

METODOLOGÍAS DE PROGRAMACIÓN I

Patrón creacional *Factory Method*

Situación de ejemplo

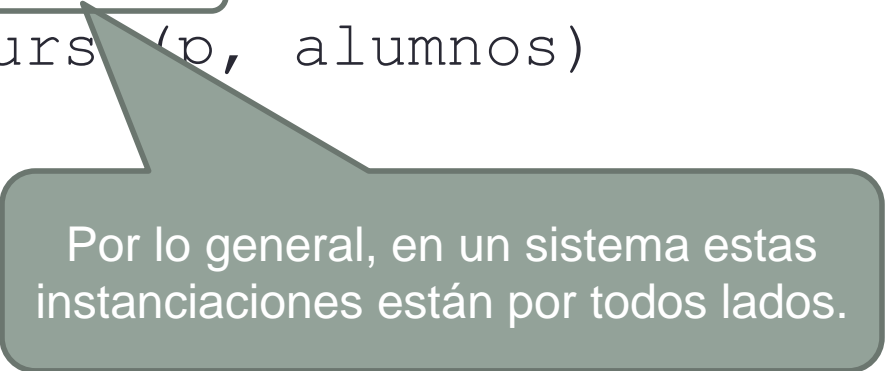
El departamento de alumnos de una universidad prepara cursos de grado y les avisa a los alumnos quien es el profesor del curso, para ello cuenta con el siguiente código:

```
class DepartamentoDeAlumnos
    Curso crearCurso(List alumnos)
        p = new Profesor()
        return new Curso(p, alumnos)
```

Situación de ejemplo

El departamento de alumnos de una universidad prepara cursos de grado y les avisa a los alumnos quien es el profesor del curso, para ello cuenta con el siguiente código:

```
class DepartamentoDeAlumnos
    Curso crearCurso(List alumnos)
        p = new Profesor()
        return new Curso(p, alumnos)
```



Por lo general, en un sistema estas instanciaciones están por todos lados.

Problema

¿Qué pasaría con el código si aparecen las clases *ProfesorSuplente* o *ProfesorVisitante*? Las cuales se usan en determinadas situaciones.

```
class DepartamentoDeAlumnos
    Curso crearCurso(List alumnos)
        if (algunaOpcion == NORMAL)
            p = new ProfesorLocal()
        if (algunaOpcion == SUPLENCIA)
            p = new ProfesorSuplente()
        if (algunaOpcion == VISITA)
            p = new ProfesorVisitante()
        return new Curso(p, alumnos)
```

Problema

Hay que cambiar por todos lados estas instanciaciones. Además de agregar condicionales para saber que crear en cada momento.

¿Qué pasaría con el código si aparecen las clases *ProfesorSuplente* o *ProfesorVisitante* en las cuales se usan en determinadas situaciones.

```
class DepartamentoDeAlumnos {
    Curso crearCurso(List alumnos)
    {
        if (algunaOpcion == NORMAL)
            p = new ProfesorLocal()
        if (algunaOpcion == SUPLENCIA)
            p = new ProfesorSuplente()
        if (algunaOpcion == VISITA)
            p = new ProfesorVisitante()
        return new Curso(p, alumnos)
    }
}
```

