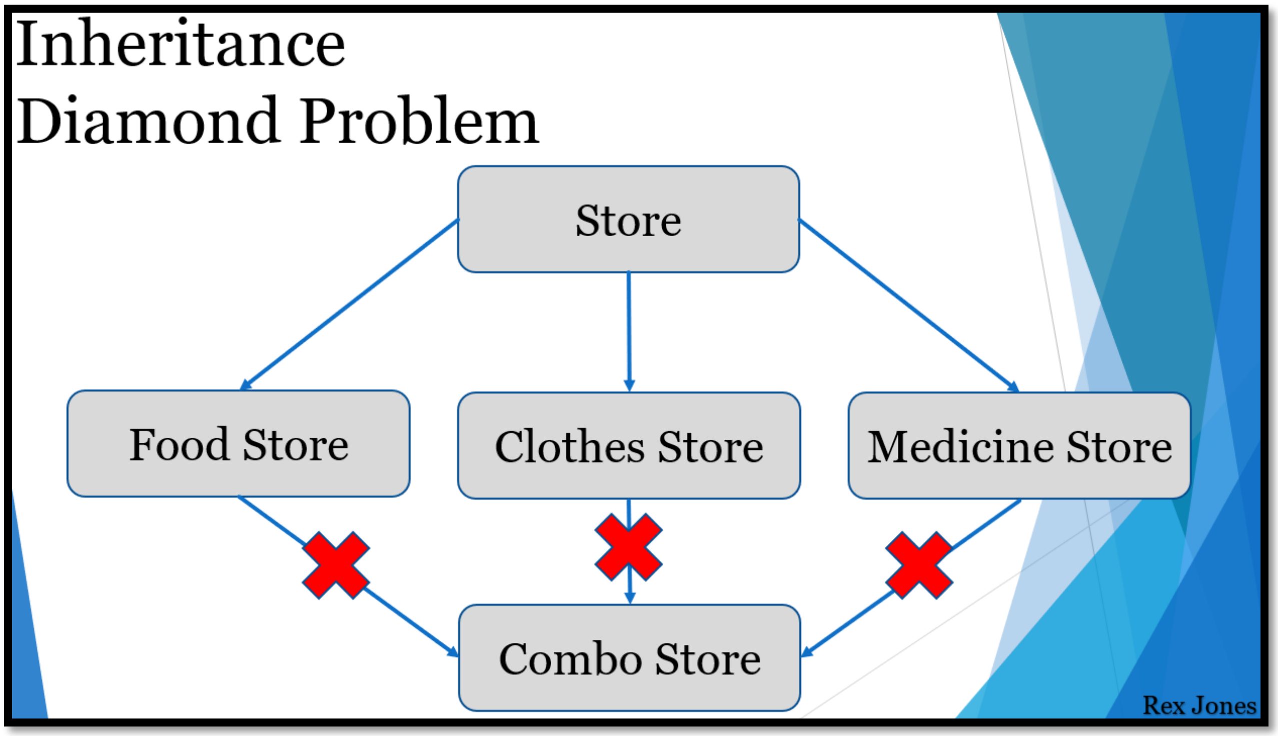
Rex Jones II

(Transcript) Interface Multiple Implementation

Multiple Interface Implementation and Extension

Multiple implementation of interfaces is a way for a class to implement more than 1 interface. That's important because inheritance cannot extend more than 1 class but an interface can extend more than 1 interface. In this diagram, we see 1 Super Class and 3 Sub Classes. Each Sub Class inherits the Super Class. Let's look at this diagram using a Store example. Store is the parent class while Food Store, Clothes Store, and Medicine Store are the child classes. With inheritance, there is no way to combine all stores into 1 Combo Store. It is impossible because we are not allowed to extend more than 1 class. This is called a diamond problem.



