Ayudantía UML

IIC2143: Ingeniería de Software
2023-1

Unified Modeling Language

Diagramas

UML permite visualizar, especificar, construir y documentar un sistema o proceso.

Diagramas que nos interesan en esta ayudantía:

- *Diagrama de **Clase***
- *Diagrama de **Secuencia***

Diagrama de Clase

Repaso de la notación vista en clases

Diagrama de Clases

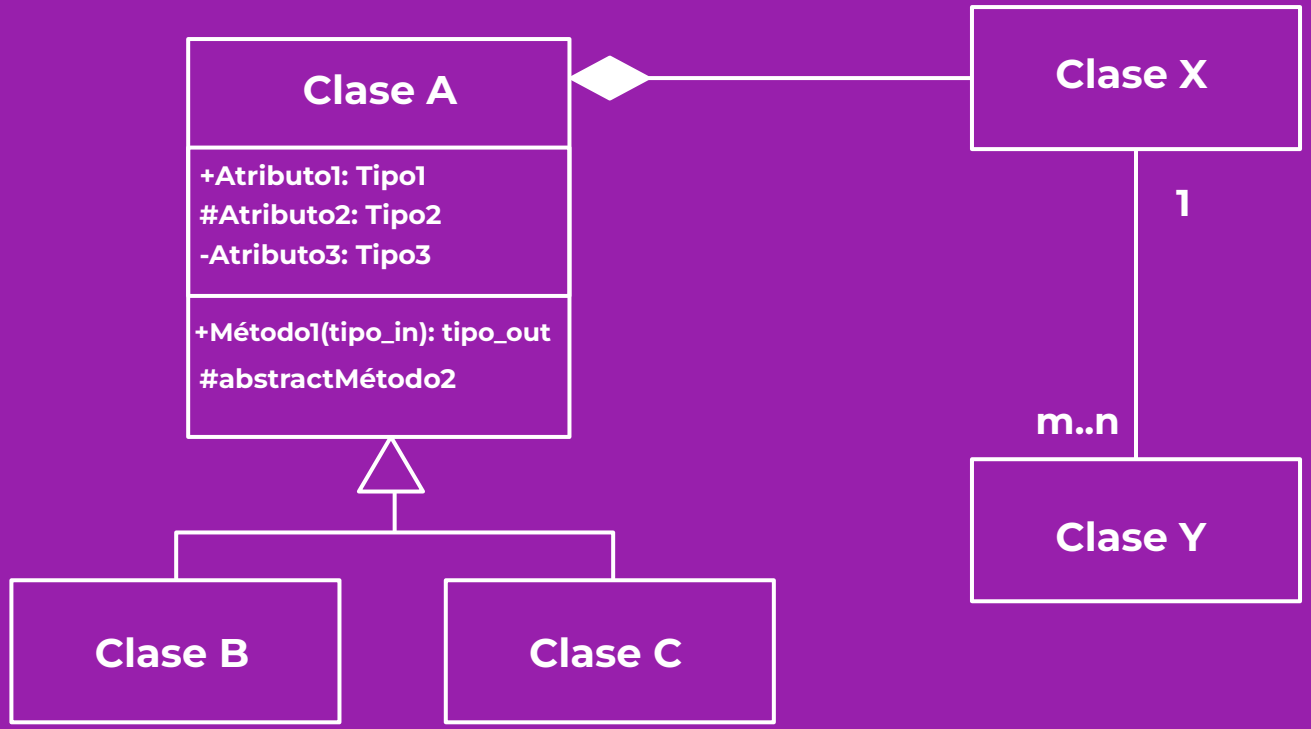


Diagrama de Secuencia

Repaso de la notación vista en clases

Diagrama de Secuencia

Diagrama de Secuencia

Permite modelar las interacciones entre objetos de un sistema cronológicamente durante un caso de uso en particular. Son útiles para observar el comportamiento de varios objetos y su interacción.