Problema

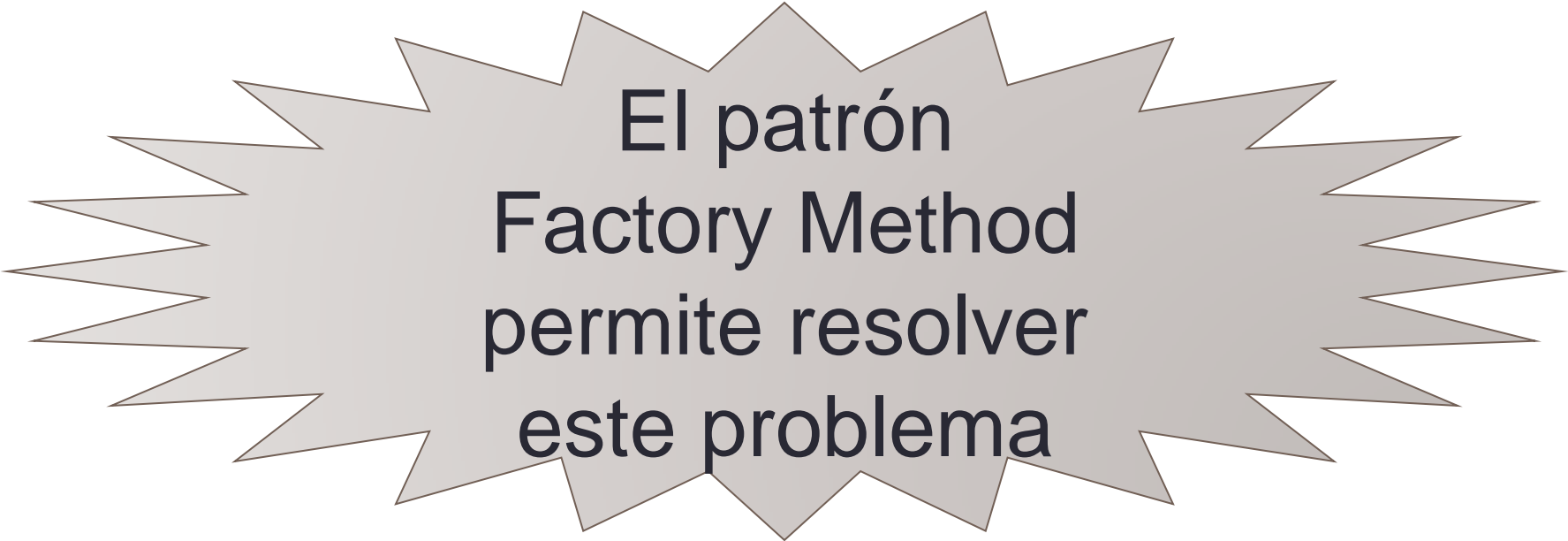
¿Qué pasaría con el código si aparecen las clases *ProfesorSuplente* o *ProfesorVisitante*? Las cuales se usan en determinadas situaciones.

```
class DepartamentoDeAlumnos
    Curso crearCurso(List alumnos)
        p = metodoMagico(algunaOpcion)
        return new Curso(p, alumnos)
```

Sería interesante contar con un método que solucione nuestro problema

Motivación

Buscamos un mecanismo donde se encapsule la creación de los objetos, de modo tal que el cliente solo pida la creación de ellos, sin especificar su clase.



