Hoja de Ruta

Semana 15 Abril

- Patrones Construccion 1
- Patrones de Dominio 1

Semana 23 Abril

- Patrones Construccion 2
- Patrones de Dominio 2

Semana 30 Abril

- Lenguaje de Patrones: Test Doubles

Federico Balaguer: federico.balaguer@lifia.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?

```java
Java
// Create a new Car object
Car auto = new Car("Toyota", "Corolla", 2023, "Blanco");

// Check the car's mileage
if (kilometraje < 10000) {

  // Create a new InsurancePolicy object with a discount
  InsurancePolicy policy = new InsurancePolicy(auto, cotización, deducible, descuento, accidentes);

} else {

  // Create a new InsurancePolicy object with a 10% surcharge
  InsurancePolicy policy = new InsurancePolicy(auto, cotización, deducible * 1.1, 0, accidentes);

}
```

Cúal es la regla de negocio en el código?

Póliza(Auto) — Reclamos

Poliza (auto)

Reclamos

¿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

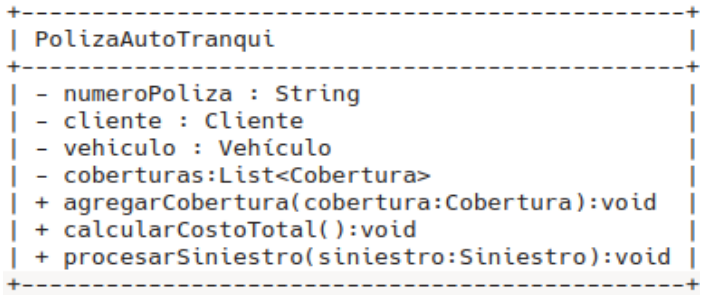
*Cotización = cotización auto

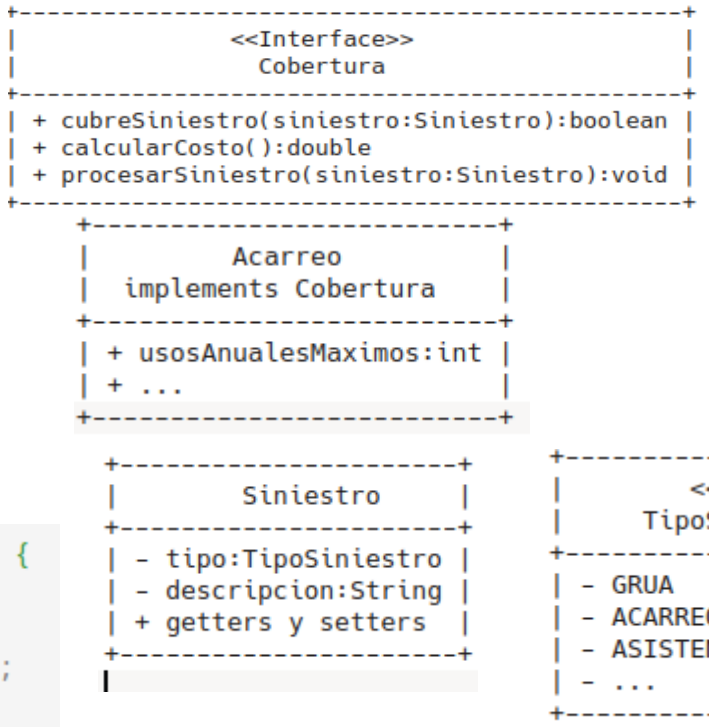
```java
public boolean checkClaim(InsurancePolicy policy, EventClaim claim) {
  // Use green for method names and keywords (return, public, boolean)
  return policy.acceptClaim(claim);
}
```

