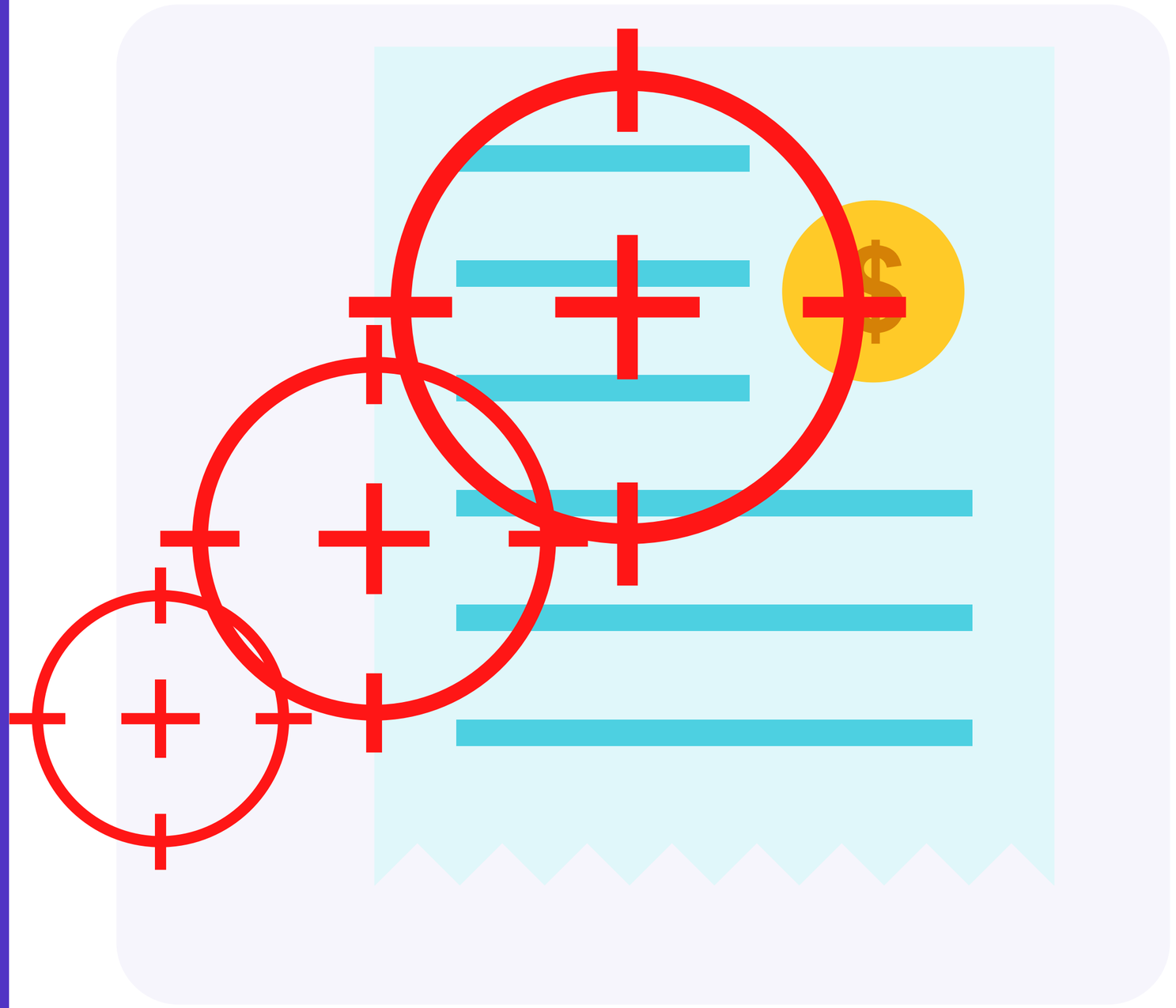
2022 – SOFTWARE ARCHITECTURE AND DESIGN

Killbill Project

https://github.com/killbill/killbill

What is KillBill



Topics

1. Architecture
2. Quality Attribues
3. Design Pattern

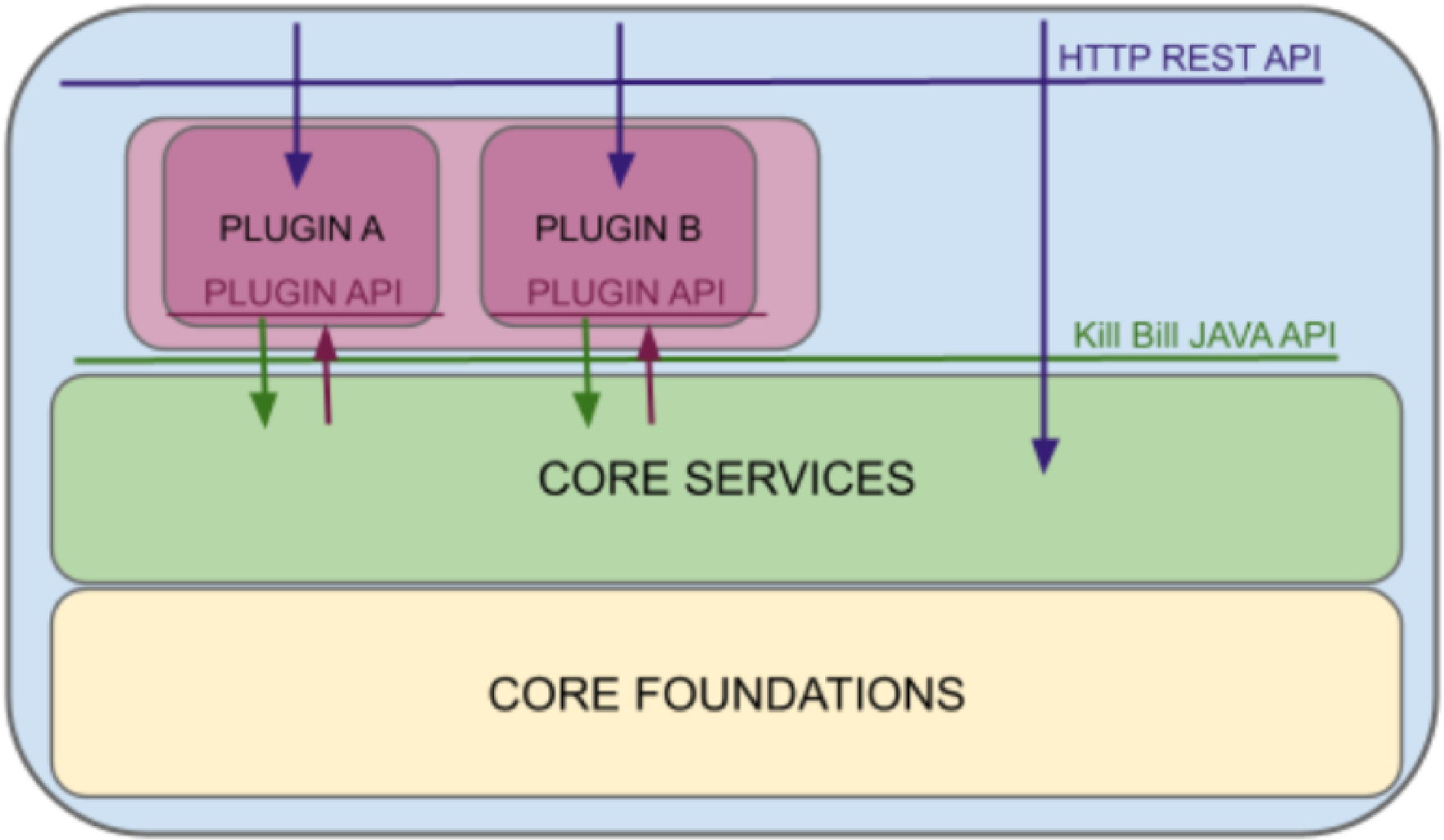
KillBill Architecture

Microkernel Architecture

Logic ของตัวระบบ Kill Bill ถูกแบ่งระหว่าง plug-in components ที่เป็นอิสระต่อกันและ core system ที่ให้ความสามารถในการปรับขยาย, การปรับปรุงแก้ไขเพิ่มเติมและการแยก logic การประมวลผลที่สามารถกำหนดเองได้

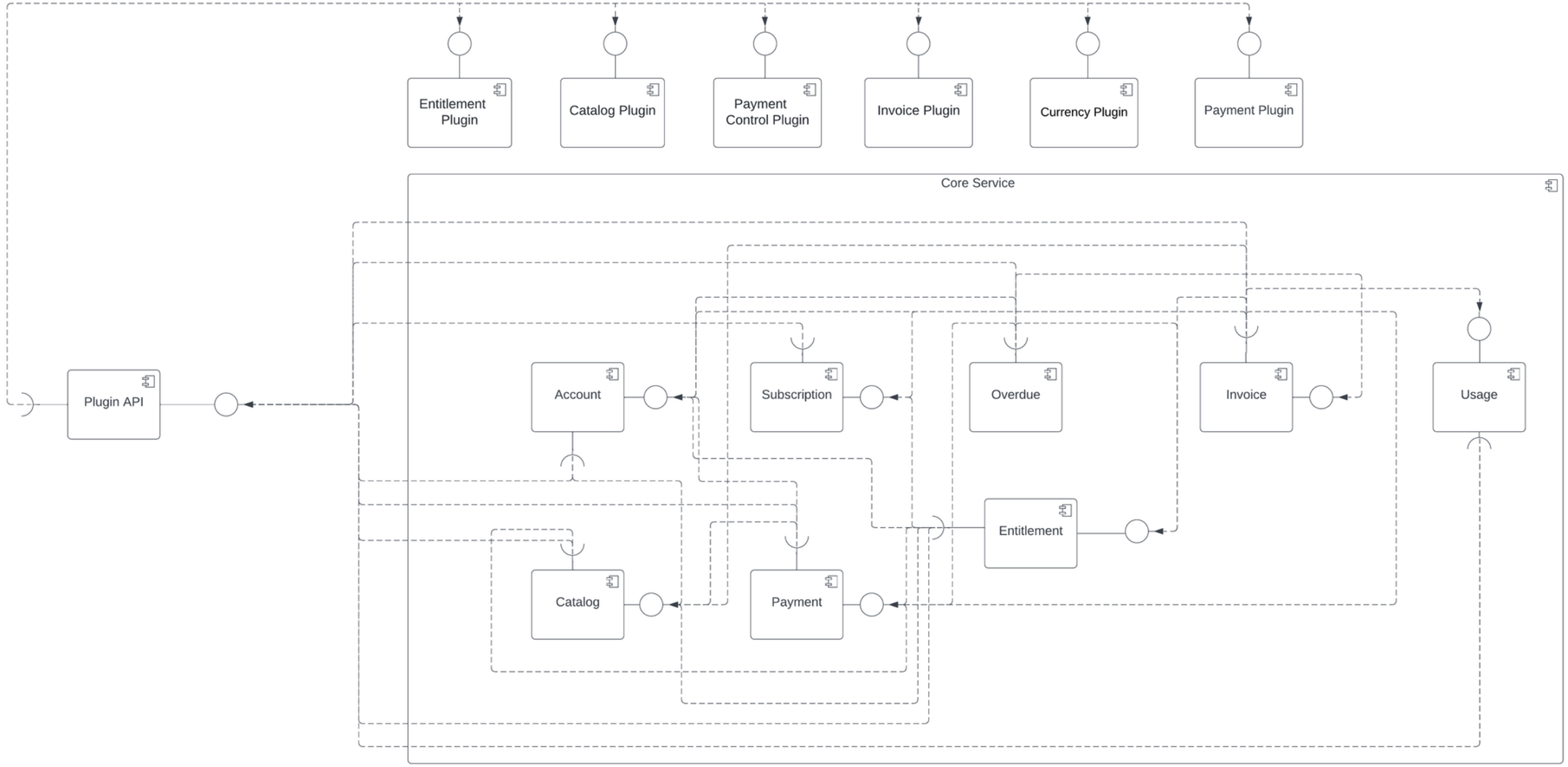
[Arch.] Diagram

Killbill Architecture



[Arch.] Diagram

Killbill Architecture



ระบบ **Kill Bill** รองรับชุดปลั๊กอิน **API** อะไรบ้าง?

