INTRODUCIÓN Á PROGRAMACIÓN ORIENTADA A OBXECTOS

Introducción

Ata agora, as aplicacións consisten en unha serie de ordes que se van executando unha detrás de outra ata chegar ao final. Para mellorar o deseño e desenvolvemento de aplicacións, podemos agrupar en funcións un conxunto de accións para realizar accións concretas, que máis tarde poderemos utilizar nas nosas aplicacións (xerar números ao azar, facer raíces, validar números de contas, de documentos.... etc).

Co tempo, remataremos cun conxunto de funcións para realizar accións comúns no ámbito de desenvolvemento no que traballamos, o que permitirá que o desenvolvemento de aplicacións sexa cada vez máis rápido e simple. Normalmente agruparemos as funcións por "categorías" formando o que se coñece como unha *libraría ou biblioteca de funcións*.

As aplicacións que desenvolvamos poderán facer uso destas bibliotecas de funcións, das bibliotecas da linguaxe e das bibliotecas de terceiros que queiramos utilizar.

Clases e Obxectos

A programación orientada a obxectos examina as aplicacións de outro xeito. Se trata de ***examinar unha aplicación como a interacción entre os obxectos que interveñen no problema.*** Deste xeito, unha aplicación non sería máis que un conxunto de obxectos que se comunican entre sí disparando uns obxectos accións por parte de outros. Un **obxecto** é un elemento que pertence a unha **clase** que define o comportamento e características comúns de todos os seus membros.

Facendo unha analoxía co mundo real, un obxecto podería ser a mesa que temos na cociña (*mesaCociña*), e a clase seria a descripción das características concretas e o comportamento (si e que as mesas teñen algún tipo de comportamento...) de todas as **Mesas.**

Outro exemplo pode ser un automóbil concreto, (*meucoche*) (Tesla Model 3, matrícula RST-90123) e outro a clase **Automobil**, que definiría as características e comportamento de todos os automóbiles (arrancar, parar, acelerar, frenar).

En Programación Orientada a Obxectos as características definidas nas clases se coñecen como **atributos** e se crean definindo variables, mentras que o comportamento se define mediante os **métodos** que se implementan utilizando funcións.

Polo tanto, unha definición dunha clase non é mais que o agrupamento baixo un nome común dunha serie de atributos (variables) que indican as características que van a ter os obxectos da clase, e unha serie de métodos (funcións) que establecen o comportamento da clase.

Unha definición dunha clase, so é eso, unha definición. Para facer algún traballo se necesitan obxectos concretos pertencentes á clase. *O proceso de creación dun obxecto dunha clase se coñece co nome de* ***instanciación****,* e na maioría das linguaxes orientadas a obxectos se realiza mediante o operador **new.** Para crear un elemento dunha clase mediante ***new*** se chama a un método especial definido na clase denominado ***constructor***. Este método encárgase normalmente de establecer os valores iniciais para o obxecto concreto que estamos creando, e o seu nome soe ser igual á clase.

Si non definimos un método constructor na clase, as linguaxes de programación normalmente proporcionan un por defecto.

Para acceder aos atributos e métodos dos obxectos se utiliza un punto (.) entre a variable que almacena o obxecto e o atributo ou método ao que queremos acceder:

meuTesla.acelera();

meuTesla.matricula="JTY98214"

Exemplo:

As lámpadas son elementos eléctricos capaces de emitir luz. Entre as súas características está a luminosidade (en lúmenes), a temperatura da cor da luz (fría ou cálida), e o consumo (en W). As accións que poden levar a cabo as lámpadas son moi limitadas, ***prender()*** e ***apagar()***.

Imos deseñar un programa que crea unha lámpada e a prende:

1.- Definición da clase:

Clase Lampada {
luminosidade e un número enteiro;
temperatura é un string;
consumo é un número enteiro;

acción prender() {

Visualizar "Lámpada encendida"

}

acción parar() {

Visualizar "Lámpada apagada"

}

}

2.- Programa

```
Lampada l=new Lampada();      // Creamos un obxecto da clase Lámpada, e o gardamos en l
l.prende();                   // Prende a lámpada
```

¿Qué é un obxecto?. Si nos observamos a unha persoa, a examinamos como un obxecto definido por dous termos distintos: os seus atributos, e o seu comportamento. Unha persoa ten atributos como a cor dos ollos, a edade, a altura, o sexo, ... etc. Tamén ten comportamentos como camiñar, falar, respirar... etc. *Básicamente, un obxecto é unha entidade que agrupa datos e comportamentos relacionados*.