10 minutos

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

Gemini dice…



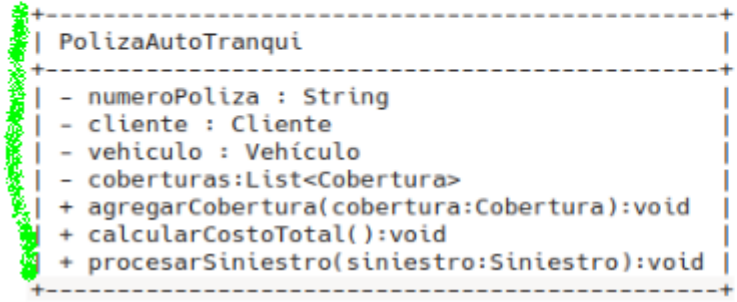


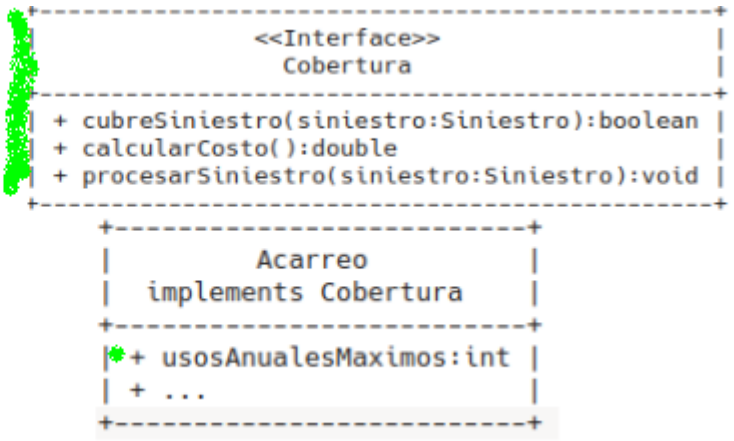

```java
public void procesarSiniestro(Siniestro siniestro) {
    for (Cobertura cobertura : coberturas) {
        if (cobertura.cubreSiniestro(siniestro)) {
            cobertura.procesarSiniestro(siniestro);
            break;
        }
    }
}
```

10 minutos

1. Evalúe la definición de clases
2. Evalúe el código

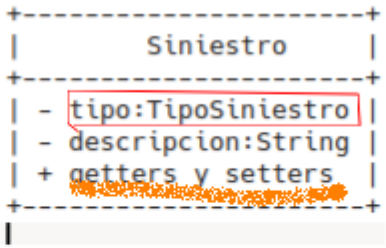
Gemini dice…

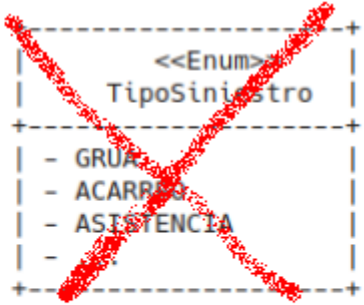





```
public void procesarSiniestro(Siniestro siniestro) {
    for (Cobertura cobertura : coberturas) {
        if (cobertura.cubreSiniestro(siniestro)) {
            cobertura.procesarSiniestro(siniestro);
            break
        }
    }
}
```

