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 (conexión, identificación, envío, desconexión) realizando internamente as operacións necesarias de xeito que o cliente se limita a "transmitir" a información. O Obxecto é unha especie de caixa que ofrece as funcionalidades necesarias para a comunicación de xeito aillado do exterior. 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, mentres que no obxecto Comunicación teremos integradas todas as operacións relacionadas.

O obxecto Comunicación é completo, non necesita nada externo para realizar todas as súas funcións, polo que pode ser volto a usar con facilidade en calquera aplicación. Ademais permite unha mellor localización dos erros, xa que as distintas funcionalidades da aplicación relativas á comunicación están completamente delimitadas neste obxecto. Por último, facilita a colaboración no desenvolvemento xa que se lle pode encargar a outra persoa a programación da clase Comunicación mentres continuamos desenvolvendo o programa principal.

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.* Si en C++ definimos unha variable como **Factura f;** , se creará un obxecto identificado por f. En cambio en Java, **Factura f;** so define unha variable capaz de almacenar unha referencia a un obxecto que debe ser creado posteriormente co operador ***new***.

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.

Para acceder a un atributo dun obxecto: ***obxecto.atributo***. Dende dentro do propio obxecto, se utiliza directamente o nome do atributo.

Pode darse o caso que dentro de un método teñamos unha variable co mesmo identificador que un atributo. Nese caso, non é posible acceder ao atributo utilizando simplemente o nome, xa que non aclara si se refire á variable do método ou ao atributo da clase. Para solucionar este problema podemos acudir á variable **this**. Esta variable é un identificador reservado que almacenará sempre a referencia ao obxecto actual. Para acceder a un atributo dun obxecto de modo inequívoco podemos antepoñer dentro da clase o identificador *this* (*this.atributo*).

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.*

O acceso aos métodos segue as mesmas regras que o acceso os atributos, facendo uso de ***this*** cando sexa necesario.

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.**

Construtores e Destrutores

O proceso de crear un Obxecto se coñece como ***instanciación***. Unha instancia é un obxecto concreto de unha clase.

Para crear un obxecto é necesario crear as estruturas de datos en memoria que representan ese obxecto concreto, en particular os atributos. Normalmente para crear esas estruturas e dar un valor inicial aos atributos se emprega un método especial denominado “construtor”, xa que é o encargado de construír o obxecto.

Si nos non escribimos un construtor que especifique o modo en que se vai a instanciar o obxectos (principalmente establecer os valores iniciais dos atributos, aínda que pode ser calquera cousa que necesitemos facer) se utiliza un construtor sen argumentos denominado ***construtor por defecto***. En Java si definimos un construtor, o construtor por defecto non existe.

O modo de declarar o construtor varía segundo a linguaxe. En Java o construtor ten que ter o mesmo nome que a clase e non se especifica valor de retorno, xa que retornará o obxecto construído.

Cando un obxecto deixa de existir, a veces é necesario realizar algun tipo de “limpeza”, como pechar conexións de rede, eliminar ficheiros temporais... etc. Cando un obxecto deixa de existir se executa un método especial da clase denominado ***destrutor*** que podemos deseñar ao noso gusto.

En algunhas linguaxes orientadas a obxectos como *C++* a vida dos obxectos está perfectamente definida, e sabemos en que instante se vai producir a súa eliminación. Sen embargo en outros como *Java*, non e posible sabelo xa que se decide durante a execución do programa en que instante é necesario eliminar os obxectos que non van a ser utilizados mediante o ***recolector de lixo.***

Atributos e Métodos estáticos

*Os **métodos ou atributos estáticos** pertencen a globalidade dos obxectos da clase (non existe unha copia para cada obxecto, se non que todos os obxectos utilizan a mesma copia) e **existen e poden ser utilizados** con independencia da existencia de un obxecto da clase.*

Normalmente os ***atributos estáticos*** utilízanse cando queremos manter un valor común a todas as instancias, pero é moi importante ter sempre en conta que cando se modifica o valor dese atributo afecta a todos os obxectos existentes (no instante actual ou no futuro) desa clase. O uso de atributos estáticos non é moi frecuente, aínda que ten moita utilidade en certas situacións.

