

Plantillas de clases (Templates)

1. Plantilla de funciones o funciones genéricas
2. Métodos genéricos
3. Plantillas de clases o clases genéricas
4. Sintaxis
5. Creación de objetos
6. Representación en UML
7. Miembros de las clases genéricas
8. Consejos

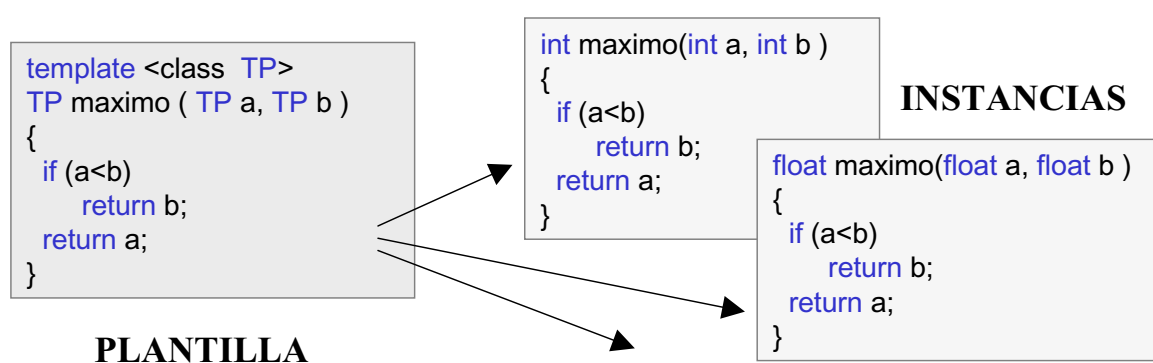
1

Plantillas de clases (Templates)

Plantillas de funciones (funciones genéricas)

Hemos visto anteriormente, que las funciones genéricas son un mecanismo C++ que permite definir una función mediante uno o varios parámetros (tipos genéricos). A partir de estas plantillas el compilador es capaz de generar código de funciones distintas que comparten ciertas características.

Las funciones así generadas se denominan **instancias** o **especializaciones** de la plantilla.

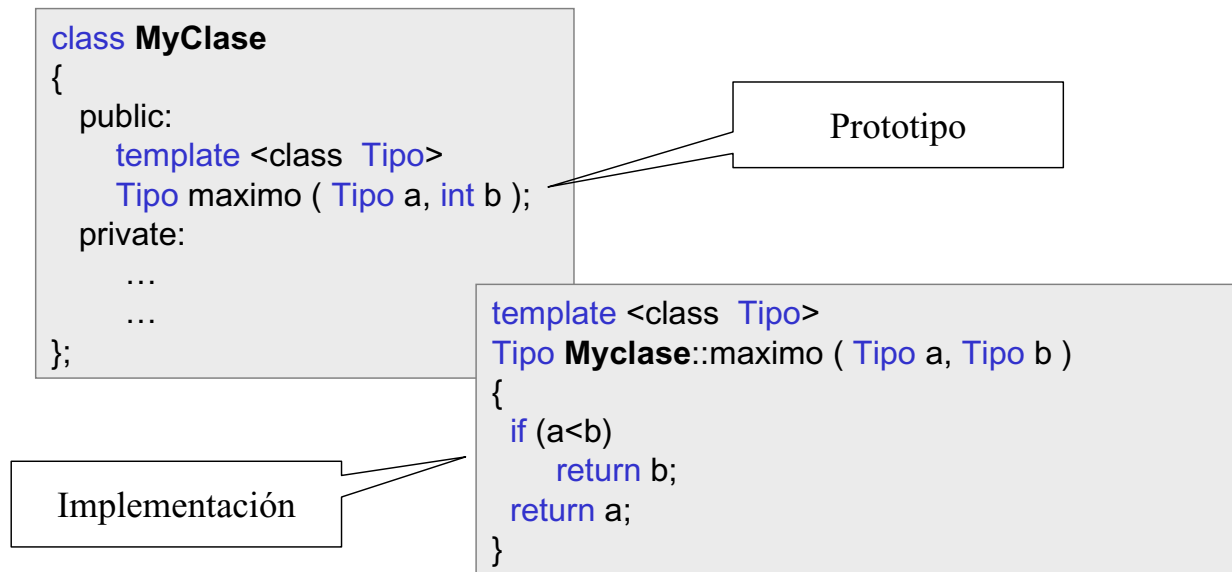


2

Plantillas de clases (Templates)

