Clases e obxectos

Sumario

Clases

Declaración de clases

Nas seccións anteriores xa se declararon algunhas clases, como o exemplo da bicicleta. A declaración dunha clase mínima é como segue:

```
class UnhaClase {
   // Declaración de atributos, construtores, e métodos
}
```

O **corpo da clase** e o que vai entre chaves `{ }` e pode conter:

- **Construtores** para inicializar novos obxectos
- Declaración de **atributos** que determinan o estado da clase e os seus obxectos
- **Métodos** para implementar o comportamento da clase e dos seus obxectos

Podemos ampliar a declaración dunha clase, tal e como xa vimos, do seguinte xeito:

```
class UnhaClase extends Pai implements unInterface {
   //Declaracion de atributos, métodos e construtores
}
```

A modo de resumo, a declaración dunha clase pode incluír os seguintes compoñentes:

- **Modificadores de acceso**, como public ou private
- O **nome da clase**, onde a primeira letra adoita a escribirse en maiúsculas
- O nome da clase pai ou superclase, especificado coa palabra **extends**

- O interface ou interfaces que implementa a clase especificado coa palabra **implements**
- O **corpo** da clase entre chaves que contén a ?lóxica? da clase

Declaración de variables membro

En Java hai varios tipos de variables:

- Os atributos dunha clase que tamén se lles chame **variables membro** dunha clase
- Variables dentro dun método ou dun bloque de que tamén se lles chama **variables locais**
- Variables que se usan na declaración dun método que tamén se lles chama **parámetros**

A clase bicicleta, vista anteriormente, utiliza as seguintes liñas de código para definir os seus atributos ou variables membro:

```
public int marcha;
public int velocidade;
```

Para declarar un atributo hai que seguir a seguinte sintaxe:

- **Modificador de acceso** (public, private, etc), é opcional
- O **tipo de dato** do atributo
- O **nome** do atributo

Os atributos da clase Bicicleta son marcha e velocidade, ambos de tipo enteiro (int). A palabra public define estes atributos como públicos, accesibles por calquera obxecto. É habitual declarar os atributos dunha clase como private e os métodos que os manipulan como public. Deste xeito, para modificar ou acceder a un atributo usarase os métodos públicos da clase. A estes métodos chámselles **métodos accesores** e son os **getter**, para obter o valor dun atributo, e os **setter**, para establecer o valor do mesmo. Isto permite seguir o que se coñece como **principio de encapsulación**. Vexámolo cun exemplo:

```
class Triangulo {
 private double base;
 private double altura;
 public double getBase() {
   return(base);
 }
 public double getAltura() {
   return(altura);
 }
 public void setBase(double bas) {
   base = bas;
 }
 public void setAltura(double al) {
   altura = al;
 }
}
```

Declaración de métodos

Xa se veu tamén como se declaran os métodos en Java:

```
 public double calculaArea (int lado) {
           // Corpo do método
}
```

A declaración dun método pode ter:

- Un **modificador de acceso** (public, private, etc.)
- **Tipo de dato** que devolve o método, en caso de que non devolva nada utilízase a palabra reservada void
- **Nome** do método, que se escribe normalmente en minúsculas se é unha palabra simple e coa primeira letra en maiúsculas se é unha plabra composta. Normalmente os métodos noméanse normalmente cun verbo, xa que indican unha acción ou comportamento dun obxecto.
- **Parámetros**, que son os valores de entrada do método

Ao nome do método xunto cos seus parámetros chámaselle a **sinatura do método**. No exemplo anterior a sinatura do método área é a seguinte:

```
calculaArea (int lado)
```

Para poñerlle nome a un método podemos usar calquera identificador válido. Por exemplo:

```
calculaArea
setX
getY
estaDefinido
```

Un método pode chamarse igual que outro pero as súas sinaturas deben ser distintas. A isto chámaselle **sobrecarga de métodos**. Por exemplo, o método suma pode estar sobrecargado se se suman números de tipo int ou de tipo float:

```
int suma (int a, int b)
float suma (float a, float b)
```

Nunha clase non pode haber dous métodos coa mesma sinatura, aínda que o valor que devolvan sexa distinto.

