# Tasarım Örüntüleri
# Visitor

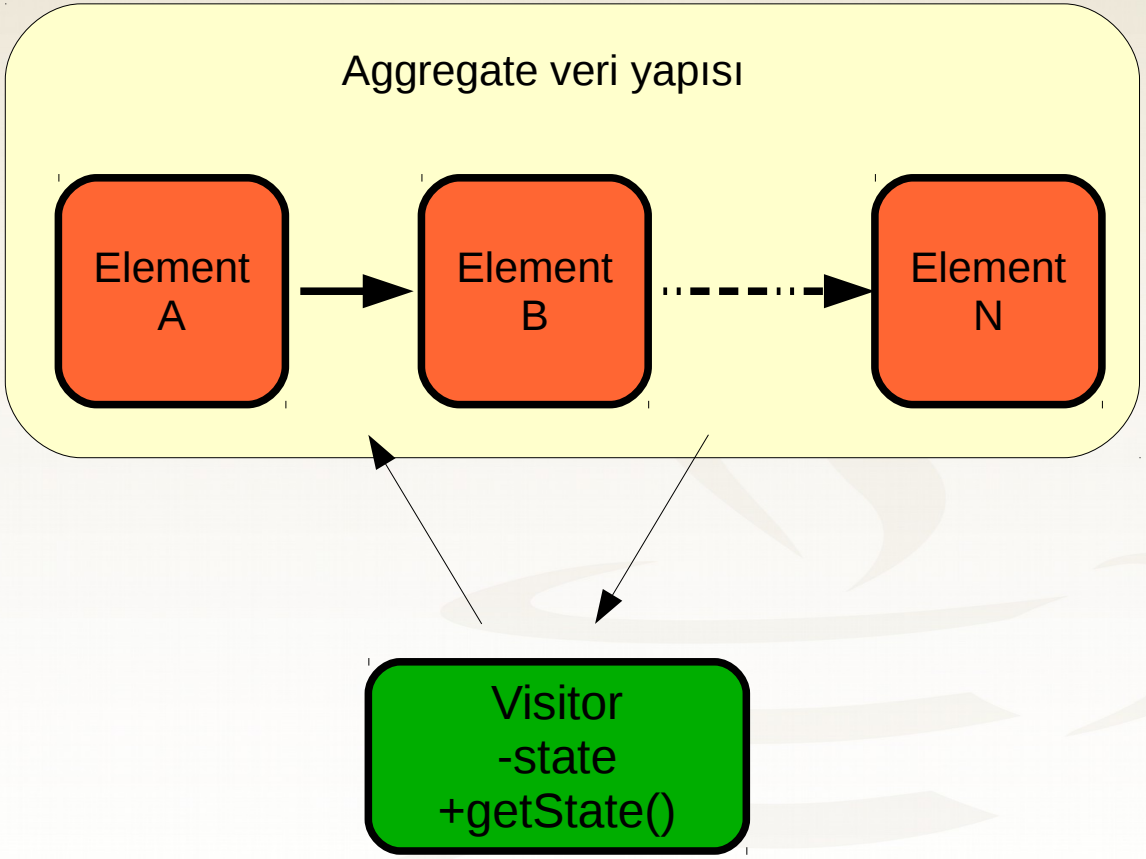
# Örüntülerin Temel Prensipleri

- GoF tasarım örüntülerinin altında yatan temel prensipler
  - Encapsulation
  - Composition
  - Abstract Data Types