1 Siniestro -> 1 Cobertura





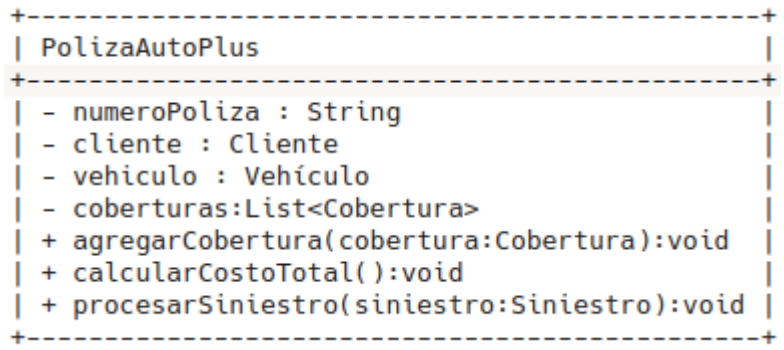
TipoSiniestro es un flag... no OO

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

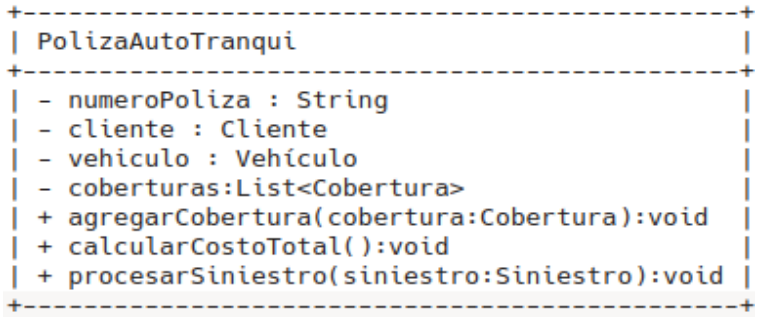
- AutoPlus
  - Grúa
    - Acarreo (5 por año)
    - Asistencia (10 por año)
  - Lentes
    - Reparación * ∞
  - Taller
    - Deducible 20% cotización
    - Presupuesto más bajo de 3
  - Roturas
    - Espejos 100%
    - Cristales 100%
    - Granizo 100%
  - Terceros
    - Internación: 300% Cotización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

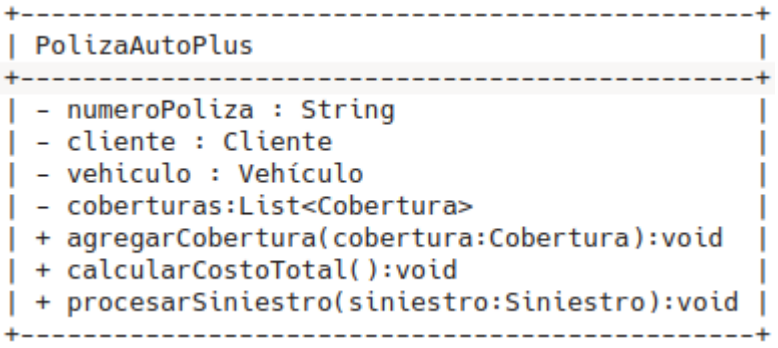
- AutoPlus
  - Grúa
    - Acarreo (5 por año)
    - Asistencia (10 por año)
  - Lentes
    - Reparación * ∞
  - Taller
    - Deducible 20% cotización
    - Presupuesto más bajo de 3
  - Roturas
    - Espejos 100%
    - Cristales 100%
    - Granizo 100%
  - Terceros
    - Internación: 300% Cotización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




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

PolizaAutoPiola y PolizaAutoPlus son lo mismo?