API ปลั๊กอินการชำระเงิน:

API เหล่านั้นถูกใช้โดยปลั๊กอินที่โต้ตอบกับเกตเวย์การชำระเงินของบุคคลที่สามหรือตัวประมวลผลการชำระเงิน

API ปลั๊กอินสกุลเงิน:

ด้วยการใช้ API นี้ ปลั๊กอินจะให้ตรรกะทางธุรกิจสำหรับการแปลงสกุลเงิน ซึ่งจากนั้นจะมีให้สำหรับปลั๊กอินการชำระเงินสำหรับการแปลงสกุลเงินตามเวลาจริง

ปลั๊กอินใบแจ้งหนี้ API:

ปลั๊กอินที่ใช้ API นี้จะถูกเรียกระหว่างการสร้างใบแจ้งหนี้เมื่อรหัสการออกใบแจ้งหนี้ของ Kill Bill คำนวณรายการแต่ก่อนที่ใบแจ้งหนี้จะคงอยู่ เพื่อให้ปลั๊กอินสามารถแก้ไขรายการ (เช่น เพื่อเพิ่มรายการภาษีเพิ่มเติม)

ระบบ **Kill Bill** รองรับชุดปลั๊กอิน **API** อะไรบ้าง?

API ปลั๊กอินควบคุมการชำระเงิน:

จุดประสงค์เพื่อสกัดกั้นการชำระเงินและเป็นผลให้ยกเลิก (กรณีการใช้งานที่เป็นการฉ้อโกง) หรือเปลี่ยนเส้นทางไปยังปลั๊กอินการชำระเงินอื่น (การหยุดทำงานของเกตเวย์การชำระเงิน หรือสถานการณ์การเพิ่มประสิทธิภาพต้นทุน)

API ปลั๊กอินของแค็ตตาล็อก:

หากการกำหนดค่าแค็ตตาล็อกแบบ XML ไม่ตรงกับความต้องการของคุณ คุณสามารถใช้กลไกจัดการแค็ตตาล็อกของคุณเองได้ผ่าน API นี้

API ปลั๊กอินการให้สิทธิ์:

API นี้ช่วยให้คุณสกัดกั้น API การให้สิทธิ์ (เช่น การสร้างหรือเปลี่ยนแปลงการสมัครรับข้อมูล) เพื่อแทรกตรรกะทางธุรกิจการสมัครรับข้อมูลของคุณเอง (กรณีการใช้งานรวมถึงการจัดการคูปองหรือการแทนที่ราคา)

KillBill

Quality Attributes

จุดเด่น

Killbill Architecture

PORTABILITY

Microkernel สามารถทำงานได้ในสภาพแวดล้อมที่หลากหลายโดยทำการเพิ่ม plug-in เข้าไปที่ระบบ

SECURITY

KillBill มีการจัดการสิทธิ์การเข้าใช้งานซึ่งออกแบบตาม Role-based access control (RBAC) ซึ่งเป็นการควบคุมการเข้าใช้งานในส่วนต่างๆ โดยเมื่อมีการเรียกใช้งานในแต่ละส่วน RBAC จะคอยเช็คผู้ใช้งานแต่ละคนว่า มีสิทธิ์ที่จะเข้าใช้งานในส่วนนี้หรือไม่

TESTABILITY

KillBill มีการแบ่งระบบออกเป็นหลาย ๆ ส่วนทำงานแยกจากกัน ทำให้เราสามารถทดสอบการทำงานได้อย่างง่ายได้ด้วยการเรียกใช้งานแต่ละส่วนที่ต้องการเพื่อการทำ testing

จุดเด่น (ต่อ)

จุดอ่อน

Killbill Architecture

SCALABILY

โดยทั่วไปแล้วการใช้รูปแบบที่เป็น Microkernel Architecture จะมีความโดดเด่นในการเพิ่ม plugin ได้ โดยไม่กระทบหรือแก้ไขต่อระบบการทำงานของแต่ละส่วน

PERFORMANCE

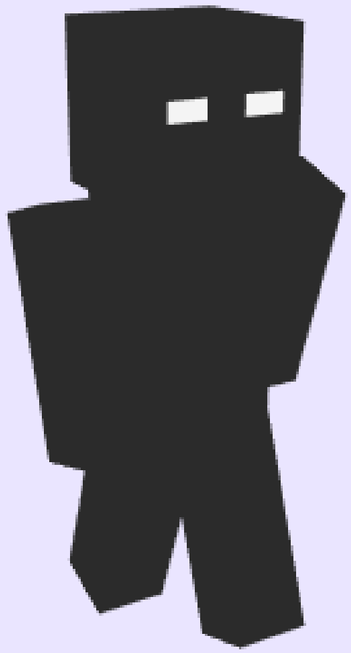
การใช้รูปแบบที่เป็น Microkernel Architecture นั้น clients ไม่สามารถติดต่อ services ได้โดยตรง ทำให้เกิด overhead ในการติดต่อส่วนกลาง (core services) ก่อน ซึ่งจะส่งผลกระทบให้เวลาในการทำงานเพิ่มขึ้น

โดยวิธีแก้ไขนั้น อาจจะเพิ่มขนาด core services ให้มากขึ้น เพื่อให้ มีการประมวลผลได้เร็วขึ้น

Killbill

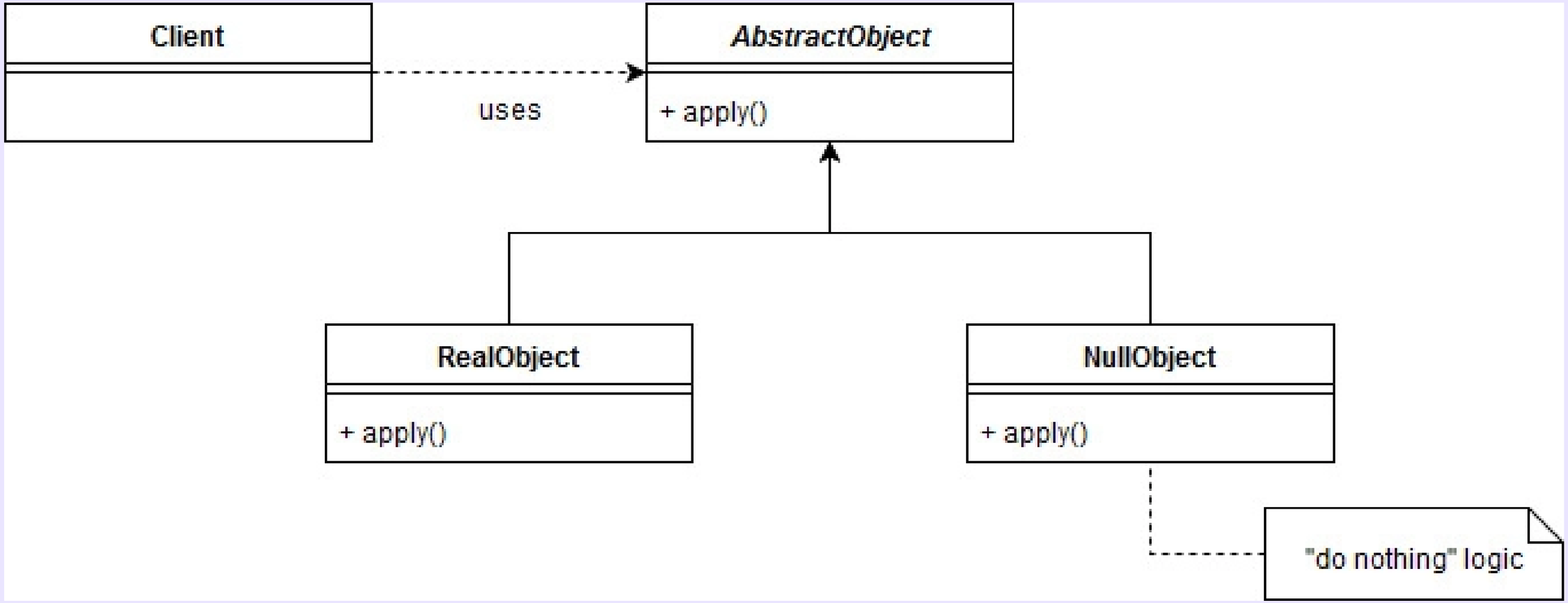
Design Pattern

Builder Adapter Null Object
Abstract Factory Singleton



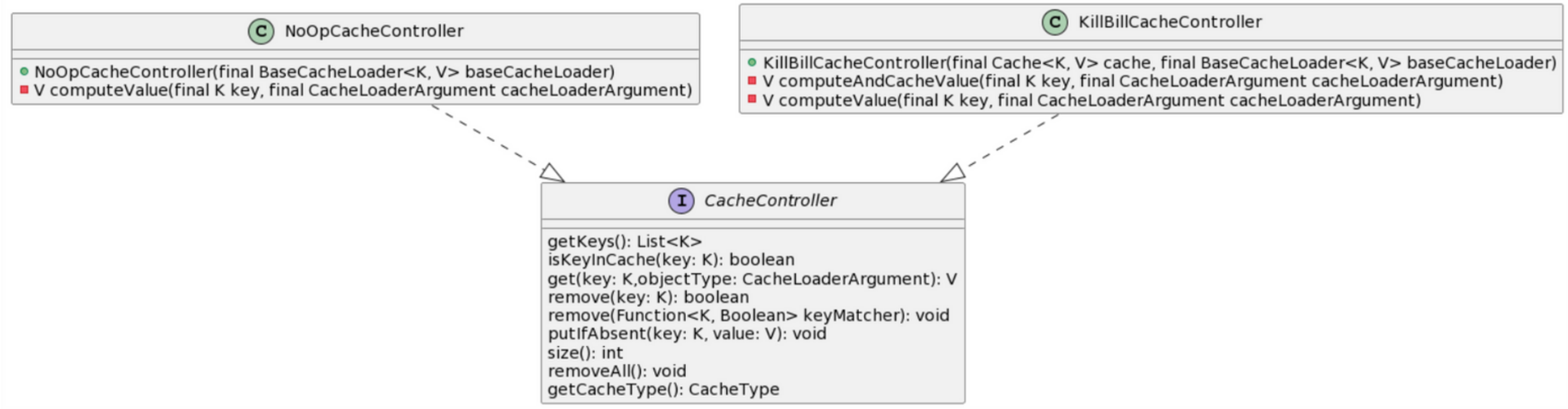
Null Object

Psuedo


Client
AbstractObject
uses
+ apply()
RealObject
+ apply()
NullObject
+ apply()
"do nothing" logic

