Clases

Dr. Mario Marcelo Béron

Generalmente un programa utiliza muchos objetos durante su ejecución. Estos objetos intercambian mensajes para cumplir con la funcionalidad programada.
Un objeto es entidad encapsulada, protegida y accesible a través de su interface y cuyo comportamiento definido es accesible por medio de un mecanismo de envío de mensajes.
Las clases son un mecanismo que permite garantizar que todos sus instancias:

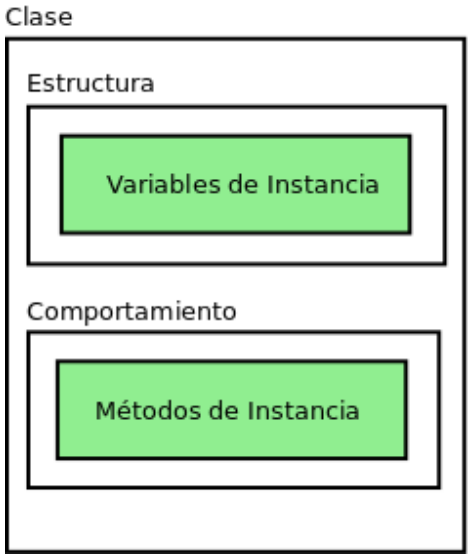
- Tengan la misma estructura y el mismo comportamiento.
- Accedan a sus datos por medio de operaciones destinadas para tal fin.

Clases

POO: Clases y Objetos

Instancias

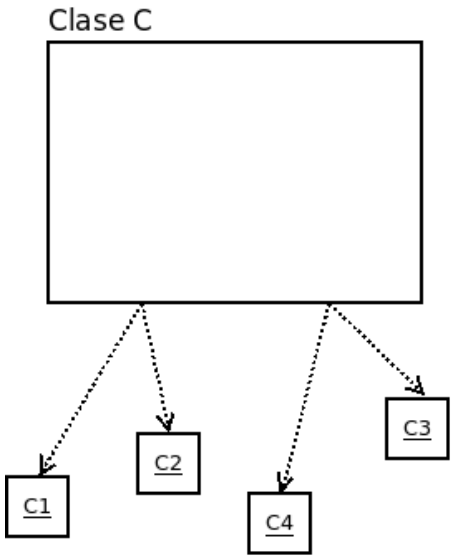
Clase



Concepto

- Una clase es un molde para la creación de objetos o sea sus instancias.
- Una clase es una abstracción de entidades del mundo real que cumplen con las siguientes características:
  - Todas las entidades del mundo real tienen las mismas características.
  - Todas las entidades siguen las mismas reglas.

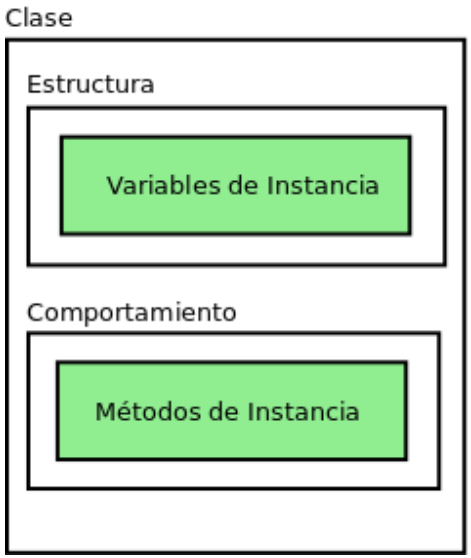
Clases



Importancia de las Clases

- Las clases permiten la creación de objetos con una misma estructura (variables de instancia) y con un mismo comportamiento (métodos de instancia).
- Las clases pueden ser vistas como mecanismos para la implementación de tipos de datos abstractos.
- Las clases representan un mecanismo muy importante para la reutilización de código.

Clases

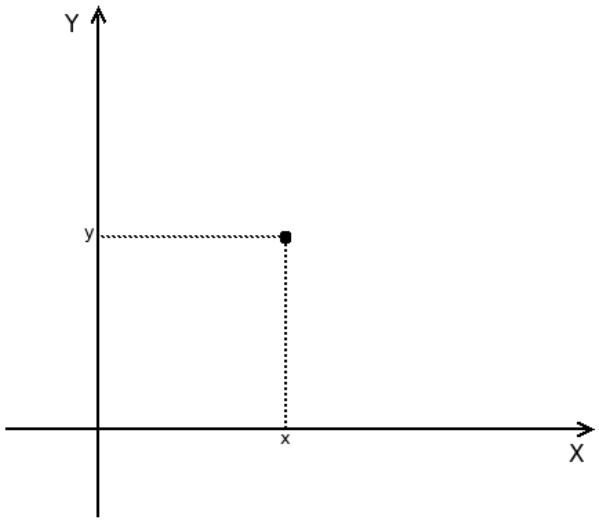


Creación de Clases

Para la creación de una clase se debe especificar:

- La estructura (Variables de Instancia)
- El comportamiento (Métodos de Instancia)

Clases



Requisitos

- Los puntos deben poseer coordenadas reales.
- Se podrán crear puntos con coordenadas (x,y) especificadas por el programador.
- Deberá ser posible conocer el valor de:
  - La coordenada X.
  - La coordenada Y.
- Se podrán realizar las siguientes operaciones:
  - Sumar dos puntos.
  - Restar dos puntos.

Ejemplo de Definición de una Clase

Esqueleto Básico para Definir un Punto

```
class Punto {
 //Definiciones
}
```

Definición de las Variables de Instancia

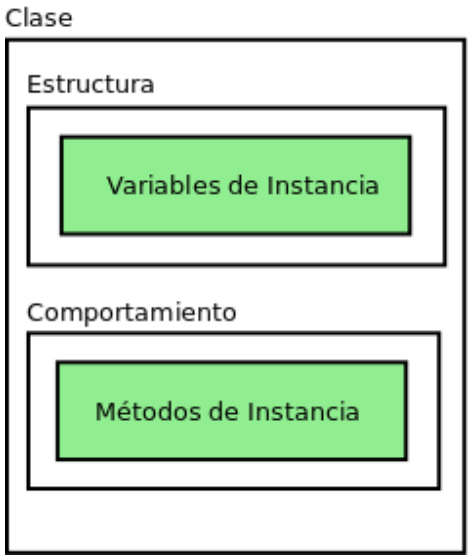
```
class Punto {
 //Variables de Instancia
 float x;
 float y;
}
```

Ejemplo de Definición de una Clase

Definición de los Métodos de Instancia

```
class Punto {
 //Variables de Instancia
 float x;
 float y;
 //Metodos de Instancia
 float getX() { return x; }
 float getY() { return y; }
 void decCoord(float deltaX, float deltaY) {
   x -= deltaX; y -= deltaY;
 }
 boolean primerCuadrante(){ return x > 0 && y > 0; }
   .....
}
```

Creación de Clases



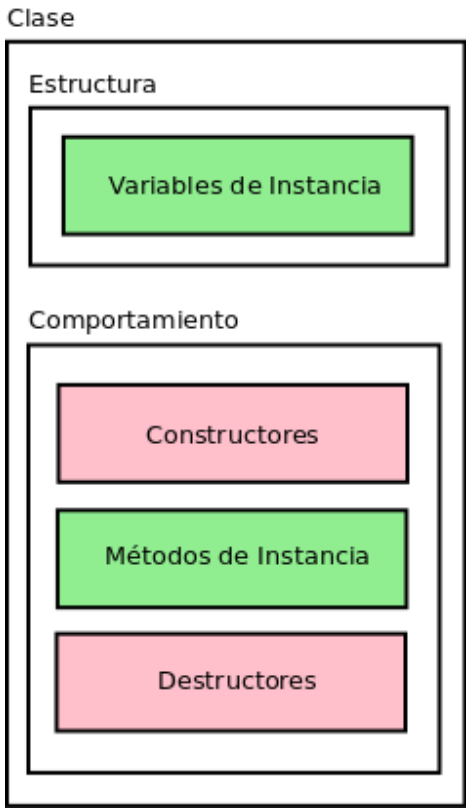
Definición de la Estructura

Esta tarea se lleva a cabo definiendo los nombres y los tipos de las variables de instancia que serán utilizadas.

