case ADD:
            return operand1 + operand2;
        case SUBTRACT:
            return operand1 - operand2;
        case MULTIPLY:
            return operand1 * operand2;
        case DIVIDE:
            return operand1 / operand2;
        }

        throw new RuntimeException(operator + "i
    }

}
```

```
public class CalculatorTest {
    public static void main(String[] args) {
       Calculate calc = new CalculateImpl();
       int result = calc.calculate(5,6,Operator..
       result = calc.calculate(result,6,Operator
       result = calc.calculate(result,1,Operator
       result = calc.calculate(result,5,Operator

       System.out.println("result=" + result);
    }
}
```

Output: result=13

Approach 2: OOP

```
public interface MathCommand<E> {
     abstract E execute(E operand1, E operand2);
}
```

```
public class AddCommand implements MathCommand<In

     @Override
     public Integer execute(Integer operand1, Inte
```

♦ Why favor com
08: ♦ Write code
Explain abstracti
How to create a
Top 5 OOPs tips
Top 6 tips to go a
Understanding C
What are good r
⊞ GC (2)
⊞ Generics (5)
⊞ FP (8)
⊞ IO (7)
⊞ Multithreading (12)
⊞ Algorithms (5)
⊞ Annotations (2)
⊞ Collection and Data
⊞ Differences Betwee
⊞ Event Driven Progr
⊞ Exceptions (2)
⊞ Java 7 (2)
⊞ Java 8 (24)
⊞ JVM (6)
⊞ Reactive Programn
⊞ Swing & AWT (2)
⊞ JEE Interview Q&A (3
⊞ Pressed for time? Jav
⊞ SQL, XML, UML, JSC
⊞ Hadoop & BigData Int
⊞ Java Architecture Inte
⊞ Scala Interview Q&As
⊞ Spring, Hibernate, & I
⊞ Testing & Profiling/Sa
⊞ Other Interview Q&A f
⊞ ▶ Free Java Interview

As a Java Architect

Java architecture & design concepts

```
        return operand1 + operand2;
    }
}
```

```
public class SubtractCommand implements MathComma

    @Override
    public Integer execute(Integer operand1, Inte
        return operand1 - operand2;
    }
}
```

```
public class MultiplyCommand implements MathComma

    @Override
    public Integer execute(Integer operand1, Inte
        return operand1 * operand2;
    }
}
```

```
public class DivideCommand implements MathCommand

    @Override
    public Integer execute(Integer operand1, Inte
        return operand1 / operand2;
    }
}
```

```
public class CalculatorTest2 {
    public static void main(String[] args) {
        MathCommand<Integer> command = new AddCo
        Integer result = command.execute(5, 6);
        command = new MultiplyCommand();
        result = command.execute(result, 6);
        command = new SubtractCommand();
        result = command.execute(result, 1);
        command = new DivideCommand();
        result = command.execute(result, 5);

        System.out.println("result=" + result);
    }
}
```

When you have more mathematical operations, add more command classes. In OOP, switch statements are unsightly and hard to maintain. The above OOP approach eliminates the need for switches. This is also a good example of the "Open-Close design principle".

Output: result=13

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 I
- ⊞ Low Latency (7)
- ⊞ Memory Managemen
- ⊞ Performance (13)
- ⊞ QoS (8)
- ⊞ Scalability (4)
- ⊞ SDLC (6)
- ⊞ Security (13)
- ⊞ Transaction Managen

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 Tuto

Approach 3: OOP

This extends **approach-2** to make the client code more elegant to use with “*”, “+”, etc.

```
import java.util.HashMap;
import java.util.Map;

public final class Calculator {

    private static final Map<Character, MathComm
                                             ne

    public Calculator() {
        init();
    }

    public void init() {
        mapOperations.put('+', new AddCommand())
        mapOperations.put('*', new MultiplyComma
        mapOperations.put('-', new SubtractComma
        mapOperations.put('/', new DivideCommand
    }

    public Integer calc(Character operator, Inte
        MathCommand<Integer> op = mapOperations.
        if (op != null) {
            return op.execute(operand1, operand2
        }
        else {
            throw new RuntimeException(operator 
        }
    }