Null Object

Killbill Design Pattern



UML Diagram

Null Object

class

public interface CacheController <K, V>

```java
public interface CacheController<K, V> {

    List<K> getKeys();
    boolean isKeyInCache(K key);
    V get(K key, CacheLoaderArgument objectType);
    boolean remove(K key);
    void remove(Function<K, Boolean> keyMatcher);
    void putIfAbsent(final K key, V value);
    int size();
    void removeAll();
    CacheType getCacheType();
}
```

Null Object

NoOpCacheController

method

private V computeValue(final K key,
final CacheLoaderArgument cacheLoaderArgument)

```
    private V computeValue(final K key, final CacheLoaderArgument cacheLoaderArgument) {
        final V value;
        try {
            value = baseCacheLoader.compute(key, cacheLoaderArgument);
        } catch (final Exception e) {
            // Remove noisy log (might be expected, see https://github.com/killbill/killbill/issues/842)
            //logger.warn("Unable to compute cached value for key='{}' and cacheLoaderArgument='{}'", key, cacheLoaderArgument, e);
            throw new RuntimeException(e);
        }
        return value;
    }
}
```

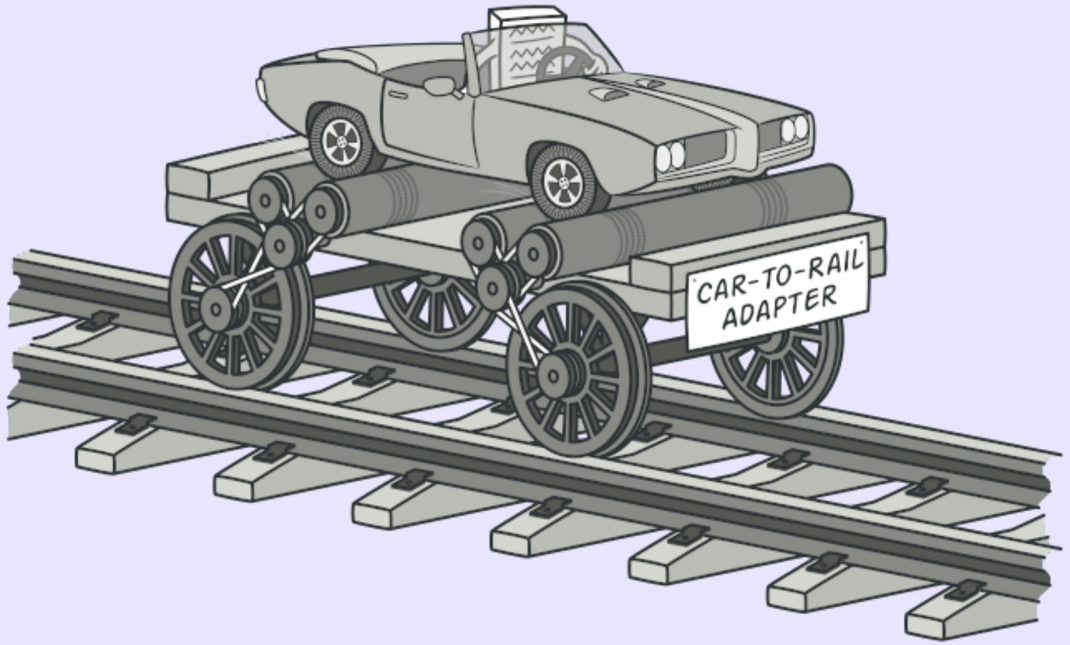
Null Object

KillBillCacheController

method

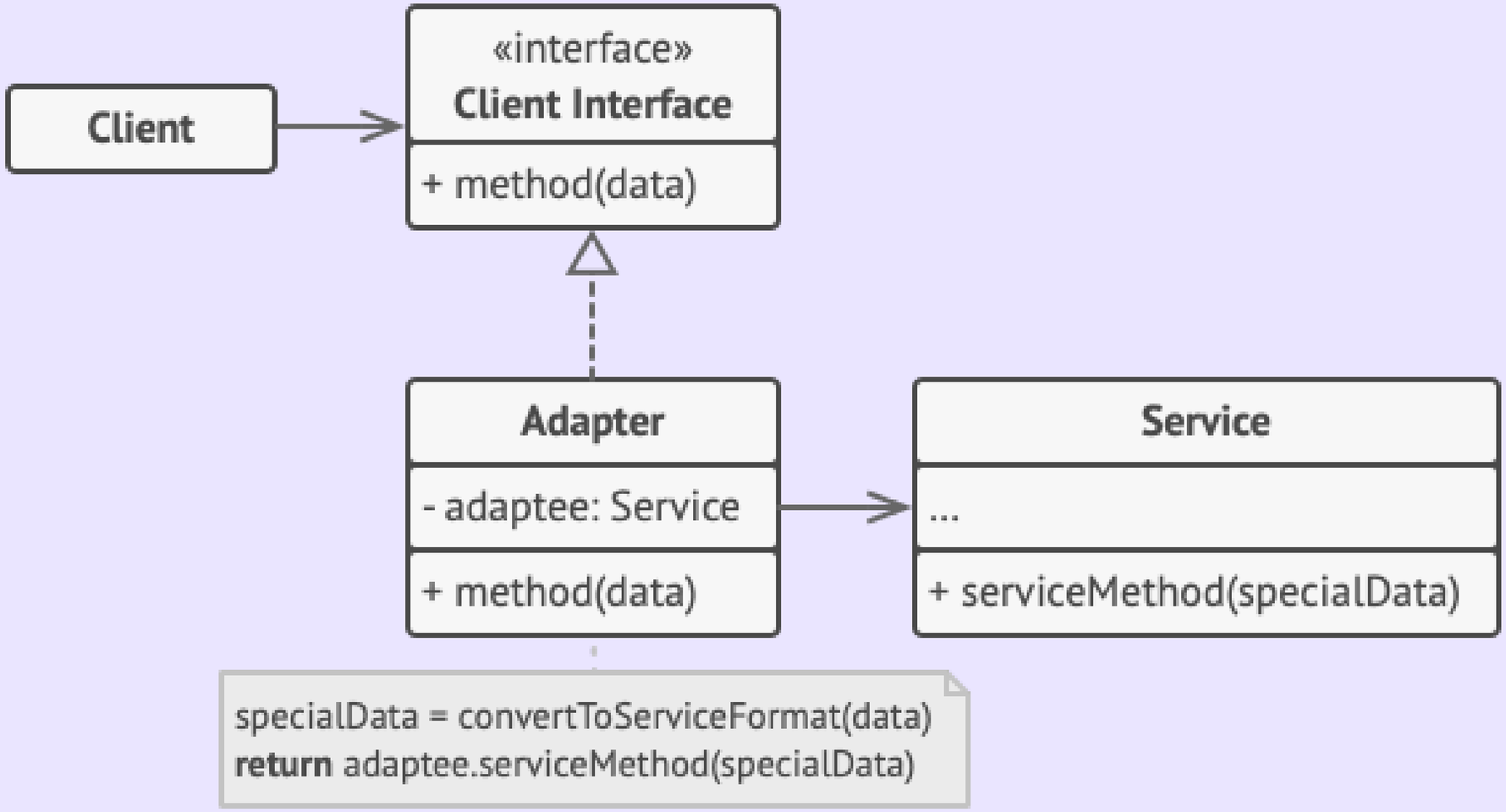
private V computeValue(final K key,
final CacheLoaderArgument cacheLoaderArgument)

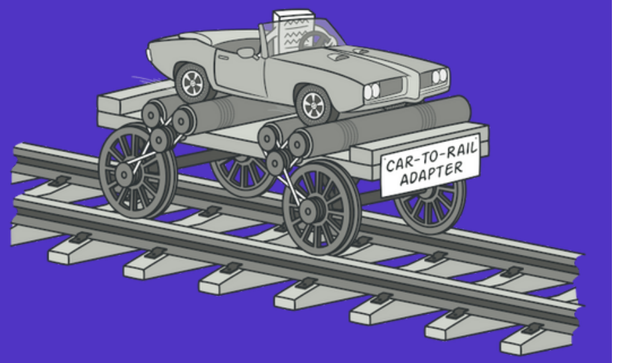
```
    private V computeValue(final K key, final CacheLoaderArgument cacheLoaderArgument) {
        final V value;
        try {
            value = baseCacheLoader.compute(key, cacheLoaderArgument);
        } catch (final Exception e) {
            // Remove noisy log (might be expected, see https://github.com/killbill/killbill/issues/842)
            //logger.warn("Unable to compute cached value for key='{}' and cacheLoaderArgument='{}'", key, cacheLoaderArgument, e);
            throw new RuntimeException(e);
        }
        return value;
    }
}
```