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

```java
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:

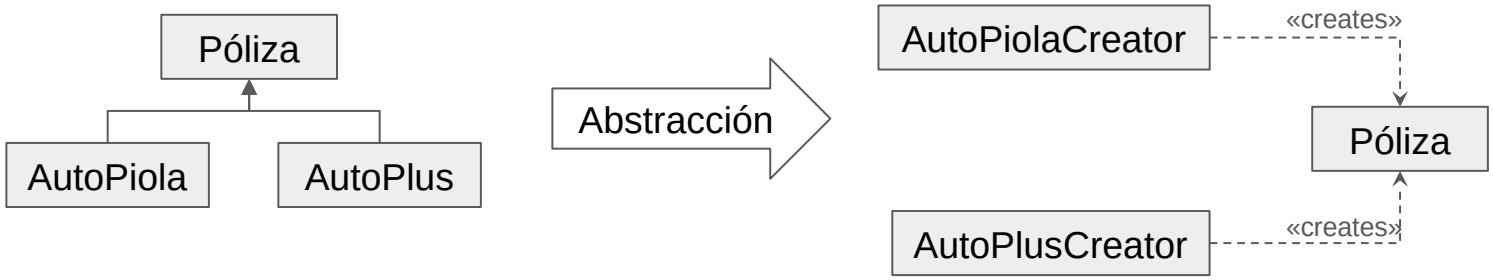
- Crear nuevas pólizas
- Actualizar las existentes con nuevas reglas

¿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?

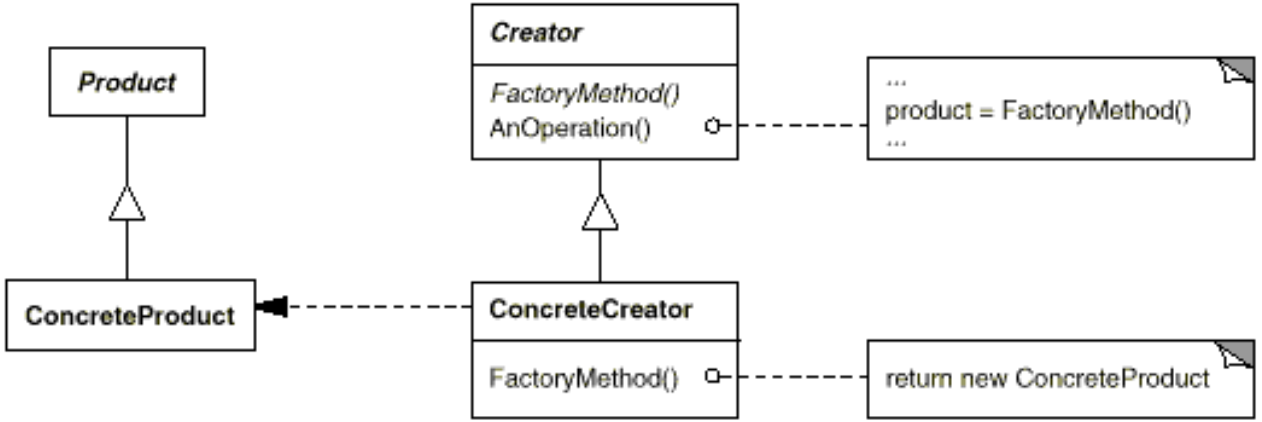
¿Y si hubiera solo una clase Póliza que se “arma” con diferentes coberturas?

- AutoPiola y AutoPlus
  - Dejan de ser casos particulares (sublclases) de Póliza
  - Pasan a ser maneras de configurar una Póliza



Factory Method

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



Factory Method

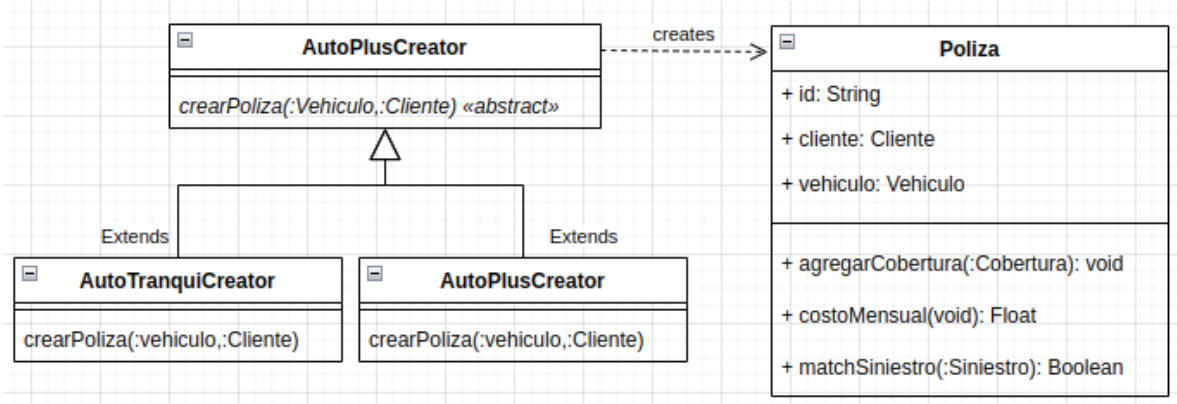
Participantes (Roles)

- 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

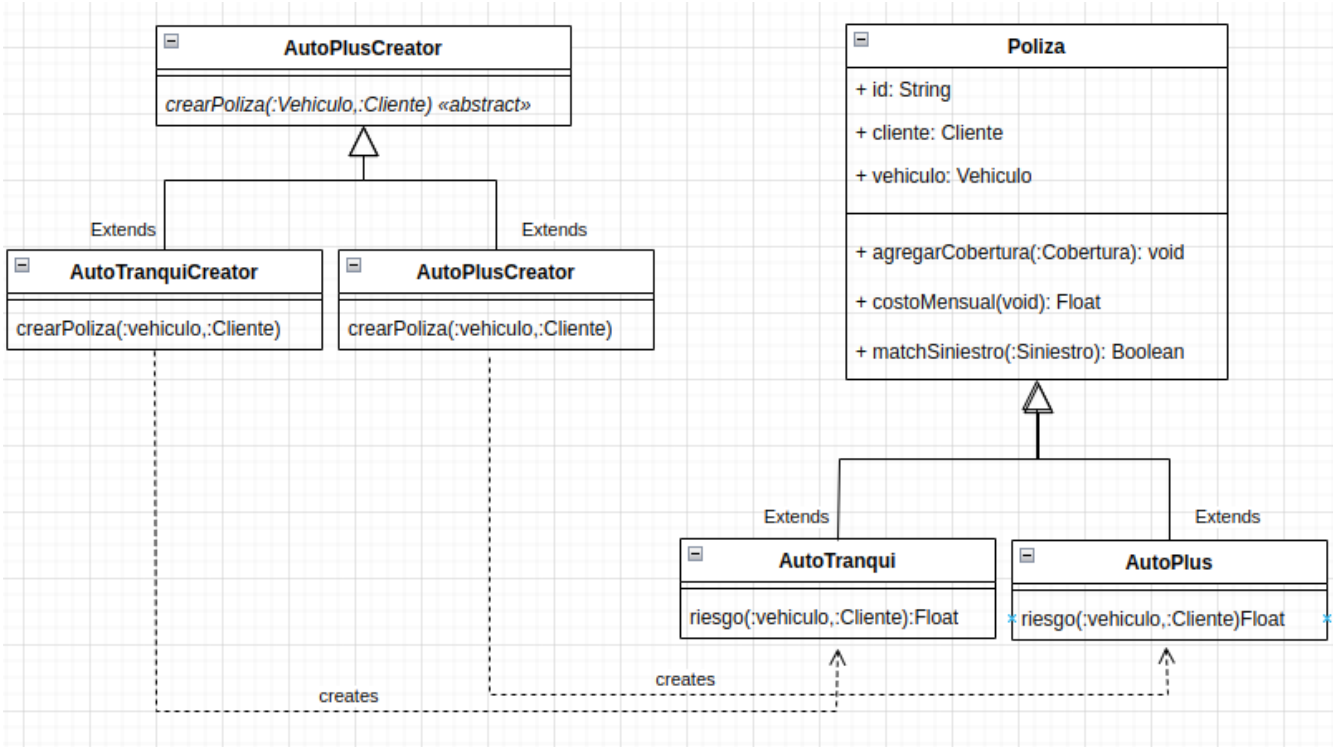
- AutoPlus
  - Grúa
