5. Desarrollo de clases.

Autor	Xerach Casanova
Clase	Programación
Fecha	@Jan 10, 2021 3:54 PM

1. Concepto de clase
    1.1. Repaso del concepto de objeto
    1.2. El concepto de clase
    2. Estructura y miembros de una clase
    2.1. Declaración de una clase.
    2.2. Cabecera de una clase
    2.3. Cuerpo de una clase
    2.4. Métodos estáticos o de clase
3. Atributos
    3.1. Declaración de atributos
    3.2. Modificadores de acceso
    3.3. Modificadores de contenido
    3.4. Atributos estáticos
4. Métodos
    4.1. Declaración de un método
    4.2. Cabecera del método
    4.3. Modificadores en la declaración de un método
    4.4. Parámetros en un método
    4.5. Cuerpo de un método
    4.6. Sobrecarga de métodos
    4.7. Sobrecarga de operadores
    4.8. La referencia this
    4.9. Métodos estáticos
5. Encapsulación, control de acceso y visibilidad
    5.1. Ocultación de atributos. Métodos de acceso
    5.2. Ocultación de métodos
    6. Utilización de los métodos y atributos de una clase
    6.1. Declaración de un objeto
    6.2. Creación de un objeto
    6.3. Manipulación de un objeto: utilización de métodos y atributos
7. Constructores
7.1. Concepto de constructor
    7.2. Creación de constructores
    7.3. Utilización de constructores
    7.4. Constructores de copia
    7.5. Destrucción de objetos
8. Empaquetado de clases
    8.1. Jerarquía de paquetes
    8.2. Utilización de paquetes
    8.3. Inclusión de una clase en un paquete
    8.4. Proceso de creación de un paquete
Mapa conceptual

1. Concepto de clase

Las clases están compuestas por atributos y métodos. Una clase especifica las características comunes de un conjunto de objetos.

Los programas a su vez están formados por un conjunto de clases, a partir de las cuales se irán creando objetos que interactúan unos con otros.

1.1. Repaso del concepto de objeto

Las **características fundamentales** de un objeto son: **la identidad** (os objetos son únicos y distinguibles entre sí, aunque pueda haber objetos exactamente iguales), **el estado** (los atributos que describen al objeto y los valores que tienen en cada momento) y **el comportamiento** (acciones que se pueden realizar sobre el objeto).

Los objetos tienen determinadas posibilidades de comportamientos (acciones) dependiendo de la familia a la que pertenezcan (un coche puede frenar, arrancar, cambiar de marcha, etc..., un círculo puede plantear acciones como calcular longitud de circunferencia, superficie, etc...).

También tienen sus atributos, cuyos valores cambian en función de la acciones que se realizan (en el caso del coche, ubicación o coordenadas, velocidad instantánea, kms. recorridos, etc..., en el caso de un círculo, tamaño del radio, color, etc...).

Todo objeto, por tanto, puede ser cualquier cosa que se pueda describir en término de atributos y acciones.

Un objeto no es más que la representación de cualquier entidad concreta o abstracta que puedas percibir o imaginar y que pueda resultar de utilidad para modelar los elementos el entorno del problema que deseas resolver.

1.2. El concepto de clase

La idea de una clase la podemos catalogar como una plantilla o modelo para cada conjunto de objetos que sean del mismo tipo, es decir, que tengan los mismos atributos y un comportamiento similar.

Una clase consiste en la definición de un tipo de objeto. Se trata de una descripción detallada de cómo van a ser los objetos que pertenezcan a esa clase, indicando qué tipo de información contendrán (atributos) y como se podrá interactuar con ellos (comportamiento).

Para ello se especifican:

- Atributos: comunes a todos los métodos que pertenezcan a esa clase.
- Métodos: que permiten interactuar con los objetos.

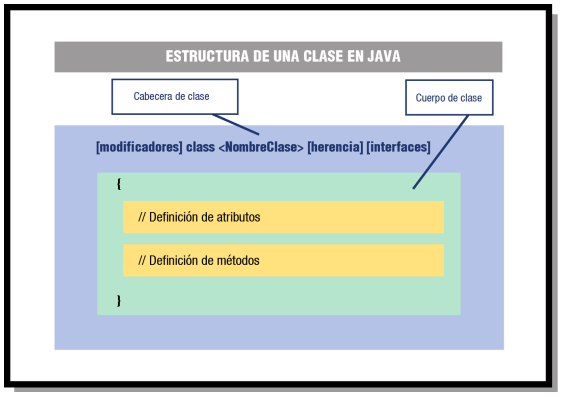
Es común confundir los términos de clase y objeto, aunque el contexto nos permite distinguir si nos referimos a una clase (definición abstracta) o a un objeto (instancia concreta de esa clase).

2. Estructura y miembros de una clase

La estructura de una clase consiste en:

Cabecera: Compuesta por una serie de modificadores de acceso, la palabra reservada class y el nombre de la clase.

Cuerpo: o contenido. se especifican o declaran los distintos miembros de la clase: atributos que caracterizan a los objetos y métodos que determinan el comportamiento de los mismos.