Importante

La estructura de una clase mantiene el estado de los objetos.

Creación de Clases

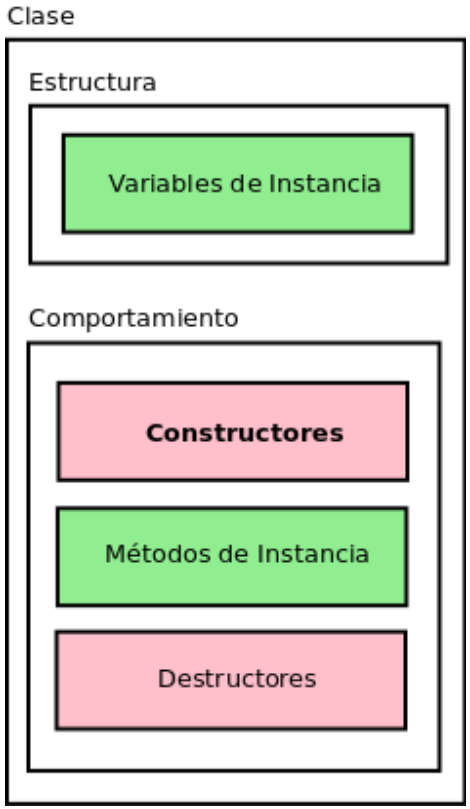


Definición del Comportamiento

Para definir el comportamiento de una clase se deben especificar:

- Los Constructores.
- Los Métodos de Instancia.
- Los Destructores.

Definición del Comportamiento



Constructores

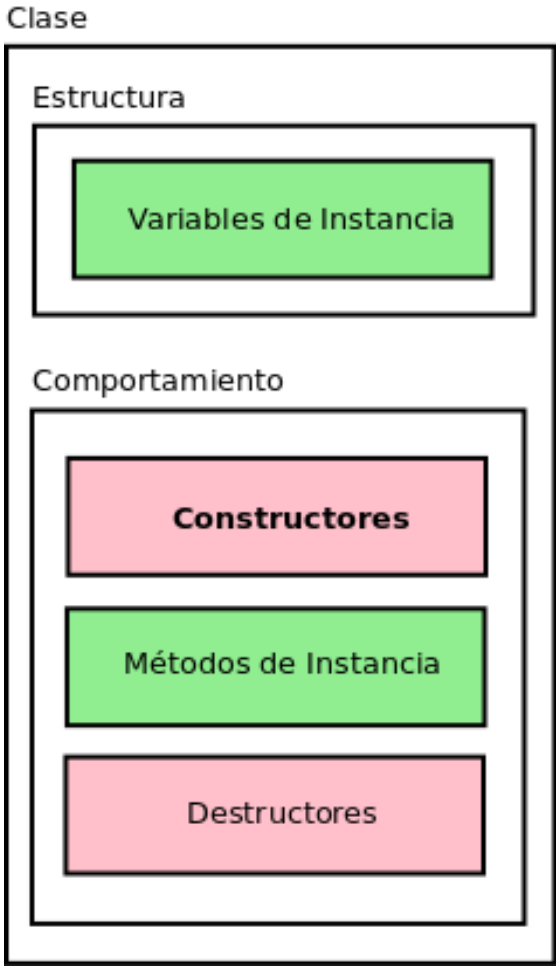
Son métodos especiales que son utilizados en la creación de instancias de una clase.

Constructores

```
Punto p1=new Punto();
Punto p1;
 p1=new Punto();
```

En ambos casos se declara p1 de tipo *Punto*. p1 es capaz de referenciar un objeto de tipo *Punto*. Luego la variable se asocia a la ejecución de una expresión *new Punto()*, la cual crea una instancia de *Punto* por la ejecución del método especial *Punto()*(El constructor).

Definición del Comportamiento



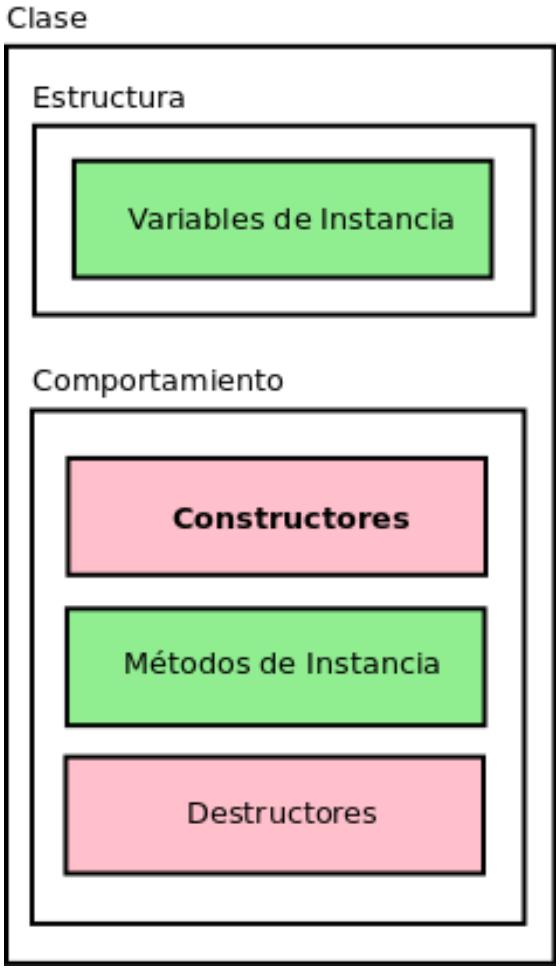
Constructores

Son métodos especiales que son utilizados en la creación de instancias de una clase.

Constructores

En Java, para cada clase **C** se le asocia un constructor **C()** capaz de crear instancias de la clase. Esto se lleva a cabo cuando dicho constructor se usa junto con la palabra reservada **new** en expresiones **new C()**.