El patrón
Factory Method
permite resolver
este problema

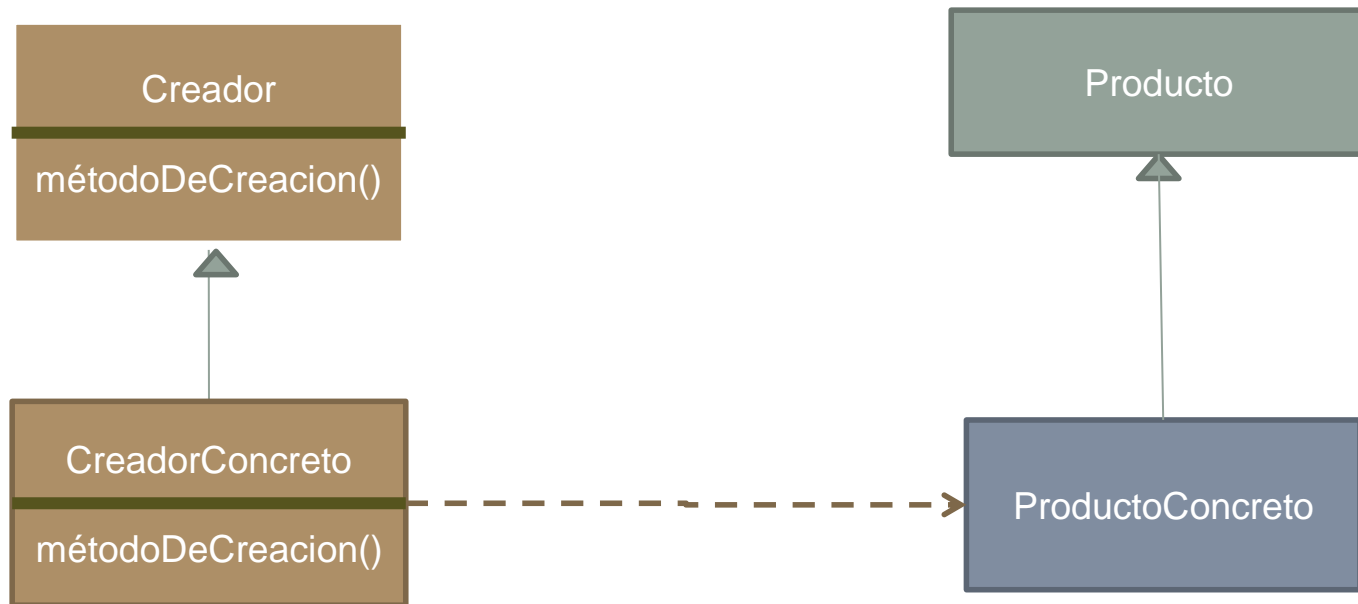
Factory method

Propósito: define una interfaz para crear un objeto, pero dejan que sean las subclasses quienes decidan que clase de objetos instanciar.

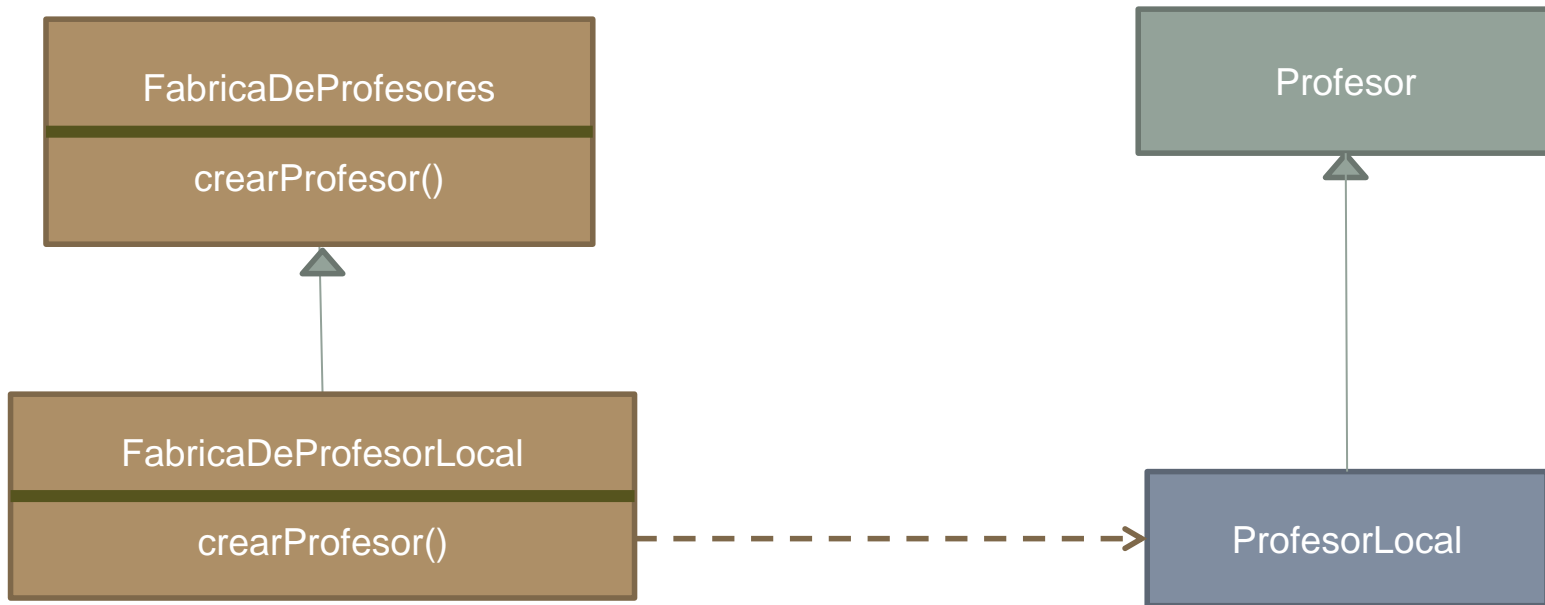
Aplicabilidad: usarlo cuando

- Una clase no puede preveer la clase de objetos que debe crear.
- Una clase quiere que sean sus subclasses quienes especifiquen los objetos que esta crea.
- Las clases delegan la responsabilidad en una de entre varias clases auxiliares y queremos localizar que subclase auxiliar concreta es en la que delega.

