Definición de Clases

Java é unha linguaxe puramente orientada a obxecto. Todo código debe pertencer a algunha clase. Para crear unha clase en Java se emprega a palabra reservada **class:**

```
public class NomeClase {
   // Definición de atributos da clase (definición de variables)
   // Definición de métodos da clase (definición de funcións)
}
```

En Java normalmente se debe codificar unha única clase por cada ficheiro de código fonte, e ese ficheiro debe chamarse ***NomeClase.java***

Definición de Métodos

Mentras que os atributos son variables que almacenarán información sobre o obxecto instanciado, os métodos son funcións que definen o comportamento dos mesmos (as accións que os obxectos serán capaces de levar a cabo). Un método dunha clase se define do seguinte xeito:

```
tipo_de_datos_devolto  identificador_do_método (definición de variables separadas por comas) {
        corpo do método implantando o algoritmo
        return resultado;
}
```

Vexamos un exemplo concreto (buscar en google **Java Date**, para examinar a documentación da clase):

```
public class Persoa {
        private String nome;
        private Date data_nacemento;

        public Persoa(String n,Date dn) {
                nome=n;
                data_nacemento=dn;
        }

        public int calculaDíasDeVida() {
                Date now;
                now=new Date();
                return ((now.getTime()-data_nacemento.getTime())/1000)/86400;
        }
}
```

nome e **data_nacemento** son atributos da clase.

A variable ***now*** so existe dentro do corpo do método ***calculaDiasDeVida***, e é unha variable local ao método.

As variables ***n*** e ***dn*** son parámetros do método Persoa (reciben a información necesaria para o método) e son variables locais ao método Persoa (so existen e se poden utilizar dentro do método persoa). Esta clase a poderíamos utilizar para crear (instanciar) obxectos, e logo facer uso deles a través dos seus métodos (interface):

```
// Creamos (instanciamos) un obxecto da clase Persoa, unha Persoa "concreta" e almacenamos a súa referencia en p
// Os valores almacenados nos atributos ("Luis" e a data 3-11-1993), definen o estado do obxecto instanciado.
Persoa p=new Persoa("Luis",new Date(1993,11,03));
int nd=p.calculaDiasDeVida();  // Almacena en nd os días de vida de "Luis"..
```

O Operador .

Como xa vimos anteriormente, si temos unha variable que identifica a posición de memoria onde almacenamos a referencia dun obxecto:

```
Persoa p=new Persoa(“Luis”,new Date(1993,11,03));
```

Para acceder aos métodos e atributos do obxecto se utiliza o operador **.** :

```
p.calculaDiasDeVida();
```

Instanciación: Constructores e Destructores

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.

Vexamos un exemplo :

```
	public class Persoa {

	String nome;
	Date data_nacemento;

	public int calculaDíasDeVida() {
		Date now;
		now=new Date();
		return ((now.getTime()-data_nacemento.getTime())/1000)/86400;
	}
}
```

Dada a definición anterior, para crear un obxecto o único modo de facelo sería facendo uso do ***constructor por defecto,*** quedando os seus atributos cun valor inicial 0 si son numéricos ou coa dirección (referencia) nula si son obxectos:

```
Persoa p=new Persoa();  // Os atributos nome e data_nacemento terán a referencia nula
p.nome=”Luis”;          // Poño a referencia do obxecto “Luis” de tipo String no atributo nome
```

Si creamos nos un constructor, o constructor por defecto xa non é utilizable, se non que teremos que utilizar o constructor proporcionado:

```
public class Persoa {
	private String nome;
	private Date data_nacemento;

	public Persoa(String n,Date dn) {
		nome=n;
		data_nacemento=dn;
	}

	public int calculaDíasDeVida() {
		Date now;
		now=new Date();
		return ((now.getTime()-data_nacemento.getTime())/1000)/86400;
	}
}
```

Para crear un obxecto da clase Persoa (instanciación) tería que facer:

```
// Creamos (instanciamos) un obxecto da clase Persoa, unha Persoa "concreta" e almacenamos a súa referencia en p
// Os valores almacenados nos atributos ("Luis" e a data 3-11-1993), definen o estado do obxecto instanciado.
Persoa p=new Persoa("Luis",new Date(1993,11,03));
```

A instanciación mediante o constructor por defecto non é válida, xa que ao proporcionar nos un constructor, non se "xenera".

Métodos e Atributos Estáticos: Método main e clase System

Java é unha linguaxe "puramente" orientada a obxectos. Eso quere decir que todo o código Java ten que pertencer forzosamente a algunha clase. O problema é o seguinte:

Si todo código ten que estar dentro de unha clase, pero as clases non son mais que "definicións de tipos de obxectos" que non teñen funcionalidade ata que instanciamos un obxecto concreto..... ¿como creamos o primeiro obxecto?. Pois facendo uso de ***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.*

Para utilizar os métodos e atributos estáticos, como non é necesario instanciar un obxecto (xa que existen con independencia dos obxectos) ***se antepón ao nome do atributo ou método o identificador da clase*** (en lugar do identificador do obxecto).