}
```

```
public class CalculatorTest {
    public static void main(String[] args) {
        Calculator calc = new Calculator();
        Integer result = calc.calc('+', 5, 6);
        result = calc.calc('*', result, 6);
        result = calc.calc('-', result, 1);
        result = calc.calc('/', result, 5);

        System.out.println("result=" + result);
    }
}
```

Output: result=13

Approach 4: OOP

This extends **approach-2** & **approach-3** to make the client code more elegant **“.” notations** [e.g. blah.calc(‘+’, 6).calc(‘*’,

⊞ JEE Tutorials (19)
⊞ Scala Tutorials (1)
⊞ Spring & HIbernate Tu
⊞ Tools Tutorials (19)
⊞ Other Tutorials (45)

Preparing for Java written & coding tests

open all | close all
⊞ ♦ Complete the given
⊞ Can you write code? (
⊞ Converting from A to I
⊞ Designing your classe
⊞ Java Data Structures
⊞ Passing the unit tests
⊞ What is wrong with th
⊞ Writing Code Home A
⊞ Written Test Core Jav
⊞ Written Test JEE (1)

How good are your...to go places?

open all | close all
⊞ Career Making Know-
⊞ Job Hunting & Resum

6).calc('-', 1).blah] with the help of "Builder" design pattern.

```java
import java.util.HashMap;
import java.util.Map;

public final class Calculator {

    Integer result = 0;

    private static final Map<Character, MathComm
                                   new HashMap<C

    public Calculator(CalculationBuilder builder
        this.result = builder.result;
    }

    public Integer getResult(){
        return this.result;
    }

    //inner static class applying the builder de
    public static class CalculationBuilder {

        protected Integer result;

        CalculationBuilder (Integer result){
            init();
            this.result = result;
        }

        public void init() {
            mapOperations.put('+', new AddComman
            mapOperations.put('*', new MultiplyC
            mapOperations.put('-', new SubtractC
            mapOperations.put('/', new DivideCom
        }

        CalculationBuilder calc(Character operat
            MathCommand<Integer> op = mapOperati
            if (op != null) {
                this.result = op.execute(result,
            } else {
                throw new RuntimeException(opera
            }
            return this;
        }

    }

}
```

```java
public class CalculatorTest {
    public static void main(String[] args) {
        //more elegant to build mathematical ope
        Calculator.CalculationBuilder calcBuilde
                                              .c
                                              .c
                                              .c
                                              .c
        Calculator calc = new Calculator(calcBui
        System.out.println("result=" + calc.getR
    }
}
```

Output: result=13

Approach 5: FP

Java 8 functional programming. You can see Lambdas, functional interfaces, default methods, and static methods in action.

```
package com.java8.examples;

import java.util.function.BinaryOperator;
import java.util.function.Function;
import java.util.Objects;

@FunctionalInterface
public interface MathOperation<Intetger>  {

    //SAM -- Single Abstract Method.
    //identifier abstract is optional
    Integer operate(Integer operand);

    default MathOperation<Integer> add(Integer o
       return (o1) -> operate(o1) +   o;
    }

    default MathOperation<Integer> multiply(Inte
         return (o1) -> operate(o1) * o;
    }

    default MathOperation<Integer> subtract(Inte
        return (o1) -> operate(o1) -   o;
     }

     default MathOperation<Integer> divide(Integ
          return (o1) -> operate(o1) / o;
     }

    default Integer getResult() {
       return operate(0);
    }

    default void print(){
        System.out.println("result=" + getResult

    }

    //static helper to initialize
    static Integer init(Integer input) {
        return input  ;
    }

}
```

```
package com.java8.examples;
```

```

public class CalculatorTest {

    public static void main(String[] args) {

        //An expressive static helper method
        MathOperation<Integer> calc = (x) -> Mat

        MathOperation<Integer> complexOp = calc.
                     .multiply(6)
                     .subtract(1)
                     .divide(5);

        complexOp.print();
    }

}
```

Output: result=13

This is a very trivial example, and some solutions could be bit of an over kill.

Q. Which one would you favor, and why?
Q. Would you provide a different solution?

Popular Posts

- ♦ 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.

‹ Explain abstraction, encapsulation, Inheritance, and polymorphism with the given code?

♥ What are the 16 technical key areas of Java programming to fast-track your career? ›

Posted in Can you write code?, Coding, Java 8, OOP

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.

↑