

# Hoja de Ruta

Semana 15 Abril

Patrones Construcción 1

Patrones de Dominio 1

Semana 23 Abril

Patrones Construcción 2

Patrones de Dominio 2

Semana 30 Abril

Lenguaje de Patrones: Test Doubles

Federico Balaguer: [federico.balaguer@lifa.unlp.edu.ar](mailto:federico.balaguer@lifa.unlp.edu.ar)



# ¿Cómo "construimos" objetos?

```
Car auto = new Car("Toyota", "Corolla", 2023 );
```

```
Car auto = new Car("Toyota", "Corolla", 2023, "Blanco");
```

```
Car auto = new Car("Toyota", "Corolla", 2023, "Blanco");  
InsurancePolicy policy = new InsurancePolicy(auto);
```

# ¿Cómo "construimos" objetos póliza?

```
1  Java
2  // Create a new Car object
3  Car auto = new Car("Toyota", "Corolla", 2023, "Blanco");
4
5  // Check the car's mileage
6  if (kilometraje < 10000) {
7
8      // Create a new InsurancePolicy object with a discount
9      InsurancePolicy policy = new InsurancePolicy(auto, cotización, deducible, descuento, accidentes);
10
11  } else {
12
13      // Create a new InsurancePolicy object with a 10% surcharge
14      InsurancePolicy policy = new InsurancePolicy(auto, cotización, deducible * 1.1, 0, accidentes);
15
16  }
```

Cúal es la regla de negocio en el código?

# ¿Cómo se usan las polizas?

```
public boolean checkClaim(InsurancePolicy policy, EventClaim claim) {  
    // Use green for method names and keywords (return, public, boolean)  
    return policy.acceptClaim(claim);  
}
```

5 minutos

- 1) Forme grupo de al menos 4
- 2) ¿Qué sabe hacer la póliza? (según este código)
- 3) ¿Qué otra cosa debería poder hacer la póliza?



# Dominio: seguros de autos (posibles coberturas)

## → AutoTranqui

- ◆ Grúa
  - Acarreo (5 por año)
  - Asistencia (10 por año)
- ◆ Lentes
  - Reparación \* 2
- ◆ Taller
  - Deducible = 20% Cotización
  - Presupuesto más bajo de 3
- ◆ Roturas
  - Cristales 50%
- ◆ Terceros
  - Internación: 300% Cotización
  - Rehabilitación: 150% Cotización

```
public boolean checkClaim(InsurancePolicy policy, EventClaim claim) {  
    // Use green for method names and keywords (return, public, boolean)  
    return policy.acceptClaim(claim);  
}
```

10 minutos

1. Diagrama UML
2. Pseudo código acceptClaim()
3. Cómo se crean instancias de AutoTranqui?

SE MUESTRA EN CLASE

SE MUESTRA EN CLASE



# Dominio: seguros de autos

## → AutoTranqui

- ◆ Grúa
  - Acarreo (5 por año)
  - Asistencia (10 por año)
- ◆ Lentes
  - Reparación \* 2
- ◆ Taller
  - Deducible 20% cotización
  - Presupuesto más bajo de 3
- ◆ Roturas
  - Cristales 50%
- ◆ Terceros
  - Internación: 300% Cotización
  - Rehabilitación: 150% Cotización
- ◆ Si varios cotitulares + daño persona -> cancelar seguro
- ◆ Si co/titular > 82 años -> acarreo a maximo 60 kms

## → AutoPlus

- ◆ Grúa
  - Acarreo (5 por año)
  - Asistencia (10 por año)
- ◆ Lentes
  - Reparación \* ∞
- ◆ Taller
  - Deducible 20% cotización
  - Presupuesto más bajo de 3
- ◆ Roturas
  - Espejos 100%
  - Cristales 100%
  - Granizo 100%
- ◆ Terceros
  - Internación: 300% Cotización
  - Rehabilitación: 150% Cotización
- ◆ Hotelería viajes (>500km )
  - \$\$ equivalente (7 noches ★★★★★)
- ◆ Si accidente Servicio Abogado
- ◆ Si daño a 3ro, servicio psicológico

