Programación II

Carla TARAMASCO

Profesora e Investigadora

DECOM & CNRS

mail : alumnos_uv@yahoo.cl

16 avril 2013

Clases, Objetos y Metodos

- Clases y objetos
  - Definir Clases
  - Constructores
  - Definición de objetos
- Métodos
  - Reglas de escritura de métodos
  - Atributos y metodos de clase
  - Sobrecarga de métodos
  - Intercambio de información con métodos
  - Recursividad
- Aplicaciones Java
  - Multiples Clases
  - Encapsulamiento
  - Paquetes
  - Ejercicio Clases, Objetos y Metodos

Recordatorio : Definir Clases

Definición de Clases

La definición general de clases que hemos usado :

```
public class Auto{
        // variables
        // metodos
}
```

Recordatorio : Definir Clases

Variables de instancia

```
public class Auto{

        private String color;
        String marca;
        boolean estado;}
```

La variable privada no será accesible fuera de la clase

Variables de clase

```
public class Auto{

        private String color;
        String marca; // variable de instancia
        static int tipo; // variable de clase
}
```

Las variables de clase son globales para todas las instancias de clase. Para definir una variable de clase se usa la palabra clave **static**

Recordatorio : Definir Clases

Variables de instancia

```
public class Auto{

        private String color;
        String marca;
        boolean estado;}
```

La variable privada no será accesible fuera de la clase

Variables de clase

```
public class Auto{

        private String color;
        String marca; // variable de instancia
        static int tipo; // variable de clase
}
```

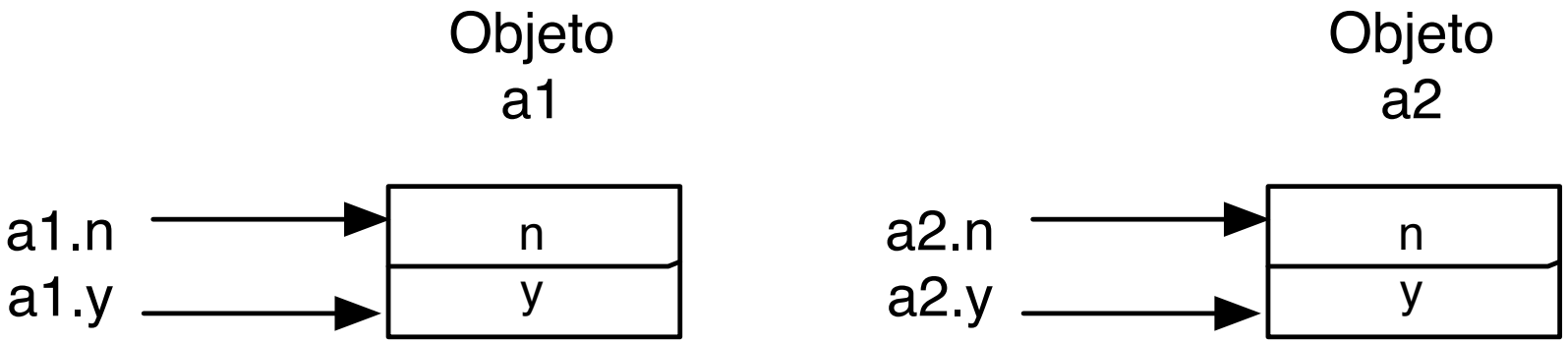
Las variables de clase son globales para todas las instancias de clase. Para definir una variable de clase se usa la palabra clave **static**

Recordatorio : Definir Clases

Variables

```
public class A{
        int n;
        float y;
        public static void main(String arg[]){
                A a1=new A(), a2=new A();}}
```