2.1. Declaración de una clase.

Estructura general:

```
[modificadores] class  <NombreClase> [herencia] [interfaces] { // Cabecera de la clase

     // Cuerpo de la clase

     Declaración de los atributos

     Declaración de los métodos

}
```

```java
/**
 *
 * Ejemplo de clase Punto 2D
 */
class Punto {
    // Atributos
    int x,y;

    // Métodos
    int obtenerX () { return x; }
    int obtenerY()  {return y;}
    void establecerX (int vx) { x= vx; };
    void establecerY (int vy) { y= vy; };
}
```

Todos los distintos programas desarrollados en unidades anteriores son una clase java: se declaraban con la palabra reservada class y contenían atributos (variables) y métodos (como mínimo el método main).

Además de indicar en la cabecera de una clase el nombre de la clase y la palabra reservada class, también podemos proporcionar más información mediante modificadores y otros indicadores como el nombre de su **superclase** (si esta clase hereda de otra), o si implementa algún **interfaz.**

Cuando se implementa una clase se debe tener en cuenta:

- Por convenio los nombres de clase empiezan por letra mayúscula, si el nombre de la clase está formado por varias palabras, también tendrán su primera letra mayúscula: Coche, JugadorFutbol...
- El archivo en el que se encuentra una clase java debe tener el mismo nombre que esa clase si queremos utilizarla desde otras clases (clase principal del archivo).
- La definición e implementación de una clase se incluye en el mismo archivo .java. En otros lenguajes como C++ podrían ir separados en archivos con extensiones .h y .cpp.

2.2. Cabecera de una clase

La declaración de una clase puede incluir los siguientes elementos en el siguiente orden:

1. Modificadores (public, abstract, final...).
2. Nombre de la clase.
3. Nombre de su clase padre (superclase), se especifica con la palabra reservada extends (extiende o hereda de).
4. Lista separada por comas de interfaces que son implementadas por la clase con la palabra implements (implementa).
5. Cuerpo de la clase encerrado entre llaves {}

Los modificadores pueden ser:

public: Indica que la clase es visible (se pueden crear objetos de esa clase) desde cualquier otra clase (desde otra parte del programa). Si no se especifica ese modificador, la clase solo podrá ser utilizada desde clases del mismo paquete. Solo puede haber una clase public en un archivo .java. El resto de clases que se definan no pueden ser públicas.

abstract: Indica que es una clase abstracta no instanciable. No es posible crear objetos de esa clase, habrá que utilizar clases que hereden de ella.

final: Indica que no podrás crear clases que hereden de ellas. final y abstract son excluyentes (o se utiliza uno o se utiliza el otro).

2.3. Cuerpo de una clase

Una clase puede no contener en su declaración atributos o métodos, pero debe contener al menos uno de los dos (la clase no puede estar vacía).

En el ejemplo de la clase Punto teníamos los atributos x e y de tipo int. Cualquier objeto de la clase punto contendrá dichos valores, que pueden coincidir o no en el contenido. Por ejemplo, declarando 3 objetos de tipo punto:

```
Punto p1,p2, p3;
```

Cada uno de ellos contendrá un par de coordenadas, puede que esos valores coincidan con los de otros objetos de tipo punto o no, pero todos han sido creado a partir de la misma clase.

También se definían métodos:

```
int obtenerX () { return x; } //devuelve el contenido de x

int obtenerY() { return y;} //devuelve el contenido de y

void establecerX (int vx) { x= vx; } //cambia el contenido de x a partir del parámetro que se le pasa

void establecerY (int vy) { y= vy; } //cambia el contenido de y a partir del parámetro que se le pasa
```

Estos métodos se pueden llamar desde cualquier objeto que sea instancia de Punto. Se trata de operaciones que permiten manipular los atributos contenidos en el objeto.

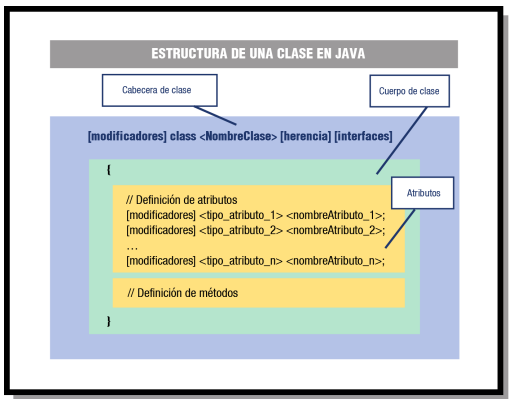
2.4. Métodos estáticos o de clase

Cada vez que instanciamos una clase se desencadenan una serie de procesos (se construye el objeto) que da lugar a la creación en memoria de un espacio físico que construye el objeto creado.

Por otro lado hay algunos miembros de la clase que no tienen sentido como partes del objeto, sino más bien como partes de la clase en sí. Estos miembros tienen sentido y existencia al margen de la existencia de cualquier objeto de dicha clase. Se trata de miembros estáticos o de clase y se definen con el modificador static, tanto en atributos como en métodos.

3. Atributos

Constituyen la estructura interna de los objetos de una clase. Es el conjunto de datos que los objetos de una determinada clase almacenan cuando son creados. Son variables cuyo ámbito de existencia es el objeto donde han sido creadas. Fuera del objeto no tienen sentido y si el objeto deja de existir, las variables también lo harán. Estos atributos son conocidos con el nombre de variables miembro o variables de objeto.