Exemplos comúns de métodos estáticos son diferentes funcións matemáticas presentes na clase Math ou a clase Arrays, que proporcionan funcións matemáticas de uso xeral, e funcións de uso común sobre táboas respectivamente.

En Java, cando invocamos a execución dunha clase a JVM busca un método estático chamado **main** dentro de esa clase para iniciar a execución do programa. Ese método debe ser declarado do seguinte xeito:

```
public static void main(String[] args) {
	// Corpo do método
}
```

args e unha táboa (array) que almacenará os parámetros utilizados na chamada a aplicación Java. Por exemplo si ordenamos a JVM que inicie a execución da clase ***Otello.class*** escribiríamos a orde:

```
java Otello
```

A JVM buscaría o método main dentro da clase Otello e comezaría a execución. Nese caso o parámetro *args* de main tería 0 elementos. Si executamos a clase Otello.class do seguinte xeito:

```
java Otello xogador1 machine
```

O método main da clase Otello tería no seu parámetro *args* dous elementos. O primeiro elemento sería “xogador1”, e o segundo elemento “machine”.

Entrada por teclado e saida por pantalla básica.

A clase System conten dous atributos *estáticos* (***in*** e ***out***) que se cando se inicia a JVM se inicializan a obxectos que se relacionan coa entrada de datos estándar (teclado) e a saída de datos estándar respectivamente (pantalla).

Estes obxectos conteñen métodos que se poden utilizar de modo mais ou menos directo para visualizar información (a través do obxecto ao que apunta o atributo estático **out** de System) ou obter información dende teclado (a través do obxecto ao que apunta ***in***). Mentras que os métodos do obxecto ao que apunta **out** (da clase *PrintStream*) ofrecen a funcionalidade desexada a través de métodos como *print*, ou *println*, os métodos do obxecto ao que apunta ***in*** (da clase InputStream) son demasiado simples, sendo necesario recurrir a clases máis complexas como Scanner que ofrecen os métodos necesarios.

Saida de datos:

System.out.println(“Hola Mundo”);

Entrada de datos:

Scanner s=new Scanner(System.in);

String line=s.readLine();

Estructura dun programa Java

Un programa orientado a obxectos non é máis que un conxunto de obxectos que crean novos obxectos e invocan métodos. Un programa Java, non é máis que o deseño das clases, estando o comportamento do programa indicado polo algoritmo deseñado no método estático main invocado.

En Java cada clase *pública* debe residir nun ficheiro que se chame igual que a clase. Esas clases poden ter un método estático ***main***, ou non. A aplicación comezará no método estático main da clase que utilicemos cando invoquemos a JVM (Máquina Virtual Java) para a execución do programa:

java Programa

Programa, é un ficheiro .class que corresponde a unha clase Java compilada ao p-code da JVM

Compilación e Execución

Unha vez deseñadas as clases necesarias para a creación da nosa aplicación, é necesario compilalas a p-code mediante o compilador Java. Normalmente invocaremos o compilador sobre a clase java que contén o método ***main*** da nosa aplicación, e se irán compilando automáticamente as clases relacionadas.

javac Programa.java

Esto xenerará un ficheiro ***.class*** en p-code por cada clase da aplicación. Para executar o programa será necesario invocar a execución da JVM co nome da clase que ten o ***main*** da nosa aplicación como argumento, sen indicar a extensión:

java Programa

Se precisa dun JDK ben instalado e das variables de entorno CLASSHPATH e PATH cos valores correctos.

Espacios de Nomes e Packages

Un programa Java está composto de varios obxectos que pertencen a clases que deseñamos para a creación da aplicación.

Os identificadores que utilizamos para os nomes das clases deben ser significativos, e suxerir do modo máis claro posible a funcionalidade ofrecida. Deste xeito si sabemos que unha clase se chama ***Math*** podemos sospeitar que se trata de unha clase con métodos matemáticos, ou si unha clase se chama ***Client*** a clase contén métodos relacionados con clientes.....

Ó mesmo tempo, habitualmente facemos uso de numerosas clases xa deseñadas que nos facilitan a programación, xa que dispoñen de métodos útiles e comúns a moitos programas. Estas clases poden ser tanto clases subministradas co propio Java (Libraría Estándar de Clases Java), clases de terceiros ou clases deseñadas por nós en outras aplicacións que nos resulten útiles na aplicación actual.