Definición del Comportamiento



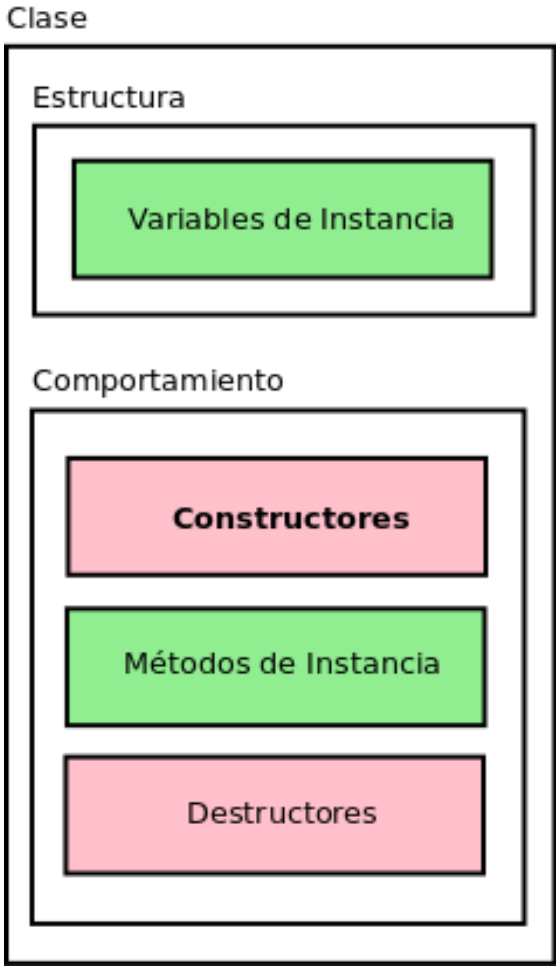
Constructores

Son métodos especiales que son utilizados en la creación de instancias de una clase.

Constructores

Java permite que el programador defina tantos constructores como crea necesario. Sólo se debe cumplir la siguiente regla: *"Todos los constructores poseen el mismo nombre de la clase y sólo difieren en el número y tipo de sus parámetros."*

Definición del Comportamiento



Constructores

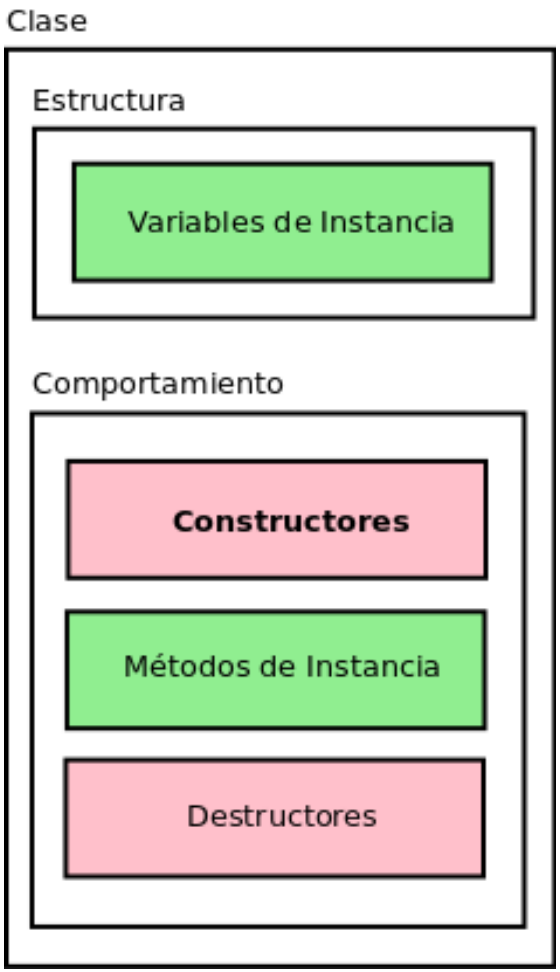
Son métodos especiales que son utilizados en la creación de instancias de una clase.

Constructores

```
Punto p1= New Punto();
```

- *p1* es una instancia de Punto.
- *p1* tiene sus variables de instancia *x* e *y* inicializadas en 0 por ser de tipo *float*.

Definición del Comportamiento



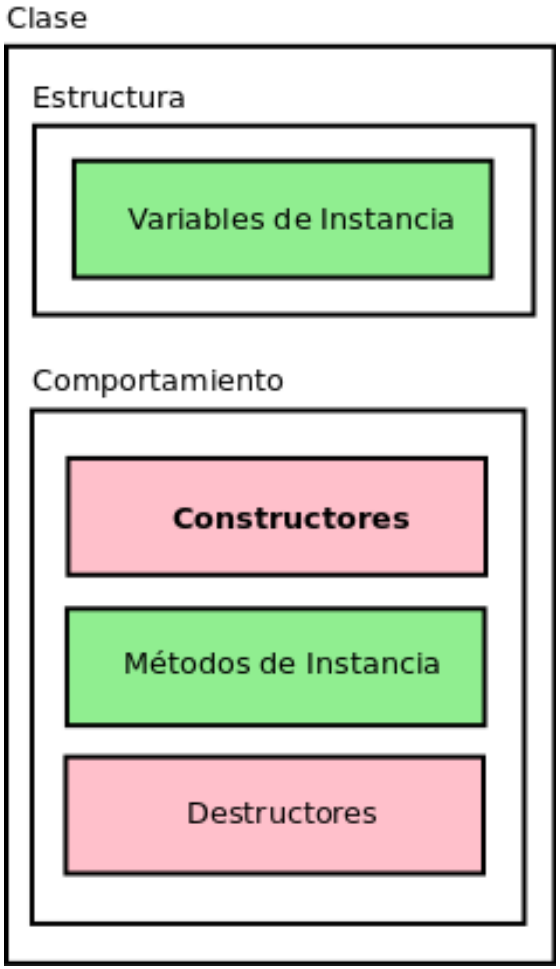
Constructores

- Llevan el mismo nombre de la clase.
- Una clase puede tener más de un constructor.
- En el caso de haber más de un constructor los mismos se diferencian por el número de argumentos.

Constructores-Función

La función principal de un constructor es la creación de instancias de un clase y por lo tanto no tiene sentido especificar el tipo de resultado.