Métodos genéricos

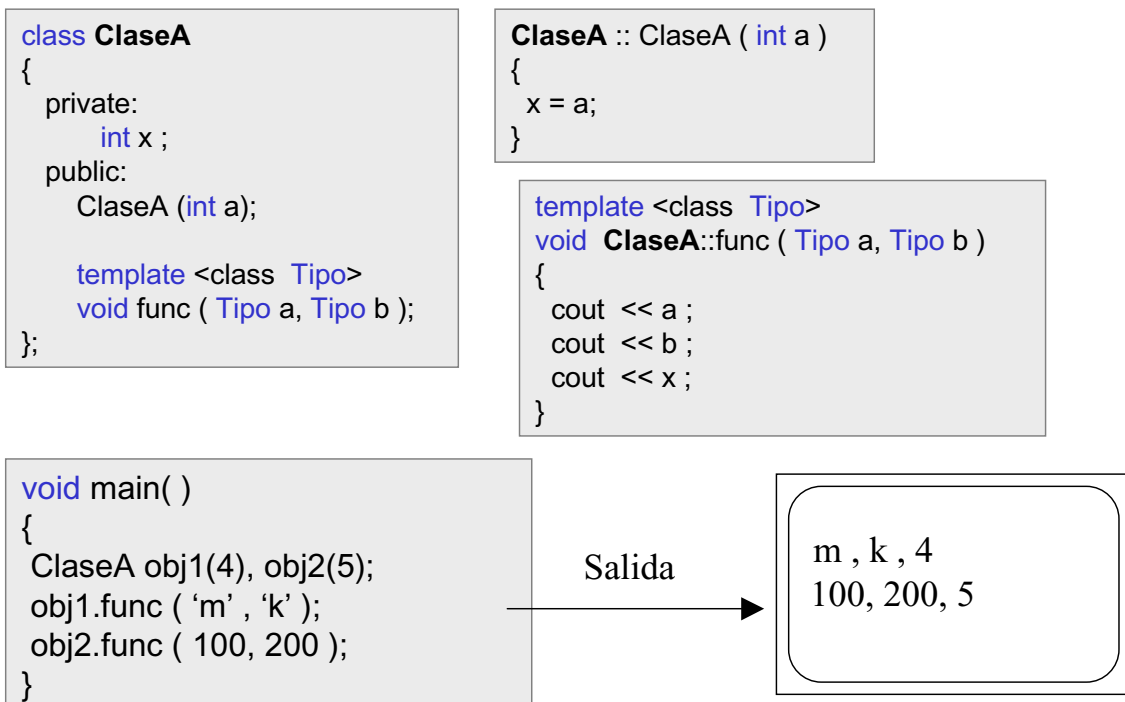
Las funciones genéricas pueden ser miembros (métodos) de clases.



3

Plantillas de clases (Templates)

Ejemplo



4

Plantillas de clases (Templates)

Plantillas de clases o Clases genéricas

Son un artificio C++ que permite definir una clase mediante uno o varios parámetros. Este mecanismo es capaz de generar infinitas clases distintas pero compartiendo un diseño común.

Al igual que en plantillas de funciones, las clases así generadas se denominan **instancias** o **especializaciones** de la plantilla.

```
class Punto
{
private:
    int coorx ;
    int coory ;
public:
    ...
};
```

Podemos diseñar la clase Punto con atributos de tipo **int**

En el caso de que necesitemos que el tipo de los puntos sea de otro tipo, tendremos que crear otra clase distinta e implementar todos los métodos.