    - Acarreo (5 por año)
    - Asistencia (10 por año)
  - Lentes
    - Reparación * ∞
  - Taller
    - Deducible 20% cotización
    - Presupuesto más bajo de 3
  - Roturas
    - Espejos 100%
    - Cristales 100%
    - Granizo 100%
  - Terceros
    - Internación: 300% Cotización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



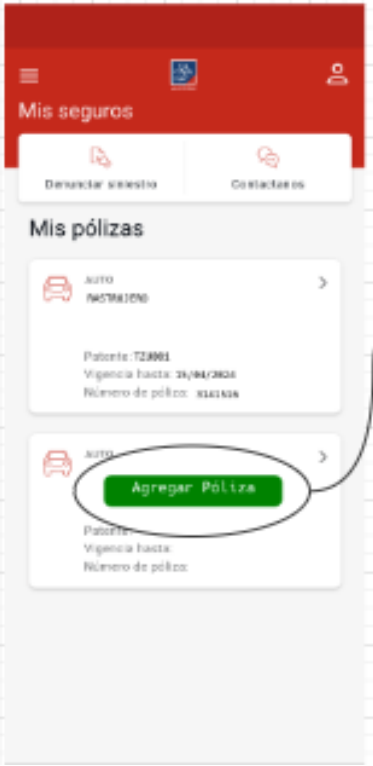
- Factory Method (GoF)
  - Define una “interface” de 1 método crearPoliza()
  - Para construir diferentes cosas relacionadas (en un dominio)
  - Cada versión del método tiene que estar en una clase creadoras (polimorfismo)
  - Las clases “creadoras” pueden tener otros comportamientos (proxima semana TypeObject)

Requerimientos 2

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



FactoryMethod: Génesis de una Póliza