La declaración de un atributo puede contener algunos modificadores como: public, private, protected o static.

Lo más normal es declarar todos o la mayoría de los atributos como privados para respetar el concepto de encapsulación. De manera que para manipular algún atributo solo se pueda realizar a través de los métodos de la clase.

3.1. Declaración de atributos

La sintaxis general es:

```
[modificadores] <tipo> <nombreAtributo>;
```

Ejemplos:

```
int x;

public int elementoX, elementoY;

private int x, y, z;

static double descuentoGeneral;

final boolean casado;
```

Cada vez que se crea un objeto, se crean tantas variables como atributos contenga ese objeto en su interior, definidas en la clase. Todas esas variables son encapsuladas dentro del objeto y solo tienen sentido dentro de él.

Dentro de la declaración de atributo encontramos:

- **Modificadores:** palabras reservadas que permiten modificar la utilización del atributo.
- **Tipo:** tipo de atributo, puede ser primitivo o referenciado.
- **Nombre.** Identificador único para el nombre del atributo, por convenio se utilizan minúsculas, en caso de varias palabras, se utiliza el primer carácter de cada palabra mayúscula a partir de la segunda palabra: nombre, nombreCandidato...

Los modificadores de un atributo pueden ser:

- **Modificadores de acceso:** son excluyentes entre sí e indican la forma de acceso al atributo de clase: private, protected, public
- **Modificadores de contenido:** No son excluyente y pueden aparecer varios a la vez: static, final.
- **Otros modificadores:** transient y volatile. Transient es para declarar atributos transitorios y volatile es para indicar al compilador que no se deben realizar optimizaciones sobre esa variable.

```
[private | protected | public] [static] [final] [transient] [volatile] <tipo> <nombreAtributo>;
```

3.2. Modificadores de acceso

Los modificadores de acceso son excluyentes. Para un atributo son:

- **Modificador de acceso por omisión (o de paquete).** Se permite el acceso a este atributo desde todas las clases del mismo paquete (package) que esta clase. No es necesario escribir ninguna palabra reservada.
- **public:** cualquier clase tiene acceso al atributo. No es habitual declarar atributos públicos (public).
- **private:** Solo e puede acceder al atributo dentro de la propia clase, el cual está oculto para cualquier otra zona de código fuera de la clase. Se recomienda que los atributos de una clase se declaren privados o

tengan acceso de paquete para garantizar la encapsulación.

- **protected:** Se permite acceder al atributo desde cualquier subclase de la clase en la que se encuentre declarado el atributo y también desde las clases del mismo paquete.

Cuadro de niveles accesibilidad a los atributos de una clase.

	Misma clase	Subclase	Mismo paquete	Otro paquete
Sin modificador (paquete)	Sí		Sí	
public	Sí	Sí	Sí	Sí
private	Sí			
protected	Sí	Sí	Sí	

3.3. Modificadores de contenido

Son no excluyentes y son los siguientes:

- **static:** convierte al atributo en común para todos los objetos de una misma clase. Todas las instancias de esa clase compartirán ese atributo con ese mismo valor.
- **final.** Indica que el atributo es una constante, su valor no puede ser modificado a lo largo de la vida del objeto. Por convenio se declaran con todas las letras en mayúsculas. Ejemplo:

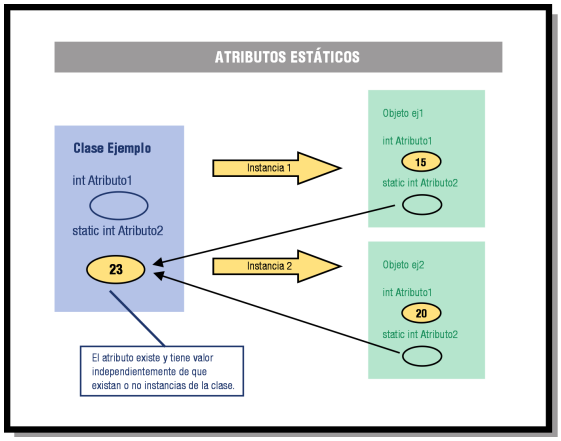
```
class claseGeometria {

   // Declaración de constantes

  public final float PI= 3.14159265;
```

3.4. Atributos estáticos

En el caso de los atributos estáticos, su existencia depende de la propia clase y no de los objetos, por lo tanto, solo habrá uno, independientemente del número de objetos que se creen. El atributo siempre será el mismo en todos los objetos y tendrá el mismo valor. Aunque no existan objetos de esa clase, el atributo si existe y puede contener un valor.



Un ejemplo sería un contador que indica el número de objetos de esa clase que se van creando durante la ejecución del programa, también habrá que implementar el código necesario para implementar el valor del atributo contador cada vez que se crea un objeto. Esto se realiza en el constructor. Otro ejemplo sería mantener un String con el nombre de la clase

```
class Punto {

  // Coordenadas del punto
```

```
  private int x, y;

  // Atributos de clase: cantidad de puntos creados hasta el momento

  public static cantidadPuntos;

  public static final nombre;
```

4. Métodos

Los métodos nos sirven para definir el comportamiento del objeto, forman parte de la estructura interna del objeto junto con los atributos.