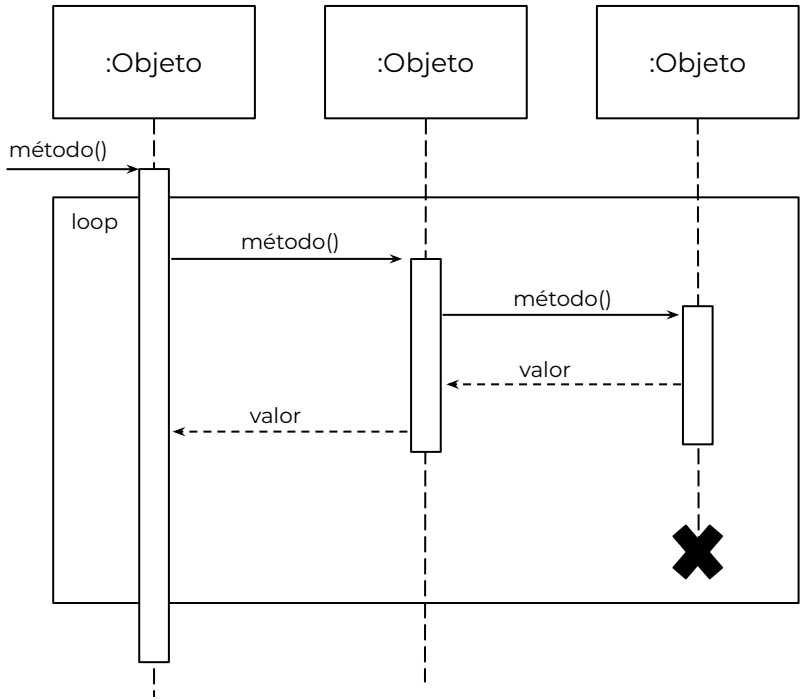


Diagrama de Secuencia

Objetos

Cada cajita es un objeto (instancia de clase)

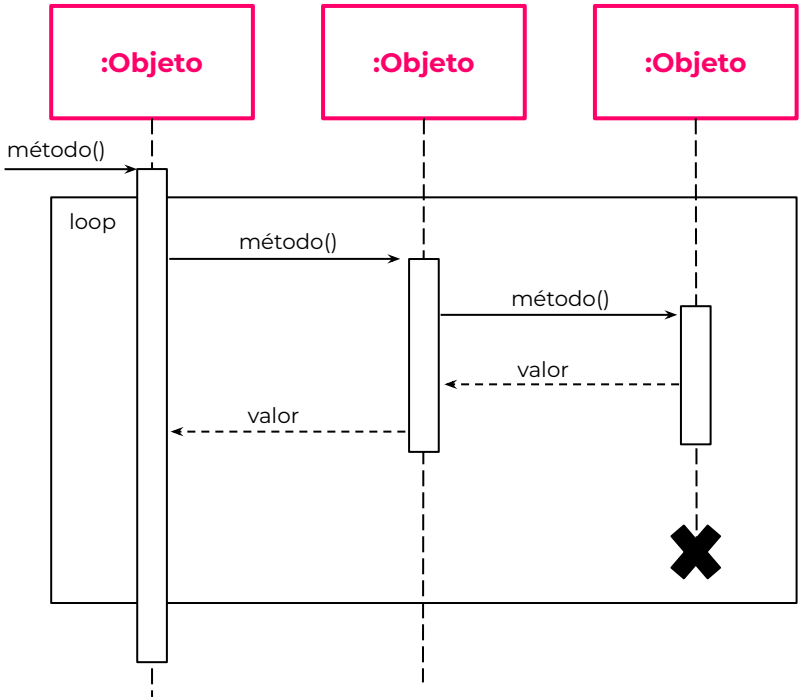


Diagrama de Secuencia

Línea de vida

La línea punteada indica la línea de vida del objeto.