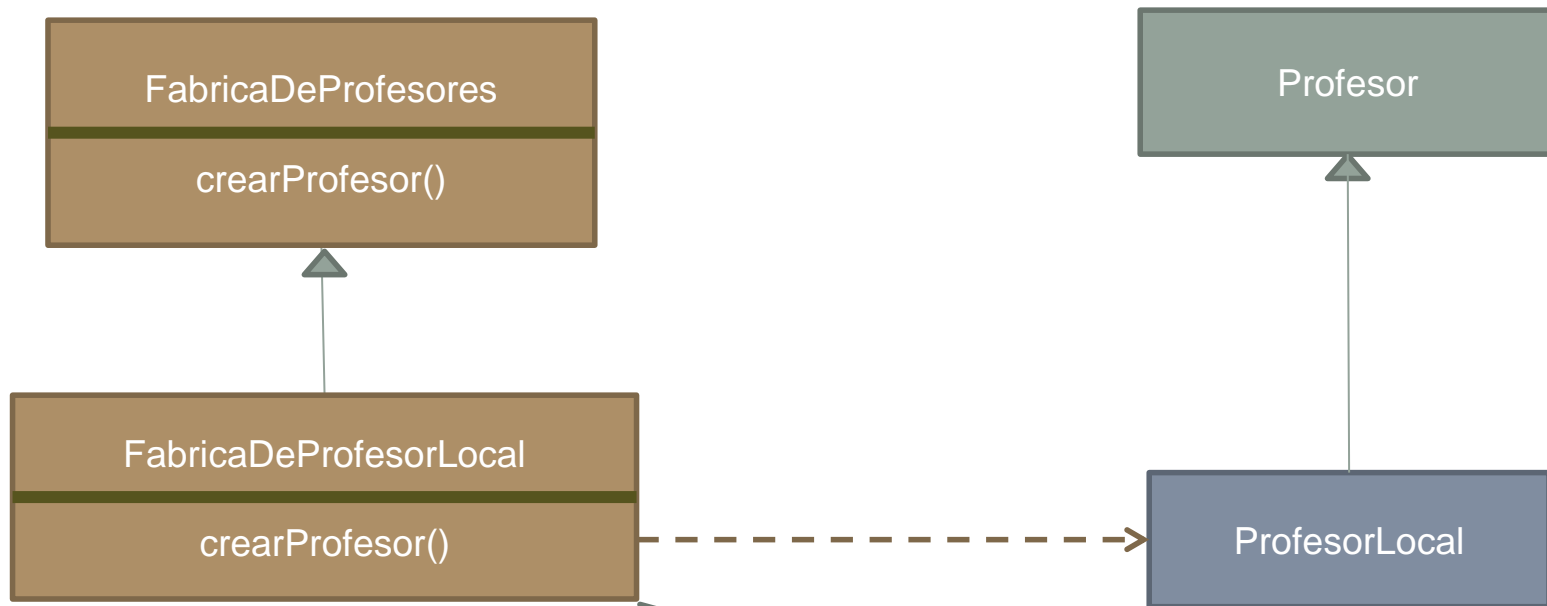
Factory method - Estructura



Factory method - Estructura



Factory method - Estructura



Debe existir una fábrica por cada producto distinto..
En nuestro ejemplo son tres fábricas (Local,
Suplente y Visitante)

Factory method - Implementación

```
abstract class FabricaDeProfesores
    static Profesor crearProfesor(int queProfesor)
        FabricaDeProfesores fabrica = null;

        if (queProfesor == LOCAL)
            fabrica = new FabricaDeProfesorLocal()
        if (queProfesor == VISITANTE)
            fabrica = new FabricaDeProfesorVisitante()
        if (queProfesor == SUPLENTE)
            . . .
        return fabrica.crearProfesor()

abstract Profesor crearProfesor()
```

Factory method - Implementación

```
abstract class FabricaDeProfesores
{
    static Profesor crearProfesor(int queProfesor)
    {
        FabricaDeProfesores fabrica = null;

        if (queProfesor == LOCAL)
            fabrica = new FabricaDeProfesorLocal();
        if (queProfesor == VISITANTE)
            fabrica = new FabricaDeProfesorVisitante();
        if (queProfesor == SUPLENTE)
            . . .

        return fabrica.crearProfesor();
    }
}
```