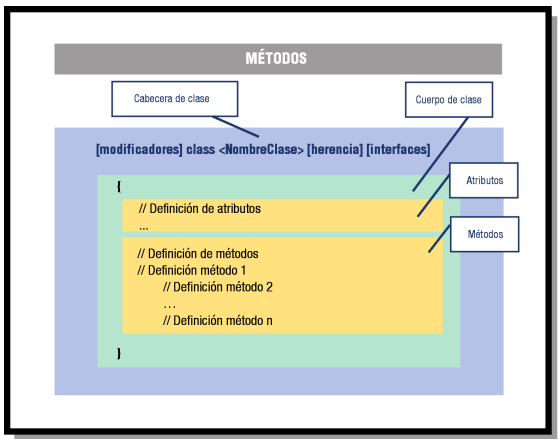
Se suelen declarar después de los atributos, aunque atributos y métodos pueden aparecer mezclados en el cuerpo, pero es aconsejable no hacerlo para mejorar la claridad y legibilidad del código.

Los métodos representan la interfaz de una clase. Son la forma que tienen otros objetos de comunicarse con ellos, solicitando información o pidiendo que se lleven a cabo acciones determinadas.

4.1. Declaración de un método

Se compone de dos partes:

- **Cabecera.** Contiene el nombre del método junto con el tipo devuelto, un conjunto de posibles modificadores y una lista de parámetros.
- **Cuerpo**. Contiene las sentencias que implementan su comportamiento, así como variables locales.



Los elementos mínimos que deben aparecer en la declaración del método son:

- Tipo devuelto por el método
- Nombre del método
- Paréntesis (opcionalmente dentro de ellos irán los parámetros que recibe).
- Cuerpo del método entre llaves.

```
int obtenerX ()

{

  /* Cuerpo del método, donde se encuentran la declaración de variables,
  sentencias y todo tipo de estructuras de control.


  ...

}
```

4.2. Cabecera del método

Puede incluir los siguientes elementos ordenados:

1. **Modificadores:** public, private y algunos más. No es obligatorio.
2. **Tipo devuelto:** tipo de dato (primitivo o referencia) que el método devuelve tras ejecutarlo. void como tipo devuelto hace que el método no devuelva ningún valor.
3. **Nombre del método.** Se aplica el mismo convenio de nomeclatura que para atributos. Además, por convenio se suele utilizar un verbo o un nombre formado por varias palabras que comiencen por verbo, seguido por adjetivos, nombres, etc... que empiecen en mayúsculas: establecerValor estaVacio moverFicha SubirPalanca girarRuedaIzquierda
4. **Lista de parámetros**. Separados por coma y entre paréntesis, cada parámetro debe incluir el tipo. Si no hay parámetros los paréntesis aparecerán vacíos.
5. **Lista de excepciones.** Se utiliza la palabra reservada throws seguida de la lista de nombres de excepciones esperada por comas, no es obligatorio aunque muchas veces es conveniente.
6. **Cuerpo del método** encerrado en llaves.

```
[private | protected | public] [static] [abstract] [final] [native] [synchronized]

<tipo> <nombreMétodo> ( [<lista_parametros>] )

[throws <lista_excepciones>]
```

4.3. Modificadores en la declaración de un método

Existen los siguientes tipos de modificadores.

- **Modificador de acceso:** igual que atributos: por omisión, public, private y protected. Tienen el mismo cometido. Acceso por parte de clases del mismo paquete, por cualquier parte del programa, solo por la propia clase o también para las subclases.
- **Modificadores de contenido.** También son static y final, pero cambia su significado. -
  - **Un método static** es igual para todos los objetos de la clase y solo tiene acceso a atributos estáticos de la clase. Pueden ser llamados sin necesidad de tener un objeto de clase instanciado. Por ejemplo, la clase Maths de java tiene métodos estáticos: Maths.abs, Math.sin, Math.cos... En cualquier caso, los objetos de una clase permiten la invocación de métodos estáticos de su clase.
  - **Un método final** es un método que no permite ser sobrescrito por clases descendientes en herencia de la clase a la que pertenece el método.
- **Otros modificadores.** Solo son aplicables a métodos: abstract, native y synchronized.
  - **native** es utilizado para señalar que el método ha sido implementado en código nativo (en un lenguaje compilado a lenguaje máquina como C o C++, solo se indica la cabecera del método, pues no tiene cuerpo en java.
  - **Un método abstracto** no tiene implementación, cuerpo vacío. Es implementado por clases descendientes, solo puede ser declarado como abstract si se encuentra en una clase abstract.
  - En un método synchronized el entorno de ejecución obliga a que cuando un proceso esté ejecutando el método, el resto de procesos que lo tengan que llamar deberán esperar a que el otro proceso termine. Es útil si sabes que determinado método va a ser llamado concurrentemente por varios procesos a la vez.

Cuadro de aplicabilidad de los modificadores.