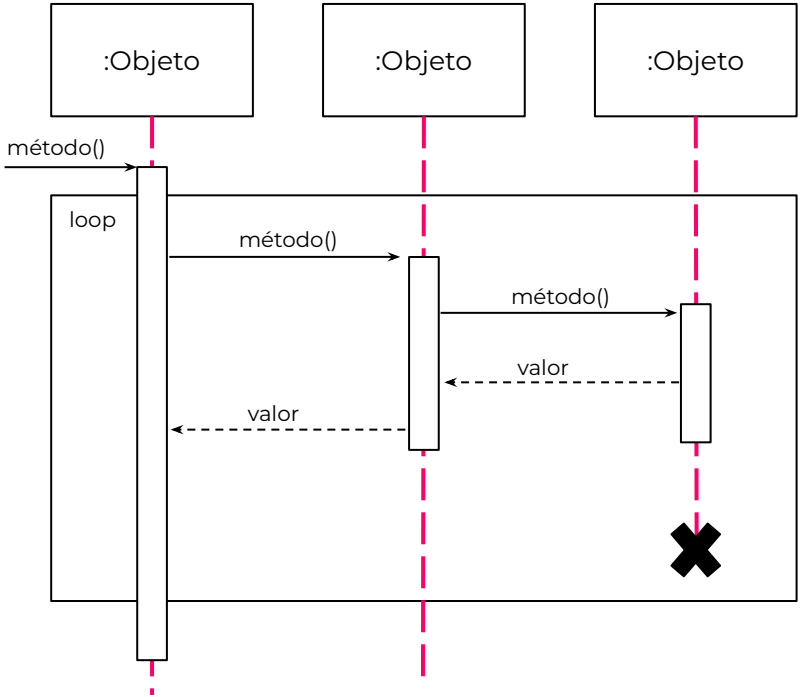


Diagrama de Secuencia

Tiempo de computación

Los rectángulos sobre la línea indican el tiempo que está involucrado el objeto en una computación.

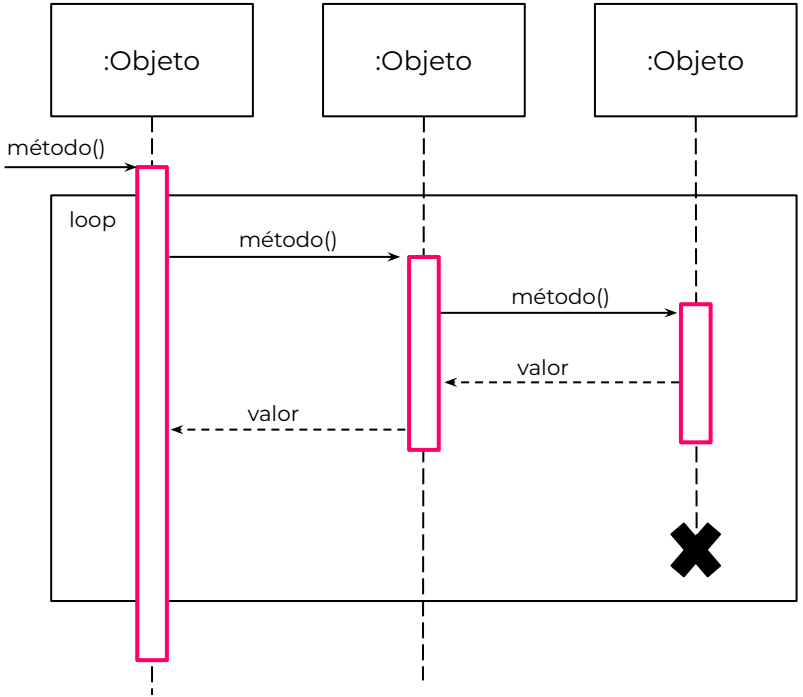


Diagrama de Secuencia

Interacciones

Las interacciones entre los objetos se indican con flechas dirigidas.

La secuencia de interacciones se lee de arriba abajo.

Hay diferentes interacciones y se refleja en el tipo de flecha.