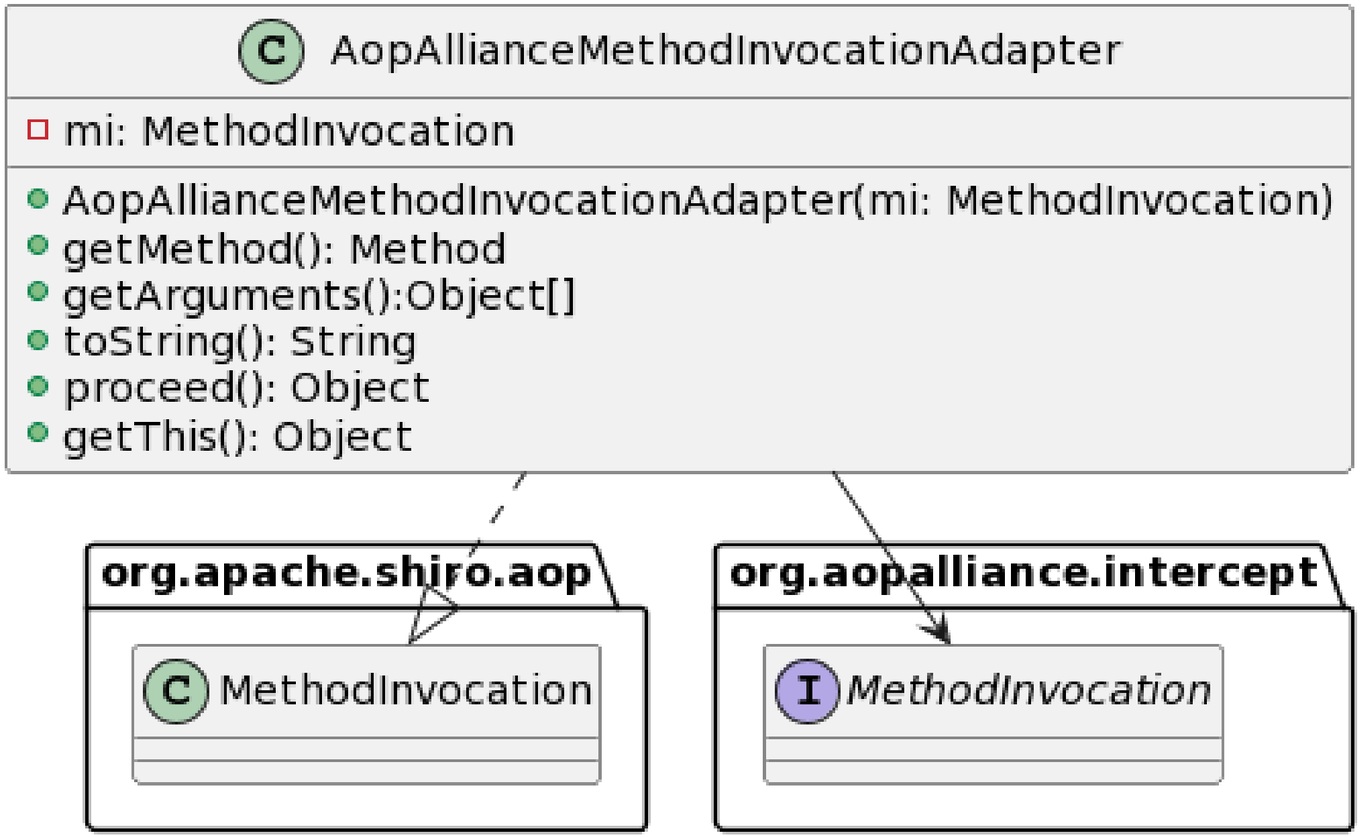
Adapter

Psuedo


«interface»
Client Interface
+ method(data)
Client
Adapter
- adaptee: Service
+ method(data)
Service
...
+ serviceMethod(specialData)
specialData = convertToServiceFormat(data)
return adaptee.serviceMethod(specialData)

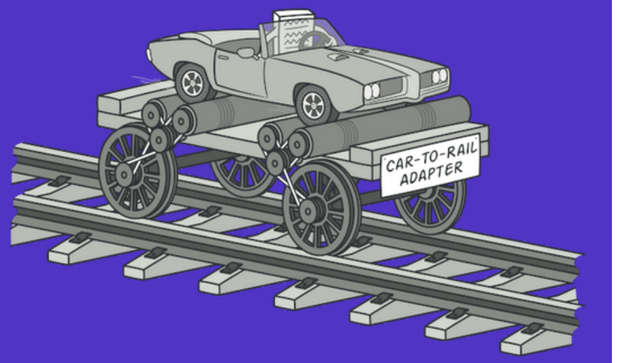


Adapter



UML Diagram

2



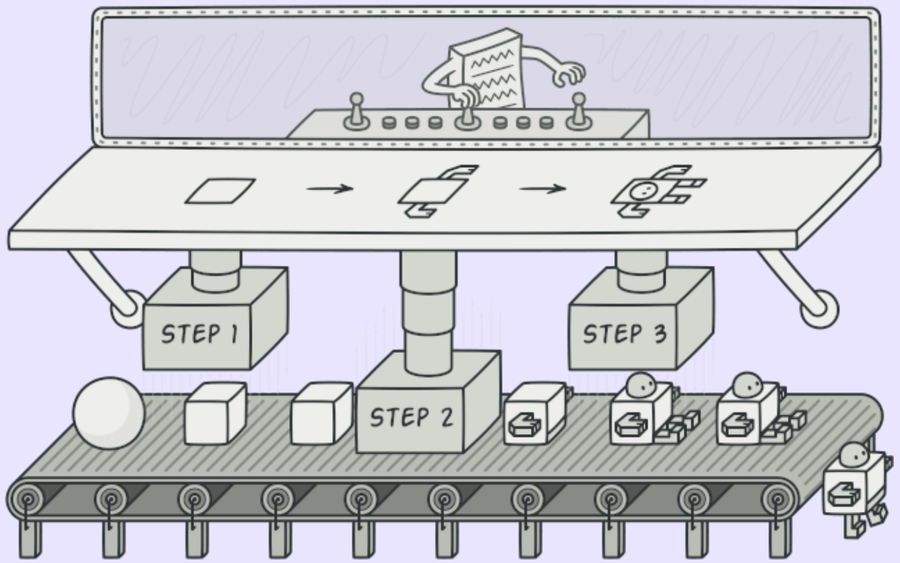
Adapter

class

public class AopAllianceMethodInvocationAdapter
implements org.apache.shiro.aop.MethodInvocation

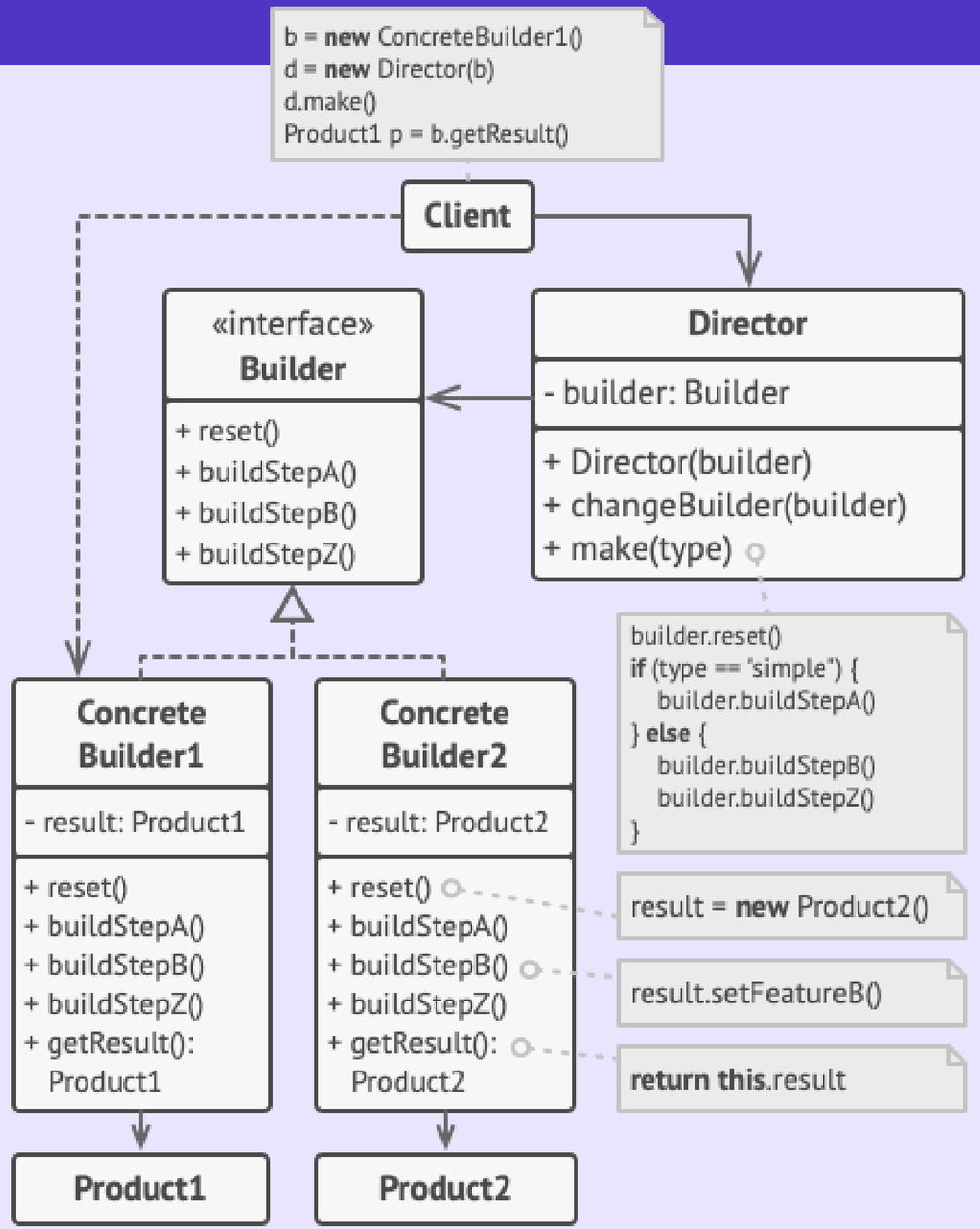
```java
import org.aopalliance.intercept.MethodInvocation;

// Taken from Shiro - the original class is private :(
public class AopAllianceMethodInvocationAdapter implements org.apache.shiro.aop.MethodInvocation {

    private final MethodInvocation mi;

    public AopAllianceMethodInvocationAdapter(final MethodInvocation mi) {
        this.mi = mi;
    }

    public Method getMethod() {
        return mi.getMethod();
    }

    public Object[] getArguments() {
        return mi.getArguments();
    }

    public String toString() {
        return "Method invocation [" + mi.getMethod() + "]";
    }

    public Object proceed() throws Throwable {
        return mi.proceed();
    }

    public Object getThis() {
        return mi.getThis();
    }
}
```