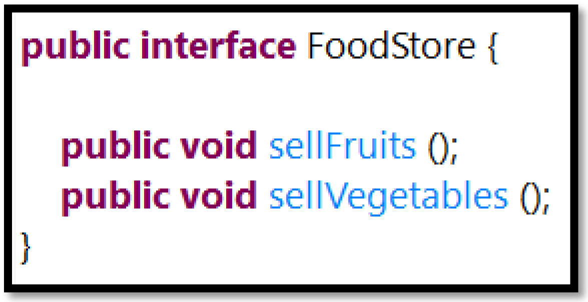
Inheritance Diamond Problem

$$\begin{tikzcd}
 & \text{Store} \arrow[dl] \arrow[d] \arrow[dr] & \\
\text{Food Store} \arrow[dr, "\times"] & \text{Clothes Store} \arrow[d, "\times"] & \text{Medicine Store} \arrow[dl, "\times"'] \\
 & \text{Combo Store} &
\end{tikzcd}$$

Do you see the diamond? Combo Store will not inherit Food, Clothes, or Medicine although it's possible in real life. However, we can combine all stores when 1 class which is Combo Store implements more than 1 store. There is no limit to the number of interfaces a class can implement.

In Eclipse, we see the FoodStore has 2 abstract methods: sellFruits and sellVegetables.

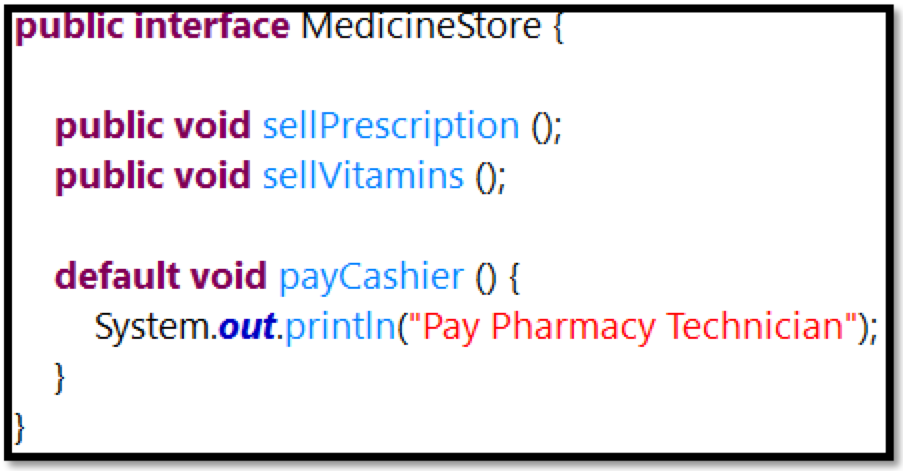
Rex Jones II



```
public interface FoodStore {

   public void sellFruits ();
   public void sellVegetables ();
}
```

The Medicine Store also has 2 abstract methods and 1 default method: sellPresecription and sellVitamins are the abstract methods, 'payCashier' is the default method with details to pay a Pharmacy Technician.

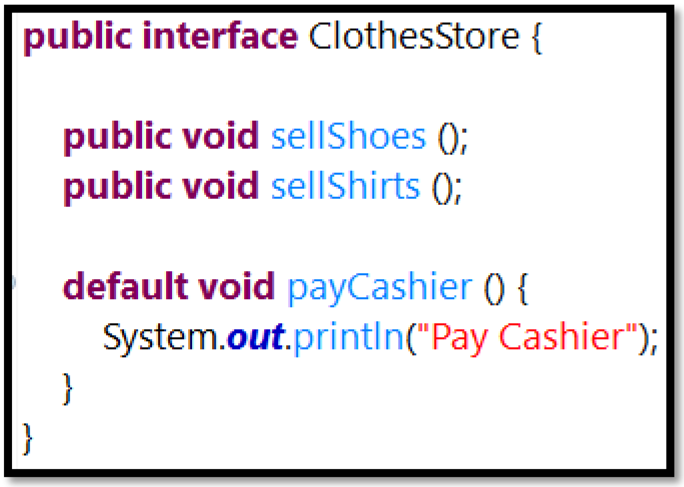


```
public interface MedicineStore {

   public void sellPrescription ();
   public void sellVitamins ();

   default void payCashier () {
      System.out.println("Pay Pharmacy Technician");
   }
}
```

Clothes Store is similar to Medicine Store. It has 2 abstract methods: sellShoes and sellShirts and the same default method named 'payCashier' but different implementation details.

