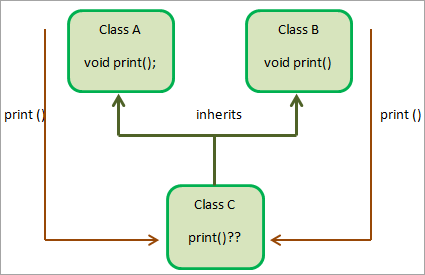
# JAVA DA MULTIPLE INHARITANCE(ÇOKLU KALITIM)

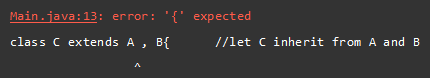
Java diğer diller(C#, C, C++...) gibi doğrudan çoklu kalıtımı desteklemez. Java da ise bu çoklu kalıtımın olmamasının nedeni; Java, programlarda basitliği hedefler. Bu, OOP özellikleri için bile geçerlidir. Dolayısıyla, çoklu kalıtım söz konusu olduğunda, birden fazla sınıftan bir sınıf türettiğimizde bazı belirsiz durumlar ortaya çıkabilir.

Şimdi programımızda birden fazla mirasa sahip olduğumuzda ortaya çıkabilecek sorunları anlayalım.



 Burada A sınıfı ve B sınıfı, C sınıfı tarafından miras alınan iki ebeveynli sınıflardır. Şimdi, hem A hem de B sınıflarının aynı ada sahip bir yönteme, yani print () olduğunu gösterdik. Dolayısıyla, C sınıfı A ve B'yi miras aldığında, print () yönteminin her iki sürümünü de alacaktır.

Bu durumda, C sınıfı hangi yöntemi miras alması gerektiğini bilemeyecektir. Çoklu miras durumunda ortaya çıkan yukarıda bahsettiğimiz belirsiz durum budur. Bu nedenle, sınıf birden fazla sınıftan miras aldığında Java bir derleyici hatası verir.



**Aşağıdaki Java programı bunu gösterecektir.**

class A{ void print(){ System.out.println('Class A :: print()'); } } class B{ void print(){ System.out.println('Class B :: print()'); } } class C extends A , B{ //let C inherit from A and B } public class Main { public static void main(String[] args) { C cObj = new C(); c.print(); } }

Çıktıdan görüldüğü gibi, Java derleyicisi, sınıfın iki sınıfı genişlettiğini bulduğunda bir hata verir.

Java, iki sınıfın eşzamanlı olarak miras alınmasına izin vermese de, bize bu sınırlamayı aşmanın bir yolunu sunar. Sınıflar yerine arayüzler kullanarak çoklu kalıtımı simüle edebiliriz. Bunu sonraki eğitimlerimizde tartışacağız.

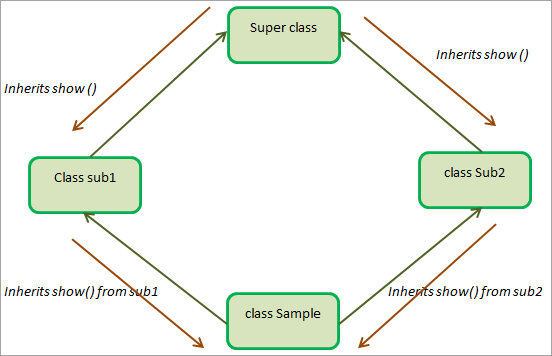
### Elmas Problemi

Çoklu miras, 'Elmas sorunu' olarak adlandırılan bir durumla sonuçlanır.

**Aşağıdaki miras sistemine sahip olduğumuzu düşünün.**

class Super{ void show(){} } class Sub1 extends super{ void show1(){} } class Sub2 extends super{ void show2(){} } class sample extends Sub1,Sub2{ //Sample class code here }

**Bu sistemi şematik olarak aşağıda gösterildiği gibi gösterebiliriz:**



Yukarıdaki diyagramdan da görebileceğimiz gibi, tüm kalıtım sistemi bir elmas şeklinde görünür. Bu elmas yapısı nedeniyle belirsizliğe 'Elmas Problemi' adı verilir.