Construtores

O construtor dunha clase é un método especial que cando se invoca permite crear instancias desa clase, é dicir, obxectos. O construtor debe ter o mesmo nome que a clase e non devolve ningún tipo de dato. Por exemplo:

```
public Bicicleta(int velocidadeIni, int marchaIni){
      marcha = marchaIni;
      velocidade = velocidadeIni;
}
```

O exemplo anterior amosa un construtor válido para a clase Bicicleta. Este construtor, ademais, inicializa dous atribtos da clase a partir dos valores dos argumentos. Para crear obxectos da clase Bicicleta escribiriamos, por exemplo:

```
Bicicleta unhaBicicleta= new Bicicleta(30, 2);
 Unha clase pode ter máis dun construtor
```

Cos construtores acontece o mesmo que cos métodos (de feito, un construtor é un método), isto é, que poden ter o mesmo nome pero distinta sinatura. Por exemplo:

```
  // Construtor sen argumentos
  Bicicleta unhaBicicleta= new Bicicleta();
  // Construtor con argumentos
  Bicicleta unhaBicicleta= new Bicicleta(30, 2);
```

Non é obrigatorio declarar un construtor para unha clase. Se non se fai úsase automaticamente o construtor da superclase. Se a superclase non ten construtor o compilador automaticamente proporciona un construtor sen argumentos. Podemos utilizar os modificadores de acceso para autorizar ou denegar acceso aos construtores, igual que se fosen métodos, aínda que normalmente os construtores teñen que poder ser accedidos dende outras clases para poder instanciar obxectos desa clase.

Paso de parámetros

Como xa se comentou anteriormente, os métodos e os construtores poden ter parámetros:

```
public int suma (int x, int y) {
  return (x + y);
}
```

Así, o método suma ten dous parámetros enteiros e devolve a súa suma. Os parámetros utilízanse no corpo do método. Durante a execución do mesmo o método tomará os valores dos argumentos que se lle pasan dende o programa que o chama. Cando se invoca un método, os argumentos utilizados deben coincidir en tipo e orde coa declaración de parámetros. Por exemplo, para o método suma anterior os argumentos teñen que ser de tipo enteiro:

```
resultado1 = suma (4, 8);
resultado2 = suma (resultado1, 100);
```

Polo tanto, temos que distinguir entre parámetros e argumentos:

- Parámetros: refírese á lista de variables dentro da declaración dun método
- Argumentos: son os valores reais que se pasan no momento de invocación do método

Podemos usar calquera tipo de parámetros dentro dun método ou construtor, isto inclúe os tipos primitivos, como double, int, etc. ou tipos referencia, tales como obxectos ou arrays. O seguinte exemplo amosa un método que ten como parámetro un array de cadeas:

```
public void almacena (String [] arrayCadeas) {
   // Corpo do método
}
```

Non podemos declarar dous parámetros con igual nome dentro do mesmo método ou construtor. O nome dun parámetro tampouco pode coincidir co nome dunha variable local do método. Por exemplo, o seguinte código dará un erro de compilación:

```
public class Proba2 {
   public void proba(int i) {
     float i; // Fallará
   }
}
```

Con todo, o nome dun parámetro pode coincidir co nome dun atributo da clase, aínda que non se recomenda.