	Clase	Atributo	Método
Sin modificador (paquete)	Sí	Sí	Sí
public	Sí	Sí	Sí
private		Sí	Sí
protected	Sí	Sí	Sí
static		Sí	Sí
final	Sí	Sí	Sí
synchronized			Sí
native			Sí
abstract	Sí		Sí

4.4. Parámetros en un método

Se colocan entre paréntesis tras el nombre del método, cada parámetro está compuesto por el tipo de dato y el nombre, separando todos los parámetros con coma. Si no hay parámetros, tan solo aparecerán los paréntesis.

Se debe tener en cuenta:

- Se pueden incluir cualquier cantidad de parámetros.
- Pueden ser de cualquier tipo: primitivos, referencias, objetos, arrays, etc...
- Una variable local del método no puede coincidir con el nombre de un parámetro.
- No puede haber dos parámetros con el mismo nombre.
- El nombre de algún parámetro puede coincidir con algún atributo de clase, este será ocultado por el parámetro, para poder acceder al atributo se utiliza el operador this.
- En java, el paso de parámetros es siempre por valor, excepto en tipos referenciados como objetos, en cuyo caso se pasa una referencia, que no podrá ser cambiada pero sí elementos de su interior, a través de sus métodos.

Es posible utilizar una construcción especial llamada **varargs,** que permite que un método pueda tener un número variable de argumentos, para ello se colocan puntos suspensivos después del tipo del cual se va a tener una lista variable, un espacio en blanco y a continuación el nombre del parámetro que aglutina la lista de argumentos variables.

Se trata de una manera transparente de pasar un array con un número variable de elementos para no tenerlo que hacer manualmente.

4.5. Cuerpo de un método

Está compuesto por una serie de sentencias java:

- Declaración de variables locales del método.
- Sentencias que implementan la lógica del método: estructuras de control, utilización de métodos de otros objetos, cálculos, cadenas, etc...
- Sentencia de devolución de valor de retorno (return). Aparece al fianl del métod permitiendo devolver la información que se le pide al método. Si el método es void no debe aparecer.

En el caso de la clase Punto teníamos los métodos de obtenerX y obtenerY. En ambos casos se devolvía un valor entero mediante la sentencia return, no recibían parámetros, ni hacían cálculos. Un método que devuelve valor de un atributo se suele llamar de tipo get o getter.

También teníamos dos métodos que servían para establecer valores a atributos del objeto (establecerX y establecerY). En este caso se le pasa un valor al método por parámetro, el cual es utilizado para variar el atributo del objeto. No se devuelve valor porque es void y no hay sentencia return. Un método que establece valores y no devuelve ninguno, se suele llamar de tipo set o setter.

4.6. Sobrecarga de métodos

Se pueden tener varias versiones de un mismo método gracias a la sobrecarga de métodos. Esto permite declarar en una misma clase varias versiones del mismo método con el mismo nombre. El compilador distingue entre varios métodos con el mismo nombre mediante la lista de parámetros del método. Desde que tiene distintos parámetros se consideran distintos métodos.

Por ejemplo, en una clase para escribir sobre un lienzo que permite utilizar diferentes tipografías en función del tipo de información que se va a escribir.

- pintarEntero(int entero)
- pintarReal(double real)
- pintarCadena (String cadena)
- pintarEnteroCadena(int entero, String cadena)

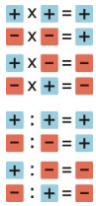
Podemos utilizar la sobrecarga de métodos de esta manera:

- pintar (int entero)
- pintar (double real)
- pintar (String cadena)
- pintarEnteroCadena (int entero, String cadena)

Como único dará error el compilador es si se utiliza otro método con el mismo número y tipo de datos de otro ya existente. Tampoco se permitiría intentar distinguirlos por el tipo de dato que devuelven.

4.7. Sobrecarga de operadores

Aunque java no lo permite, otros lenguajes de programación permiten la sobrecarga de operadores, para darle otro significado dependiendo del tipo de objetos con el que se va a operar.



En algunos casos puede resultar útil, ya que estos operadores resultan intuitivos.

4.8. La referencia this

la palabra referencia this consiste en una referencia al objeto actual. Resulta útil para evitar ambigüedad entre el nombre del parámetro de un método y el nombre de un atributo cuando ambos tienen el mismo.

this es una referencia a la propia clase en la que te encuentras, por tanto te puedes referir a un atributo del objeto con: this.atributo.

En el ejemplo de la clase Punto, this podría utilizarse si el parámetro pasado en el método establecerX se llamase igual que el atributo.

```
void establecerX (int x)
{
     this.x= x;
}
```

4.9. Métodos estáticos

Un método estático puede ser usado directamente desde la clase, sin necesidad de crear una instancia del método. Son conocidos como métodos de clase (frente a los métodos de objeto).

Los métodos estáticos no manipulan atributos de instancias, solo atributos estáticos y suelen ser utilizados para realizar operaciones comunes a todos los objetos.

Por ejemplo.

- Acceso a atributos específicos de clase: incremento o decremento de contadores de clase...
- Operadores genéricas relacionadas con la clase, por ejemplo en una clase NIF que permite trabajar con el DNI y la letra, que proporciona funciones adicionales para calcular la letra de un número de DNI, este método puede ser interesante para ser usado desde fuera de la clase de manera independiente a la existencia de objetos tipo NIF.