Çoklu kalıtım için daha önce açıklandığı gibi, buradaki belirsizlik, örnek sınıfının kalıtımın bir sonucu olarak elde ettiği show () yönteminin çok sayıda kopyasıdır. Birden fazla yol olduğundan, show () yöntemini gerçekleştirir ve hangi kopyanın devralınacağı konusunda bir belirsizlik vardır.

Çoklu kalıtımda olduğu gibi, elmas sorunu da arayüzler kullanılarak çözülebilir.

### **Java'da Kalıtım Kuralları**

Şimdiye kadar kalıtımın temellerini ve Java'daki kalıtım türlerini gördük, hadi kalıtım içeren uygulamaları geliştirirken uymamız gereken bazı kuralları listeleyelim.

**Kural 1: Java, Çoklu Devralmayı desteklemez**

Daha önce gördüğümüz gibi, Java herhangi bir biçimde çoklu kalıtımı desteklemez. Bir programda birden fazla miras alma girişiminde bulunulduğunda, Java derleyicisi derleyici hatası verir.

**Kural # 2: Java, Döngüsel Devralmaya izin vermez**

Bir döngü oluşturmak için kendini miras alan bir sınıfa Döngüsel Miras denir.

**Aşağıdaki sınıfları, sınıf1 ve sınıf2'yi düşünün.**

*sınıf1, sınıf 2'yi genişletir {}*

sınıf2, sınıf 1'i genişletir {}

Yukarıdaki durum bir döngü oluşturur. Java'da buna izin verilmez.

Kural 3: Özel veri üyelerini devralamayız

Özel erişime sahip üyelerin devralınamayacağını 'Erişim değiştiriciler' konusunda zaten öğrendik.

Kural 4: Yapıcılar miras alınmaz

Oluşturucular, sınıf nesneleri oluşturmak için kullanıldığından, bir sınıftan diğerine miras alınmazlar.

Kural 5: Alt nesnelere erişmek için ebeveyn referansını kullanabiliriz

Zaten çeşitli miras örnekleri gördük. Üst sınıf türünde bir referans oluşturabilir ve ardından buna new anahtar sözcüğü kullanılarak oluşturulmuş türetilmiş bir sınıf nesnesi atayabiliriz.

Süper bir üst sınıfsa ve alt sınıf süperden türetilmişse aşağıda gösterildiği gibi.

class super{} class sub extends super{ public static void main(String[] args){ super s\_obj = new sub(); } }

Kural # 6: Türetilmiş kuruculardaki super () nedeniyle üst yapıcılar çalıştırılır

Kurallardan biri, kurucuların miras alınamayacağını belirtir. Ancak üst sınıfın kurucusu, türetilmiş sınıfın yapıcısı içinde super () çağrısı kullanılarak çalıştırılabilir.

Java'da çoklu kalıtım interface'ler arasında bulunur. Bir interface birden fazla interface'i extend edebilir.  
Üstelik bir sınıf bir tek sınıfı extend edebilirken birden fazla interface'i implement edebilir. interface'ler arasında çoklu kalıtımın C++'daki çoklu kalıtımda ortaya çıkan problemi yaratmamasının sebebi interface'lerin verisinin olamamasıdır. En fazla static final tanımlı sabit tanımlayabilirsiniz.  C++'daki çoklu kalıtım problemi daha çok verilerin tekrarlı olarak ebeveyn sınıftan gelmesiyle ilgilidir. Bu problemi bir ölçüde "virtual inheritance"  ile çözüyoruz. Ama bu çözüm fazla sezgisel olması nedeni ile tam bir çözüm olarak kabul edilmiyor. Ayrıca çözemediği bazı durumlar da bulunuyor.

Rapor Hazırlanırken: <https://tr.myservername.com/scrum-artifacts-product-backlog> Sitesinde ki verilerden yararlanılmıştır