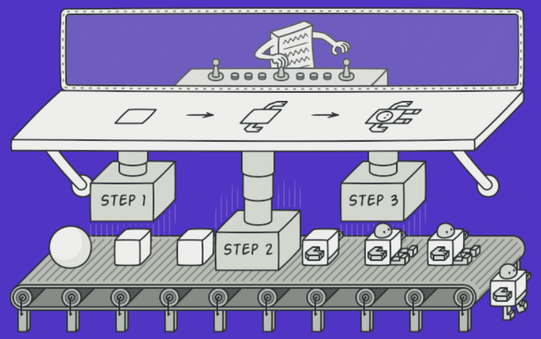
Killbill Design Pattern



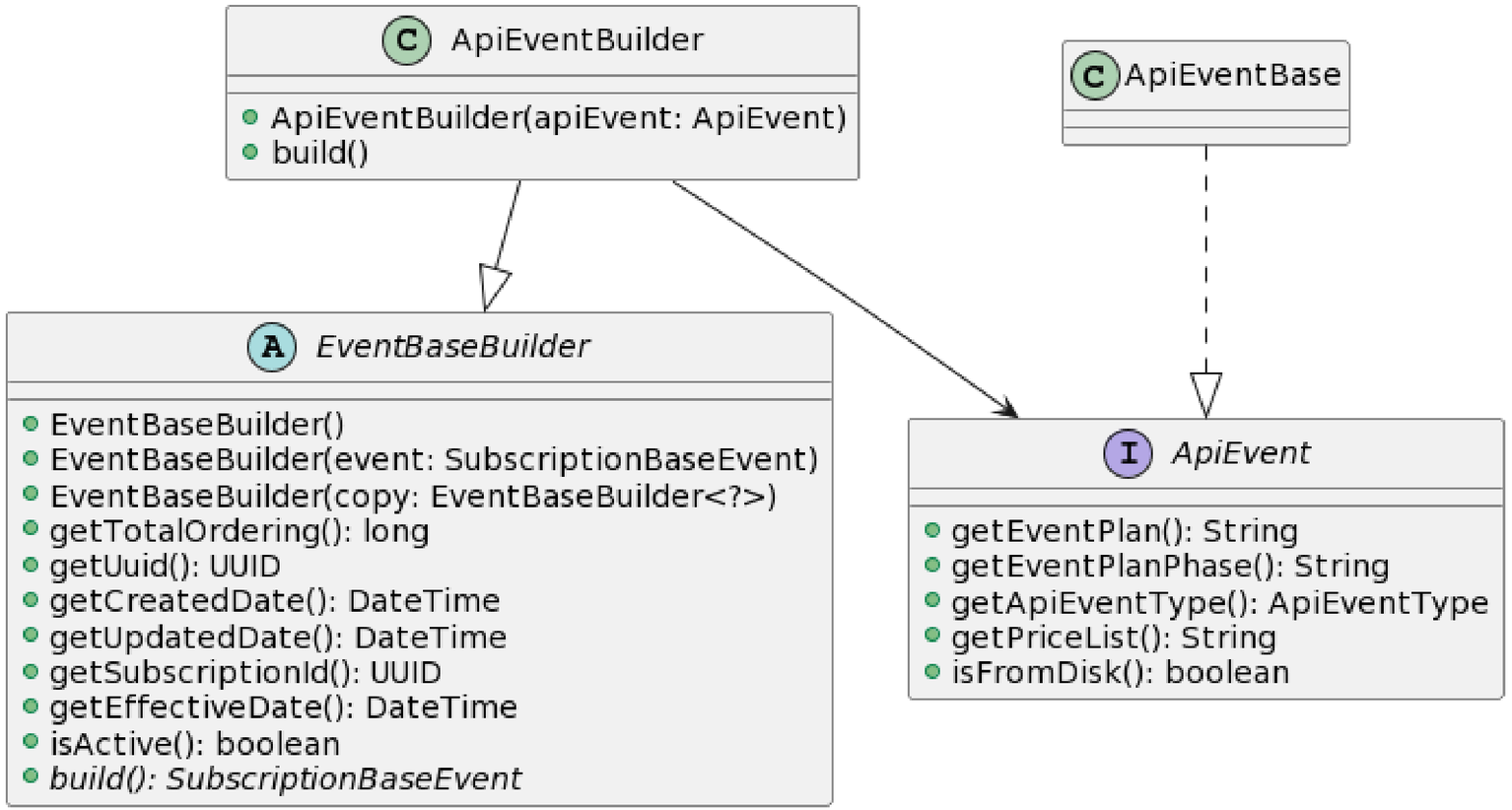
Builder

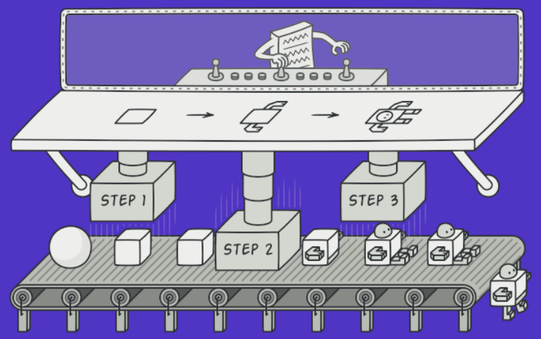
Pseudo


b = new ConcreteBuilder1()
d = new Director(b)
d.make()
Product1 p = b.getResult()
Client
«interface»
Builder
+ reset()
+ buildStepA()
+ buildStepB()
+ buildStepZ()
Director
- builder: Builder
+ Director(builder)
+ changeBuilder(builder)
+ make(type)
builder.reset()
if (type == "simple") {
builder.buildStepA()
} else {
builder.buildStepB()
builder.buildStepZ()
}
Concrete
Builder1
- result: Product1
+ reset()
+ buildStepA()
+ buildStepB()
+ buildStepZ()
+ getResult():
Product1
Concrete
Builder2
- result: Product2
+ reset()
+ buildStepA()
+ buildStepB()
+ buildStepZ()
+ getResult():
Product2
result = new Product2()
result.setFeatureB()
return this.result
Product1
Product2


STEP 1
STEP 2
STEP 3

Builder


ApiEventBuilder
ApiEventBuilder(apiEvent: ApiEvent)
build()
ApiEventBase
EventBaseBuilder
EventBaseBuilder()
EventBaseBuilder(event: SubscriptionBaseEvent)
EventBaseBuilder(copy: EventBaseBuilder<?>)
getTotalOrdering(): long
getUuid(): UUID
getCreatedDate(): DateTime
getUpdatedDate(): DateTime
getSubscriptionId(): UUID
getEffectiveDate(): DateTime
isActive(): boolean
build(): SubscriptionBaseEvent
ApiEvent
getEventPlan(): String
getEventPlanPhase(): String
getApiEventType(): ApiEventType
getPriceList(): String
isFromDisk(): boolean

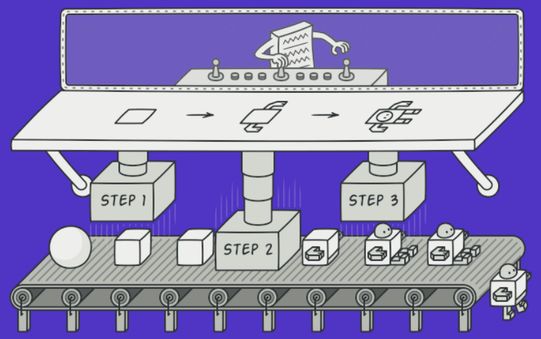


Builder

abstract class

public abstract class EventBaseBuilder
<T extends EventBaseBuilder<T>>


```java
public abstract class EventBaseBuilder<T extends EventBaseBuilder<T>> {

    private long totalOrdering;
    private UUID uuid;
    private UUID subscriptionId;
    private DateTime createdDate;
    private DateTime updatedDate;
    private DateTime effectiveDate;

    private boolean isActive;

    public EventBaseBuilder() {
        this.uuid = UUIDs.randomUUID();
        this.isActive = true;
    }


    public EventBaseBuilder(final SubscriptionBaseEvent event) {
        this.uuid = event.getId();
        this.subscriptionId = event.getSubscriptionId();
        this.effectiveDate = event.getEffectiveDate();
        this.createdDate = event.getCreatedDate();
        this.updatedDate = event.getUpdatedDate();
        this.isActive = event.isActive();
        this.totalOrdering = event.getTotalOrdering();
    }

    public EventBaseBuilder(final EventBaseBuilder<?> copy) {
        this.uuid = copy.uuid;
        this.subscriptionId = copy.subscriptionId;
        this.effectiveDate = copy.effectiveDate;
        this.createdDate = copy.getCreatedDate();
        this.updatedDate = copy.getUpdatedDate();
        this.isActive = copy.isActive;
        this.totalOrdering = copy.totalOrdering;
    }
```

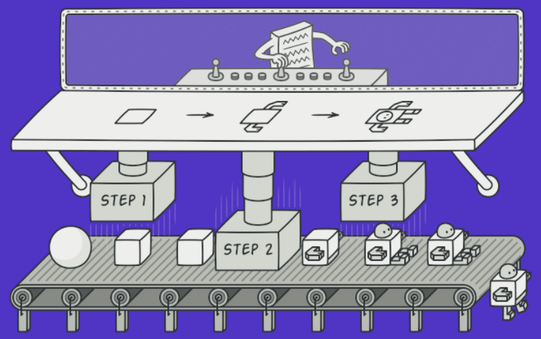


Builder

Api EvenBaseBuilder.java

public ApiEventBase build()

```
91      public ApiEventBase build() {
92          final ApiEventBase result;
93          if (apiEventType == ApiEventType.CREATE) {
94              result = new ApiEventCreate(this);
95          } else if (apiEventType == ApiEventType.TRANSFER) {
96              result = new ApiEventTransfer(this);
97          } else if (apiEventType == ApiEventType.CHANGE) {
98              result = new ApiEventChange(this);
99          } else if (apiEventType == ApiEventType.CANCEL) {
100             result = new ApiEventCancel(this);
101         } else if (apiEventType == ApiEventType.UNCANCEL) {
102             result = new ApiEventUncancel(this);
103         } else if (apiEventType == ApiEventType.UNDO_CHANGE) {
104             result = new ApiEventUndoChange(this);
105         } else {
106             throw new IllegalStateException("Unknown ApiEventType " + apiEventType);
107         }
108         return result;
109     }
110 }
```

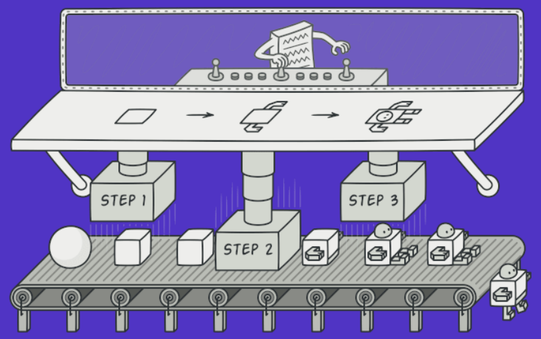


Builder

interface

public interface ApiEvent extends SubscriptionBaseEvent

```java
package org.killbill.billing.subscription.events.user;

import org.killbill.billing.subscription.events.SubscriptionBaseEvent;


public interface ApiEvent extends SubscriptionBaseEvent {

    public String getEventPlan();

    public String getEventPlanPhase();

    public ApiEventType getApiEventType();

    public String getPriceList();

    public boolean isFromDisk();

}
```