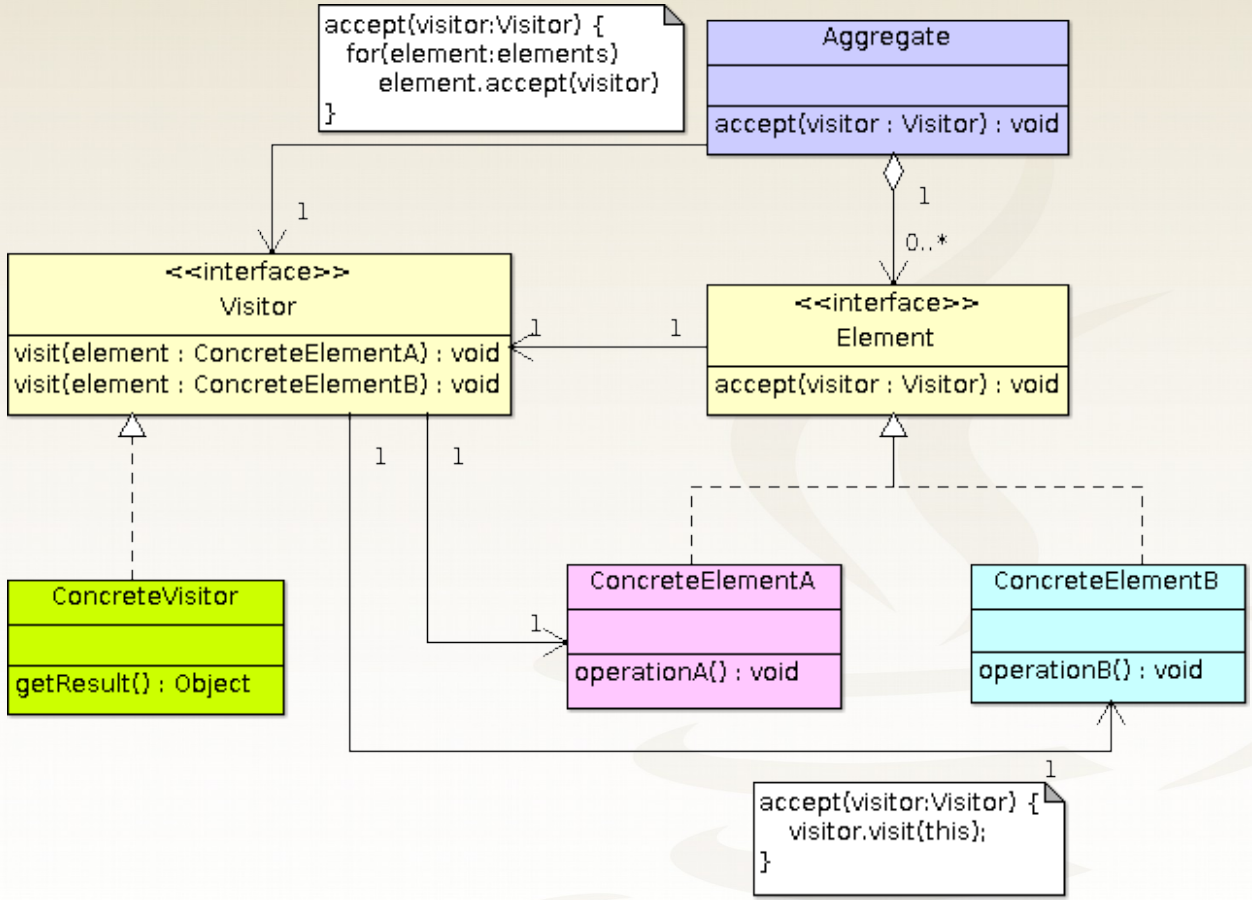
# Visitor Örüntüsü

- Bir **veri yapısının üzerinde yapılacak işlemleri** veri yapısının dışında **ayrı bir yerde encapsule etme**yi sağlayan bir örüntüdür
- Bu sayede veri yapısını kirletmeden **farklı farklı operasyonları tanımlamak mümkün** hale gelmektedir