En la biblioteca Java se encuentran multitud de clases que proporcionan métodos estáticos, que son útiles para cálculos auxiliares, conversiones de tipos, etc... La mayoría de clases del paquete java.lang ofrecen métodos estáticos para hacer conversiones:

static String valueOf(int i). Devuelve en formato string un valor int.

```
String enteroCadena = String.valueOf(23);
```

static int paresInt(String s). Método estático de la clase Integer, analiza la cadena pasada por parámetro y la transforma en un int.

```
int cadenaEntero = Integer.parseInt("-12");
```

Este tipo de clases se utilizan como una especie de caja de herramientas que contienen métodos que pueden ser utilizados desde cualquier parte, suelen ser métodos públicos.

5. Encapsulación, control de acceso y visibilidad

Dentro de la POO es muy importante el concepto de ocultación. Los modificadores de acceso en java permiten especificar el ámbito de visibilidad de los miembros de una clase.

En función de la visibilidad que se desee que tengan los objetos o miembros de los objetos, se eligen los distintos modificadores.

Los modificadores de acceso determinan si una clase puede utilizar miembros de otra clase. Existen dos niveles:

- **A nivel general (nivel de clase):** visibilidad de la propia clase: público (public) o privada al paquete (sin modificador)
- **A nivel de miembros:** especificación, miembro por miembro de su nivel de visibilidad: público, privado al paquete, privado (private) y protegido protected).

5.1. Ocultación de atributos. Métodos de acceso

Los atributos de una clase se suelen declarar como privados o como mucho protected, pero no como public. Así evitamos que se puedan manipular inadecuadamente desde otro objeto.

Para poder manipular estos atributos creamos métodos públicos que permiten acceder a esos atributos de manera controlada: métodos getter o setter. Muchos programadores utilizan el nombre en inglés para estos métodos: setNombre, getNombre, setX, setY...

Por otro lado también podemos tener métodos que no devuelven el atributo en sí, sino un cálculo realizado con él. Por ejemplo, podemos tener getDNI, al que se le pasa el número del DNI y nos devuelve el DNI junto con la letra, calculando en el método la propia letra.

También podríamos tener un método setDNI, que deje almacenar el DNI junto con la letra, siempre y cuando el cálculo de la letra a partir de los números sea el correcto. El código que comprueba que es un DNI correcto se encuentra dentro del método setDNI y si no es correcto devuelve un error de validación (por ejemplo, lanzando una excepción). En cualquier caso, la letra del DNI no tiene por qué almacenarse nunca, al ser un cálculo

5.2. Ocultación de métodos

Los métodos de una clase pertenecen a su interfaz y es lógico que se declaren como públicos, pero se puede dar el caso de necesitar algunos métodos privados a la clase. Son métodos que realizan operaciones intermedias o auxiliares y que son utilizados por los métodos que sí son públicos. Se suelen declarar como privados porque no son de interés fuera del contexto del propio objeto.

En el ejemplo del DNI, la propia comprobación de la letra del DNI se podría calcular en un método privado que luego utilice el método get o set para interactuar con el exterior.

6. Utilización de los métodos y atributos de una clase

Una vez implementada una clase con sus atributos y método, podemos proceder a utilizarla instanciando objetos de esa clase o interaccionar con ellos. Esto se hace con el operador new.

6.1. Declaración de un objeto

Se declara exactamente igual que una variable de cualquier tipo.

```
Punto p1;
Rectangulo r1, r2;
Coche cocheAntonio;
String palabra;
```

Todas estas variables en realidad son referencias (punteros o direcciones de memoria), que apuntan a un objeto (zona de memoria) de la clase indicada en la declaración.

Un objeto recién declarado (referencia recién creada) no apunta a nada, la referencia está vacía o es una referencia nula (de hecho la variable del objeto contiene el valor null). Solamente es una variable que existe y está preparada para guardar una dirección de memoria donde se encuentra el objeto al que hace referencia. **El objeto aún no existe (no ha sido creado o instanciado).**

Para que esta variable o referencia apunte realmente a un objeto (contenga una referencia o dirección de memoria que apunte a una zona de memoria en la que hay espacio reservado para un objeto), es necesario crear o instanciar el objeto. Para ello se utiliza el operador new.

6.2. Creación de un objeto

Para crear un objeto o instancia de clase, es necesario utilizar el operador new, seguido del nombre del constructor de la clase y sus paréntesis con los parámetros. En el caso de no tener parámetros se pueden omitir los paréntesis.

```
p1= new Punto ();
r1= new Rectangulo ();
r2= new Rectangulo;
cocheAntonio= new Coche();
palabra= String;
```

El constructor de una clase es un método especial que toda clase tiene y coincide con el nombre de la clase. Se encarga de crear y construir el objeto, solicitando la reserva de memoria necesaria para los atributos e inicializarlos a algún valor si es necesario.

El entorno de ejecución de java es el que se encarga de reservar la memoria para la estructura del objeto a partir de ejecutar un método constructor. Entre estas tareas se encuentra: solicitar una zona de memoria disponible, reservar memoria para los atributos, enlazar la variable objeto en esa zona, etc...).

El método constructor no es necesario implementarlo si no se quiere hacer, ya que Java se encarga de dotar de un constructor por omisión o por defecto, pero el constructor por omisión no tiene parámetros ni realiza tareas adicionales.