Variables de Instancia

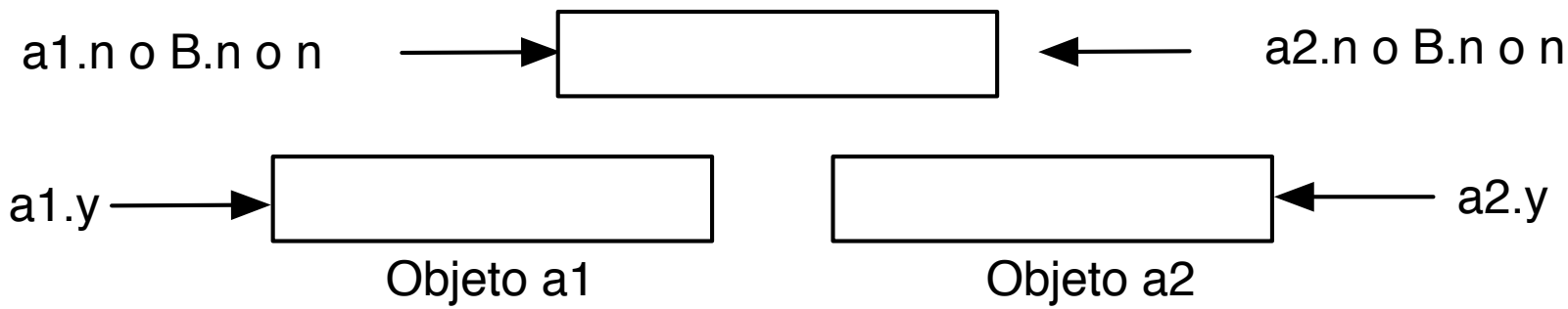


Recordatorio : Definir Clases

Variables

```
public class A{
        static int n;
        float y;
        public static void main(String arg[]){
                A a1=new A(), a2=new A();}}
```

Variables de Clase



Recordatorio : Definir Clases

Constantes

```
public class Auto{

        private String color;
        String marca; // variable de instancia
        static int tipo; // variable de clase
        final int DER=1; // constante
}
```

Las constantes son útiles para definir valores comunes a todos los métodos de un objeto.

Recordatorio : Clases, objetos y métodos

Cual es la diferencia entre :

- Auto a ;
- int a ;

La primera no reserva un espacio en memoria para un objeto del tipo Auto sino que una referencia a un objeto del tipo Auto. El lugar de la memoria sera asignado al realizar *new Auto() ;*. Esto ultimo crea un lugar de para el objeto de tipo Auto y entrega como resultado su referencia. Por ende, *a=new Auto() ;* crea un objeto del tipo Auto y pone su referencia en a.

Recordatorio : Definir Métodos

Métodos

Si el método es publico será accesible de cualquier programa. Si el método no retorna valor se debe usar **void**.

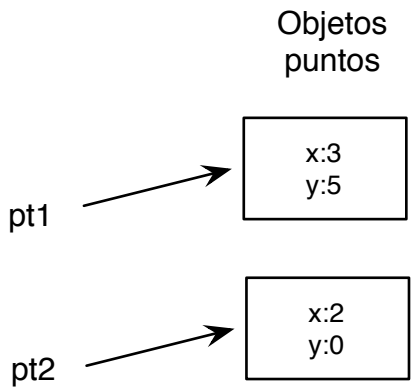
```
public class Auto{

        String color;
        String marca;
        boolean estado;

        public void startMotor(){
                if (estado== true)
                        System.out.println("Auto encendido");
                else {
                        estado=true;
                        System.out.println("Ok se encendio");}}}
```

De la misma manera que tenemos variables de clase y de instancia existen métodos de clase y de instancia.
Para definir métodos de clase se debe usar la palabra **static** (al igual que en las variables de clase, los métodos de clase están disponibles para cualquier instancia de la clase).

Recordatorio : Clases, objetos y métodos

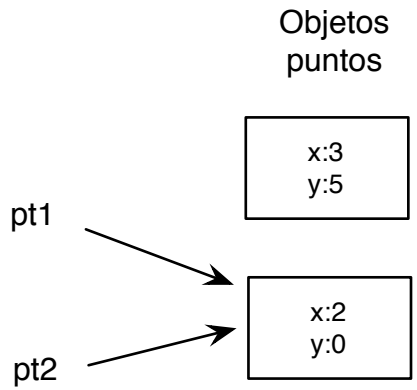
Supongamos que tenemos una clase Punto con dos instancias *Punto a,b ;*.
Le asignamos los valores (3,5) al punto 1 y (2,0) al punto 2 (*pt1.x=3 ;pt1.y=5 ;*).