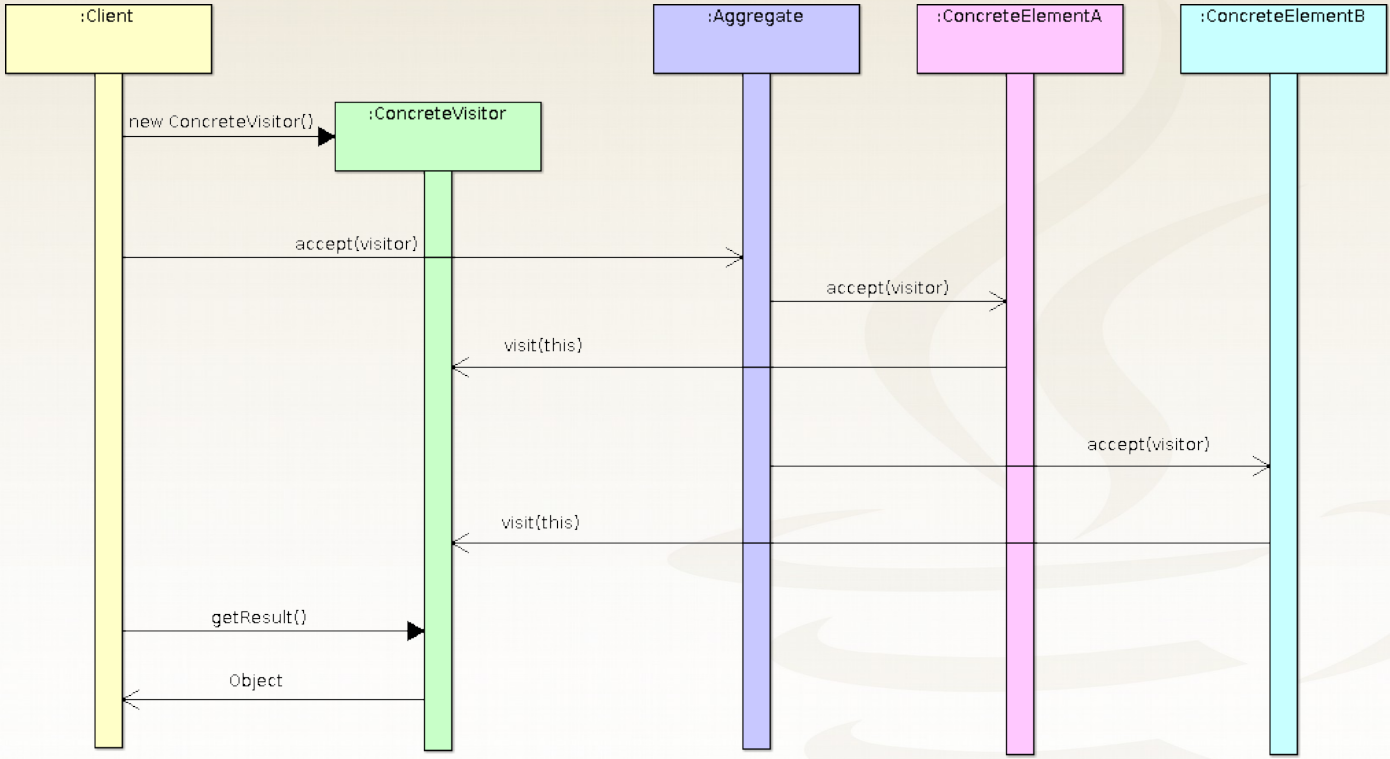
# Visitor Örüntüsü



# Visitor Sınıf Diagramı



# Visitor Akış Diagramı



# Java ve Visitor

- Java'da Visitor örüntüsünü implement etmek için genellikle **double dispatch** yöntemi kullanılır

# Java ve Visitor: Single Dispatch

```
public class A {
    public void sayHello() {
        System.out.println("Hello A");
    }
}

public class B extends A {
    public void sayHello() {
        System.out.println("Hello B");
    }
}
```

```
A a1 = new A();
A a2 = new B();


a1.sayHello();
a2.sayHello();
```

```
Hello A
Hello B
```

# Java ve Visitor: Single Dispatch

```
public class A {

}
public class B extends A {

}
```

```
public class Visitor {
    public void visit(A a) {
        System.out.println("visit A");
    }

    public void visit(B b) {
        System.out.println("visit B");
    }
}
```

```
A a1 = new A();
A a2 = new B();

Visitor v = new Visitor();

v.visit(a1);
v.visit(a2);
```

```
visit A
visit A
```

# Java ve Visitor: Double Dispatch

```
public class A {
    public void accept(Visitor v) {
        v.visit(this);
    }
}

public class B extends A {
    public void accept(Visitor v) {
        v.visit(this);
    }
}
```

```
A a1 = new A();
A a2 = new B();

Visitor v = new Visitor();

v.visit(a1);
v.visit(a2);
```

```
a1.accept(v);
a2.accept(v);
```

```
visit A
visit B
```

# Java ve Visitor: instanceOf ile Double Dispatch

```
public interface Element {
}

public class A implements Element {
}

public class B implements Element {
}
```

```
public abstract class Visitor {
     public void visit(Element e) {
          if(e instanceof A) {
               this.visit((A)e);
          } else if(e instanceof B) {
               this.visit((B)e);
          }
     }

     protected abstract void visit(A a);

     protected abstract void visit(B b);
}
```

# Java ve Visitor: instanceOf ile Double Dispatch

```
public class ConcreteVisitor extends Visitor {
     @Override
     protected void visit(A a) {
          System.out.println("visit A");
     }
     @Override
     protected void visit(B b) {
          System.out.println("visit B");
     }
}
```

```
E e1 = new A();
E e2 = new B();

Visitor v = new ConcreteVisitor();

v.visit(e1);
v.visit(e2);
```

```
visit A
visit B
```

# Visitor Örüntüsünün ASM İçerisinde Kullanımı

- ASM genel amaçlı bir **bytecode manipulation kütüphanesi**dir
- OpenJDK, Groovy, Kotlin gibi derleyilerden CGLib, Gradle gibi araçlara kadar **pek çok yerde** kullanılmaktadır
- ASM içerisinde **Visitor temel bir örüntü** olarak karşımıza çıkmaktadır

# Visitor Örüntüsünün Sonuçları

- **Operasyonlar** aggregate veri yapısı içerisinde farklı bileşenlere dağılmayıp, **sadece Visitor nesnesinde toplanmış** olur
- Aggregate üzerinden çalışacak **yeni operasyonları eklemek** yeni bir **Visitor implemantasyonu eklemek kadar kolay**dır
- Öte yandan aggregate veri yapısına **farklı bir eleman eklemek** Visitor tarafında da değişiklik yapmayı gerektirecektir

# İletişim



JAVA Eğitimleri

## Kurumsal Java Eğitimleri

www.java-egitimleri.com

info@java-egitimleri.com

@javaegitimleri


youtube.com/c/
KurumsalJavaEğitimleri