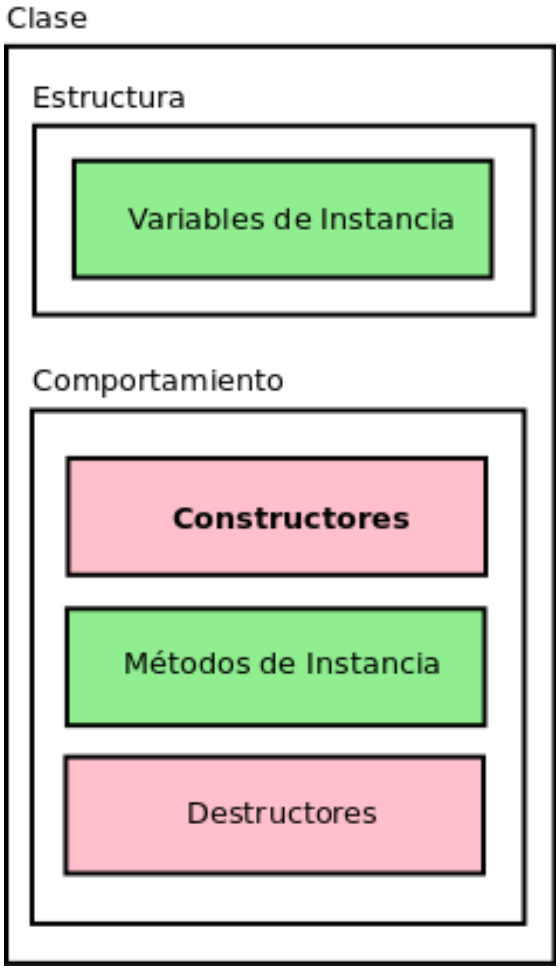
Definición del Comportamiento



Constructores

```
NombreDeLaClase(listaDeParametros){
  Sentencias
}
```

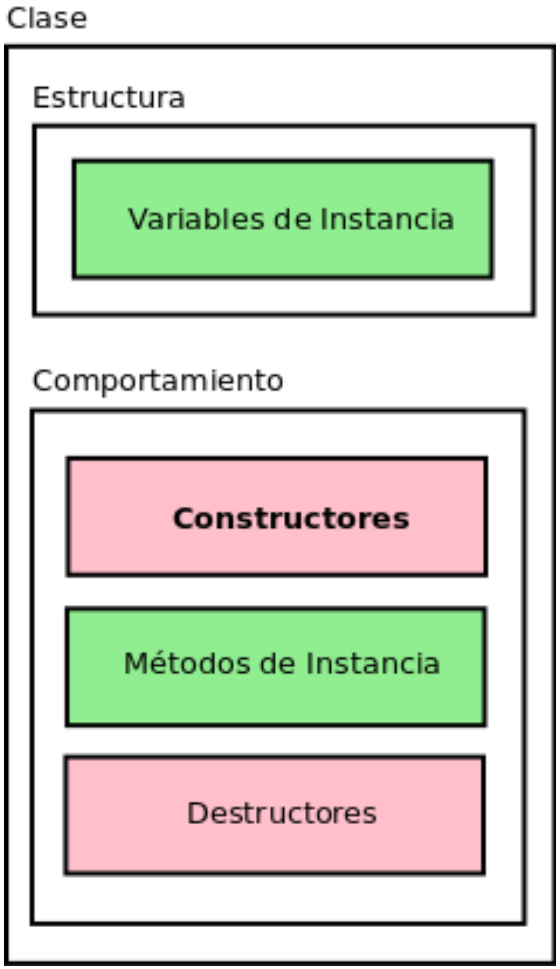
Definición del Comportamiento



Constructores

Es siempre posible y generalmente muy útil definir más de un constructor. En este caso, generalmente los constructores difieren en los valores asignados a las variables de instancia, o también en la forma de asignar esos valores.

Definición del Comportamiento

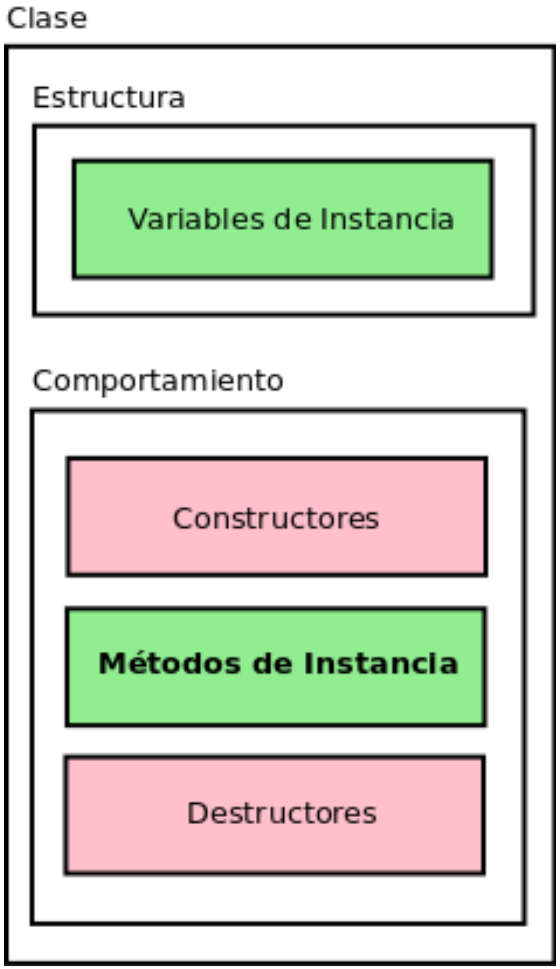


Constructores

```
class Punto {
 // Constructores
 Punto() { x = 0; y = 0;  }

 Punto(float cx, float cy){x=cx; y=cy;}
  //Variables de Instancia
  float x;
  float y;
      ...
}
```

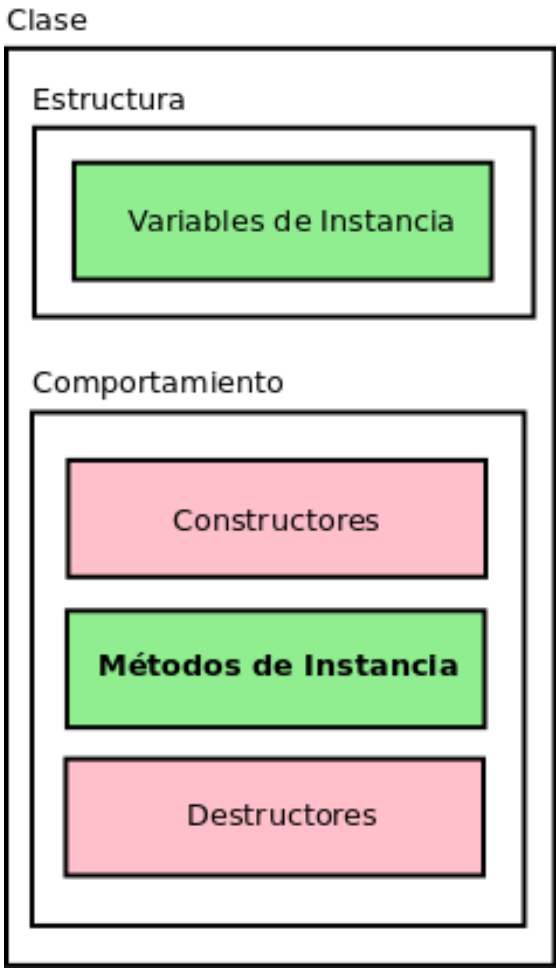
Definición del Comportamiento



Regla

La única forma de garantizar que se programan entidades independiente del contexto y por lo tanto reutilizables es garantizando que ninguna cápsula accede directamente a las variables de instancia de otra cápsula. Para modificar estas variables se invoca a través del envío de mensajes a métodos de acceso a tales variables. Dichos métodos deben ser programados en las cápsulas.

Definición del Comportamiento



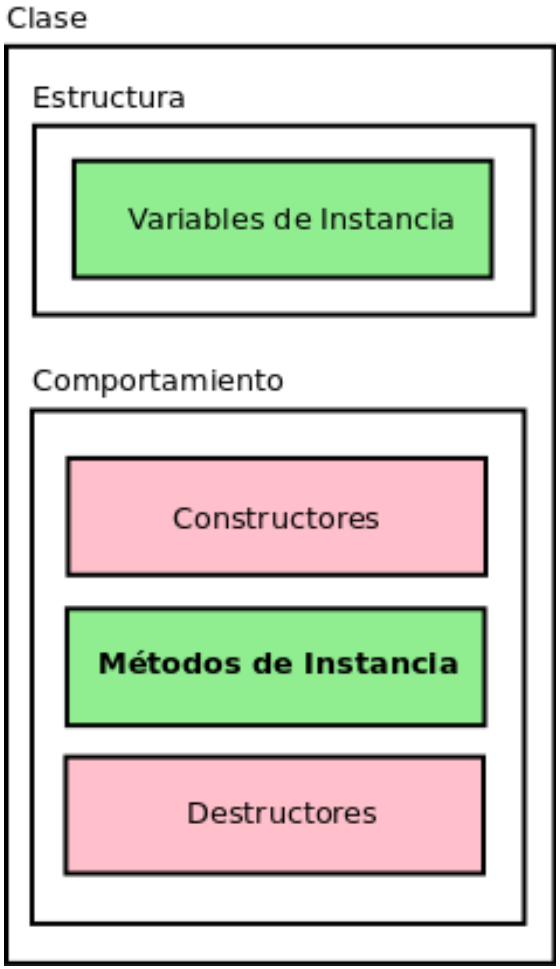
Responsabilidades Impuestas por la Regla Anterior

- En la definición de los métodos de instancia no se debe escribir código que acceda a las variables de instancia de otro objeto.
- Es necesario que las interfaces provean métodos adecuados para el acceso a las variables de instancia.