Os ***métodos estáticos*** son moito máis utilizados. Serven para poder invocar métodos sen necesidade de crear un obxecto da clase. Se utilizan cando desexamos agrupar métodos de temática común, sen que teña sentido instanciar un obxecto, como por exemplo operacións matemáticas

```
// 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) throws Exception {
                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.

Unha vez definida a clase, podemos instanciar 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.

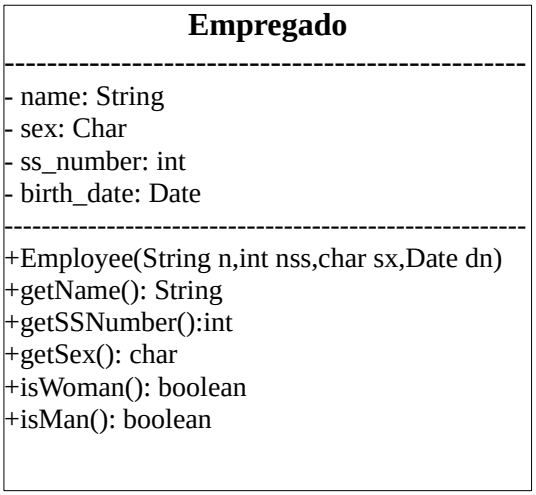
```
Employee l=new Employee(“Luis”,233434344, ‘H’, new Date(1987,12,06));
Employee s=new Employee(“Susana”,51534344, ‘M’, new Date(1991,11,08));
```

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());
```

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.



Cando nos enfrontamos ao deseño dunha solución a un problema mediante orientación a obxectos, o máis importante é a identificación dos obxectos que interveñen no problema, os seus atributos, e os métodos que deben implantar. Posteriormente analizamos como se relacionan eses obxectos entre sí para crear cada un dos casos de uso da aplicación.

O programa consistirá no deseño das clases e na instanciación dos obxectos necesarios para iniciar os casos de uso.

Un xeito bastante eficaz de identificar os obxectos do problema consiste en seleccionar no problema os **sustantivos** (que corresponderán normalmente a obxectos ou atributos) e os **verbos** que poden corresponder con métodos dos obxectos.

Exemplo:

*Queremos crear un **xogo de taboleiro** que consiste nun **taboleiro** de 8x8 **casiñas** nas que se atopan inicialmente catro **pezas** no centro, dúas brancas e dúas negras aliñadas en diagonal. Un **xogador** so pode **poñer unha peza** nunha casiña na que pecha en horizontal ou diagonal pezas do xogador contrario entre dúas fichas propias, nese caso todas as pezas pechadas na xogada cambian de cor cambiando a **quenda** ao outro xogador. O xogo **remata** cando ningún xogador pode efectuar unha xogada gañando o xogo o que teña máis fichas da súa cor.*

Deseño

Na POO é primordial identificar os ***obxectos*** e os ***actores*** que van a intervir no problema que estamos modelando co software. Para elo, podemos *examinar os substantivos e o s verbos* que aparecen no enunciado. Os substantivos normalmente ou ben non teñen relevancia, ou son obxectos, ou son atributos dun obxecto, mentres que os verbos poden ser métodos (funcións) dunha clase.

Outro elemento a identificar son os **casos de uso**, é decir, todas as situacións ou escenarios posibles que debe controlar o software e a resposta que debe ofrecer. Parte fundamental do caso de uso son os **actores**. Un actor é o elemento que orixina a escena ou caso de uso. No exemplo anterior podemos distinguir os seguintes elementos:

- Temos un ***Xogo de Taboleiro***
- Temos un ***Taboleiro*** de 8x8 casiñas
- Temos ***casiñas***
- Temos ***Pezas,*** que poden ser de dúas cores distintas, branco e negro
- Temos 2 ***Xogadores***
- Temos unha ***quenda***, e un *cambio de quenda*
- ***Os Xogadores poñen*** pezas no taboleiro
- O ***Xogo nalgún momento remata***

