Conceptos Básicos de OOP e JAVA

Conceptos Básicos de OOP

Introducción

Un algoritmo é unha serie de pasos definidos para realizar unha acción concreta. Por exemplo, os pasos a seguir para cociñar unha tortilla de patacas poden definirse mediante un algoritmo. Calqueira persoa que siga esa secuencia de pasos e conte co material necesario (datos de entrada) rematará cunha tortilla de patacas (datos de saída).

Un programa ou aplicación e unha serie de algoritmos combinados para realizar algo máis complexo. Podemos facer un símil entre elaborar a tortilla de patacas (algoritmo) ou elaborar un menú para unha voda (aplicación/programa).

Existen diversos xeitos de enfocar a elaboración de aplicacións:

- A máis básica consiste simplemente en determiñar a secuencia de pasos completa a seguir para solucionar o problema. Esta técnica é útil para problemas moi simples, pero non permite a elaboración de solucións complexas fácilmente.
- Unha mellor aproximación é identificar o conxunto de tarefas a realizar sin entrar en detalles, e logo ir solucionando cada unha das tarefas. Para solucionar estas tarefas, se repite o procedemento ata que a tarefa é o suficientemente simple para aplicar o método indicado no punto anterior. Esta técnica coñécese como *deseño descendente* ou ***top-down*** e permite solucionar problemas complexos e facer unha boa división do traballo para que poda ser realizado simultáneamente por varias persoas.
- Unha terceira técnica consiste en identificar os elementos que interveñen no problema, e como deben interaccionar entre eles para conseguir o que se desexa. Coñécese co nome de Programación Orientada a Obxecto, e é un paso máis sobre o deseño *top-down*.

Vexamos como exemplo a elaboración dun menú para unha voda.

Podemos tomar unha aproximación á solución como esta :

1.- Unha vez pensado que imos a elaborar, e*mpezamos a describir todas as accions a realizar unha por unha hasta chegar a ter elaborado todo o menú, as mesas postas e colocadas, e a decoración do local.* Si realizamos todas esas accións unha por unha, remataremos con todo preparado para a comida.

Ou como esta (Orientada a Obxectos):

2.- En lugar de describir as accións necesarias para conseguir ter todo listo, identificamos en primeiro lugar os obxectos que interveñen no problema:

- O **Menú** a servir.
- A **Disposición** das mesas e cubertos.
- A **Decoración** da sala.

- Indicamos os pasos a seguir para que a sala quede perfectamente decorada (Decoracion)

- Indicamos os pasos a seguir para que as mesas e cubertos estén perfectamente (Disposición)

- Elaboramos o Menú (Menu). O menú consiste nunha serie de pratos a servir (**Pratos**). Para cada Prato indicamos os pasos a seguir para a súa elaboración.

- Por último, indicamos os pasos a seguir para que todo esté preparado para a comida, que serían: Crear o Menu, Crear a Decoración, Crear a Disposición

Estos tres pasos poden realizarse además de xeito simultáneo.

A primeira aproximación, permite solucionar o problema concreto do que se trata, pero ao consistir simplemente nun conxunto de instruccións a seguir resulta maís difícil entender a totalidade, e polo tanto correxir calqueira fallo. Además, si dispoñemos de varias persoas para a tarefa é moi difícil repartir o traballo (os pasos) para que se realice de modo eficiente.

Un cambio simple como variar o menú, a decoración ou a disposición da mantelería, sería bastante laborioso e confuso xa que teríamos que ir a modificar as secuencias de instruccións no programa apropiadas a cada caso. Outra consecuencia é que os pasos non se poden reutilizar dunha maneira sinxela para outras tarefas, como unha *merenda*, un *almorzo*, un *cumpleanos*..... necesitaríamos reescribir case todo.

A segunda aproximación en cambio, permite entender moito mellor o funcionamento, e polo tanto realizar modificacións, xa que os conxuntos de instruccións están organizados por funcionalidades. Esta organización además permite un reparto de tarefas eficiente, xa que cada persoa pode encargarse dunha tarefa concreta en lugar de dispoñer únicamente dunha lista de instruccións a seguir. Os cambios son máis simples: Para variar a decoración sei exactamente donde están o conxunto de pasos a realizar.

O máis importante sen embargo é que permite unha reutilización das instruccións dun modo moi eficiente. Podo ter instruccións que indican como realizar distintos tipos de Decoración, instruccións para a realización de multitude de Pratos e instruccións con distintas Disposicións de mesas, mantelería e cubertos. Si quixera cambiar un Prato do menú, simplemente elaboraría as instruccións para a realización de ese Prato (so si non as tivera xa realizadas de outra vez) e o engadiría ao menu en sustitución do outro Prato. Distintas combinacións de diferentes Decoracións, Pratos, e Disposicións me permitirían organizar vodas, cumpleanos, festas informais, almorzos, ceas de traballo.... etc. dun modo moi simple. Cada elemento que programe podrei utilizalo fácilmente en novas aplicacións.