O problema, e que é fácil que varias clases se chamen do mesmo modo ofrecendo funcionalidades distintas, e que queiramos facer uso de todas elas na mesma aplicación.

A solución ofrecida por Java (e outras linguaxes de programación) consiste no uso de *espazos de nomes* en Java implantados mediante a palabra reservada ***package.***

Java organiza as clases en “packages”. Cando non se indica nada, as clases pertencen ao “package por defecto”, e deben estar situadas nunha carpeta raíz das carpetas de clases (as carpetas de clases son as indicadas no CLASSPATH).

Para utilizar calqueira clase que non pertenza ao package por defecto é necesario importala mediante a palabra reservada ***import***.

Para evitar colisións nos nomes das clases, os nomes dos packages deben ser únicos, polo que normalmente se utilizan URL (os desenvolvedores de software habitualmente teñen dominios, ou traballan para compañías que os teñen) co dominio de primeiro nivel ao principio:

package com.iesrodeira.xavi;

Esto indicaría que a clase deseñada a continuación pertence ao package “com.iesrodeira.xavi”, e debe estar situada a partir dunha carpeta raiz de clases (as indicadas no CLASSPATH) na carpeta

com->iesrodeira->xavitaboada. A partir de ahí temos garantido que os nomes das clases serán únicos, xa que en caso de existir outra clase co mesmo nome se atopará nun *package* diferente.

Dentro do package podemos seguir agrupando por funcionalidade:

package com.iesrodeira.xavi.graph;

As clases deste package, serán distintas das clases en ***package.com.iesrodeira.maria.grah***, aínda que se chamen igual.

Para utilizar as clases dun package é necesario importalas ou antepoñer o nome completo do package ao nome da clase. Imaxinemos que dentro do package *com.iesrodeira.xavi.graph* temos desenvolta unha clase Triangle. Para instanciar un obxecto, poderíamos poñer ao principio do ficheiro:

import com.iesrodeira.xavi.graph.Triangle;

e logo poderíamos instanciar un obxecto mediante:

Triangle t=new Triangle().

Alternativamente poderíamos facer:

com.iesrodeira.xavi.graph.Triangle t= new com.iesrodeira.xavi.grap.Triangle();

Para cargar a clase, a JVM buscaría unha definición de Triangle (Triangle.class) a partir de todas as carpetas indicadas no classpath na ruta *com->xavi->graph*.

NOTA: ***Normalemente a carpeta actual sempre forma parte do CLASSPATH***

Herdanza

Unha vez definida unha clase, é posible crear novas clases derivadas que heredan os atributos e métodos, e permiten a agregación de novos métodos e a modificación dos existentes.

Se explicará con máis detalle nun documento dedicado ao tema.

```
public class  Animal {
        protected String name;

        public void talk() {
                System.out.println("I don't talk");
        }
}
class Bird extends Animal {
        public Bird() {
                name="Bird";
        }

        public void fly() {
                System.out.println("I'm flying");
        }
}

Bird b=new Bird();          // Instanciamos un obxecto Bird, que é un Animal
b.talk();                   // Chamamos o método heredado de Animal, visualiza "I don't talk"
b.fly();                    // Chamamos ao método fly da clase Bird, visualizar "I'm flying"
```

Protección e Encapsulación: friendly, public, protected e private

Un dos obxectivos da programación orientada a obxectos e conseguir que a funcionalidade e correcto funcionamento dos obxectos dunha clase sexa seguro sen importar as accións que se podan levar a cabo fora delas.

Para acadar isto é necesario impedir que se modifiquen os valores dos atributos de xeito inapropiado, polo que se debe impedir o acceso mediante un sistema de protección de acceso reglado por catro palabras reservadas:

- **friendly**: É o comportamento por defecto si non se especifica. So se pode acceder dende clases do mesmo package.
- **public**: O acceso é público. Pódese acceder libremente.
- **protected**: So se pode acceder dende a propia clase, ou dende clases derivadas.
- **private**: So se pode acceder dende a propia clase.

Pódense utilizar os mesmos modificadores para protexer o acceso ás clases (as clases **private** so teñen sentido si se declaran dentro de outra clase):

```
class Test {
        int friendly_attribute;
        public int public_attribute;
        private int private_attribute;
        protected int protected_attribute;

        int friendly_method() {}
        public int public_method() {}
        private int private_method() {}
        protected int protected_method() {}
}

Test t=new Test();
t.friendly_atrribute=1;     // Correcto si estamos escribindo dentro do package por defecto
t.public_attribute=1;       // Correcto
t.private_attribute=1;      // ERROR, salvo que a sentencia esta dentro dun método da clase Test
t.protected_attribute=1;    // ERROR, salvo que a sentencia estea dentro dun método da clase Test ou dentro dunha clase  derivada.
```