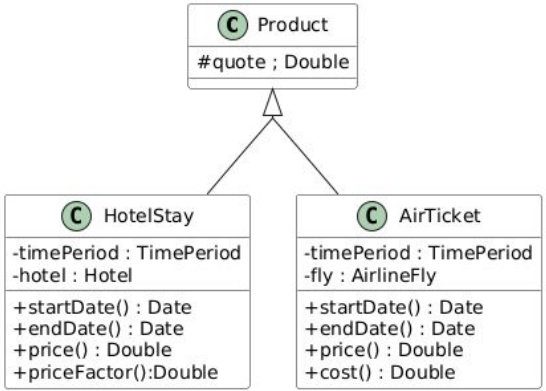
Hablando del elefante blanco…

¿Qué otro code smell detectan?



HotelStay

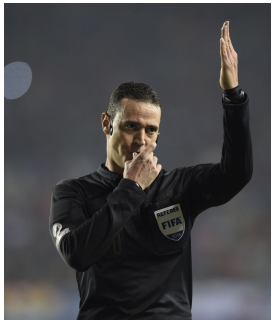
```
startDate() {
return timePeriod.start; }

endDate() {
return timePeriod.end; }

price(){
  return timePeriod.duration() * hotel.nightPrice() * hotel.discountRate(); }

priceFactor(){
  return this.quote / this.price(); }
```

AirTicket

```
startDate() {
return timePeriod.start; }

endDate(){
return timePeriod.end; }

price() {
  return fly.price() * fly.promotionRate(); }

cost() {
  return this.quote; }
```

Código duplicado! ⇒ Pull up Method

- ¿Qué debemos considerar para ver si se puede realizar?
- ¿Cuales son los pasos?
- ¿Cómo nos aseguramos que preservamos el comportamiento?



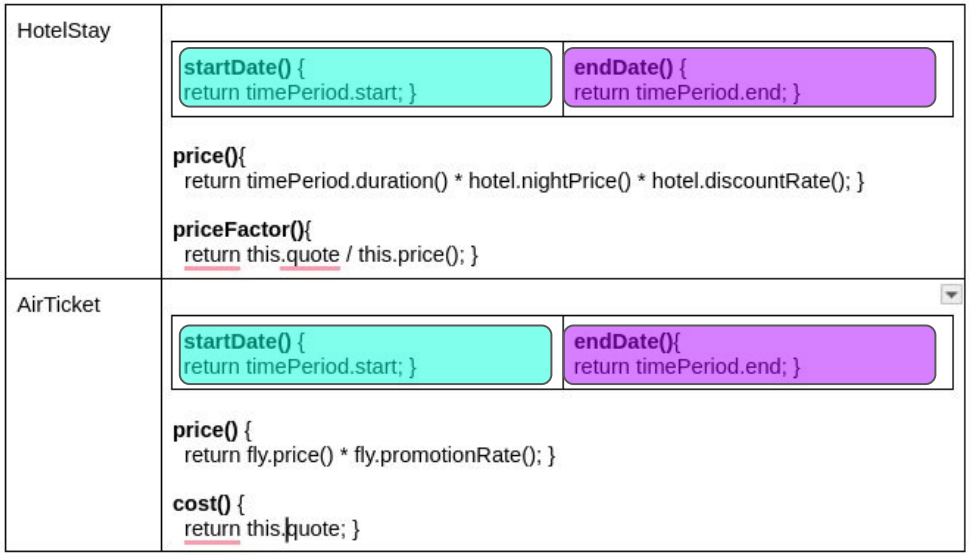
Resolver con algún compañero

- Nosotros vamos a publicar **una** de las posibles soluciones

Una posible resolución a Código Repetido

Código Duplicado (p. 76 @ Refactoring, Fowler)

- Def: misma Estructura de Código en varios lugares.
  - Expresiones repetidas
  - Puede incluir todo el método
- En este ejemplo (simple), los métodos marcados están completamente duplicados
  - startDate(): son iguales en ambas clases
  - endDate(): son iguales en ambas clases
- Se acceden variables (fields) del objeto guardado en la variable *timePeriod*
  - "timePeriod.**start**"
  - "timePeriod.**end**"
  - Este code smell quedará para ser analizado más adelante.



Pull Up Method (p. 322 @ Refactoring, Fowler)

- Consideraciones de **startDate()**
  a. Las implementaciones son iguales
  b. Existe una superclase: *Product*
  c. Product no tiene un método con ese nombre
- Mecánicas (adaptada a este ejemplo)
  a. Cree un método en la superclase (Product) y copie el cuerpo del método: *startDate()*
     i. ∃ referencias a la variable timePeriod ⇒ realizar PullUp Field de timePeriod (*)
  b. Borre el método en una de las subclases
  c. Compile y corra los test
  d. Siga borrando el método en cada subclase, compilando y testeando hasta que solo quede el método en la superclase *Product*.
  e. Verifique que en las llamadas al método el tipo (de parámetros y variables) se ajuste al tipo de la superclase.