## → AutoPlus

- ◆ Grúa
  - Acarreo (5 por año)
  - Asistencia (10 por año)
- ◆ Lentes
  - Reparación \* ∞
- ◆ Taller
  - Deducible 20% cotización
  - Presupuesto más bajo de 3
- ◆ Roturas
  - Espejos 100%
  - Cristales 100%
  - Granizo 100%
- ◆ Terceros
  - Internación: 300% Cotización
  - Rehabilitación: 150% Cotización
- ◆ Hotelería viajes (>500km )
  - \$\$ equivalente (7 noches ★★)
- ◆ Si accidente Servicio Abogado
- ◆ Si daño a 3ro, servicio psicológico

```
+-----+
| PolizaAutoPlus |
+-----+
| - numeroPoliza : String |
| - cliente : Cliente |
| - vehiculo : Vehículo |
| - coberturas:List<Cobertura> |
| + agregarCobertura(cobertura:Cobertura):void |
| + calcularCostoTotal():void |
| + procesarSiniestro(siniestro:Siniestro):void |
+-----+
```

```
public void procesarSiniestro(Siniestro siniestro) {
    for (Cobertura cobertura : coberturas) {
        if (cobertura.cubreSiniestro(siniestro)) {
            cobertura.procesarSiniestro(siniestro);
            break;
        }
    }
}
```

# PolizaAutoPiola y PolizaAutoPlus son lo mismo?

```
+-----+
| PolizaAutoTranqui |
+-----+
| - numeroPoliza : String
| - cliente : Cliente
| - vehiculo : Vehículo
| - coberturas:List<Cobertura>
| + agregarCobertura(cobertura:Cobertura):void
| + calcularCostoTotal():void
| + procesarSiniestro(siniestro:Siniestro):void
+-----+
```

```
+-----+
| PolizaAutoPlus |
+-----+
| - numeroPoliza : String
| - cliente : Cliente
| - vehiculo : Vehículo
| - coberturas:List<Cobertura>
| + agregarCobertura(cobertura:Cobertura):void
| + calcularCostoTotal():void
| + procesarSiniestro(siniestro:Siniestro):void
+-----+
```

```
public void procesarSiniestro(Siniestro siniestro) {
    for (Cobertura cobertura : coberturas) {
        if (cobertura.cubreSiniestro(siniestro)) {
            cobertura.procesarSiniestro(siniestro);
            break;
        }
    }
}
```

```
public void procesarSiniestro(Siniestro siniestro) {
    for (Cobertura cobertura : coberturas) {
        if (cobertura.cubreSiniestro(siniestro)) {
            cobertura.procesarSiniestro(siniestro);
            break;
        }
    }
}
```

5 Minutos: ¿cómo se puede eliminar el código repetido?

# Requerimientos 1

La compañía de seguros se dispone a:

- Crear nuevas pólizas
- Actualizar las existentes con nuevas reglas

¿Cuál es el principal problema?

- Familia de algoritmos?
- Cambios de estado de un objeto en tiempo de ejecución?
- Una definición única que puede ser “customizable” a casos específicos?
- Adaptar protocolos entre objetos?
- Manejar la configuración de objetos para que tengan diferente comportamiento?