A orientación a obxectos é unha técnica de programación, en principio independente da linguaxe de programación utilizada. Sen embargo, hoxe en día existen multitude de linguaxes que nos proporcionan as estructuras necesarias para utilizar esta técnica dun xeito eficiente, son as ***Linguaxes Orientadas a Obxecto***.

Clases e Obxectos: Contructores

O concepto básico da Programación Orientada a Obxectos é a ***Clase***.

Unha Clase é unha descripción xenérica de todos os obxectos que podan pertenecer a esa clase. Por exemplo, a clase *Automóbil* sería unha descripción xenérica de calqueira automóbil. Estas descripcións constan de dúas partes diferenciadas:

1. As características xerais (***atributos***), como a cor, o número de rodas, o motor, a cilindrada, etc.... algunhas desas características como o motor poden ser obxectos de outra clase, coma

o motor (un motor ten características como cilindrada, tipo de combustible, potencia..... que son do motor, non do vehículo).

2. As accións que son capaces de efectuar os obxectos de esta clase (***métodos***), que serían os pasos a seguir para realizar as distintas accións que son capaces de realizar os obxectos de esta clase. Por exemplo, arrancar, parar, acelerar, cambiar de marcha..... etc. Estes pasos están agrupados por funcionalidade en diversas *funcións.*

Un Obxecto é un elemento que pertence a unha clase, e polo tanto dispón dos métodos e atributos definidos na clase. Por exemplo un *BMW serie 3 con matrícula 2345JTM* podería ser un Obxecto da Clase Vehículo, e polo tanto dispon dos atributos descritos na clase Vehículo, e é capaz de realizar as accións definidas na clase Vehículo. *O procedemento para crear un Obxecto denomínase instanciación*, e se realiza invocando a un método especial (acción) definido na clase denominado ***constructor***, que se encarga de construir o obxecto e normalmente de dar un valor inicial ás súas características concretas (atributos).

En Java, o constructor é un método que se chama igual que a clase, e se invoca co operador ***new.*** Por exemplo: ***Vehiculo v=new Vehiculo("BMW serie 3","2345JTM");*** instanciaría (crearía) o obxecto **v.**

Como indicamos anteriormente, ***unha clase non é mais que unha definición xenérica*** polo que as clases non teñen atributos nin métodos, únicamente as definen. Sen embargo, en certas situacións pode interesarnos almacenar información común a calqueira obxecto que queiramos instanciar (por exemplo, si queremos levar conta do número de vehículos que levamos instanciados), ou definir accións que queremos poder realizar sen necesidade de instanciar un obxecto. Nesas circunstancias podemos definir métodos e atributos ***static*** que existen nun ámbito global, independentemente dos obxectos instanciados.

Herencia, Sobreposición e Sobrecarga

Unha das grandes ventaxas da programación orientada a obxectos é a reutilización de código. Estos tres mecanismos (herencia, sobreposición e sobrecarga) permiten unha reutilización eficiente.

Herencia

A herencia consiste en definir novas clases tomando como referencia unha clase base. Deste xeito únicamente necesitaríamos engadir as características e funcionalidade que faltan na clase base, ou variar na nova clase as instruccións para realizar determiñanda acción.

Supoñamos por exemplo que temos definida unha clase denominada **Prato**, que ten como características unha *lista de Ingredientes*, e como métodos único un método denominado *cocinado.* Podería crear novas clases derivadas de **Prato** (Tortilla, RodaballoOForno, EnsaladaMixta), e todas elas terían automáticamente unha *Lista de Ingredientes* e un método *cocinado*, que coinciden exactamente cos definidos na clase Prato. Logo se modificaría a instrucción de cocinado e a lista de ingredientes dentro de cada "subclase" para que se comporten do modo axeitado. Unha vez feito esto, podería elaborar unha tortilla:

Tortilla t=new Tortilla();
t.cocinado();

un *RodaballoOForno....*

RodaballoOForno r=new RodaballoOForno();
r.cocinado();

etc....

Posteriormente incluso podería crear a nova clase TortillaPatacas heredando de Tortilla, engadindo como ingredente as patacas, e alterando os pasos indicados no método *cocinado....*

Tanto as Tortilla, TortillaPatacas, RodaballoOForno ... etc serían Pratos.

Sobreposición e Sobrecarga

Cando nunha clase derivada variamos a modo de traballar dunha acción definida na clase base (como cocinado, en Tortilla ou en TortillaPatacas), decimos que estamos sobrepoñendo o método (**Sobreposición**).

Tamén é posible indicar distintos modos de realizar unha mesma acción segundo a información que se suministre, o que se realiza mediante a codificación de varios métodos co mesmo nome, variando únicamente no tipo de información que reciben. Esta técnica coñécese como **sobrecarga**.

Encapsulación

A progamación orientada a obxectos persigue a reutilización do código do xeito máis seguro posible, esto se consigue facendo que cada Obxecto se comporte sempre do modo no que foi deseñado sen ter en conta o a manipulación que se poda realizar dende fora de ese obxecto. Para conseguilo, é fundamental que os obxectos non podan ser alterados ou influidos de xeitos que non se pensaron ó ser deseñados.