Importante

Las reglas descriptas previamente no son chequeadas por el compilador. Es responsabilidad del programador cumplirlas.

Definición del Comportamiento



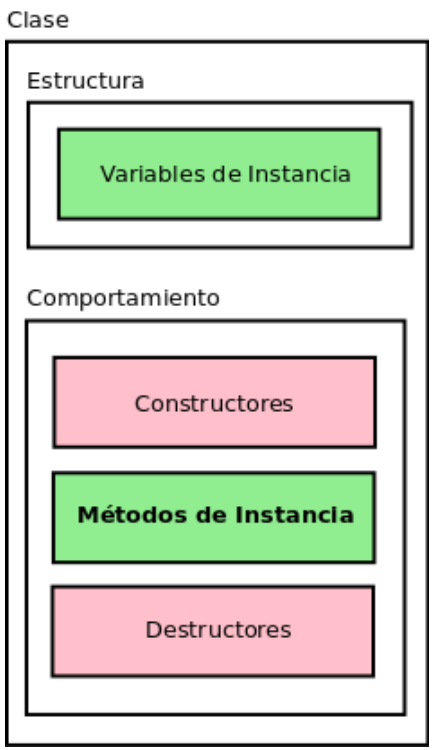
Sugerencia

- Es altamente recomendado utilizar métodos que permitan acceder al valor de las variables de instancia. Estos métodos son conocidos con el nombre de **observadores**.
- Es altamente recomentado utilizar métodos para modificar el valor de las variables de instancia. Estos métodos son conocidos con el nombre de **modificadores**.

Definición del Comportamiento

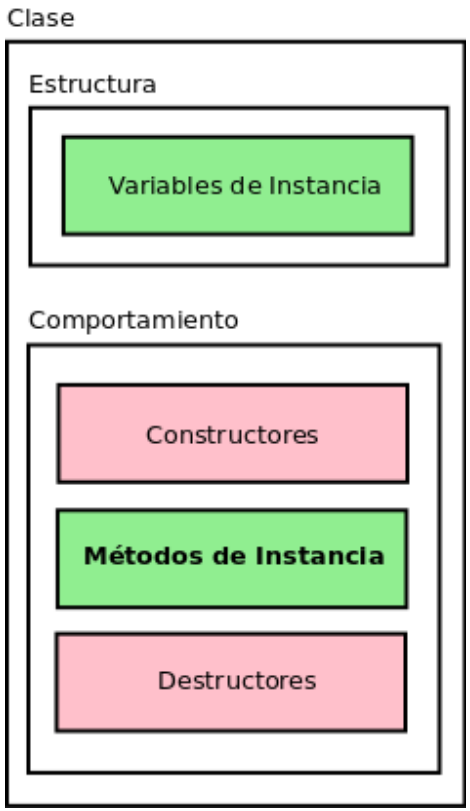
Métodos de Instancia - Sintáxis

<Mod. Acc> <Tipo de Result.> <Ident.> (<Tipo P1> .. <Tipo PN>) Cuerpo



- Modificador de Acceso: Se verá luego.
- Tipo de Resultado: Es un tipo simple o el nombre de una clase.
- Identificador: Es una secuencia de letras y dígitos debiendo la primera ser una letra.
- Lista de Parámetros Formales: Está constituida por una lista de cero o más pares Tipo Par que definen los tipos e identificadores de los parámetros formales del método.

Definición del Comportamiento



Métodos de Instancia-Cuerpo

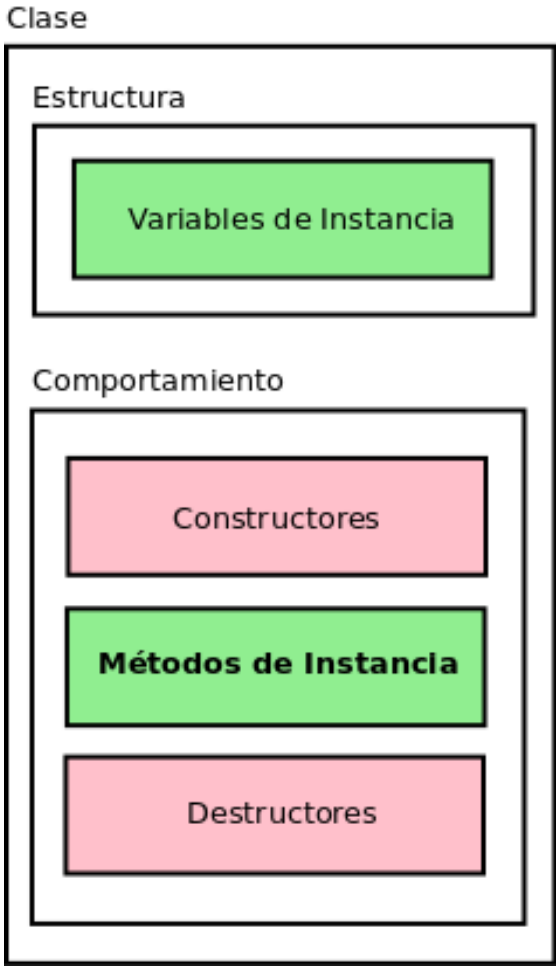
Consiste de un grupo de sentencias delimitados por llaves.

$$\{ \text{ sentencias } \}$$

Importante

Si el método devuelve un resultado éste será el valor resultante de la expresión que está asociada a la primera sentencia **return** que fue ejecutada.