Referencias a objetos



Recordatorio : Clases, objetos y métodos

Que pasa si hacemos : *pt1=pt2 ;* ?.



Lo que ocurre es que copia en pt1 la referencia al contenido de pt2, así pt1 y pt2 referencian al mismo objeto y no a dos objetos que tienen el mismo valor. En este caso Java con el recolector de basura libera el espacio en memoria del otro objeto.

Paso de argumentos a los métodos

Cuando llama a un método con parámetros de objetos, las variables que pasa al cuerpo del método lo hacen por *referencias*, lo que significa que las instrucciones aplicadas a esos objetos dentro del método afectara los objetos originales también (esto incluye arreglos). Recuerde que los tipos de dato primitivos son pasados por *valor*, por lo cual, si fueron modificados al interior de un método, esta modificación no se mantiene de un método a otro.

```
public class PasoRef{
        int unoaCero(int [] arg){
                int cont=0;
                for (int i=0;i<arg.length;i++){
                        if(arg[i]==1){
                                cont++;arg[i]=0;
                        }}return cont;}
public static void main (String arg[]){
        int arr[]={1,3,1,1,1,3,6};
        PasoRef test = new PasoRef();
        int numUno;
        System.out.println("Valores del arreglo : ");
        for (int i=0;i<arr.length;i++){
                System.out.print(arr[i]+ "  ");}
        System.out.println("");
        numUno=test.unoaCero(arr);
        System.out.println("Cantidad de 1 = " + numUno);
        System.out.println("Valores del arreglo : ");
        for (int i=0;i<arr.length;i++){
                System.out.print(arr[i]+"  ");}
        System.out.println("");}}
```

Recordatorio : Intercambio de información entre los métodos

- Por valor : el método recibe una copia del valor del argumento ; el método trabaja sobre esa copia que podrá modificar sin que esto tenga incidencia al valor efectivo del argumento.
- Por referencia : el método recibe la referencia del argumento con la cual trabaja directamente, por lo cual, el método podrá modificar el valor efectivo del argumento.

Autoreferencia : La palabra clave this

Para hacer una referencia a la instancia actual de una clase usamos la palabra clave this. Así, la palabra clave *this* se usa para referirse al objeto actual o a las variables de instancia de este objeto. No se usa this en los metodos declarados como *static*.

```
t=this.x;
this.miMetodo();


t=x;
miMetodo();
```

Se puede omitir la palabra clave this para las variables de instancia, esto depende de la existencia o no de variables con el mismo nombre en el ámbito local.

Nombre de variables y la palabra clave this

Java busca la definición de las variables primero en el ámbito actual, después en el exterior hasta la definición del método actual. Si esa variable no es local, entonces java busca una definición de ella como instancia de la clase actual y por ultimo de una super clase. En estos casos se usa por ejemplo : *this.test* para referirse a la variable de instancia y solo *test* para referirse a la variable local.

```
public class Prueba{
        int test=10;
        void prinTest(){
                int test = 20;
                System.out.println("test = " + test );
                System.out.println("test = " + this.test );}
        public static void main(String arg[]){
                Prueba miObj=new Prueba();
                miObj.prinTest();}}
```

Sobrecarga : Métodos con el mismo nombre y argumentos diferentes

En java se pueden crear métodos con el mismo nombre pero con definiciones diferentes. No hay necesidad de métodos completamente diferentes que realizan en esencia lo mismo, simplemente se comportaran distinto en base a la entrada.
Al llamar a un método, si hay múltiples declaraciones de este, java hará coincidir su nombre, numero y tipo de argumentos (no considera el tipo de dato de retorno)para seleccionar que definición de método utilizar.
Si se quiere realizar múltiples declaraciones de métodos en la clase, lo único que se debe hacer es crear definiciones distintas todas con el mismo nombre, pero con diferentes listas de parámetros (es necesario que cada lista de parámetros sea única). En la sobrecarga de métodos java no diferencia una variable definida como *int a* o *final int a*.