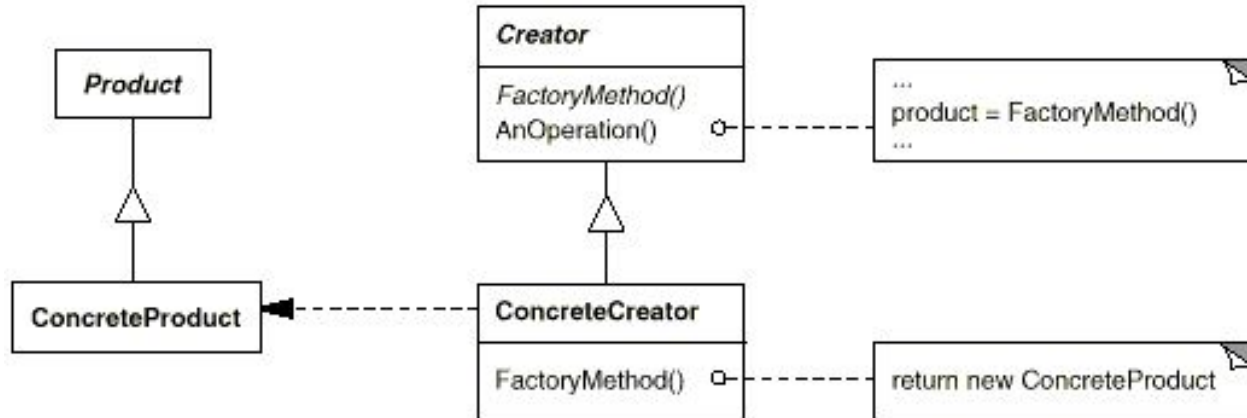
¿

•

SE MUESTRA EN CLASE

# Factory Method

- Intención: Define una “interface” para la creación de objetos, mientras permite que subclasses decidan qué clase se debe instanciar.
- Alias: Virtual Constructor



# Factory Method

## Participantes (Roles)

- **Product**
  - Define la interface of objetos creados por el “factory method”
- **Concrete Product**
  - Implementa la interface definida por el Product
- **Creator**
  - Declara el “factory method” (abstracto o con comportamiento default)
- **ConcreteCreator**
  - Implementa el “factory method”

# Qué parametros se necesitan para crear Pólizas?

## → AutoTranqui

- ◆ Grúa
  - Acarreo (5 por año)
  - Asistencia (10 por año)
- ◆ Lentes
  - Reparación \* 2
- ◆ Taller
  - Deducible 20% cotización
  - Presupuesto más bajo de 3
- ◆ Roturas
  - Cristales 50%
- ◆ Terceros
  - Internación: 300% Cotización
  - Rehabilitación: 150% Cotización
- ◆ Si varios cotitulares + daño persona -> cancelar seguro
- ◆ Si co/titular > 82 años -> acarreo a maximo 60 kms

## → AutoPlus

- ◆ Grúa
  - Acarreo (5 por año)
  - Asistencia (10 por año)
- ◆ Lentes
  - Reparación \* ∞
- ◆ Taller
  - Deducible 20% cotización
  - Presupuesto más bajo de 3
- ◆ Roturas
  - Espejos 100%
  - Cristales 100%
  - Granizo 100%
- ◆ Terceros
  - Internación: 300% Cotización
  - Rehabilitación: 150% Cotización
- ◆ Hotelería viajes (>500km )
  - \$\$ equivalente (7 noches ★★★★★)
- ◆ Si accidente Servicio Abogado
- ◆ Si daño a 3ro, servicio psicológico



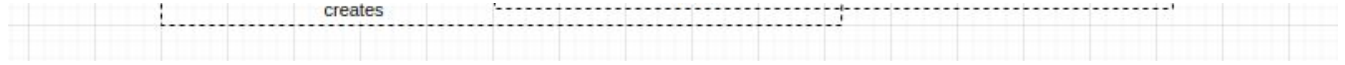
SE MUESTRA EN CLASE

# Requerimientos 2

Los diferentes tipos de póliza calculan el riesgo a su manera.



SE MUESTRA EN CLASE



## FactoryMethod: Génesis de una Póliza



```
public void agregarPoliza(creator:PolizaCreator, cliente:Cliente, auto:Vehiculo) {  
    // Validar si el auto ya está asegurado  
  
    // Agregar el auto a la lista de autos asegurados  
    autosAsegurados.add(creator.crearPoliza(auto, cliente));  
}
```