Se puede declarar un objeto e instanciarlo en la misma línea:

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

6.3. Manipulación de un objeto: utilización de métodos y atributos

Una vez declarado e instanciado el método, ya existe en el entorno de ejecución y puede ser manipulado en el programa, haciendo uso de sus atributos o método.

Para acceder al miembro de un objeto se utiliza el operador punto:

```
<nombreObjeto>.<nombreMiembro>
```

Ejemplos:

```
Punto p1, p2, p3;

p1= new Punto();

p1.x= 5; //esto no es recomendable porque implica declarar el atributo como público dentro de la clase.

p1.y= 6;

System.out.printf ("p1.x: %d\np1.y: %d\n", p1.x, p1.y);

System.out.printf ("p1.x: %d\np1.y: %d\n", p1.obtenerX(), p1.obtenerY());

p1.establecerX(25);

p1.establecerX(30);

System.out.printf ("p1.x: %d\np1.y: %d\n", p1.obtenerX(), p1.obtenerY());
```

7. Constructores

En el ciclo de vida de un objeto se distinguen las fases de:

- Constructor del objeto.
- Manipulación y utilización del objeto accediendo a sus miembros.
- Destrucción del objeto.

El constructor es un método especial con el mismo nombre de la clase que se encarga de construir el objeto.

7.1. Concepto de constructor

Es un método que tiene el mismo nombre de la clase a la que pertenece y no devuelve ningún valor. Su única función es proporcionar el mecanismo de creación de instancias u objetos de clase.

Un constructor es, al fin y al cabo, una especie de método (algo especial) y como java soporta la sobrecarga de métodos, también se permite la sobrecarga de constructores (disponer de varios constructores).

Toda clase tiene al menos un constructor, que si no se define, el propio compilador lo hará por nosotros. El constructor por defecto se encarga de inicializar los atributos de la clase a sus valores por defecto (0 para numéricos, null para referencias, false para boolean...)

Una vez se incluye un constructor personalizado a una clase, el compilador ya no incluye constructor por defecto y no se podrá utilizar. Si quieres que tu clase tenga constructor sin parámetros tendrás que escribir el código.

7.2. Creación de constructores

Los constructores indican:

- El tipo de acceso.
- El nombre de la clase (el mismo que el de un constructor).
- Lista de parámetros que puede aceptar.
- Si lanza o no excepciones.
- Cuerpo del constructor (bloque de código).

Si se crea un constructor con parámetros y no se ha implementado el constructor por defecto, el intento de utilización del constructor por defecto lanza un error de compilación.

Ejemplo de constructor:

```
public Punto (int x, int y)
{

     this.x= x;
     this.y= y;
     cantidadPuntos++;  // Suponiendo que tengamos un atributo estático cantidadPuntos

}
```

En este caso el constructor recibe dos parámetros y reserva espacio para los atributos, también asigna valores iniciales a los dos atributos y por último incrementa un atributo estático llamado cantidadPuntos

7.3. Utilización de constructores

La forma de utilizar un constructor es igual que la del constructor por defecto, pero teniendo en cuenta que si hemos declarado parámetros en el método constructor, tendremos que asignar valores para esos parámetros:

```
Punto p1
p1 = new Punto(10,7);
```

En este caso estamos utilizando el constructor implementado y no el constructor por defecto.

7.4. Constructores de copia

Una forma de iniciar un objeto puede ser mediante copia de los valores de atributos de otro objeto ya existente.

Si generamos dos objetos exactamente iguales, basados en la misma clase, puedes hacer que el segundo objeto se genere con los mismos valores que el otro que ya existe. A este proceso de clonación se le llama constructor copia o constructor de copia.

Este constructor es un método constructor que recibe como parámetro una referencia al objeto cuyo contenido se desea copiar:

```
public Punto (Punto p)
{

     this.x= p.obtenerX();

     this.y= p.obtenerY();

}
```

El constructor recibe como parámetro un objeto del mismo tipo (clase Punto), inspecciona el valor de sus atributos x e y, y los reproduce en los atributos del objeto en proceso de construcción. Para utilizar este constructor podríamos hacer lo siguiente:

```
Punto p1, p2;

p1= new Punto (10, 7);

p2= new Punto (p1);
```

7.5. Destrucción de objetos

Cuando un objeto deja de ser utilizado, los recursos usados por él: memoria, acceso a archivos, conexiones bbdd... Deben ser liberados para ser utilizados por otros procesos.

La destrucción de objetos es trabajo del recolector de basura (garbage collector). Este proceso busca periódicamente objetos que ya no están referenciados y los marca para ser eliminados. Luego los elimina cuando considera oportuno.

En java también es posible implementar un método destructor de clase finalize();

El método finalize es llamado por el recolector de basura cuando se va a destruir el objeto (esto no se sabe cuando va a suceder. Si ese método no existe se ejecuta un destructor por defecto.

Se recomienda que si un objeto utiliza determinados recursos de los cuales no hay garantía que el entorno de ejecución los vaya a eliminar, se implemente el método finalize en tus clases. Si lo único que utilizan nuestras clases es espacio en memoria para los atributos, estos si son liberados sin problemas.