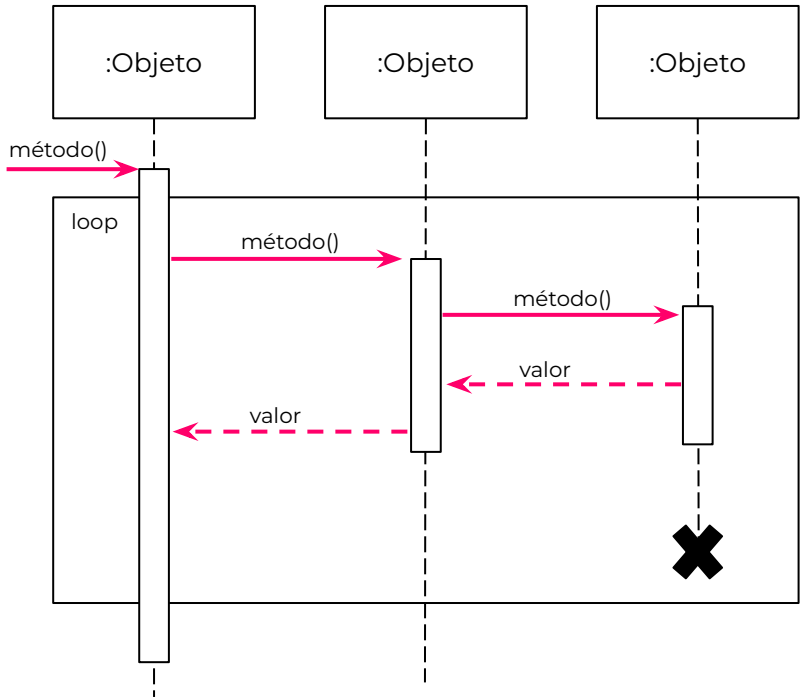


Diagrama de Secuencia

Métodos: argumentos y valores retornados

Se especifica el método llamado.

Es posible especificar los argumentos y los valores retornados correspondientes a las interacciones entre objetos al agregar dicha información en las interacciones.

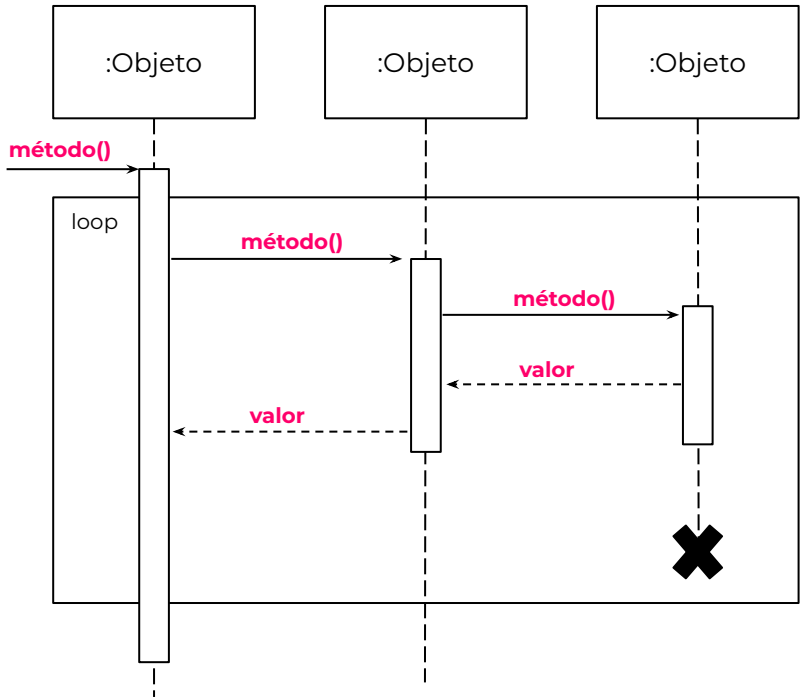


Diagrama de Secuencia

Eliminación de objetos

Es posible especificar cuando se eliminan algunos objetos del diagrama.

Un objeto puede ser eliminado por otro, o puede auto-eliminarse.