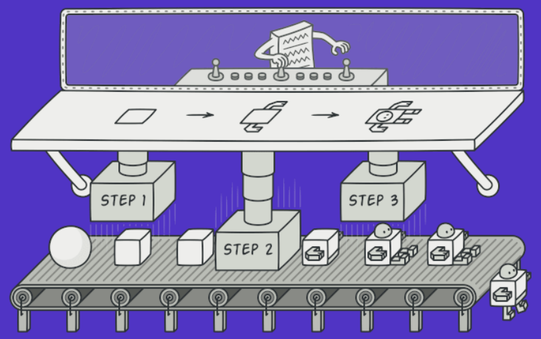
Builder

class

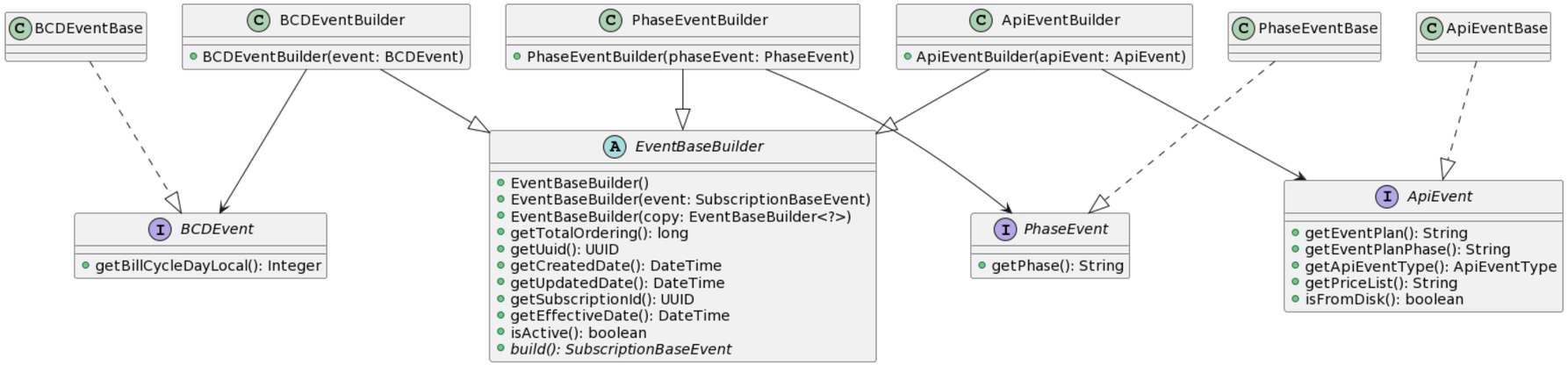
public class ApiEventBase extends EventBase implements ApiEvent

```java
public class ApiEventBase extends EventBase implements ApiEvent {

    private final ApiEventType apiEventType;
    // Only valid for CREATE/CHANGE
    private final String eventPlan;
    private final String eventPlanPhase;
    private final String eventPriceList;
    private final boolean fromDisk;

    public ApiEventBase(final ApiEventBuilder builder) {
        super(builder);
        this.apiEventType = builder.getApiEventType();
        this.eventPriceList = builder.getEventPriceList();
        this.eventPlan = builder.getEventPlan();
        this.eventPlanPhase = builder.getEventPlanPhase();
        this.fromDisk = builder.isFromDisk();
    }

    @Override
    public ApiEventType getApiEventType() {
        return apiEventType;
    }