Sobrecarga

```
import java.awt.Point;
public class Rectangulo{
        int x1=0;
        int x2=0;
        int y1=0;
        int y2=0;


Rectangulo consRect(int x1,int y1,int x2,int y2){
        this.x1=x1;
        this.y1=y1;
        this.x2=x2;
        this.y2=y2;
        return this;}
```

Sobrecarga

```
Rectangulo consRect(Point p1, Point p2){
        x1=p1.x;
        y1=p1.y;
        x2=p2.x;
        y2=p2.y;
        return this;}


Rectangulo consRect(Point p1, int a, int l){
        x1=p1.x;
        y1=p1.y;
        x2=(x1+a);
        y2=(y1+l);
        return this;}
```

Sobrecarga

```
void printRect(){
System.out.print("Mi Rectangulo " + x1 + ", "+y1 + ", "+x2 + ", "+y2);}


public static void main (String args []){
        Rectangulo rect = new Rectangulo();
        System.out.print("Llamar a constRect con las coordenadas : 25,25,50,50:");
        rect.consRect(25,25,50,50);
        rect.printRect();
        System.out.print("Llamar a constRect con los puntos (10,10) y (20,20):");
        rect.consRect(new Point(10,10),new Point(20,20));
        rect.printRect();
        System.out.print("Llamar a constRect con los puntos (10,10) y los valores 50,50 :");
        rect.consRect(new Point(10,10),50,50);
        rect.printRect();}
```

Programa con varias Clases

- Un archivo fuente por clase : para ejecutar un programa compuesto de varias clases en archivos separados es necesario compilar todas las clases y luego ejecutar la clase que tenga el método *main* (que probablemente llamaremos *public class Main()*). Las clases deben estar en la misma carpeta o darle la ruta para que pueda acceder a ellas.
- Un archivo fuente para todas las clases : para esto solo, una clase que será la que contenga el método *main* debe ser *public*. La maquina virtual accede a la clase publica y por ende, al método *main*. Las otras clases pueden estar dentro de la clase *public* o fuera de ella ya sea al inicio o al final del archivo.

Constructor

El constructor asigna memoria para el objeto y automatiza el proceso de inicialización de este. Un constructor será un método que

- tiene el mismo nombre de que la clase,
- puede tener argumentos
- no tiene un valor de retorno(return).

```
public class Punto{
        int x,y;
        public Punto (int abs, int ord){
                x=abs;
                y=ord;}}
```

Cuando la clase tiene un constructor debemos crear los objetos llamando al constructor. En este caso : *Punto a = Punto (1,2);*

Constructor - ejercicio

Crear una clase Punto. Crear 3 constructores

- uno que no reciba argumentos y que inicialice *x* e *y* en 0
- uno que defina *x=y=abs* y reciba como argumento (int abs)
- uno que reciba (int abs, int ord)

Cree un método para mostrar en pantalla y tres métodos para desplazar para desplazar el punto

- el primero recibe 2 enteros
- el segundo recibe 1 enteros
- el tercero recibe 1 short

Pruebe que pasa si llama al método desplazar con una variable declarada como byte.

Use esta clase Punto en el ejercicio anterior de construcción del rectángulo.

Constructor - ejercicio

Crear una clase Punto. Crear 3 constructores

- uno que no reciba argumentos y que inicialice *x* e *y* en 0
- uno que defina *x=y=abs* y reciba como argumento (int abs)
- uno que reciba (int abs, int ord)

Cree un método para mostrar en pantalla y tres métodos para desplazar para desplazar el punto

- el primero recibe 2 enteros
- el segundo recibe 1 enteros
- el tercero recibe 1 short

Pruebe que pasa si llama al método desplazar con una variable declarada como byte.
Use esta clase Punto en el ejercicio anterior de construcción del rectángulo.