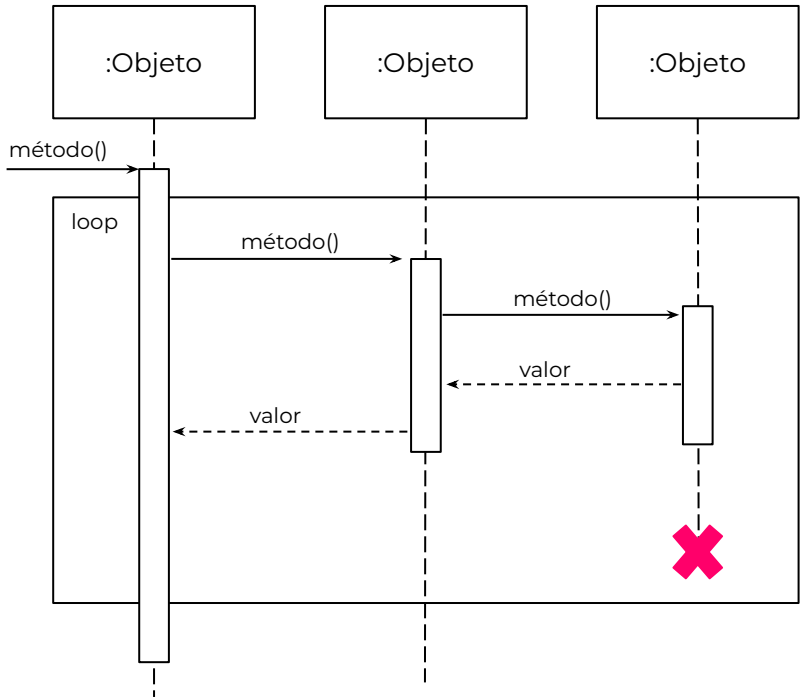
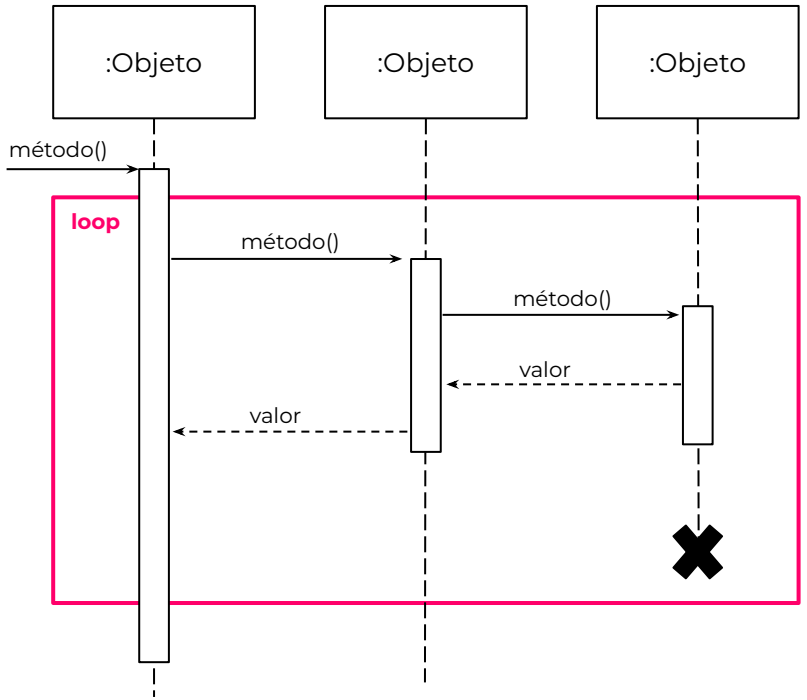


Diagrama de Secuencia

Loops y condicionales

Se utilizan operadores para mostrar interacciones más complejas como:

- *alt: if then else*
- *loop: for*
- *opt: if*



Ejercicio

Interrogación 2
2022-2

Ejercicio

A continuación se presenta el código Ruby que sirve para implementar un sistema de facturación. El objeto de tipo PaymentSystem mantiene una lista de las órdenes y de los productos que se manejan.