O medio para impedilo, consiste en limitar o acceso aos atributos e métodos desde o exterior do obxecto en múltiples grados. *O feito de permitir o acceso aos atributos e métodos únicamente do modo previsto, se coñece como encapsulación.*

En JAVA utlízanse os modificadores ***public, private, protected*** **e** ***friendly*** antes da definición do atributo (*public int edade;*) ou o método (*private int acelera(int aceleracion, int segundos);*) para indicar o grado de acceso que proporcionamos aos atributos e métodos.

Polimorfismo

Gracias ao Polimorfismo é posible utilizar obxectos de distinta clase como si foran da mesma. Durante a execución da aplicación é posible determiñar qué método se debe invocar. Por exemplo, si facemos o seguinte:

Prato p=new RodaballoOForno();

Prato q=new TortillaPatacas();

O Polimorfismo garantiza que si chamamos a p.cocinado(); se invocará ao método de cocinado de RodaballoOForno, e si chamamos ao método q.cocinado(); se invocará ao método de cocinada da TortillaPatacas, aínda que tanto **p** e **q** son da mesma clase, **Prato**.

O que non se podería facer e chamar a métodos específicos de RodaballoOForno en p, xa que p é de tipo Prato. O polimorfismo o que nos garantiza é que os métodos de Prato chamen á versión apropiada.

Excepcións e tratamento de erros

As excepcións son un sistema de xestión de erros. Tradicionalmente os erros nas aplicacións se xestionaban mediante códigos de erro, por exemplo:

```
boolean c;
c=verificaContaCorrente(“3567324ABD21”);
if (!c) System.out.println(“Código erróneo”);
else {
        // Lóxica da aplicación
}
```

Esta aproximación presenta dous graves problemas:

- A lóxica de control de erros (os if para comprobar a corrección dos codigos de erro) fan que o programa sexa máis difícil de leer e mais longo, necesitando escribir moitas liñas de código.
- Este sistema permite ao programador obviar os códigos de erro e non controlar os posibles fallos, o que tradicionalmente ocurre con moita frecuencia.

Para solucionar esto, se deseñaron as ***excepcións***. Unha ***exception*** é un sinal (en realidade un Obxecto con información sobre o erro) que se lanza cando se produce un erro. No momento en que se lanza esa sinal temos dúas opcions:

- Indicamos expresamente que non nos imos ocupar de ese erro. Nese caso, o código que chama debe ocuparse do tratamento.
- Capturamos o sinal e nos ocupamos de xestionar o erro mediante un bloque ***try {} catch***. En ningún caso podemos ignorar a posibilidade de erro.

Por exemplo, o caso anterior quedaría como sigue:

```
boolean c;
try {
        verificaContaCorrente(“3567324ABD21”);
        // Lóxica da aplicación
} catch(Exception e) {
        System.out.println(“Error: ”+e.getMessage());
}
```

Conceptos Básicos de JAVA

Características fundamentais dos programas JAVA

JAVA é unha linguaxe orientada a obxectos, as súas características principais son:

- Java é unha linguaxe pseudocompilada. Se compila a un código intermedio que é executado por unha máquina virtual Java (JVM).
- Nos proporciona un mecanismo de herencia simple: *Unha clase Java so pode heredar de unha única clase pai*.

- En Java (salvo os tipos primitivos) únicamente existen obxectos e clases. Non pode existir código fora dunha clase.
- Todas as clases Java descenden da clase Object.
- As clases Java se organizan en **packages**, si non o especificamos, a clase pertenece ao package por defecto. O nome do package é unha secuencia de identificadores separados por punto, correspondendo cada identificador a unha carpeta, indicando donde debe atoparse a clase dende a carpeta principal das clases (CLASSPATH).
- Unha aplicación Java comenza polo método ***static main*** da clase indicada ao invocar a execución da JVM (comando ***java***)

Estructura do código fonte JAVA

Como comentamos anteriormente, en Java todo o código debe ir dentro de algunha clase. A estructura das aplicacións Java é normalmente a seguinte:

- Cada clase pública vai nun ficheiro fonte que ten como nome o nome da clase e extensión .java
- A clase principal da aplicación (pola que comenza a execución) debe ter un método main coa forma ***public static void main(String[] args);*** é por esta clase pola que se inicia a execución da aplicación, e a JVM levará a cabo as instruccións presentes neste método.
- As clases se poden organizar en ***packages***. A misión dos packages é organizar as Clases de xeito que non existan conflictos entre clases distintas que teñen o mesmo nome. Unha aplicación debe importar as clases que non pertencen ao seu package para poder utilizalas mediante a sentencia ***import***
- En Java únicamente se manipulan obxectos, salvo os tipos de datos primitivos, que existen por motivo de eficiencia, aínda que Java dispón de clases que ofrecen a funcionalidade equivalente. Eses tipos primitivos son **byte, short, int, long, char, boolean, double e float**