Na programación procedural, como vimos anteriormente, o código se coloca en funcións distintas que reciben unha serie de datos e producen uns resultados, actuando dende o punto de vista da aplicación como unha 'caixa negra'.

A programación orientada a obxecto agrupa nunha única entidade, os datos e os métodos que manipulan eses datos.

Na programación procedural é relativamente común o uso de variables globais (accesibles dende calqueira parte da aplicación). Aínda que esto pode parecer cómodo provoca que un erro no valor da variable provoque comportamentos inesperados en distintas partes da aplicación, facéndoa difícil de manter e dificultando a reutilización do código. A programación orientada a obxectos, incorpora toda a funcionalidade relacionada dentro da mesma clase, evitando o uso de variables globais e facilitando a reutilización das clases deseñadas en diferentes aplicacións e eliminando os efectos colaterais das variables globais.

Para garantizar o correcto funcionamento dos obxectos pertencentes a unha clase en calqueira circunstancia e preciso evitar que se alteren os valores dos atributos de modo non apropiado. Esto se pode conseguir ocultando o acceso aos atributos e métodos, regulando os distintos graos de acceso a atributos e métodos dende outras clases. Únicamente se debería ter o acceso mínimo necesario a métodos e atributos dende as clases externas.

Para ilustrar a diferencia entre programación procedural e orientada a obxecto imaxinemos un programa que quere enviar información a un servidor remoto. Para facer eso, é necesario conectarse, identificarse, enviar a información e pechar a conexión. No caso de programación procedural sería:

cliente ------> conexion- identificacion-envío-desconexión ----> --->destiño

En OOP o esquema sería:

cliente envia-----> obxecto Comunicación -> -->destiño

O obxecto Comunicación terá os métodos necesarios para implementar todas as accións relativas á comunicación, por exemplo, enviar información, e realizará internamente as operacións necesarias. E certo que as 4 accións necesarias se poderían integrar en unha única función, pero para realizar outra operación de comunicación distinta sería necesaria unha función diferente, mentras que no obxecto Comunicación teremos integradas todas as operacións relacionadas.

O obxecto Comunicación además é completo, non necesita nada externo para realizar todas as súas funcións, polo que pode ser reutilizado con facilidade en calqueira aplicación. Además permite unha mellor localización dos erros, xa que as distintas funcionalidades da aplicación están completamente delimitadas nos obxectos correspondentes. Por último, facilitan a colaboración no desenvolvemento xa que cada persoa do equipo de programación pode ocuparse do deseño dun obxecto distinto da aplicación.

Obxectos

Os Obxectos son o bloque de construción básico das aplicacións orientadas a obxecto. Unha aplicación orientada a obxectos consiste nun conxunto de obxectos que están a utilizar uns métodos proporcionados por outros, o que se chama "intercambiando mensaxes".

Neste contexto, unha "mensaxe" non é mais que o proceso polo que dende un obxecto se invoca un método de outro.

Un Obxecto está composto de datos (características do obxecto, representadas en variables e denominadas ***atributos***) e métodos (accións que é capaz de levar a cabo ese obxecto, representadas en funcións e denominadas ***métodos***).

Para crear un Obxecto é necesario ***instanciar*** unha clase. Este proceso é o encargado de xerar as estruturas na memoria necesarias para almacenar o obxecto e de darlles o valor inicial. O xeito de crear obxectos varía segundo a linguaxe, pero podemos distinguir dous grupos:

- **Instanciación estática:** Se crea un obxecto igual que unha variable, cunha simple definición. A variable identifica a dirección de memoria onde se almacena o obxecto.

Exemplo:

Factura f; // Instancia un obxecto Factura, f identifica a dirección de memoria onde se almacena o obxecto.

f é un simple identificador que representa a dirección de memoria onde se atopa almacenado o obxecto:

f (0x732323FF)
Obxecto factura

- **Instanciación dinámica:** Se crea dinámicamente en memoria o obxecto mediante un operador da linguaxe (normalmente se chama ***new***) retornando a dirección de memoria ondee se está creado. A variable **almacena** a dirección de memoria onde se atopa o obxecto, é un "punteiro" ou "apuntador" ao obxecto.