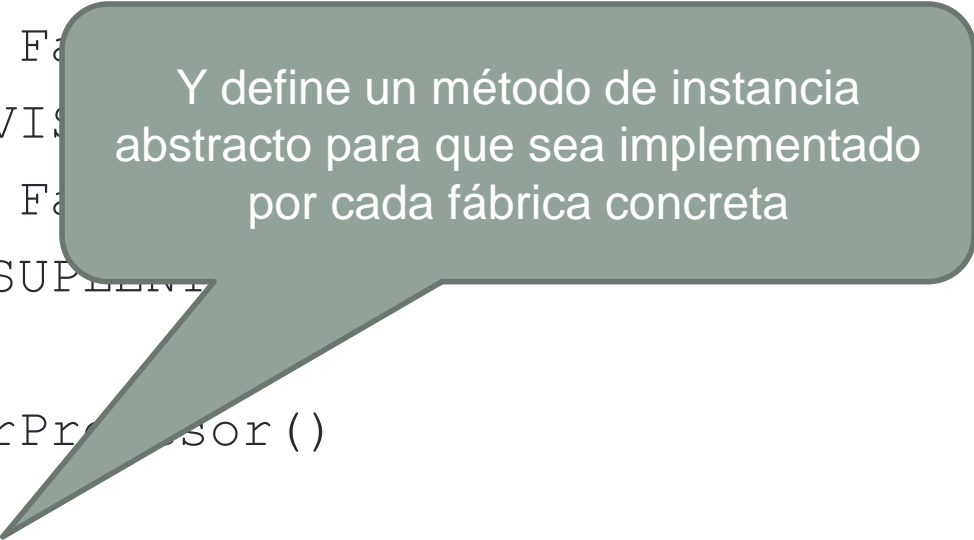
abs Implementamos en la clase abstracta un método de clase que es el que, según la opción indicada, crea la instancia de la fábrica concreta correspondiente.

Factory method - Implementación

```
abstract class FabricaDeProfesores
{
    static Profesor crearProfesor(int queProfesor)
    {
        FabricaDeProfesores fabrica = null;

        if (queProfesor == LOCAL)
            fabrica = new FabricaLocal();
        if (queProfesor == VISITA)
            fabrica = new FabricaVisita();
        if (queProfesor == SUPLENTE)
            . . .
        return fabrica.crearProfesor();
    }
}

abstract Profesor crearProfesor()
```



Y define un método de instancia abstracto para que sea implementado por cada fábrica concreta


Factory method - Implementación

```
class FabricaDeProfesorLocal : FabricaDeProfesor

    Profesor crearProfesor()
        return new ProfesorLocal()
```

Factory method - Implementación

```
class FabricaDeProfesorLocal : FabricaDeProfesor  
  
    Profesor crearProfesor()  
        return new ProfesorLocal()
```



La fábrica concreta devuelve la instancia de profesor que corresponde.

Factory method - Implementación

```
class FabricaDeProfesorLocal : FabricaDeProfesor

    Profesor crearProfesor()
        p = new ProfesorLocal()
        p.setNombreYApellido(leerPorTeclado())
        p.setMateria(recuperarDesdeBaseDeDatos())
        p.setAyudante(new
                        Ayudante(leerDatosAyudante()))

    return p
```

En este mismo método se "elabora" todo el producto. Por ejemplo se puede leer los datos del profesor, realizar la composición que corresponda, etc.

Factory method - Implementación

```
abstract class FabricaDeProfesores
{
    static Profesor crearProfesor(int queProfesor)
    {
        FabricaDeProfesores fabrica = null;

        if (queProfesor == LOCAL)
            fabrica = new FabricaDeProfesorLocal();
        if (queProfesor == VISITANTE)
            fabrica = new FabricaDeProfesorVisitante();
        if (queProfesor == SUPLENTE)
            . . .

        return fabrica.crearProfesor();
    }
}
```

El método mágico

Factory method - Implementación

```
class DepartamentoDeAlumnos
    Curso crearCurso(List alumnos)
        p = FabricaDeProfesor.crearProfesor(algunaOpcion)
        return new Curso(p, alumnos)
```

Factory method - Implementación

```
class DepartamentoDeAlumnos
    Curso crearCurso(List alumnos)
        p = FabricaDeProfesor.crearProfesor(algunaOpcion)
        return new Curso(p, alumnos)
```

¿Cuál es la modificación que hay que hacer en este método si a futuro se agregan más tipos de profesores?

Factory method – Ventajas

- Este patrón elimina la necesidad de ligar clases específicas de la aplicación a nuestro código.
- El código solo trata con la interfaz del Producto.
- Crear objetos dentro de una clase con un método de fabricación es siempre más flexible que hacerlo directamente.