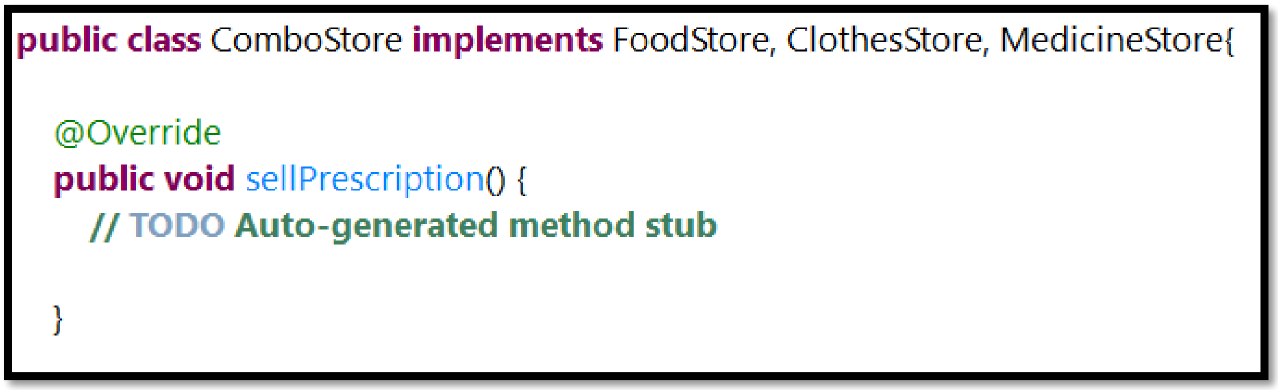


```
public interface ClothesStore {

   public void sellShoes ();
   public void sellShirts ();

   default void payCashier () {
      System.out.println("Pay Cashier");
   }
}
```

To combine all 3 interface stores, we go to the Combo Store then it implements FoodStore, ClothesStore, and MedicineStore. Just like when implementing 1 interface, we must add all unimplemented methods. There should be 6 overridden methods: sellPrescription, sellVitamins, sellShoes, sellShirts, sellFruits, and sellVegetables.



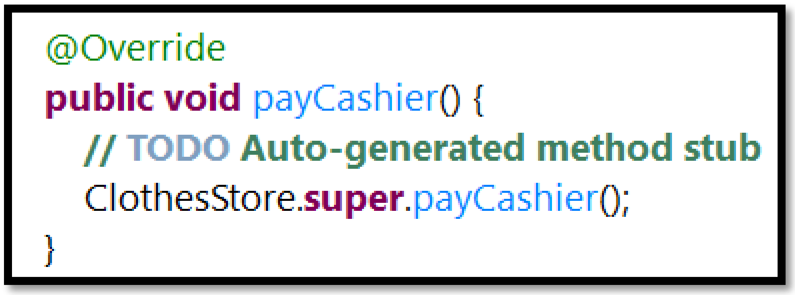
```
public class ComboStore implements FoodStore, ClothesStore, MedicineStore{

  @Override
  public void sellPrescription() {
    // TODO Auto-generated method stub

  }
```

Save and there's 1 more error "Duplicate default methods named payCashier with the parameters () and () are inherited from the types MedicineStore and ClothesStore".

For a quick fix, we can override the default method or change the parameters. This is also a diamond problem because both interfaces have the same default method name payCashier and the compiler cannot decide which super method to use. Override payCashier from ClothesStore. Go to the method and we see Override – ClothesStore.super.payCashier().