```

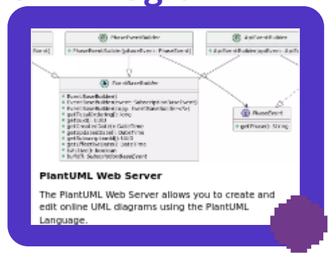
Killbill Design Pattern

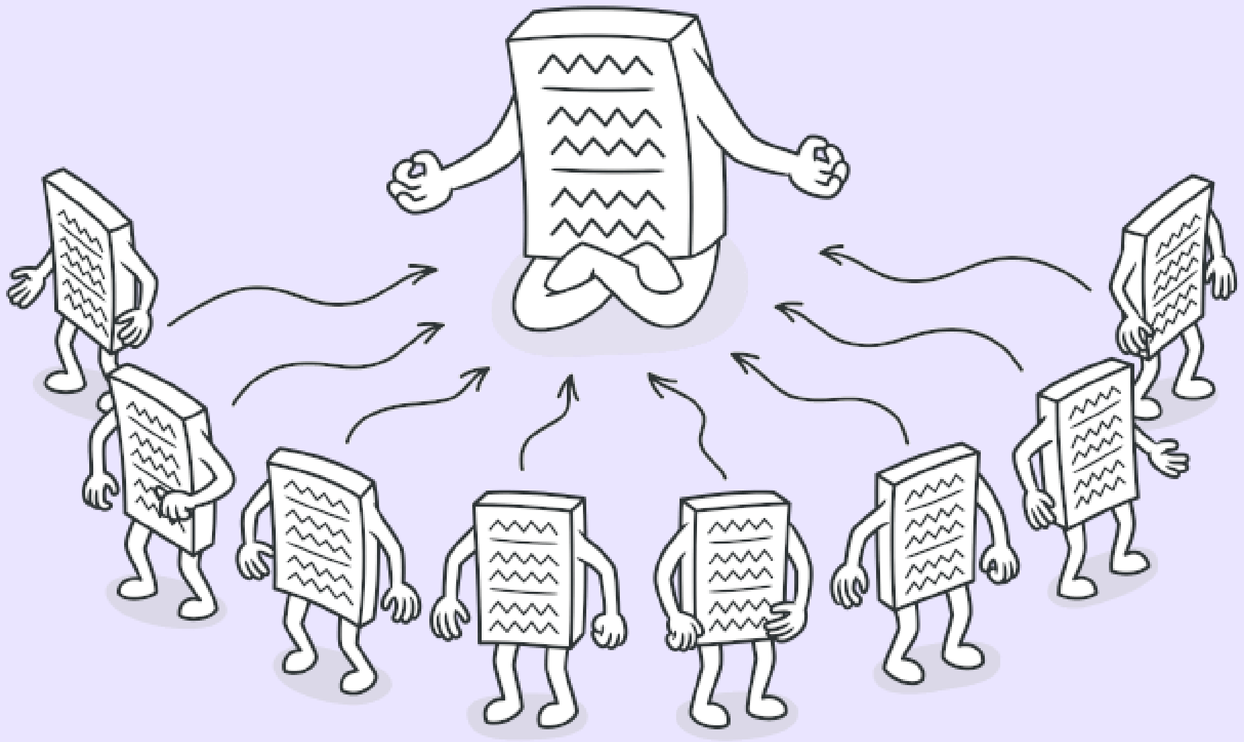


Builder



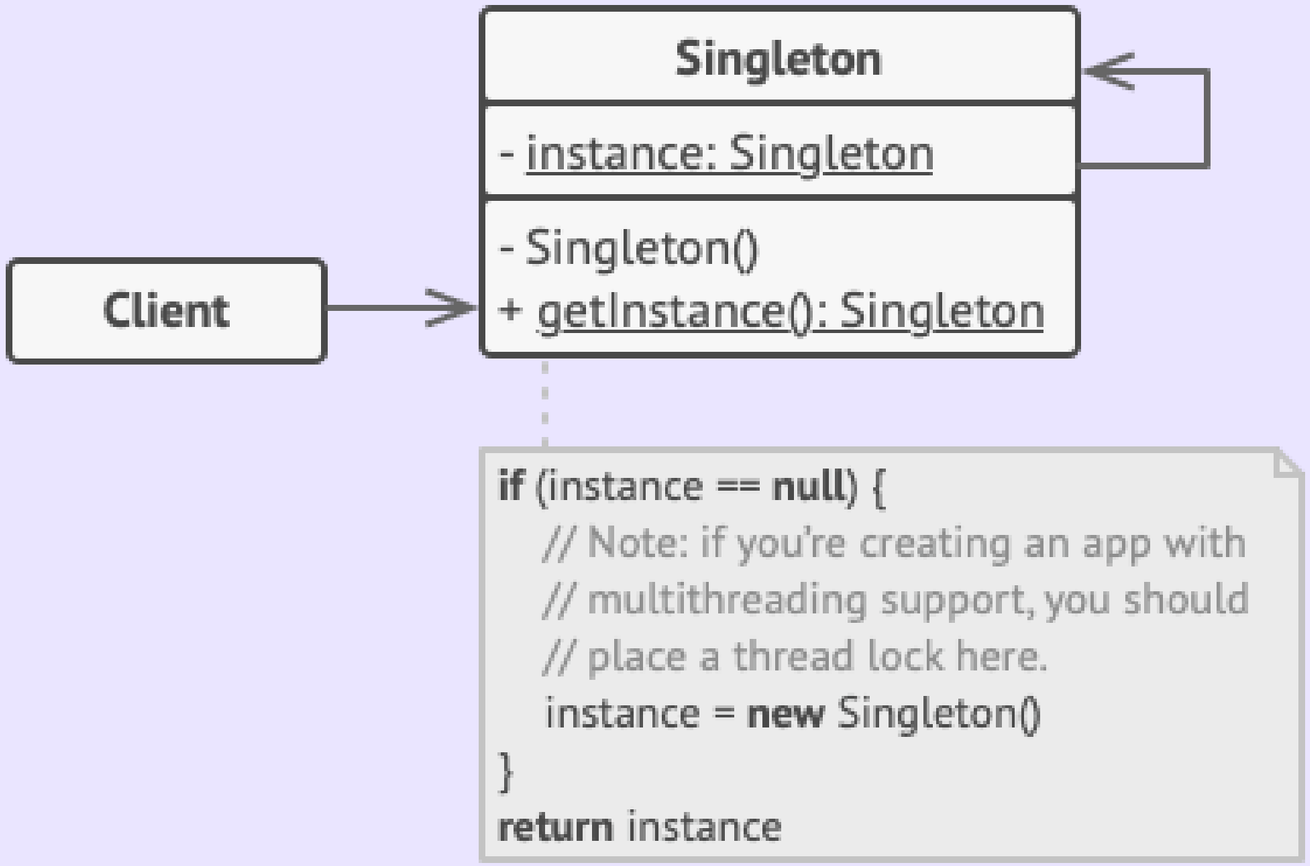
UML Diagram

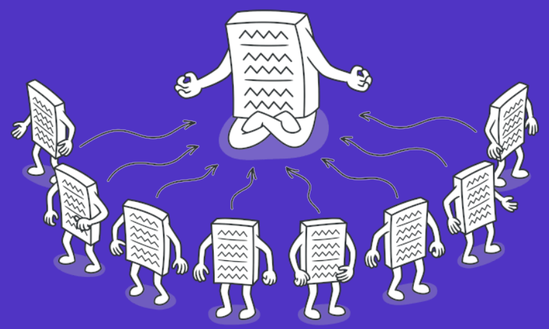




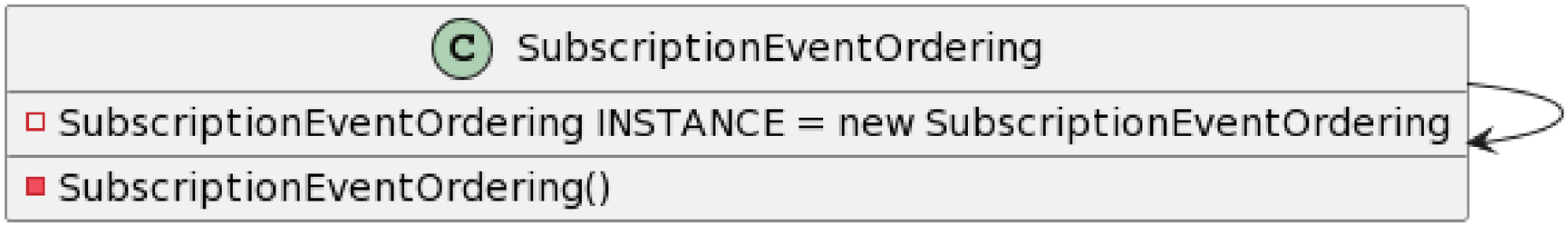
Singleton

Psuedo

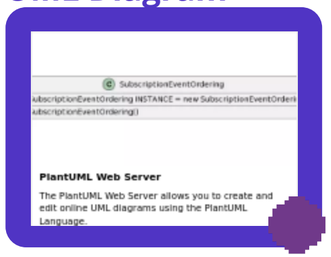

Singleton
- instance: Singleton
- Singleton()
+ getInstance(): Singleton
Client
if (instance == null) {
// Note: if you're creating an app with
// multithreading support, you should
// place a thread lock here.
instance = new Singleton()
}
return instance

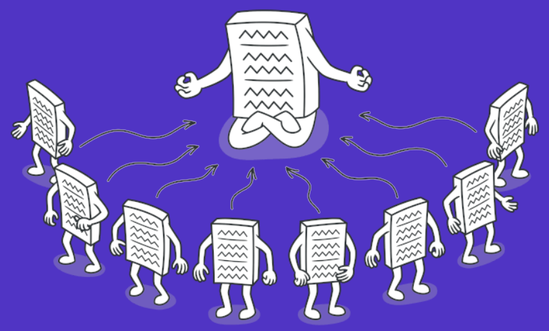


Singleton



UML Diagram



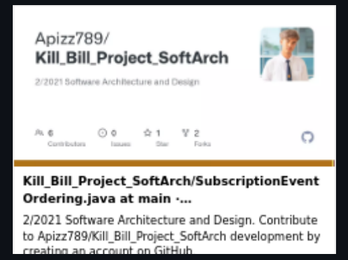


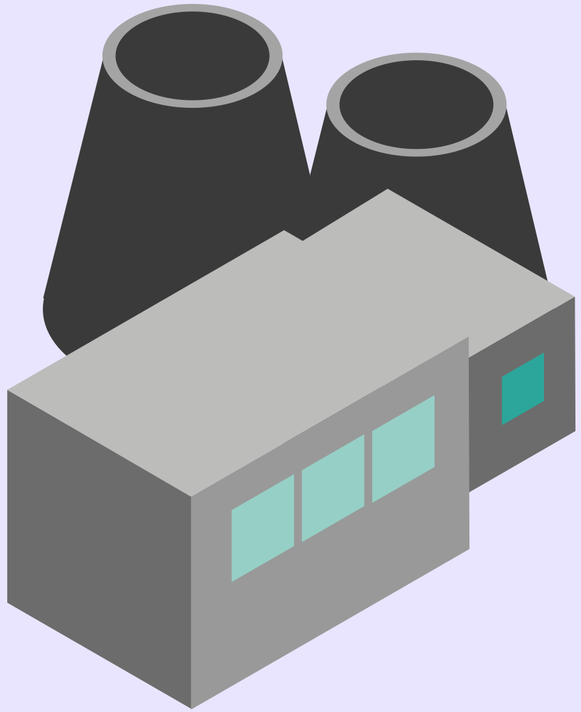
Singleton

class

public class SubscriptionEventQrdering extends EntitlementOrderingBase

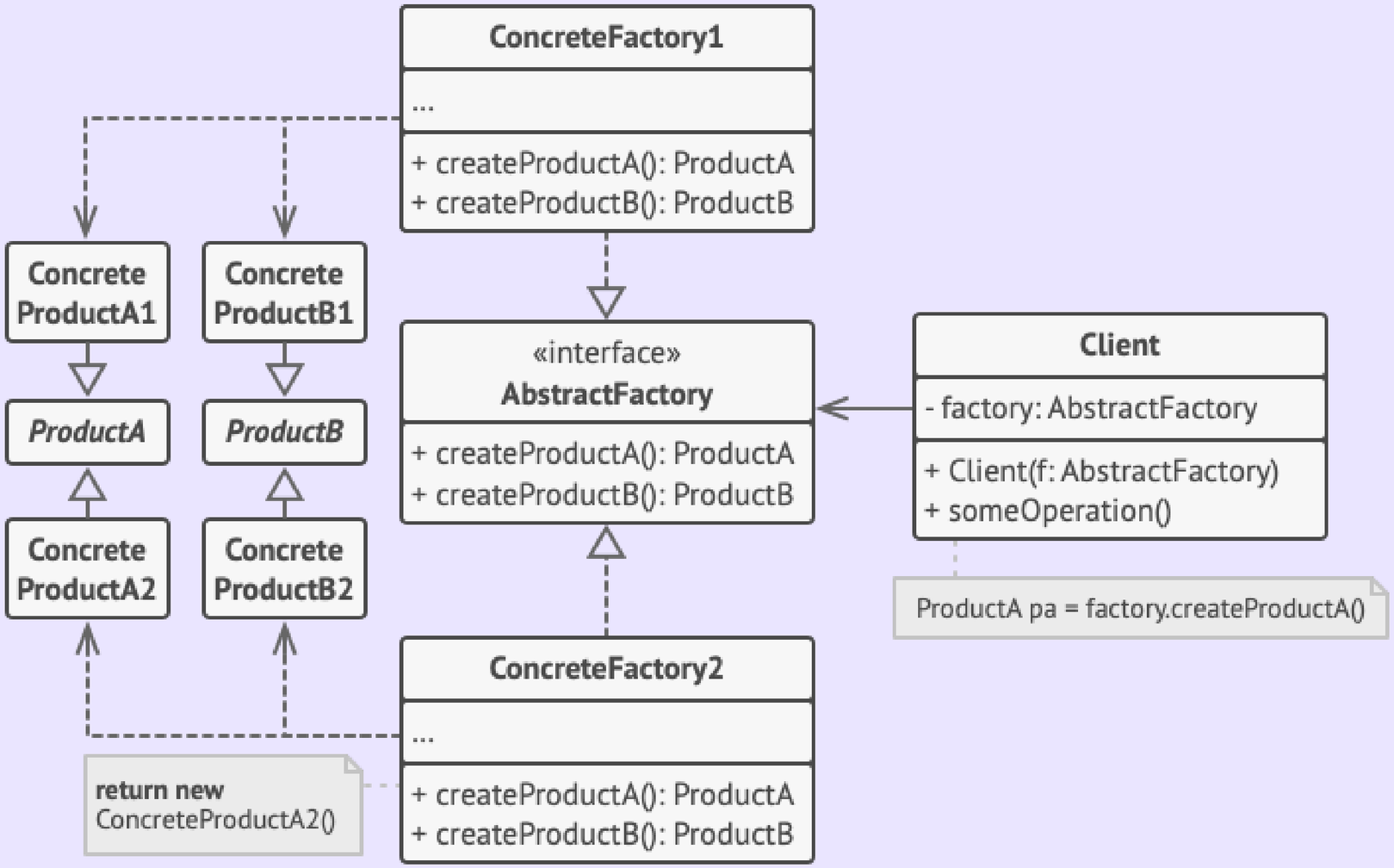
```
//
public class SubscriptionEventOrdering extends EntitlementOrderingBase {

    private static final SubscriptionEventOrdering INSTANCE = new SubscriptionEventOrdering();

    private SubscriptionEventOrdering() {}

    public static List<SubscriptionEvent> sortedCopy(final Entitlement entitlement, final InternalTenantContext internalTenantContext) {
        return sortedCopy(ImmutableList.<Entitlement>of(entitlement), internalTenantContext);
    }

    public static List<SubscriptionEvent> sortedCopy(final Iterable<Entitlement> entitlements, final InternalTenantContext internalTenantContext) {
        return INSTANCE.computeEvents(entitlements, internalTenantContext);
    }

    private List<SubscriptionEvent> computeEvents(final Iterable<Entitlement> entitlements, final InternalTenantContext internalTenantContext) {
        // Compute base events across all entitlements (already ordered per entitlement)
        final LinkedList<SubscriptionEvent> result = computeSubscriptionBaseEvents(entitlements, internalTenantContext);

        // Add blocking states at the right place
        BlockingStateOrdering.insertSorted(entitlements, internalTenantContext, result);

        // Final cleanups
        removeOverlappingSubscriptionEvents(result);

        return result;
    }
```