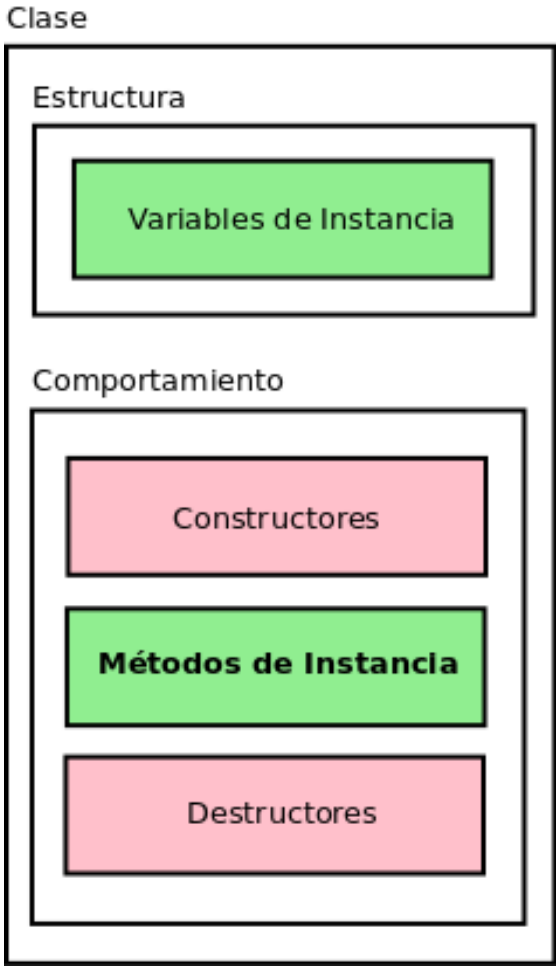
Definición del Comportamiento



Métodos de Instancia-Observadores/Modificadores

```
// Observadores
float getX() { return x;}
float getY() { return y;}
// Modificadores
void incCoord(float deltaX, float
    deltaY) {
 x = x + deltaX; y = y + deltaY ;
}
```

Definición del Comportamiento

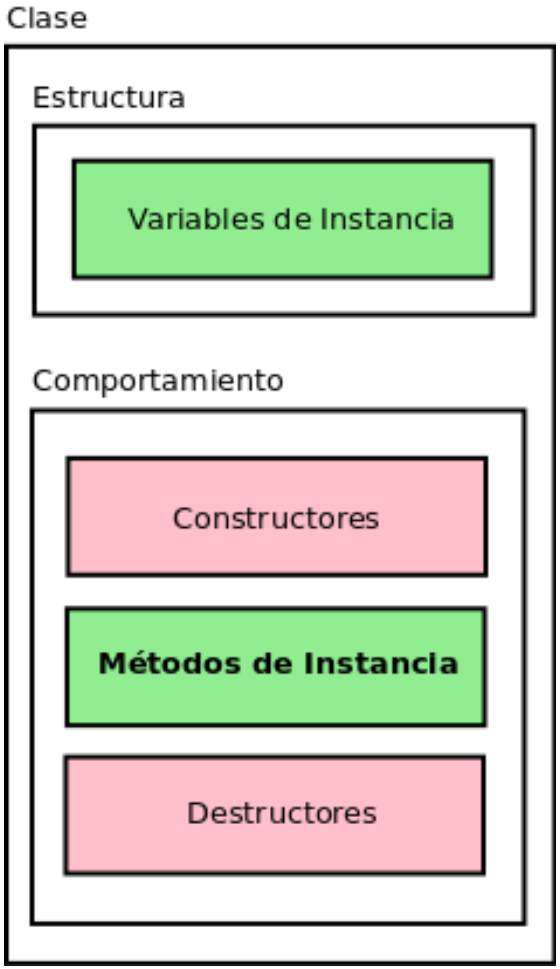


La Referencia this

Se utiliza cuando:

- Se desea referenciar a una variable de instancia de la clase que se está definiendo.
- Se desea invocar un método de la clase que se está definiendo.

Definición del Comportamiento

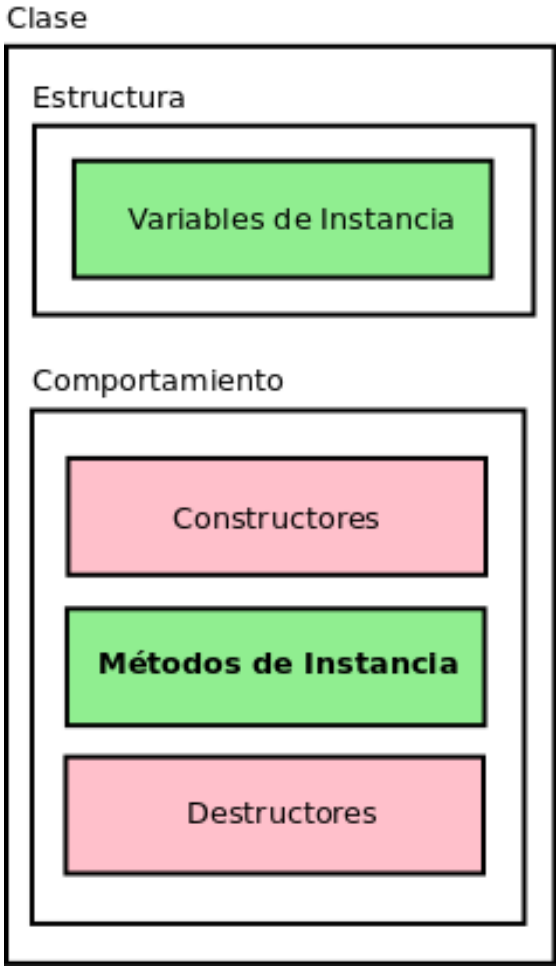


La Referencia this

```
Punto(float x, float y) {
  this.x = x;
  this.y = y;
}
```

En este ejemplo los nombres de los parámetros son los mismos que los nombres de las variables de instancia. La referencia *this* permite claramente identificar el acceso a las variables de instancia de las utilizaciones de los parámetros.

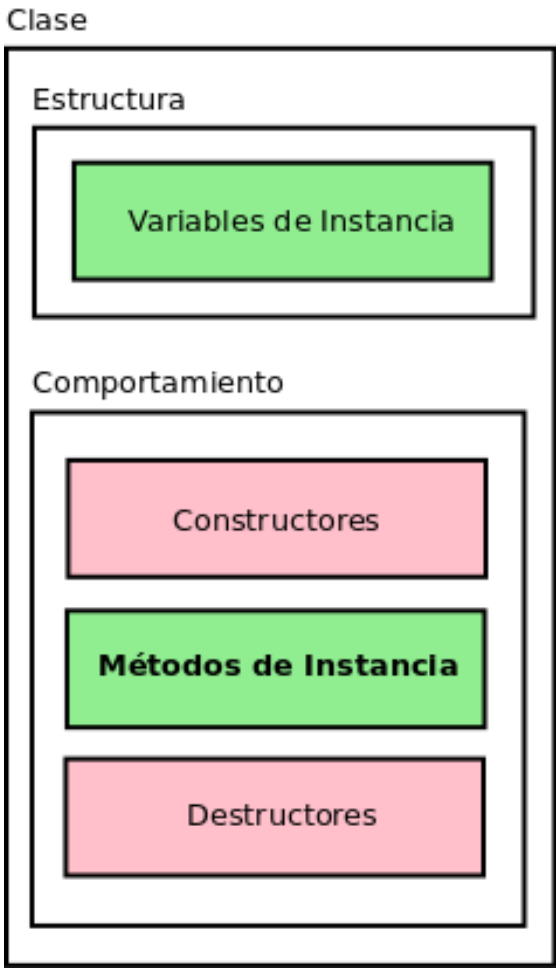
Definición del Comportamiento



La Referencia this

En el caso de los métodos, la referencia *this* se necesita cuando un método que se está definiendo en la clase necesita invocar a otro método definido en la misma clase.