Paso de parámetros por valor

Pasar un parámetro por valor significa que calquera cambio que se faga ao valor do parámetro dentro do método será válido unicamente dentro dese método. Cando o método remata o parámetro deixa de existir (de feito, o que se pasa é unha copia do parámetro) e, polo tanto, as modificacións feitas nel, non se conservan. En Java **os parámetros pásanse sempre por valor**. Vexamos o seguinte exemplo:

```
public class PasoPorValor {
   public static void main(String[] args) {
       int x = 3;
       // Invoca ao método passMetodo con x como argumento
       passMethod(x);
       // Imprime por pantalla o valor de x para ver se cambiou
       System.out.println("Despois de invocar a passMetodo, x = " + x);
   }
   // O método passMetodo
   public static void passMethod(int p) {
       p = 10;
   }
}
```

A saída do programa será:

```
Despois de invocar a passMetodo, x = 3
```

Paso de obxectos como parámetros

Podemos usar un obxecto como argumento dun método. **Tamén se pasará por valor**, pero se modificamos o seu estado, é dicir, os seus atributos, estas modificacións permanecerán incluso despois de que o método remate. Isto é así porque o que se pasa como argumento é unha referencia ao obxecto (un punteiro), e non o obxecto en si mesmo. Polo tanto, tanto o programa que chama ao método como o método chamado teñen unha copia idéntica da referencia ao obxecto e ambos os dous se referirán a exactamente o mesmo obxecto (non a unha copia do obxecto). Así pois, calquera modificación dentro do método afectará ao obxecto fóra do método.

Obxectos

Creación de obxectos

As clases proporcionan un molde a partir do cal se poden crear obxectos. Podemos ter unha clase Alumno pero se nunha aula temos 40 alumnos matriculados haberá que instanciar 40 veces a clase Alumno, por cada un deles, co seu nome concreto, DNI, etc.

Declaración de obxectos

A declaración dun obxecto (en realidade teríamos que dicir a declaración dunha variable que apunta a un obxecto) faise igual que con calquera outra variable, pero o tipo de dato da variable será unha clase. Por exemplo:

```
// Variable de tipo primitivo (int)
int x;
// Variable que referencia un obxecto de tipo Punto
```

```
Punto meuPunto;
```

A variable meuPunto é unha referencia a un obxecto de tipo Punto. Terá un valor indeterminado até que se cree realmente o obxecto, instanciando a clase Punto.

Instanciación de obxectos

No exemplo anterior, a declaración da variable meuPunto non crea un obxecto da clase Punto. Para elo é necesario utilizar o operador `new` tal e como se amosa no seguinte exemplo:

```
Punto meuPunto = new Punto(23, 94);
```

Incluso podemos non declarar a variable de tipo obxecto para acceder a un atributo ou método do mesmo. No seguinte exemplo, temos unha clase Rectangulo cun atributo altura e accedemos a el sen crear unha variable de tipo Rectangulo (aínda que si se crea a referencia ao chamar a new Rectangulo())

```
int altura = new Rectangulo().altura;
```

Inicialización de obxectos

Inicializar un obxecto implica dar valores aos seus atributos cando o creamos. Isto realízase normalmente a través dun construtor. Se a clase Punto do exemplo anterior ten un construtor con dous argumentos de tipo enteiro teríamos o seguinte código:

```
 public class Punto {
   private int x = 0;
   private int y = 0;
   //construtor
   public Punto(int a, int b) {
     x = a;
     y = b;
   }
}
```

Como xa se comentou unha clase pode ter varios construtores pero con diferentes sinaturas. A clase Punto podería ter un construtor por defecto, sen parámetros, e outro para inicializar os seus atributos, como o anterior.

Uso de obxectos

Unha vez creados os obxectos podemos traballar con eles, invocando os seus métodos, e cambiar o seu estado, a través da modificación dos seus atributos.

Acceso aos atributos dun obxecto

Accedemos aos atributos dun obxecto co operador `.` Para elo, escribimo o nome do obxecto seguido do operador de acceso (o punto) e o nome do atributo. Por exemplo:

```
System.out.println("Largo: " + rectUn.largo + " Alto: " + rectUn.alto);
```

Obxectos diferentes pertencentes á mesma clase teñen cada un unha copia das variables membro (atributos) da clase. Polo tanto cada obxecto de tipo Rectangulo terá as súas propias variables. No exemplo, largo e alto. Co operador de acceso accedemos a un atributo dun obxecto concreto (o obxecto que aparece precedendo ao nome do atributo).