La orden incluye una serie de items (uno por cada producto comprado) que deben ser impreso uno por línea.

```ruby
class PaymentSystem
   @@orders
   @@products
   def self.process
       orders.each do |order|
           order.printOrder
       end
   end
   def self.getProduct (code)
       products.each |prod|
           if code == prod.code
               return prod
           end
   end
   def self.addProduct(product)
       products << product
   end
   def self.addOrder(order)
       orders << order end
   end
end
```

```ruby
class Product
   attr_reader :code, :name, :price
   def initialize (code, name, price)
       @code = code
       @name = name
       @price = price
   end
end
```

```ruby
class Item
   @productCode
   @quantity
   def printLine
    product = PaymentSystem.getProduct(@productCode)
    amount = product.price * quantity
    puts "#{quantity} #{product.code}
         #{product.name} #{product.price} #{amount}"
   end
end
```

```ruby
class Order
   @items
end
```

```ruby
class RetailOrder < Order
   def addItem (code, quantity)
       theItem = Item.new (productCode, quantity)
       @items << item
   end
   def printOrder
       items.each do |item|
           item.printLine
       end
   end
end
```

Ejercicio

A. Dibuje el ***diagrama de clases*** *que corresponde al código dado (incluya todo lo que puede obtenerse a partir del código).*

B. Dibuje un ***diagrama de secuencia*** *que muestre como se lleva a cabo la impresión de una factura cuando el objeto de tipo RetailOrder recibe un mensaje printOrder.*

```
class PaymentSystem
   @@orders
   @@products
   def self.process
       orders.each do |order|
           order.printOrder
       end
   end
   def self.getProduct (code)
       products.each |prod|
           if code == prod.code
               return prod
           end
   end
   def self.addProduct(product)
       products << product
   end
   def self.addOrder(order)
       orders << order end
   end
end
```

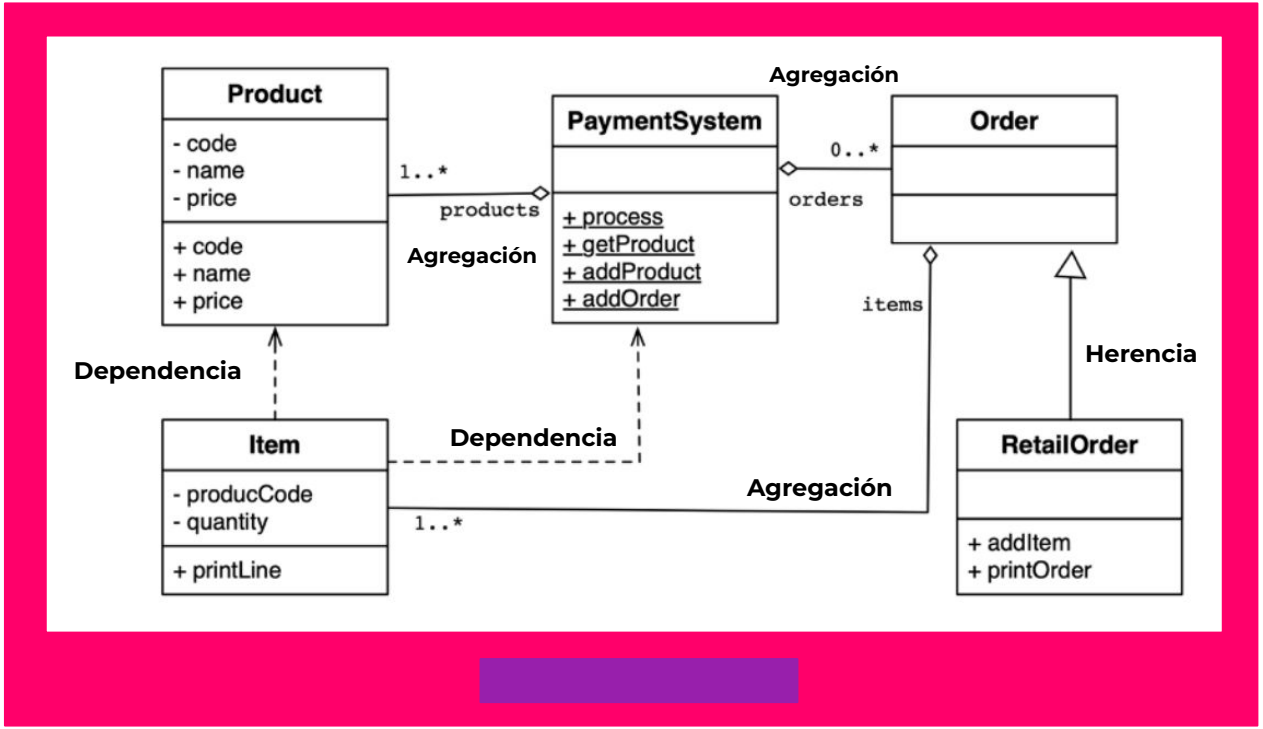
```
class Product
   attr_reader :code, :name, :price
   def initialize (code, name, price)
       @code = code
       @name = name
       @price = price
   end
end
```

```
class Item
   @productCode
   @quantity
   def printLine
    product = PaymentSystem.getProduct(@productCode)
    amount = product.price * quantity
    puts "#{quantity} #{product.code}
         #{product.name} #{product.price} #{amount}"
   end
end
```