Definición del Comportamiento

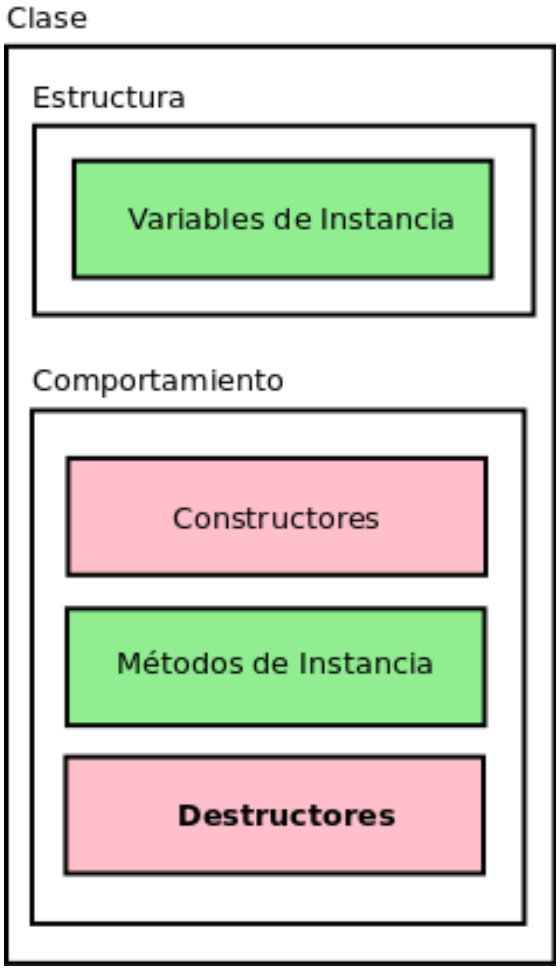


La Referencia this

```
// modificador - decrementa Coordenada
    X
 void decCoordX(float deltaX) {
  x -= deltaX;
 }

// modificador - decrementa Coordenada
    X
 void decCoordX(float deltaX) {
  //invoca decCoord() local
  this.decCoord(deltaX, 0);
 }
```

Definición del Comportamiento



Destructores

Un destructor se utiliza cuando se necesitan realizar acciones de limpieza cuando se destruye el objeto.

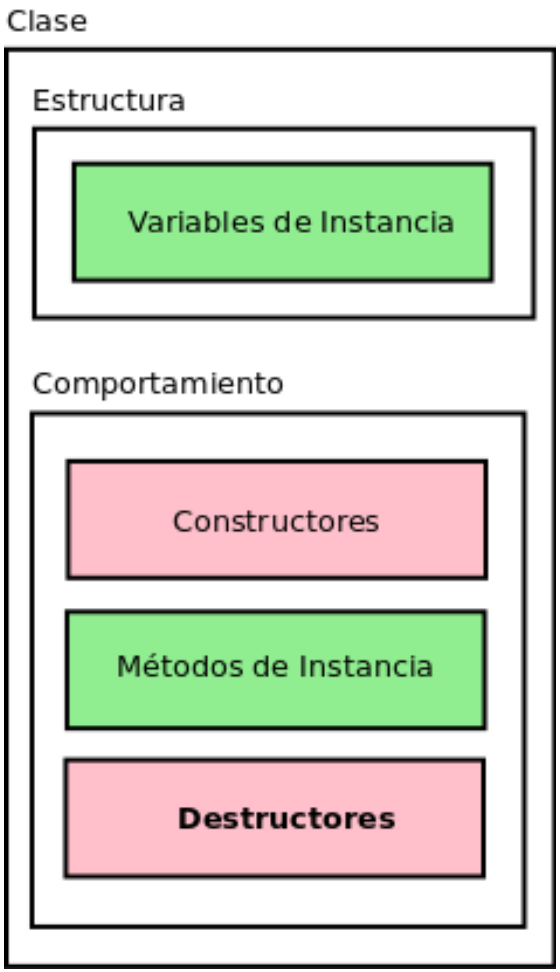
Importante

En Java no se definen destructores. Esta sección de la clase es sólo a nivel informativo.

Definición del Comportamiento

Destructores-Sintáxis

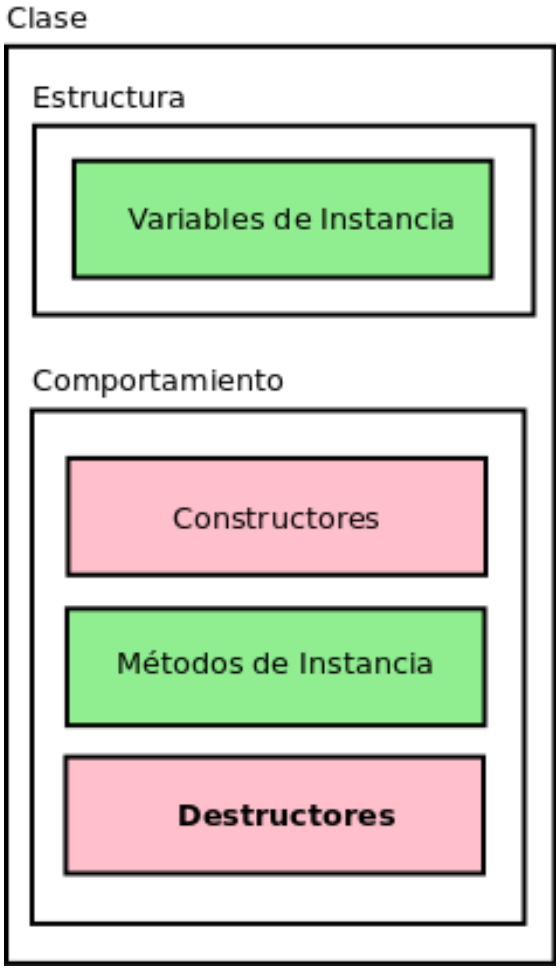
Símbolo Identificador (Lista de Parámetros Formales) Cuerpo



Destructores

Un destructor siempre lleva el nombre de la clase pero se distingue del constructor porque va precedido por algún símbolo especial. En el caso de C++ dicho símbolo es ˜.

Definición del Comportamiento



Destructores

Los destructores son útiles en clases las cuales usan punteros a bloques de memoria ya que cuando el objeto se destruye se debe liberar el bloque de memoria. Un destructor puede realizar esa tarea a la perfección.

Especificadores de Acceso

Reglas de Accesibilidad

Modificador	**Accesible a partir del código de**
public	Cualquier clase
protected	(*)
private	(*)
ninguno	Clases dentro de su paquete

Nota

Las clases con modificadores *private* o *protected* son clases especiales en java. Su estudio está fuera del alcance de este curso.

Especificadores de Acceso

Reglas de Accesibilidad

- La clase Punto al ser definida sin utilizar ningún modificador estará accesible sólo dentro del respectivo paquete.
- Java asocia a la clase Punto a un paquete particular sin identificador lo que es conveniente en la etapa de desarrollo y test de la clase.