Exemplo:

Factura f=new Factura(); // Instancia un obxecto Factura, f almacena a dirección de memoria onde se almacena o obxecto.

f é un identificador que representa a dirección de memoria onde se atopa a dirección onte está almacenado o obxecto:

f (0x23523625) (0x732323FF)
0x732323FF → Obxecto factura

*Linguaxes como C++ soportan tanto instanciación estática como dinámica. **Java** únicamente soporta instanciación dinámica.*

Atributos

Os atributos son as variables que almacenan os valores que describen as características concretas dun obxecto. **O conxunto actual de valores almacenados se coñece co nome de estado do obxecto.** Durante a execución da aplicación, os distintos obxectos poden ir variando o seu estado.

Comportamento (métodos)

Os métodos son os algoritmos que describen as distintas accións que é capaz de levar a cabo un obxecto implantados coma funcións. Deste xeito é posible indicarlle a un obxecto que realice unha acción determiñada invocando ao método correspondente.

Clases

Unha clase é a ***definición*** dos atributos e métodos que conformarán todos os obxectos que se creen de esa clase. Unha aplicación OOP consiste na definición das distintas clases que interveñen na aplicación, indicando os atributos e métodos dos que están compostos (por suposto, definindo os algoritmos dos distintos métodos), e a posterior instanciación (creación) dos obxectos que interveñen na aplicación. Eses obxectos irán invocando métodos e instanciando novos obxectos para conseguir realizar a súa función. *Unha clase **define** a funcionalidade, pero que realimente a leva a cabo é un **obxecto** concreto.* Unha clase é como unha simple "molde" a partir do que se crean os obxectos.

Definición de Atributos

Os atributos dunha clase se definen declarando variables do tipo apropiado para almacenar os valores desexados.

Na maior parte das ocasións é desexable impedir a manipulación externa dos atributos, de modo que impidamos colocar valores nos mesmos que impidan o correcto comportamento do obxecto. Isto se consigue mediante os modificadores de acceso. Normalmente os niveis de acceso que se poden definir son:

- *público* – Todo o mundo ten acceso
- *privado* – So se pode acceder dende dentro da propia clase
- *protexido* – So se pode acceder dende a propia clase ou unha clase derivada

Facendo os atributos privados se garante que dende ningunha parte da aplicación se vai a poder alterar o estado do obxecto de modos non previstos.

Definición de Métodos

A funcionalidade dos obxectos dunha clase se describe codificando os algoritmos correspondentes mediante funcións. Estas funcións deben describir:

- O tipo de dato producido, si a linguaxe é con tipo (resultado devolto mediante ***return***)
- O identificador do método (o seu nome)
- As variables onde se reciben os datos, si a linguaxe é con tipo indicando os seus tipos (os parámetros)

Do mesmo xeito que cos atributos, é posible illar o acceso aos métodos. Sen embargo, so ten sentido facer privados ou protexidos os métodos que non son realmente funcionalidades dos obxectos si non que se utilizan como funcións auxiliares para crear as funcionalidades desexadas.

*O conxunto de métodos públicos dunha clase se denomina **interface da clase.** A interface dunha clase define a súa funcionalidade.*

Para poder utilizar toda a potencialidade dun obxecto, é necesario coñecer completamente a súa interface. Polo tanto é necesaria unha completa documentación da interface de xeito terceiras perosas podan obter esa información de xeito simple. Oracle ofrece para ***Java*** unha documentación completa das clases subministradas co JDK vía web, é moi importante consultala para poder utilizar os obxectos básicos ofrecidos pola linguaxe, como Date, String, Calendar.... etc.

Un conxunto de métodos típicos na maioría das clases son métodos especiais encargados de facilitar o acceso para ler ou modificar atributos privados (que non son accesibles dende fora da clase). Deste xeito estaremos forzando a pasar por unha función para alterar os valores dos atributos, o que da a oportunidade para poder filtrar os valores incorrectos.