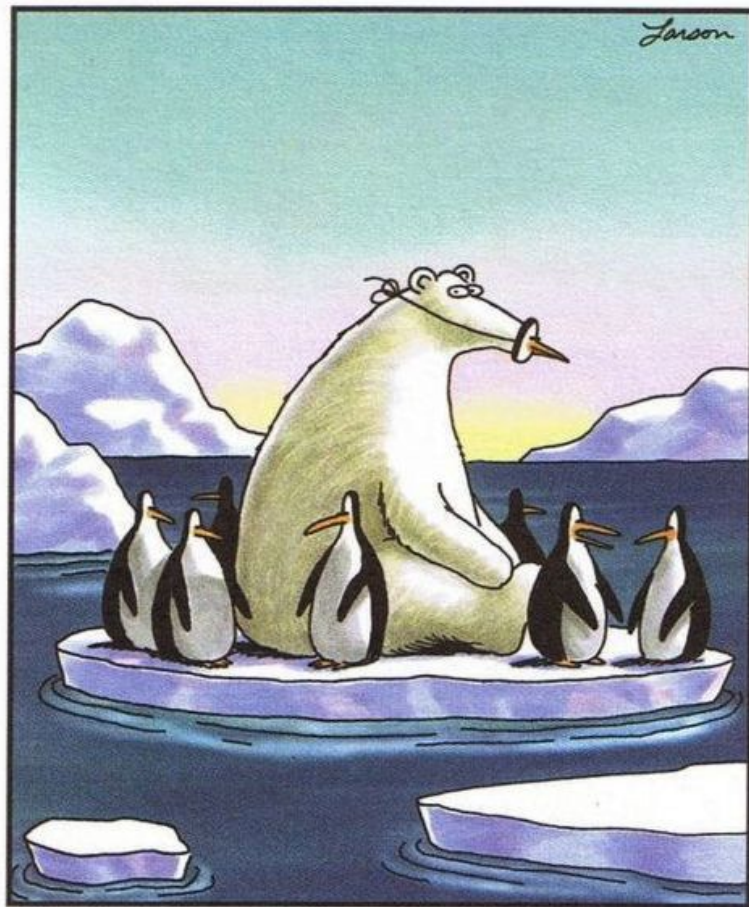
# Vale la pena Factory Method?

Si!

- Abstrae la construcción de un objeto
  - No acoplamiento entre el objeto que necesita un objeto y el objeto creado
- Facilita agregar al sistema nuevos tipos de productos
  - Cada Producto “solo” requiere un ConcreteCreator (\*)

No!

- Agrega complejidad en el código
  - 1 llamada a un constructor vs diseñar e implementar varios roles
- Cada Producto “solo” requiere un ConcreteCreator (!)



And now Edgar's gone. ...  
Something's going on around here.

## ¿Quién ha visto este tipo de código?

```
Customer customer = findCustomerId(10);

if (customer != null) {
    System.out.println("Customer email: " + customer.getEmail());
} else {
    System.out.println("Customer with ID 10 not found.");
}
```

```
27 Customer customer = findCustomerId(10);
28
29 if (customer != null) {
30     String email = customer.getEmail();
31     if (email != null){
32         System.out.println("Customer email: " + email);
33     } else{System.out.println("Customer email: not in the system");
34     }
35 } else {
36     System.out.println("Customer with ID 10 not found.");
37 }
```

¿Que significa?

¿Por qué tantos condicionales? ¿Cual es el problema?

```
40  Customer customer = findCustomerById(10);
41
42  if (customer != null) {
43      Riesgo riesgo = customer.getRiesgo();
44      if (riesgo != null){
45          riesgo.actualizarRangoFechasHoy();
46      } else {//no hacemos nada}
47  }
48 } else {//nada por hacer}
49     System.out.println("Customer with ID 10 not found.");
50 }
```

# Null Object Pattern (Woolf&Johnson)

**Objetivo:** Proporciona un sustituto para otro objeto que comparte la misma interfaz pero no hace nada. El NullObject encapsula las decisiones de implementación de cómo "no hacer nada" y oculta esos detalles de sus colaboradores

**Alias:** Stub

**Consecuencia:**

- Elimina todos los condicionales que verifican si la referencia a un objeto es NULL
- Hace explícito elementos del dominio que hacen “nada”



Ahora que saben de que se trata lean el paper :-)



# Así quedaría el código gracias a NullObject para Customer y Riesgo

```
Customer customer = findCustomerId(10);  
Riesgo riesgo = customer.getRiesgo();  
riesgo.actualizarRangoFechasHoy();
```



```
Customer customer = findCustomerId(10);  
(customer.getRiesgo()).actualizarRangoFechasHoy();
```



```
((this.findCustomerId(10)) customer.getRiesgo()).actualizarRangoFechasHoy();
```

# Para la semana que viene

1. Leer NullObject
2. Responder las siguientes preguntas
  - a. Ventajas
  - b. Desventajas
  - c. Piense en un caso (**real**) en donde se podría aplicar
3. Leer FactoryMethod del GOF
  - a. Traer preguntas