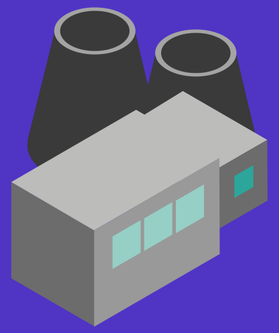




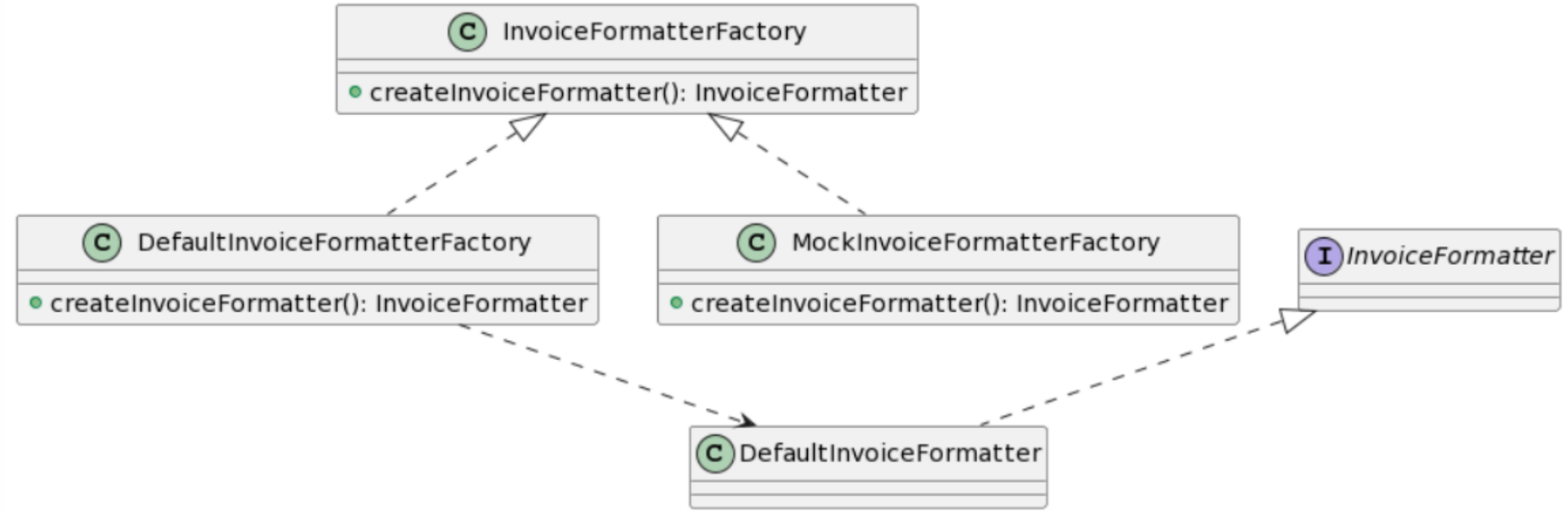
Abstract Factory

Pseudo

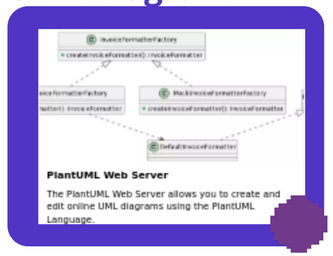

ConcreteFactory1
...
+ createProductA(): ProductA
+ createProductB(): ProductB
Concrete ProductA1
Concrete ProductB1
ProductA
ProductB
Concrete ProductA2
Concrete ProductB2
«interface»
AbstractFactory
+ createProductA(): ProductA
+ createProductB(): ProductB
Client
- factory: AbstractFactory
+ Client(f: AbstractFactory)
+ someOperation()
ProductA pa = factory.createProductA()
ConcreteFactory2
...
+ createProductA(): ProductA
+ createProductB(): ProductB
return new ConcreteProductA2()

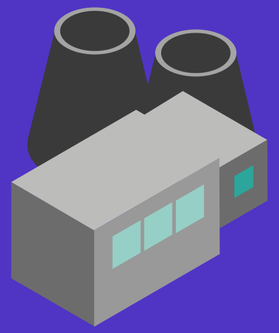


Abstract Factory



UML Diagram





Abstract Factory

interface

public interface InvoiceFormatterFactory


```java
public interface InvoiceFormatterFactory {

    public InvoiceFormatter createInvoiceFormatter(TranslatorConfig config, Invoice invoice, Locale locale, CurrencyConversionApi currencyConversionApi,
                                                   ResourceBundleFactory bundleFactory, InternalTenantContext context);
}
```

class

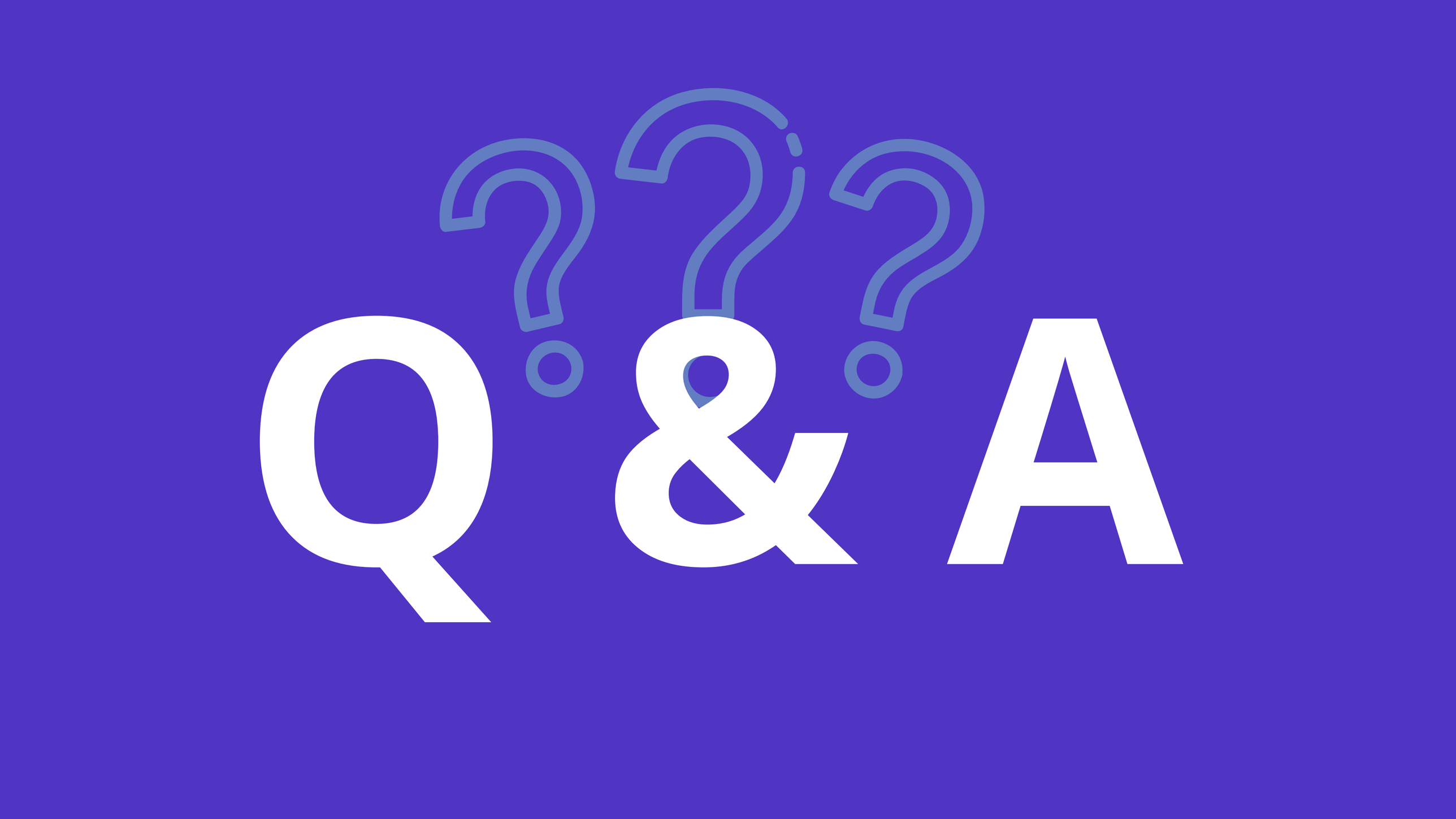
public class DefaultInvoiceFormatterFactory
implements InvoiceFormaterFactory


```java
public class DefaultInvoiceFormatterFactory implements InvoiceFormatterFactory {

    public DefaultInvoiceFormatterFactory() {
    }

    @Override
    public InvoiceFormatter createInvoiceFormatter(final TranslatorConfig config, final Invoice invoice, final Locale locale, final CurrencyConversionApi currencyConversionApi
                                                   final ResourceBundleFactory bundleFactory, final InternalTenantContext context) {
        return new DefaultInvoiceFormatter(config, invoice, locale, currencyConversionApi, bundleFactory, context);
    }
```

class

public MockInvoiceFormatterFactory implements InvoiceFormatterFactory


```java
public class MockInvoiceFormatterFactory implements InvoiceFormatterFactory {

    @Override
    public InvoiceFormatter createInvoiceFormatter(final TranslatorConfig config, final Invoice invoice, final Locale locale, final CurrencyConversionApi currencyConversionApi
        return null;
    }
}
```


Q & A

62010029 นางสาวกฤติกามาส สุโนภักดิ์
62010090 นางสาวคัมภีรดา ภู่ทอง
62010175 นายชวกร เหลาแก้ว
62010215 นางสาวโชติกา ตระกูลนุช
62010285 นางสาวณัฐมน บุญประเสริฐ
62010292 นายณัฐวัฒน์ จันทร์วิสิทธิ์
62011019 นายอภิรักษ์ อุลิศ

MEMBER

พร้อมบวก
กับทุกคน
พร้อมชน
ทุกปัญหา

โชติกา

1

#พลังซอร์ฟอาร์ฯ
พลังคลื่นเต่า ย๊ากก

พรรค
พลังคลื่นเต่า

โปรดเลือก

คัมภีรดา
ภู่ทอง

เป็นผู้ว่าฯ เอสเอดี.

2

บัคต่อ แก้ต่อ
แล้วก็กลับมาบัคใหม่
#บ้าจริง

เบอร์
3

เลือก
กฤติกามาส
สุโนภักดิ์

คิวบริลวิปัตย์

ผู้สมัคร
ผู้ว่าฯ เอสเอดี.

ปลื้ม
อภิรักษ์

4

เปลี่ยน
อธิการฯ

#เราทำได้

5

เลือก ชวกร ทำ ซอร์ฟอาร์ฯ น่าอยู่

โปรดเลือก "ณัฐมน" เป็นผู้ว่าฯ ซอร์ฟอาร์ฯ

ผู้สมัครอิสระ ตัวจริง

6

ต้อง หยุด เรียน

ซอร์ฟอาร์เปลี่ยนแน่

#เลือกณัฐมน มาเป็นทีม

SADMORE>

ทำ (ตอน)เดดไลน์

เลือก

ณัฐวัฒน์ จันทร์วิสิทธิ์

เป็นผู้ว่าฯ ซอร์ฟอาร์ฯ

7

SOFTWARE ARCHITECTURE AND DESIGN

22 พ.ค. **2565**

อย่าลืมไปเลือกตั้งผู้ว่าฯ กทม.

THANK YOU