Examinando estes elementos, podemos concluir que o **Xogo de Taboleiro é un Obxecto** que ten como **atributo un Taboleiro de 8x8 casiñas**. A súa vez, o Tableiro podería ser un obxecto formado por casiñas organizadas en filas e columnas, e as casiñas serían simplemente un atributo que almacena un valor. Alternativamente, dado o simple do taboleiro (non ten métodos claros, por exemplo)**, podemos deseñalo como un atributo do Xogo consistente nunha táboa (array) de 8x8 valores enteiros.** As **casiñas** serían simplemente os elementos do array**.** As **pezas** poderían ser simplemente un valor numérico que indicaría a cor da peza.

Os xogadores son elementos externos á aplicación e son quenes realizan as xogadas. Si queremos que a aplicación poda xogar contra un humano teríamos que crear un obxecto Xogador capaz de seleccionar xogadas válidad no taboleiro seguindo unha estratexia. A **quenda** é claramente un atributo do xogo que podemos representar cun número que coincida coa cor da peza, e *o* **cambio de quenda** e un simple cambio no valor dese número que cambiaría tamén a cor da peza. *O remate do xogo se produce cando ningún xogador poda xogar,* de xeito que o Xogo debe ser capaz de determiñar si a partida remata ou non

Os casos de uso serían:

- CASO 1: Un xogador pon una ficha nunha casiña válida
  - Neste caso, se cambian as fichas pechadas do xogador contrario
  - Se verifica si a partida remata. Si remata se contan as fichas e se notifica o gañador, se non, se cambia de quenda si o xogador contrario ten movementos posibles, se non se continúa na mesma quenda.

- CASO 2: Un xogador pon unha ficha nunha casiña non válida.

- Se indica o erro

Para a realización do deseño en POO se empregan unha serie de diagramas que conforman UML (Unified Model Language), que inclúen numerosos diagramas como o Diagrama de Clases, que amosa as Clases que interveñen no problema, o diagrama de casos de uso que indican os casos de uso da aplicación e o seu comportamento ou o diagrama de secuencia que describe a secuencia de accións necesarias para a realización dos casos de uso.

Clase Xogo

Atributos:
tableiro[8][8]
quenda

Métodos:
xogar_peza
comprobar_vencedor
cambio_quenda
comprobar_fin

Codificación

A codificación se realizaría nunha linguaxe de programación Orientada a Obxectos, no noso caso Java. En pseudocódigo sería algo así:

Inicio:

1. Creamos o obxecto ***xogo*** – Inicialmente a quenda será 0 e terá as pezas colocadas no centro
2. Mentras non remate o xogo (***xogo.comprobar_fin()*** sexa falso)
   a) Se pon unha peza (***xogo.xogar_peza()***). Este método se encargará de determiñar a posición na que quere poñer a peza o xogador correspondente ao quenda, por exemplo, solicitando cunha mensaxe e unha entrada por teclado ou recibindo a entrada mediante un click nun entorno gráfico si o xogador é humano, ou mediante un algoritmo que seleccione a casiña óptima si o xogador é a propia aplicación.... Si a xogada é correcta, se reviran as pezas axeitadas e se non, se notifica.
   b) Se cambia de quenda si a xogada é correcta e o outro xogador ten movementos posibles, se non se continúa na mesma quenda.
3. Se determiña o gañador (***xogo.comprobar_vencedor()***) e se visualiza a mensaxe apropiada

Fin.

Sería algo así:

```
Crear  obxecto xogo
Mentras !xogo.comprobar_fin()
        Pedir fila e columna da xogada para o xogador que lle toque
        Si (xogo.xogar_peza(fila,columna) e correcto)
                xogo.cambia_quenda();
        Fin-Si
Fin Mentras
t=xogo.comprobar_vencedor();
Visualizar “Gañou o xogador “ t
```

Faltaría un método de Xogo para visualizar o tableiro e visualizalo ao finalizar cada xogada, noutro caso estaríamos xogando “a cegas”. Bastará implantar os distintos métodos da clase Xogo de xeito que fagan a labor descrita e teremos a aplicación terminada.