



CLASES Y OBJETOS.

Febrero / 2020

PROGRAMACIÓN ORIENTADA A OBJETOS
INGENIERÍA INFORMÁTICA

Objetivos

Presentación de la asignatura.

Caracterizar los conceptos de Clase y Objeto.

Sumario

- ◇ Presentación de la asignatura.
- ◇ Conceptos de Clase y Objeto.
- ◇ Representación de clase con UML.

Bibliografía

- ◇ *Como programar en Java.*
- ◇ *Aprenda Java como si estuviera en primero.*

Presentación de la asignatura

- ◇ Tema I. Conceptos básicos del Modelo de Objetos.
- ◇ Tema II. Herencia.
- ◇ Tema III. Programación Visual.

Presentación de la asignatura

- ◇ Tema IV. Ficheros, genericidad, excepciones y sobrecarga de operadores
- ◇ Tema V. Proyecto de Curso.

Clase y objetos

Una **clase** es una agrupación de **datos** (variables o campos) y de **funciones** (métodos) que operan sobre esos datos. A estos datos y funciones pertenecientes a una clase se les denomina **variables** y **métodos** o **funciones miembro**.

Clase y objetos

La programación orientada a objetos se basa en la programación de clases. Un programa se construye a partir de un conjunto de clases.

Clase y objetos

Una vez definida e implementada una clase, es posible declarar elementos de esta clase de modo similar a como se declaran las variables del lenguaje (de los tipos primitivos int, double, String, ...). Los elementos declarados de una clase se denominan **objetos** de la clase.

Clase y objetos

De una única clase se pueden declarar o crear numerosos **objetos**.

Clase y objetos

La **clase** es lo genérico: es el patrón o modelo para crear **objetos**. Cada objeto tiene sus propias copias de las variables miembro, con sus propios valores, en general distintos de los demás objetos de la clase. Las clases pueden tener variables *static*, que son propias de la clase y no de cada objeto.

Características de las clases

- ◇ Todas las variables y funciones de Java deben pertenecer a una clase. No hay variables y funciones globales.
- ◇ Java tiene una jerarquía de clases estándar de la que pueden derivar las clases que crean los usuarios.

Características de las clases

- ◇ Si una clase deriva de otra (**extends**), hereda todas sus variables y métodos.
- ◇ Si una clase contenida en un fichero no es **public**, no es necesario que el fichero se llame como la clase.

Características de las clases

- ◇ En un fichero se pueden definir varias clases, pero en un fichero no puede haber más que una clase public. Este fichero se debe llamar como la clase public que contiene con extensión *.java. Con algunas excepciones, lo habitual es escribir una sola clase por fichero.

Características de las clases

- ◇ Una clase sólo puede heredar de una única clase (en Java no hay herencia múltiple). Si al definir una clase no se especifica de qué clase deriva, por defecto la clase deriva de Object. La clase Object es la base de toda la jerarquía de clases de Java.

Características de las clases

- ◇ Los métodos de una clase pueden referirse de modo global al objeto de esa clase al que se aplican por medio de la referencia **this**.

Características de las clases

- ◇ Las clases se pueden agrupar en **packages**, introduciendo una línea al comienzo del fichero (`package packageName;`). Esta agrupación en packages está relacionada con la jerarquía de directorios y ficheros en la que se guardan las clases.

Composición de una clase

Una clase puede ser constituida por:

- ◇ Variables miembros (atributos)
- ◇ Variables miembros de clase (**static**)
- ◇ Variables finales.
- ◇ Métodos.
- ◇ Métodos de clase (static).
- ◇ Constructores.

Composición de una clase

Variables miembros (atributos)

Cada objeto, es decir cada ejemplar concreto de la clase, tiene su propia copia de las variables miembro. Las variables miembro de una clase (también llamadas campos) pueden ser de tipos primitivos (**boolean**, **int**, **long**, **double**, ...) o referencias a objetos de otra clase (composición).

Composición de una clase

Variables miembros (atributos)

Un aspecto muy importante del correcto funcionamiento de los programas es que no haya datos sin inicializar. Por eso las variables miembro de tipos primitivos se inicializan siempre de modo automático, incluso antes de llamar al **constructor**

Composición de una clase

Variables miembros (atributos)

Las variables miembro pueden ir precedidas en su declaración por uno de los modificadores de acceso: **public**, **private**, **protected** y **package** (que es el valor por defecto y puede omitirse).

Composición de una clase

Variables miembros de clase (**static**)

Una clase puede tener variables propias de la clase y no de cada objeto. A estas variables se les llama variables de clase o variables **static**. Las variables **static** se suelen utilizar para definir constantes comunes para todos los objetos de la clase

Composición de una clase

Variables miembros de clase (**static**)

Las variables de clase son lo más parecido que Java tiene a las variables globales de C/C++.

Composición de una clase

Variables miembros de clase (**static**)

Las variables de clase se crean anteponiendo la palabra **static** a su declaración. Para llamarlas se suele utilizar el nombre de la clase (no es imprescindible, pues se puede utilizar también el nombre de cualquier objeto).

Composición de una clase

Variables finales

Una variable de un tipo primitivo declarada como final no puede cambiar su valor a lo largo de la ejecución del programa. Puede ser considerada como una constante, y equivale a la palabra **const** de C/C++.

Composición de una clase

Métodos

Los métodos son funciones definidas dentro de una clase. Salvo los métodos **static** o de clase, se aplican siempre a un objeto de la clase por medio del operador punto (.). Dicho objeto es su argumento implícito.