Invocación de métodos

Para chamar aos métodos dun obxecto empregamos a mesma sintaxe que para acceder aos seus atributos. Novamente, hai que recordar que accedemos a un método cun obxecto concreto (o obxecto que aparece precedendo ao nome do método):

```
// Método con dous argumentos
  referenciaObxecto.nomeMetodo(arg1, arg2);
// Método sen argumentos
  referenciaObxecto.nomeMetodo();
```

O recolector de lixo

No capítulo I xa se falou do recolector de lixo como unha das características importante da linguxe. Java permite crear tantos obxectos como se queira sen ter que preocuparse de destruílos unha vez que xa non se precisan. Existe un proceso chamado **recolector de lixo** (garbage collector) encargado de facelo. Con todo, tamén se pode eliminar explicitamente a referencia dunha variable (momento no que poderá ser eliminada) asignándoa ao valor `null`. Noutras linguaxes de programación (como na linguaxe C) este traballo ten que facelo o programador pero en Java o garbage collector periodicamente libera memoria utilizada por obxectos que xa non están referenciados. De feito, quen o activa e a propia JVM (*Java Virtual Machine*).

Outra volta ás clases

Veremos a continuación outros aspectos relacionados coas clases en Java.

Devolución de valores dende métodos

Cando invocamos a un método dende un programa a execución dese método remata cando acontece algunha das seguintes situacións:

1. Complétanse todas as sentenzas dentro do método
2. Execútase unha sentenza return
3. Lánzase unha excepción (verémolo noutra unidade)

Unha vez que o método remata o fluxo de execución volve ao programa principal dende o que se invocou o método.

Os métodos poden devolver valores, como vimos cando falamos da súa declaración. O valor que devolve un método especifícase, polo tanto, na propia declaración do método. Para devolver o valor usamos a sentenza **`return`** dentro do corpo do método.

Os métodos declarados coa palabra void non devolven ningún valor polo que non é necesario usar a sentenza return neles. Se devolvemos un valor dende un método void obteremos un erro de compilación. Analogamente, calquera método que non sexa declarado como void debe ter unha sentenza return co correspondente valor de retorno. Non podemos devolver un boolean se declaramos o método para que devolva un int, é dicir, o tipo de dato devolto debe coincidir co tipo declarado no método:

```
public int getArea() {
  return largo * alto; // As variables largo e alto debes ser int
}
```

O método getArea() anterior devolve un int, é dicir, un tipo primitivo, pero un método pode tamén devolver un obxecto, é dicir, un tipo referencia:

```
public Bicicleta maisRapida(Bicicleta b1, Bicicleta b2) {
  Bicicleta aBiciMaisRapida;
  // Código para calcular a bici máis rápida
  return aBiciMaisRapida;
}
```

A palabra reservada this

Dentro dun método ou construtor a palabra reservada this é unha referencia ao obxecto actual, é dicir, ao obxecto que está chamando ao método ou construtor. Con this podemos acceder a atributos e a métodos dun obxecto concreto. Adoita a usarse this para distinguir os parámetros dun método ou construtor dos atributos dese obxecto cando estes se chaman igual. Por exemplo:

```
public class Punto {
           public int x = 0;
           public int y = 0;

           //construtor
           public Punto (int x, int y) {
              this.x = x;
              this.y = y;
           }
  }
```

Non anterior exemplo os parámetros do construtor e os atributos do obxecto chámanse igual, polo que temos que usar a palabra reservada this para distinguilos. Para referirnos ao atributo x da clase Punto o construtor debe usar usar this.x. Tamén podemos usar this nos construtores tal e como se amosa no seguinte exemplo para inicializar atributos:

```
public class Rectangulo {
            private int x, y;
            private int largo alto;
            public Rectangulo() {
                this(0, 0, 0, 0);
            }
}
```

Modificadores de acceso

Unha clase ten métodos e atributos. Os modificadores de acceso determinan como outras clases poden usar eses atributos ou invocar aos métodos desa clase. Hai catro posibles modificadores de acceso: **public**, **protected**, **sen modificador** e **private**. Na seguinte táboa vemos que implicacións ten cada un destes modificadores:

Modificador	Clase	Paquete	Subclase	Resto
public	SI	SI	SI	SI
protected	SI	SI	SI	NON
Sen modificador	SI	SI	NON	NON
private	SI	NON	NON	NON

A primeira columna (clase) indica se a propia clase ten acceso aos seus métodos e atributos. Vemos que isto sempre é así, como é lóxico.