Pérdida de tiempo

Posibilidad de error

5

Plantillas de clases (Templates)

Plantillas de clases o Clases genéricas

```
class Puntoentero
{
private:
    int coorx ;
    int coory ;
public:
    ...
};
```

El sistema de plantillas nos permite definir una clase genérica que permita crear **especializaciones** o **instancias** de dicha plantilla para cualquier tipo especificado por un parámetro.

```
class Puntofloat
{
private:
    float coorx ;
    float coory ;
public:
    ...
};
```

INSTANCIAS

```
class Puntolong
{
private:
    long int coorx ;
    long int coory ;
public:
    ...
};
```

```
template <class TIPO>
class Punto
{
private:
    TIPO coorx ;
    TIPO coory ;
public:
    ...
};
```

PLANTILLA

6

Plantillas de clases (Templates)

Sintaxis

Las plantillas de clases o clases genéricas tienen la siguiente sintaxis:

```
template <lista de tipos genéricos>  
class Nombre_clase  
{  
  ...  
};
```

La *lista de tipos* contiene los tipos genéricos a los que precede la palabra **class**

- ➔ La palabra reservada **template** indica que se va a declarar una plantilla.
- ➔ Las plantillas se declaran normalmente en un archivo de cabecera.
- ➔ La *lista de tipos* contiene los tipos genéricos separados por comas.

7

Plantillas de clases (Templates)

Creación de objetos

¿Cómo se crea un objeto de la clase Punto con atributos de tipo float?

```
template <class TIPO>  
class Punto  
{  
  private:  
    TIPO coorx ;  
    TIPO coory ;  
  public:  
    ...  
};
```

PLANTILLA

```
class Punto  
{  
  private:  
    float coorx ;  
    float coory ;  
  public:  
    ...  
};
```

INSTANCIA

```
...  
Punto <float> obj ;  
...
```

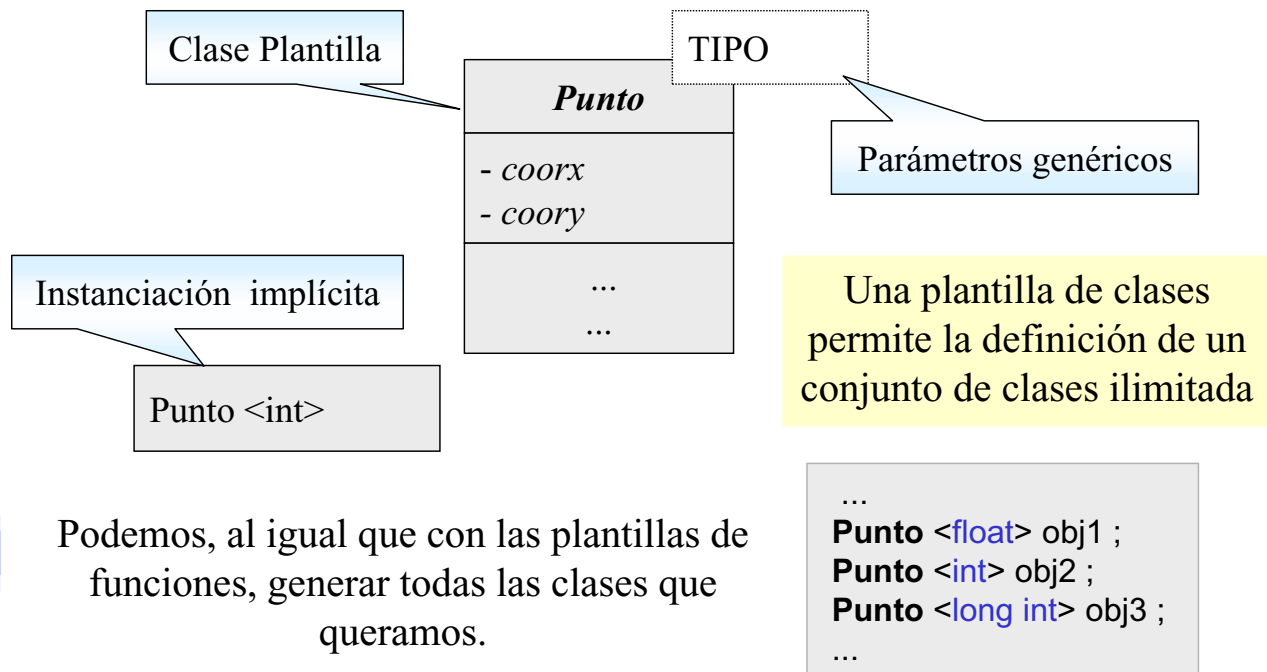
Se escribe el nombre de la plantilla de clase seguido por los tipos con los que se declara entre <>