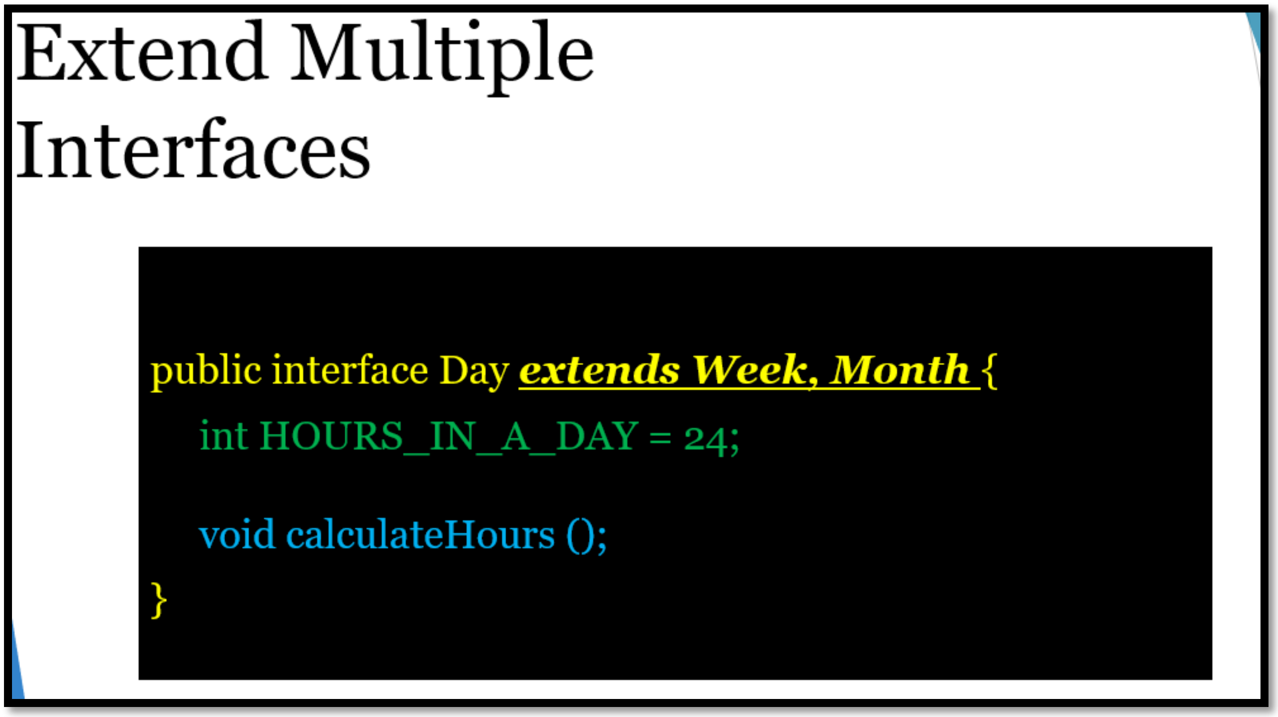


```
@Override
public void payCashier() {
  // TODO Auto-generated method stub
  ClothesStore.super.payCashier();
}
```

Recall that a class cannot extend more than 1 class but an interface can extend more than 1 interface. Let's go to the FoodStore interface. It extends another interface called EmailServiceProvider which is another interface from the previous session How To Implement An Interface.

Save and ComboStore will have a different error because the abstract methods from EmailServiceProvider must be implemented. Add unimplemented methods then go to the bottom and we see each method: createEmailMessage, openJunkSpamFolder, closeEmailProvider. In this example, multiple interfaces are extended after adding a comma and another interface name. The interface is Day then it extends 2 interfaces: Week and Month.

Rex Jones II



Extend Multiple Interfaces

```
public interface Day extends Week, Month {
    int HOURS_IN_A_DAY = 24;

    void calculateHours ();
}
```

That completes the Java Object-Oriented Programming Series. We covered 4 traits of the OOP's concept: Encapsulation, Inheritance, Polymorphism, and Abstraction. I hope you learned something from this OOP series. If you did, Subscribe, Connect, and Follow me because I have so much more content to share with our Testing Community.