A segunda columna (paquete) indica se as clases que están no mesmo paquete que a clase teñen acceso aos atributos e métodos desa clase.

A terceira columna (subclase) indica se as subclases que herdan da clase e están noutros paquetes teñen acceso aos atributos e métodos da superclase.

A última columna (resto) indica se calquera clase, independentemente de que estea ou non no mesmo paquete ou sexa unha subclase, teñen acceso aos atributos e métodos desa clase.

Os modificadores de acceso aféctannos cando usamos código de terceiros (por exemplo, as clases da API de Java) e cando escribimos unha clase e temos que decidir que nivel de acceso queremos que se teña sobre os seus métodos e atributos:

- O modificador **private** especifica que os métodos e atributos só poden ser accedidos dentro da propia clase
- O modificador **protected** especifica que os métodos e atributos só poden ser accedidos polas clases que estean dentro do mesmo paquete, ou polas subclases independentemente do paquete onde estean.
- O modificador **public** especifica que os métodos e atributos poden ser accedidos por calquera clase
- Sen non usamos modificador de acceso só poden acceder aos métodos e atributos as clases dentro do mesmo paquete e as subclases noutros paquetes

Á hora de escoller un modificador de acceso aconséllase:

- Usar modificadores de acceso o máis restritivos posibles (private)
- Non usar atributos con modificadores de acceso públicos, para manter a encapsulación

Atributos e métodos de clase (static)

Atributos

Cando creamos varios obxectos a partir dunha clase cada un deses obxectos ten unha copia dos atributos da clase pero cuns valores determinados.Por exemplo, no caso da clase Alumno cada obxecto alumno ten un nome e un curso no que está matriculado, é dicir, ese nome e ese curso son **variables de instancia** porque pertencen ao ese obxecto concreto. Nalgunhas ocasións pode interesarnos ter variables (atributos) que sexan comúns a todos os obxectos dunha clase (por exemplo, para levar un contador de cantas instancias existen dunha clase). Para elo usamos a palabra reservada `static`. Os atributos que teñan a palabra `static` diante son atributos ou variables de clase. Teñen un valor para a clase pero non para cada un dos obxectos (instancias) desa clase. O acceso ás variables de clase realízase co operador de acceso habitual (o **.**) pero poñendo o nome da clase en lugar do nome da instancia. Por exemplo:

```
class Exemplo {
```

```
  static int contador = 0;
  int x;
   // Cada vez que se crea un obxecto, incrementase o contador global asociado a clase.
  public Exemplo() {
    contador++;
  }
}

class ExemploDemo {
  public static void main(String[] args) {
    Exemplo ej1 = new Exemplo();
    Exemplo [] vej = new Exemplo[100];
    for (int i = 0; i < 100; i++)
      vej[i] = new Exemplo();
    int numTotal = Exemplo.contador;
    System.out.println("Número de obxectos creados: " + numTotal);
 }
}
```

As variables de tipo static son, en certos aspectos, similares ás variables globais dalgunhas linguaxes de programación. A linguaxe Java non ten variables globais, pero si static, ás que se pode ter acceso dende calquera instancia dunha clase.