Composición de una clase

Métodos de clase (static)

Análogamente, puede también haber métodos que no actúen sobre objetos concretos a través del operador punto. A estos métodos se les llama métodos de clase o **static**.

Composición de una clase

Métodos de clase (static)

Análogamente, puede también haber métodos que no actúen sobre objetos concretos a través del operador punto. A estos métodos se les llama métodos de clase o **static**.

Composición de una clase

Métodos de clase (static)

Los métodos y variables de clase se crean anteponiendo la palabra **static**. Para llamarlos se suele utilizar el nombre de la clase, en vez del nombre de un objeto de la clase (por ejemplo, *Math.sin(ang)*, para calcular el seno de un ángulo).

Composición de una clase

Constructores

Un punto clave de la Programación Orientada Objetos es el evitar información incorrecta por no haber sido correctamente inicializadas las variables. Java no permite que haya variables miembro que no estén inicializadas.

Composición de una clase

Constructores

Java inicializa siempre con valores por defecto las variables miembro de clases y objetos. El segundo paso en la inicialización correcta de objetos es el uso de constructores.

Composición de una clase

Constructores

Un **constructor** es un método que se llama automáticamente cada vez que se crea un objeto de una clase. La principal misión del constructor es reservar memoria e inicializar las variables miembro de la clase.

Composición de una clase

Constructores

Los **constructores** no tienen valor de retorno (ni siquiera **void**) y su nombre es el mismo que el de la clase. Su argumento implícito es el objeto que se está creando.

Composición de una clase

Constructores

Una clase tiene varios constructores, que se diferencian por el tipo y número de sus argumentos (son un ejemplo típico de métodos sobrecargados). Se llama **constructor por defecto** al constructor que no tiene argumentos.

Composición de una clase

Constructores

Al igual que los demás métodos de una clase, los constructores pueden tener también los modificadores de acceso **public**, **private**, **protected** y **package**. Si un constructor es **private**, ninguna otra clase puede crear un objeto de esa clase.

Composición de una clase

Constructores

El **constructor** es tan importante que, si el programador no prepara ningún constructor para una clase, el compilador crea un constructor por defecto, inicializando las variables de los tipos primitivos a su valor por defecto, y los Strings y las demás referencias a objetos a null.

Proceso Unificado de Desarrollo

Rational Unified Process- RUP

RUP es una propuesta de proceso o metodología para el desarrollo de software orientado a objeto que utiliza UML (Unified Modeling Language) para describir un sistema.

Modelos Gráficos del UML

Diagramas de estructura estática:

- Diagrama de clases.

- Diagrama de objetos.

Diagrama de clase

Nombre de la clase
Variables (Atributos)
Constructores
Métodos

Diagrama de clase

Nombre de la clase

- atributo1 : <TD>

+ atributo2 : <TD> [,]

atributo3 : <TD> []

+ Constructor()

+ Constructor(parametro : <TD>)

+ Met(parametro : <TD>) : <TD>

+ Met2(p1 : <TD> , p2 : <TD>) : <TD>

+ Met3() : <TD>

Diagrama de clase

Donde:

- Implica que el atributo, método o constructor es **private**.

- + Implica que el atributo, método o constructor es **public**.

- # Implica que el atributo, método o constructor es **protected**.

Diagrama de clase

Donde:

<TD> Se sustituye por el tipo de dato.

[] Arreglo.

[,] Matriz.

Diagrama de clase

Para la elaboración de los diagramas de clases puede utilizar el software *Visual Paradigm*.

Caso de estudio 1

Una persona se puede definir por su nombre, apellidos y año de nacimiento. De igual forma una persona tiene una edad en la cual depende de forma directa el año actual.

Modele lo descripto anteriormente en una clase. Utilice *UML*

Caso de estudio 1

Persona

-m_nombre : string
-m_apellidos : string
-m_yearNacimiento : int

+Persona()
+Persona(_nombre : string, _apellidos : string, _yearNacimiento : int)
+getEdad(_currentYear : int) : int
+getNombre() : string
+setNombre(m_nombre : string) : void
+getApellidos() : string
+setApellidos(m_apellidos : string) : void
+getYearNacimiento() : int
+setYearNacimiento(m_yearNacimiento : int) : void

Caso de estudio 2

Una circunferencia se define por su radio y un punto 2D el cual se define por sus coordenadas en el eje X y Y. De la circunferencia se puede determinar su área y perímetro.

Modele lo descripto anteriormente en una clase. Utilice *UML*

Caso de estudio 2

Punto2D

-m_coordX : double

-m_coordY : double

+Punto2D()

+Punto2D(_x : double, _y : double)

+getCoordX() : double

+setCoordX(_coordX : double) : void

+getCoordY() : double

+setCoordY(_coordY : double) : void

Caso de estudio 2

Circunferencia

-m_centro : Punto2D

-m_radio : double

+Circunferencia()

+Circunferencia(_radio : double, _x : double, _y : double)

+Circunferencia(_radio : double, _centro : Punto2D)

+getCentro() : Punto2D

+setCentro(_centro : Punto2D) : void

+getRadio() : double

+setRadio(_radio : double) : void

Estudio independiente

Diseñe el diagrama de clase en UML una clase Temperatura con responsabilidades para almacenar un valor de temperatura en grados Celsius(C) y obtener su correspondiente en grados Fahrenheit(F) ($F = C * 9/5 + 32$).

UNIVERSIDAD DE MATANZAS

cosechando el saber

FIN