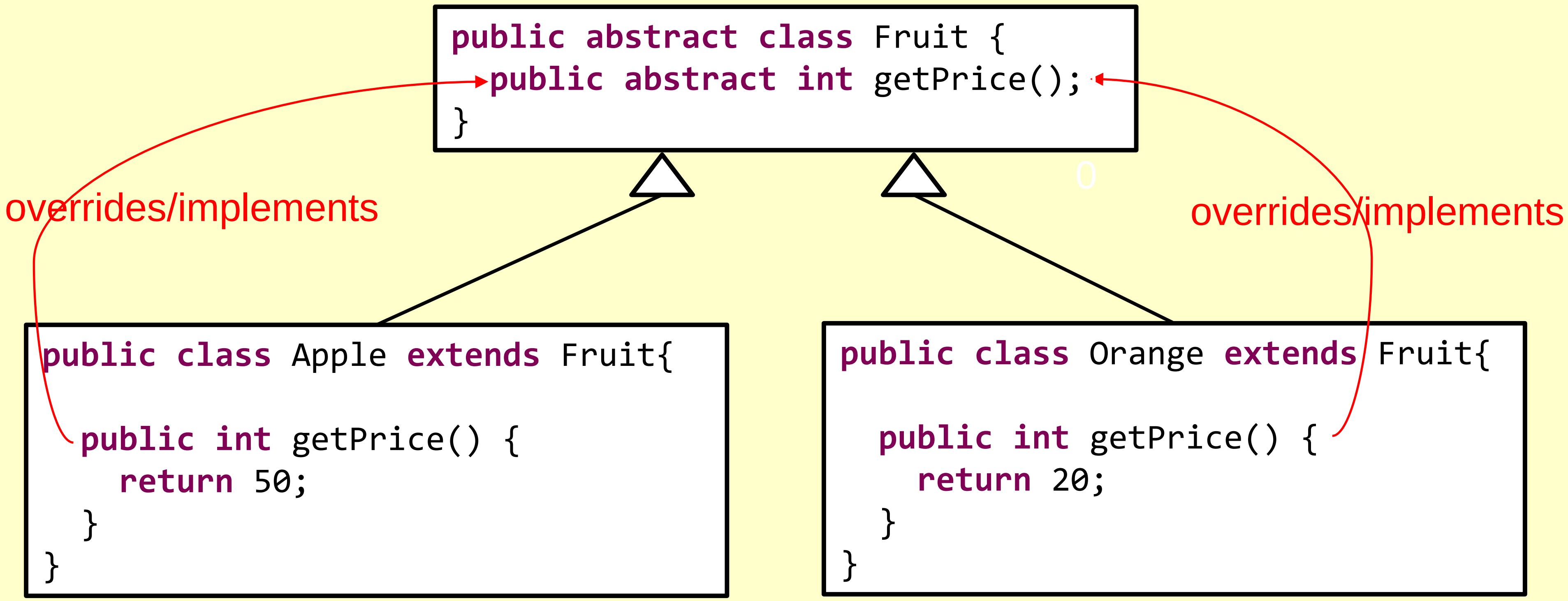
Dynamic Polymorphism典型使用結構與優點

首先，先建立樹狀結構，通常Superclass定義抽象Method，Subclass實作各自Method



```
public abstract class Fruit {
  public abstract int getPrice();
}
```

```
public class Apple extends Fruit{

  public int getPrice() {
    return 50;
  }
}
```

```
public class Orange extends Fruit{

  public int getPrice() {
    return 20;
  }
}
```

Dynamic Polymorphism典型使用結構與優點

接著，在需要Subclass當作參數的Method上，改為利用Superclass當作參數，以避免為每一個Subclass產生重複的程式碼

```
public class Shop {

  public int checkout(Fruit fruit, int count) {
    return fruit.getPrice() * count;
  }
  ~~public int checkout(Apple apple, int count) {~~
    ~~return apple.getPrice() * count;~~
  ~~}~~
  ~~public int checkout(Orange orange, int count) {~~
    ~~return orange.getPrice() * count;~~
  ~~}~~
}
```

Dynamic Polymorphism典型使用結構與優點

最後，使用端可動態餵入Subclass的Object Instance

```
public class Customer {

  public static void main(String[] args) {
    Apple apple = new Apple();
    Shop shop = new Shop();
    int totalPrice =   shop.checkout(apple, 10);
    System.out.println(totalPrice); // output: 500
  }
}
```

Call by reference

```
public class Shop {
  public int checkout(Fruit fruit, int count) {
    return fruit.getPrice() * count;
  }
}
```

Dynamic polymorphism achieved
by dynamic method binding

```
public class Apple extends Fruit{
  public int getPrice() {
    return 50;
  }
}
```

Dynamic Polymorphism典型使用結構與優點

- 帶來的好處是什麼?

假設要新增一個Strawberry Class，那麼Shop Class 的checkout完全不需要修改! 易於擴充!