(*) más sobre sigue…

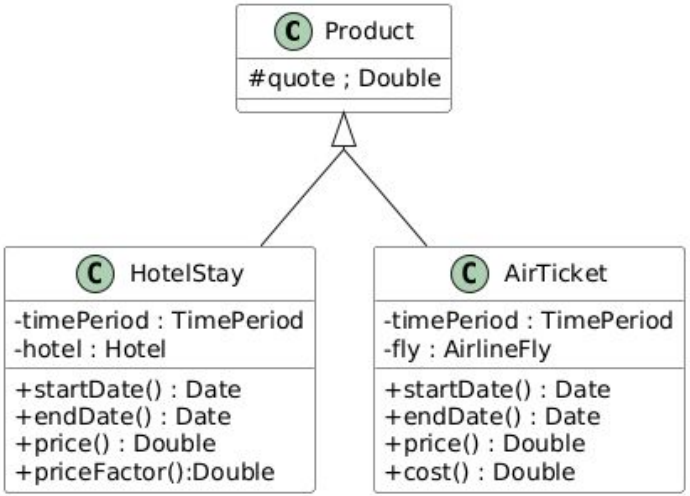
Revisitando Fundamentos de Programación Orientada a Objetos

Herencia

- Subclases heredan métodos y declaración de variables.

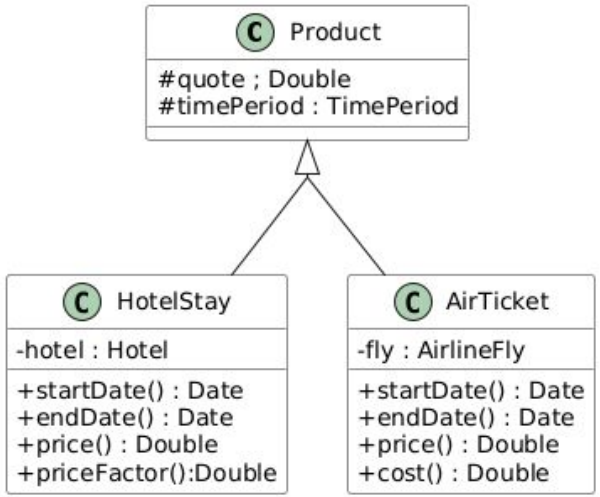
Encapsulamiento

- Las instancias son responsables de su estado (encapsulamiento)
  - Directamente: Variables no accesibles desde otros objetos
  - Indirectamente: No hay métodos getters y setters
- Instancias acceden a las variables declaradas y heredadas
- Jerarquías en donde las subclases no pueden acceder a variables declaradas en una superclase es un **code smell**
  - Posiblemente el comportamiento asociado a ese estado se puede descomponer en otra clase que requiera composición y delegación de responsabilidades



Revisitando Pull Up Field(p. 320 @ Refactoring, Fowler)

1. **Consideraciones**
   a. La superclase Product no tiene una variable similar
   b. La variable timePeriod es privada en ambas subclases
2. **Mecánica** (adaptada a este caso)
   a. Asegurarse que todos los accesos a la variable son consistentes
   b. La variable tiene en todas las subclases el mismo nombre ⇒ No es necesario renombrar
   c. Declarar la variable en la superclase
      i. Considere las reglas de Java(*) y declare la variable como **protegida**
   d. Borrar la declaración de la variable en cada subclase
   e. Compile y corra los test



(*) otros lenguajes pueden tener otras reglas de visibilidad

Pull Up Method (p. 322 @ Refactoring, Fowler)

- Consideraciones de **endDate()**
  a. Las implementaciones son iguales
  b. Existe una superclase: *Product*
  c. Product no tiene un método con ese nombre
- Mecánicas (adaptada a este ejemplo)
  a. Cree un método en la superclase (Product) y copie el cuerpo del método: *endDate()*
     i. Ya se realizó PullUp Field de timePeriod
  b. Borre el método en una de las subclases
  c. Compile y corra los test
  d. Siga borrando el método en cada subclase, compilando y testeando hasta que solo quede el método en la superclase *Product*.
  e. Verifique que en las llamadas al método el tipo (de parámetros y variables) se ajuste al tipo de la superclase.

Modelo Resultante

Seguimos la mecánica del libro:

1. Pull Up startDate()
   a. Pull Up Field (timePeriod)
2. Pull Up endDate()

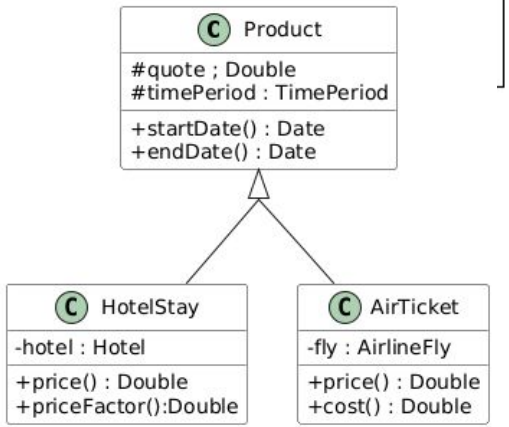
Podríamos haber hecho:

1. Pull Up Field (timePeriod)
2. Pull Up startDate()
3. Pull Up endDate

Product	**startDate()** {
	return timePeriod.start; }
	
	endDate() {
	return timePeriod.end; }

HotelStay	**price()**{
	return timePeriod.duration() * hotel.nightPrice() * hotel.discountRate(); }
	
	priceFactor(){
	return this.quote / this.price(); }

AirTicket	**price()** {
	return fly.price() * fly.promotionRate(); }
	
	cost() {
	return this.quote; }



El código resultante no es perfecto

Existen code smells ⇒ eventuales refactorings