Métodos

Tamén aos métodos pódeselles aplicar o modificador static nas súas declaracións. Os métodos static invócanse co nome da clase, sen necesidade de crear unha instancia da clase para elo (igual que para os atributos de clase:

```
// Sintaxe para invocar a un método de clase
NomeClase.nomeMetodo(argumentos)
```

Os métodos static úsanse normalmente para acceder a atributos de clase, é dicir, atributos static. Por exemplo, podemos engadir un método static á clase Bicicleta para acceder ao atributo static numeroDeBicicletas:

```
public static int dimeNumeroDeBicicletas () {
  return numeroDeBicicletas;
}
```

Normalmente, os métodos estáticos empréganse para levar a cabo accións comuns a todas as instancias dunha clase. Os métodos estáticos incorporan a palabra reservada static como modificador do método antes do tipo de retorno e despois do modificador de acceso:

```
public static int suma(int x, int y){
}
```

Un exemplo de método static é o método main(), porque a aplicación debe acceder a el para executarse, antes de que se cree calquera instancia.

Inicialización de atributos

As variables de instancias, é dicir, os atributos dos obxectos, inicialízanse cos construtores. Pola contra, un atributo dunha clase pode inicializarse na súa declaración, como podemos ver no seguinte exemplo:

```
public class Almacen {
  //Inicializa capacidade a 10
  public static int capacidade = 10;

  //Inicializa cheo a false
  private boolean cheo = false;
}
```

Clases encadeadas (nested)

Java permite definir unha clase dentro doutra clase. A estas clases chámaselles clases **encadeadas** ou aniñadas. Por exemplo:

```
class ClaseExterna {
  ...
  class ClaseEncadeada {
  ...
  }
```

```
}
```

As clases encadeadas poden acceder a atributos privados da clase externa. Igual que os métodos e os atributos unha clase encadeada pode ser static. Se a clase encadeada non é static chámaselle clase interna (**inner class** en inglés). Vexamos un exemplo:

```
class ClaseExterna {

   static class ClaseEncadeadaEstatica {
   }

   class InnerClass {
   }
}
```

Os principais motivos para usar clases encadeadas son:

- Permiten agrupar clases que se usan nun ámbito concreto
- Incrementan a encapsulación
- Permiten escribir código máis lexible e polo tanto máis doado de manter

Tipos enum

O tipo enum representa unha enumeración de constantes. Por exemplo os valores NORTE, SUR, LESTE e OESTE, ou os días da semana, que non varían. Como son constantes van en maiúsculas e declárase como segue:

```
public enum Dias {
  LUNS, MARTES, MERCORES, XOVES, VENRES, SABADO, DOMINGO
}
```

Podemos usar o tipo enum cando queiramos representar un conxunto finito de constantes. Por exemplo:

```
public enum Dias {
                LUNS, MARTES, MERCORES,
                XOVES, VENRES, SABADO, DOMINGO
        }

public class EnumDemo {
        Dias meusDias;
        public EnumDemo(Dias meusDias) {
                this.meusDias = meusDias;
        }
        public void dimeDescricion() {
                switch (meusDias) {
                        case LUNS: System.out.println("Os luns son complicados");
                                        break;
                        case VENRES: System.out.println("Os venres son fantÃ¡sticos");
                                        break;
                        case SABADO:
                        case DOMINGO: System.out.println("Fin de semana: o mellor!");
                                        break;
                        default:         System.out.println("Entre semana, nin fu, nin fa");
                                        break;
                }
        }

        public static void main(String[] args) {
                EnumDemo primeiroDia = new EnumDemo(Dias.LUNS);
                primeiroDia.dimeDescricion();
                EnumDemo terceiroDia = new EnumDemo(Dias.MERCORES);
                terceiroDia.dimeDescricion();
                EnumDemo quintoDia = new EnumDemo(Dias.VENRES);
                quintoDia.dimeDescricion();
                EnumDemo sextoDia = new EnumDemo(Dias.SABADO);
                sextoDia.dimeDescricion();
                EnumDemo ultimoDia = new EnumDemo(Dias.DOMINGO);
                ultimoDia.dimeDescricion();
        }
}
```

Anotacións

As anotacións proporcionan información sobre un programa pero non forman parte do código. Utilízanse para xerar documentación automática, mediante a ferramenta javadoc, ou para dar información ao compilador javac facilitando a depuración do código.