```java
public void agregarPoliza(creator:PolizaCreator, cliente:Cliente, auto:Vehiculo) {
  // Validar si el auto ya esta asegurado

  // Agregar el auto a la lista de autos asegurados
  autosAsegurados.add(creator.crearPoliza(auto, cliente));

}
```

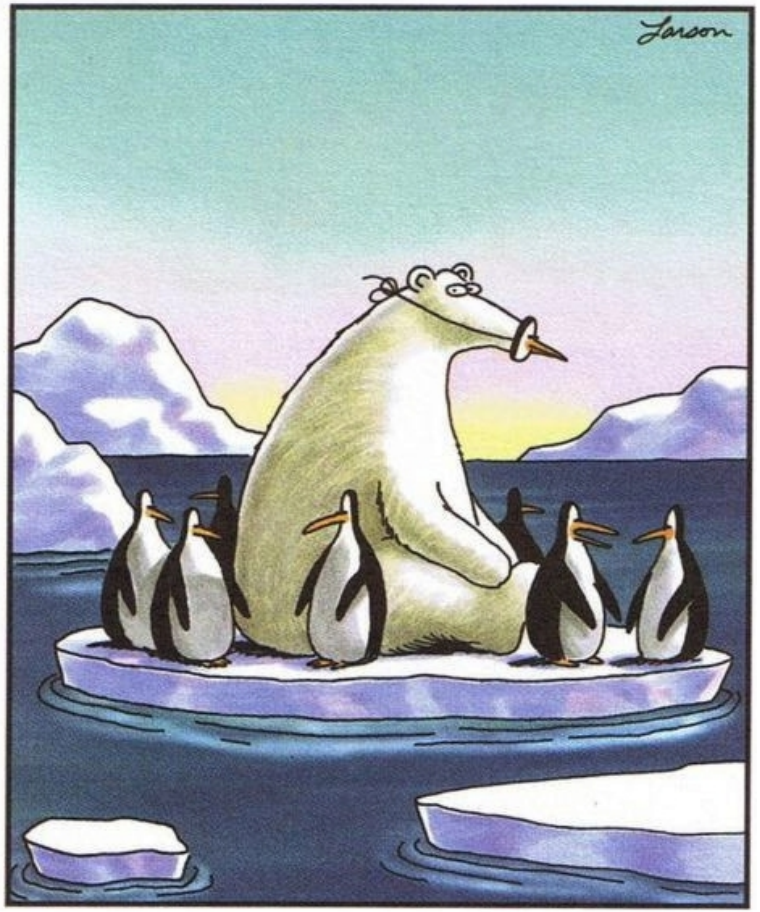
Vale la pena Factory Method?

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 codigo
  - 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.