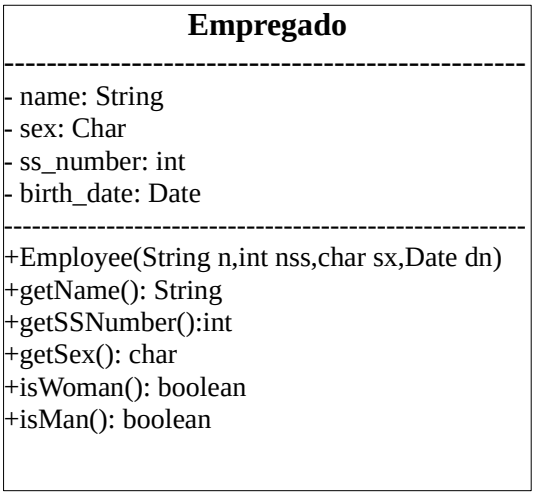
Estas funcións se denominan **setter** (para alterar o valor dun atributo privado) e **getter** (para obter o valor dun atributo privado)

Outros métodos “especiais” son os encargados de inicializar os valores dos atributos no momento de crear un novo obxecto. Estes métodos se denominan ***constructores***, e deben existir obrigatoriamente. Si nos non definimos un método constructor para unha clase, a propia linguaxe xenera un sen argumentos.

A diferencia dos outros métodos, nos métodos constructores non se indica o valor devolto, xa que sempre devolverán a referencia do novo obxecto creado, e normalmente reciben como argumentos os valores iniciais que desexamos poñer nos atributos. **En Java, os constructores deben chamarse igual que a clase.**

UML

Para a realización do deseño de aplicacións en POO se empregan unha serie de diagramas que conforman UML (Unified Model Language). Algúns destes diagramas son o ***Diagrama de Clases***, que amosa as Clases que interveñen no problema indicando os atributos e métodos relevantes, o **Diagrama de Casos de Uso** que describe os distintos escenarios que debe contemplar a aplicación indicando quén inicia a execución do escenario (Actor) e todas as posibles respostas do sistema que se vai a construír, ou o ***Diagrama de Secuencia*** que describe a secuencia de accións necesarias por parte dos distintos obxectos para a realización dos casos de uso. O seguinte exemplo amosa unha descrición dunha clase en UML, e a súa codificación en Java.



```
// Definición da Clase Empregado – Non ten funcionalidade, so sirve de "molde" para crear obxectos Empregado
//
Class Employee {
  // ----> Definición de Atributos
        // Nome do empregado. So accesible dende os métodos definidos dentro da clase Empregado
        private String name;
        // Número da seguridade social. So accesible dende os métodos definidos dentro da clase Empregado
        private int ss_number;
        // Sexo. Debe ser H ou M (home ou muller). So accesibledende os métodos definidos dentro da clase Empregado
        private char sex;
        // Data nacemento. O seu tipo é un obxecto da clase Date, e so é accesible dende métodos da clase Empregado
        private Date birth_date;

  // -----> Definición de Métodos
        // Método constructor.
        // Constrúe un obxecto Empregado co nome, numeros de seguridade social e sexo indicados.
        public Employee(String n,int nss,char sx,Date dn) {
                if ( (sx!='M') && (sx!='H')) throw new Exception("Sexo erróneo");
                name=n;
                ss_number=nss;
                sex=sx;
                birth_date=ds;
        }

        public String getName() {
                return name;
        }

        publico int getSSNumber() {
                return ss_number;
        }

        public char getSex() {
                return sex;
        }

        public boolean isWoman() {
                return (sex=="M");
        }

        public boolean isMan() {
                return (sex=='H');
        }
}
```

A clase definida aquí establece os atributos e métodos que van a ter os obxectos Employee que instanciemos. É como un "molde" que se utiliza para crear as estructuras en memoria para o obxecto. Poderíamos facer o seguinte.

```
Employee l=new Employee("Luis",233434344, 'H', new Date(1987,12,06));
Employee s=new Employee("Susana",51534344, 'M', new Date(1991,11,08));
```

No exemplo anterior instanciamos dous obxectos distintos pertencentes a clase Employee. Para poder facelo, necesitamos instanciar tamén dous obxectos Date, que almacenan a data de nacemento de cada un de eles. A instanciación é dinámica, polo que dentro deles atópase a dirección de memoria onde están situados os obxectos creados.

Unha vez instanciados os obxectos é posible utilizalos seus métodos.

```
if (l.isMan())    System.out.println("Benvido, señor "+l.getName());
else              System.out.println("Benvida, señorita "+l.getName());
```