Si en un momento dado es necesario garantizar el proceso de finalización, se puede recurrir al método runFinalization() de la clase System para forzarlo, ya que el entorno de ejecución de java solo marca los objetos para ser borrados pero no lo tiene por qué hacer en ese instante.

Este método se encarga de llamar a todos los métodos marcados por el recolector para ser destruidos.

Para implementar un destructor se debe tener en cuenta:

- El nombre del método se debe llamar finalize();
- No recibe parámetros
- Solo puede haber uno.
- No devuelve valores (debe ser void)

8. Empaquetado de clases

La encapsulación de la información dentro de las clases permite llevar a cabo el proceso de ocultación. Conforme aumenta la complejidad de una aplicación, es probable que se necesite que algunas clases puedan tener acceso a parte de la implementación de otras debido a la relación que se establezca entre ellas cuando diseñas tu modelo de datos.

Se suele hablar de un nivel de superior de encapsulamiento llamado empaquetado.

Un paquete es un conjunto de clases relacionadas entre sí y agrupadas bajo el mismo nombre. Se encuentran en el mismo paquete todas esas clases que forman una biblioteca o que reúnen algún tipo de característica

común.

8.1. Jerarquía de paquetes

Los paquetes en java se organizan jerárquicamente de manera similar a la estructura de carpetas de un dispositivo de almacenamiento.

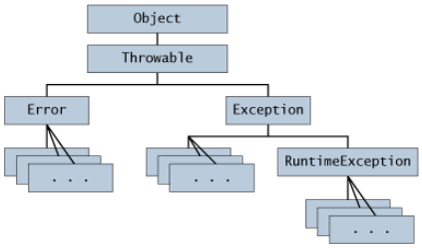
- Las clases serían los archivos
- Cada paquete es una carpeta que contiene esas clases.
- Cada paquete puede contener otros paquetes.
- Para hacer referencia a una clase dentro de una estructura de paquetes hay que indicar la trayectoria completa desde el paquete raíz hasta el paquete en el que se encuentra la clase.

La estructura de paquetes de java permite organizar y clasificar las clases, evitando conflictos de nombres y facilitando la ubicación de una clase dentro de una estructura jerárquica.

También permite el control de acceso a miembros de las clases desde otras clases del mismo paquete gracias a los modificadores.

Las clases proporcionadas por java en sus bibliotecas son miembros de distintos paquetes organizados jerárquicamente. Ese conjunto de paquetes forman la API de java.

Las clases básicas se encuentran en java.lang, las de entrada y salida se encuentran en java.io, en java.math se encuentran clases para trabajar con números grandes y de precisión.



8.2. Utilización de paquetes

Es posible acceder a cualquier clase de cualquier paquete, siempre que esté disponible en nuestro sistema, para ello debemos usar la estructura jerárquica del paquete. Esto se hace utilizando el operador punto para especificar cada subpaquete

```
java.lang.String;
java.util.regex.Patern.
```

Existe la posibilidad de indicar si se desea trabajar con las clases de uno o varios paquetes, de manera que cuando se vaya a utilizar una clase de esos paquetes no es necesario indicar toda la trayectoria. Esto se realiza con import.

```
import java.lang.String; //este es un ejemplo, peroeste paquete no es necesario importarlo ya que es el paquete por defecto.
import java.util.regex.Patern.
```

Una vez importado el paquete solo se tiene que utilizar el nombre de la clase y no toda su trayectoria para trabajar con ella.

Si vamos a trabajar con varias clases de un mismo paquete podemos utilizar el símbolo asterisco para indicar que queremos importar todas las clases del mismo paquete.

```
import java.util.*;
import java.util.regex.*;
```

8.3. Inclusión de una clase en un paquete

Se realiza mediante la palabra reservada package seguida del nombre del paquete en la cabecera del programa.

```
package paqueteEjemplo;

class claseEjemplo {

   ...

}
```

Esta sentencia se debe incluir en cada archivo fuente de cada clase. Si en un archivo fuente hay distintas clases definidas, todas formarán parte del mismo paquete.

Si no se incluye ninguna sentencia package, el compilador considera que esas clases forman parte del paquete por omisión (paquete sin nombre asociado al proyecto).

Dentro de un mismo paquete no pueden haber clases con el mismo nombre para evitar ambigüedad, pero si pueden existir en paquetes distintos.

Al igual que una clase debe tener el mismo nombre que el archivo fuente, la estructura de los paquetes debe corresponder al mismo nivel de directorios y carpetas para que el compilador sea capaz de localizar todos los paquetes.

El compilador debe tener conocimiento de donde comienza la estructura de carpetas definida por los paquetes en la cual se encuentran las clases. Para ello se utiliza el ClassPath.

8.4. Proceso de creación de un paquete

Se recomienda seguir los siguientes pasos.

- Poner nombre al paquete: suele ser habitual utilizar el dominio de internet de la empresa que lo ha creado. Para el caso de miempresa.com se podría utilizar com.miempresa.
  - Crear una estructura jerárquica de carpetas equivalente a la estructura de subpaquetes. La ruta de la raíz de esa estructura debe estar especificada en el ClassPath de java.
  - Especificar a que paquete pertenecen la clase o clases del archivo .java usando la sentencia package.

Mapa conceptual