Especificadores de Acceso

Reglas de Accesibilidad

Si se pretende que la clase Punto forme parte del paquete java.classes1, cuyo directorio deberá figurar en la variable CLASSPATH, entonces al inicio del archivo donde se define la clase punto se debe escribir:

```
package java.classes1;
class Punto {
```

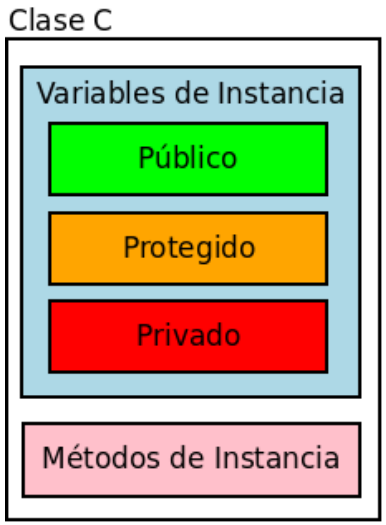
Especificadores de Acceso

Reglas de Accesibilidad

Si a la definición anterior de la clase le agregamos el modificador de acceso **public** entonces la clase puede ser accedida desde cualquier sitio del exterior.

```
package java.classes1;
public class Punto {
```

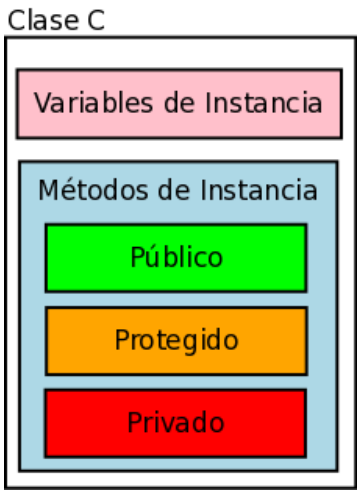
Especificadores de Acceso



Variables de Instancia

Modificador	**Accesible por:**
public	Cualquier clase.
protected	La clase, Clases del paquete, subclases.
private	La clase
ninguno	La clase, Clases del paquete

Especificadores de Acceso



Métodos de Instancia

Modificador	**Accesible por:**
public	Cualquier clase.
protected	La clase, Clases del paquete, subclases.
private	La clase
ninguno	La clase, Clases del paquete

Especificadores de Acceso

Lecciones Aprendidas

- Las variables de instancia almacenan el estado del objeto y por consiguiente son en general privadas.
- El programador debe proveer métodos adecuados para la consulta y modificación de las estructuras de datos internas de una clase.
- Los métodos de instancia son utilizados desde el exterior, por consiguiente son en general públicos.
- API (Application Programming Interface) de una clase en Java es el conjunto de métodos públicos y los constructores de la misma.
- Los métodos privados son accesibles al código de la clase y por lo tanto pueden ser vistos como métodos auxiliares.
- Las variables declaradas dentro de los métodos son variables locales al mismo. Dichas variables dejan de existir cuando termina la ejecución del método.

Compilación de la Clase Punto

Pasos para la Compilación

1. Se escribe el código con un editor de textos (wordpad, notepad, kate, emacs, vi, etc.)
2. Se graba el programa con extensión *.java*. En este caso *Punto.java*.
3. En una terminal ejecutar el siguiente comando: javac [-opciones] Punto

Comentario

La clase Punto no es ejecutable porque no posee un método *main*. Una forma adecuada de probar el funcionamiento de la clase consiste en crear una clase que:

- Posibilite crear instancias de la clase Punto.
- Permita enviar mensajes correspondientes a la API de Punto.

Prueba de la Clase Punto

Creación de la Clase TestPunto

```
public class TestPunto {
// Clase para probar la Clase Punto.
// Aquí se escriben las variables auxiliares y
// metodos auxiliares si los hubiere.

   public static void main(String args[]) {
  // Aquí se escribe el codigo de la clase
  // o sea se crean instancias de Punto y se
  // envian los mensajes correspondientes, etc.
    }
}
```

Prueba de la Clase Punto

Métodos Complementarios Usuales

Son métodos que aunque no sean mencionados en la lista de requisitos son necesarios en la definición de una clase:

- Igualdad.
- Crear una copia de un objeto de la clase.
- Crear una representación textual.

Ejercicio

Implementar los métodos mencionados previamente.

Sobrecarga de Métodos

Conceptualización

Consiste en la posibilidad de que en una misma clase se puedan definir dos o más métodos con el mismo nombre pero con signatura diferente. Por signatura diferente se entiende una lista diferente de parámetros sea cuanto a sus tipos o su orden.

Comentarios

- Los constructores están siempre sobrecargados porque tienen nombre igual al de la clase.
- Cuando un método está sobrecargado, la selección del método a ejecutar se llevará a cabo a partir de el número, tipo y orden de los parámetros.

Sobrecarga de Métodos

Ejemplo

```
// Incrementa-Decrementa las coordenadas
// en valores determinados
 public void incCoord(float deltaX, float deltaY)
 public void decCoord(float deltaX, float deltaY)
// Incrementa-Decrementa las coordenadas en 1
 public void incCoord()
 public void decCoord()
```

Métodos con un Número Variable de Parámetros

Ejemplo

```
  //Definición
  public impuesto(double tasa, int... vals) {
   double impTotal = 0.0;
   for(int val : vals) impTotal += val;
   return impTotal*tasa;
}
//Invocación
double calc1 = 0.0, calc2 = 0.0;
calc1 = impuesto(0.45, 12, 34, 44, 25);
calc2 = impuesto(0.21, 56, 34, 11, 23, 45, 2, 45, 67);
```

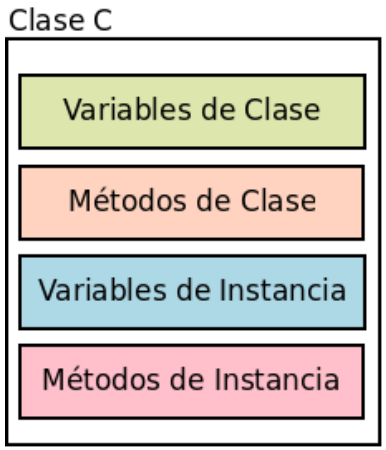
Métodos con un Número Variable de Parámetros

Ejemplo

Ejercicio: Defina el método mayorX que recibe como parámetros un número variable de puntos y retorna el punto con la coordenada X mayor.

Ejercicio: Muestre formas de utilización del método definido en el ejercicio anterior.

Clases



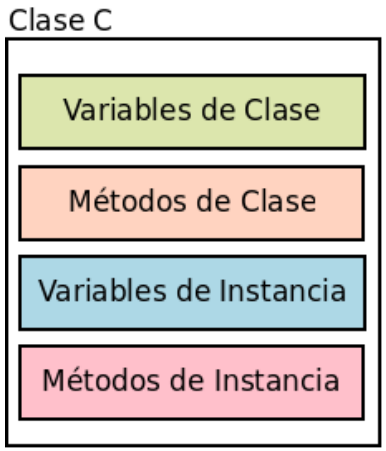
Clases

Las clases también son objetos y por consiguiente es natural que poseean:

- Variables de Clase.
- Métodos de Clase.

Generalmente este tipo de variables y métodos se utilizan con propósito de administración de los objetos de la clase.

Clases



Variables y Métodos de Clase

Tanto las variables como los métodos de clase se distinguen porque luego del modificador de acceso sigue la palabra clave **static**. Esta palabra indica que en el contexto de clases e instancias existe una única copia de estas entidades. Dicha copia reside en la clase.

Clases

Clases

- Clases Anidadas.
- Clases Abstractas
- Definir clases a partir de otras clases (Composición, Herencia).
- Etc.

Conclusión

- Una clase es un molde para la creación de objetos.
- Una clase está compuesta por:
  - Variables de Clase.
  - Métodos de Clase.
  - Variables de Instancia.
  - Métodos de Instancia.
- Cada uno los elementos mencionados en el ítem previo pueden tener diferentes tipos de acceso.
  - Público.
  - Protegido.
  - Privado.
  - No especificado.