Esta instrucción genera una clase normal a partir de la plantilla de clase
Instanciación implícita

8

Plantillas de clases (Templates)

Representación de una plantilla de clase en UML



9

Plantillas de clases (Templates)

Miembros de las clases genéricas

Los miembros de las clases genéricas se definen y declaran igual que los de las clases normales.

```
template <class TIPO>  
class Punto  
{  
    private:  
        TIPO coorx ;  
        TIPO coory ;  
    public:  
        Punto(TIPO a, TIPO b);  
        TIPO acc_x( );  
        TIPO acc_y( );  
        void mu_x( TIPO a);  
        void mu_x( TIPO a);  
        void visualizar( );  
};
```

Hay que señalar que la funciones miembro, son a su vez plantillas con los mismos parámetros que la clase genérica a la que pertenecen.

```
template <class TIPO>  
Punto<TIPO>::Punto(TIPO a, TIPO b)  
{  
    coorx = a;  
    coory = b;  
}
```

```
template <class TIPO>  
TIPO Punto<TIPO>::acc_x( )  
{  
    return coorx;  
}
```

10

Plantillas de clases (Templates)

Miembros de las clases genéricas

```
template <class TIPO>
class Punto
{
private:
    TIPO coorx ;
    TIPO coory ;
public:
    Punto(TIPO a, TIPO b);
    TIPO acc_x( );
    TIPO acc_y( );
    void mu_x( TIPO a);
    void mu_y( TIPO a);
    void visualizar( );
};
```

```
template <class TIPO>
void Punto<TIPO>::mu_x(TIPO a )
{
    coorx = a;
}
```

```
template <class TIPO>
void Punto<TIPO> :: visualizar( )
{
    cout << coorx ;
    cout << coory;
}
```

Plantillas de clases (Templates)

CONSEJOS

- ◆ Es aconsejable realizar el diseño y una primera depuración con una clase normal antes de convertirla en una clase genérica.

Es más fácil imaginarse el funcionamiento referido a un tipo concreto que a entes abstractos.

- ◆ Es más fácil entender los problemas que pueden presentarse si se maneja una clase concreta.
- ◆ En estos casos es más sencillo ir de lo particular a lo general.

Plantillas de clases (Templates)

Ejemplo

```
template <class T1, T2>
class ClaseB
{
private:
    T1 x ;
    T2 y;
public:
    ClaseB (T1 a, T2 b);
    void visualizar ( );
};
```

```
template <class T1, T2>
ClaseB< T1, T2> :: ClaseB (T1 a, T2 b)
{
    x = a;
    y = b;
}
```

```
template <class Tipo>
void ClaseB < T1, T2> :: visualizar ( );
{
    cout << "Valor de x : " ;
    cout << x << endl ;
    cout << "Valor de y : " ;
    cout << y ;
}
```

```
void main( )
{
    ClaseB <int, char> obj1(5, 'p') ;
    ClaseB <bool, float> obj2(true, 9.0) ;
    obj1.visualizar( );
    obj2.visualizar( );
}
```

Salida

```
Valor de x:5
Valor de y: p
```