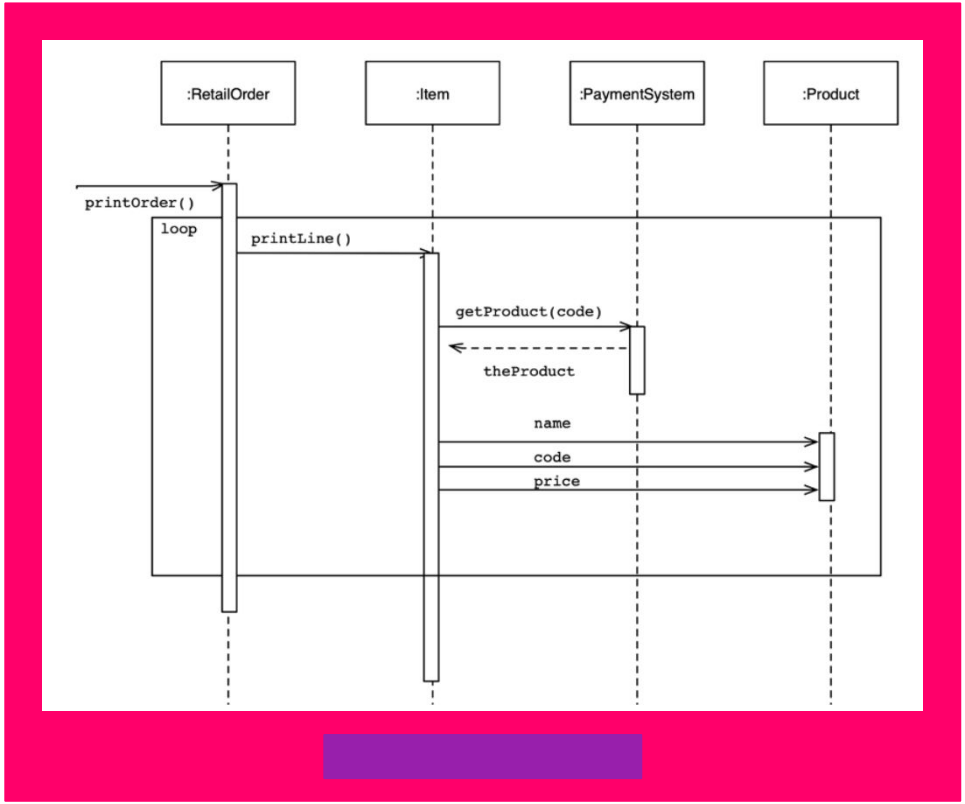
```
class Order
   @items
end
```

```
class RetailOrder < Order
   def addItem (code, quantity)
       theItem = Item.new (productCode, quantity)
       @items << item
   end
   def printOrder
       items.each do |item|
           item.printLine
       end
   end
end
```

Solución - Diagrama de Clases



Solución - Diagrama de Secuencia



Gracias!

IIC2143: Ingeniería de Software
2023-1