Constructor - ejercicio

```
class PuntoT{
        int x,y;
        public PuntoT (int abs, int ord){
                x=abs;
                y=ord;}
        public PuntoT (){
                x=0;
                y=0;}
        public PuntoT (int abs){
                x=y=abs;}
        void desplazar(int dx, int dy){
                x+=dx;
                y+=dy;}
        void desplazar(int dx){
                x+=dx;
        }
        void desplazar(short dy){
                y+=dy;
        }
        public void mostrar(){
                System.out.println("punto de coordenadas : "+x+" " + y);}
}
```

Constructor - ejercicio

```
public class Rectangulos1{
        int x1=0;
        int x2=0;
        int y1=0;
        int y2=0;
        Rectangulos1 consRect(int x1,int y1,int x2,int y2){
                this.x1=x1;
                this.y1=y1;
                this.x2=x2;
                this.y2=y2;
                return this;}
        Rectangulos1 consRect(PuntoT p1, PuntoT p2){
                x1=p1.x;
                y1=p1.y;
                x2=p2.x;
                y2=p2.y;
                return this;}
        Rectangulos1 consRect(PuntoT p3, int a, int l){
                x1=p3.x;
                y1=p3.y;
                x2=(x1+a);
                y2=(y1+l);
                return this;}
        void printRect(){
                System.out.print(" Valores " + x1 + ", "+y1 + ", "+x2 + ", "+y2);
                System.out.println(" ");}
```

Constructor - ejercicio

```
public static void main (String args []){
Rectangulos1 rect = new Rectangulos1();
System.out.print("Llamar a constRect con las coordenadas : 25,25,50,50:");
rect.consRect(25,25,50,50);
rect.printRect();
System.out.print("Llamar a constRect con los puntos (10,10) y (20):");
rect.consRect(new PuntoT(10,10),new PuntoT(20));
rect.printRect();
System.out.print("Llamar a constRect con los puntos (0,0) y los valores 50,50 :");
rect.consRect(new PuntoT(),50,50);
rect.printRect();
PuntoT a=new PuntoT(3,5);
PuntoT b=new PuntoT();
PuntoT c=new PuntoT(4);
a.mostrar();
int v=2,w=4;
a.desplazar(v,w);
a.mostrar();
b.mostrar();
b.desplazar(v);
b.mostrar();
short z=3;
c.mostrar();
c.desplazar(z);
c.mostrar();
byte p=7;
a.desplazar(p);
a.mostrar();}}
```

Reglas de los Constructores

- Los constructores no retornan ningún valor. Por lo cual, no debe figurar nada antes del nombre del constructor *public void Punto()*.
- Podemos declarar una clase sin constructor. En este caso, para instanciar los objetos se usa : *Punto a = Punto () ;*, como si existiera un constructor por defecto sin argumentos. Desde que la clase posee al menos un constructor este constructor *por defecto* no podrá ser utilizado.
- Un constructor no puede ser llamado directamente desde otro método.
  Por ejemplo : *Punto a = Punto (2,3) ;*
  a.Punto (4,6) ;

Construcción e inicialización de un objeto

La creación de un objeto conlleva :

- Inicialización por defecto de todos los campos del objeto (null)..
- Inicialización explícita al declarar los campos de un objeto.
- Ejecución de las instrucciones del constructor.

```
public class A{
        public A (...){...} //constructor
        int n=10;
        int p;}
```

Si ejecutamos la instrucción : *A a = new A() ;* que ocurre ?

Si los campos del objeto son constantes, es decir, definidos con la palabra clave *final*, estos deberán ser inicializados a mas tardar por el constructor y evidentemente no podan ser modificados posteriormente. A diferencia de los objetos las variables locales no son inicializadas de manera implicita. Toda variable local debe ser inicializada antes de ser usada.

Llamar a un constructor dentro de otro constructor

No se puede llamar directamente a un constructor (*a.Point(2,3)*). Pero si se puede llamar a un constructor al interior de otro constructor de la misma clase. Para esto se usara la palabra clave *this*.

```
public class PuntoT{
        int x,y;
        public PuntoT (int abs, int ord){
                x=abs;
                y=ord;
                System.out.println("Constructor con dos argumentos "+x+" "+ y);}
        public PuntoT (){
                this(0,0);
                System.out.println("Constructor sin argumentos");}
        public static void main(String arg[]){
                PuntoT a=new PuntoT(3,5);
                PuntoT b=new PuntoT();}}
```

Recursividad

En Java es posible usar la recursividad ya sea en los metodos de instancia o en los metodos de clase (*static*)..

- Directa : un método tiene, en su definición, una llamada a si mismo.
- Cruzada : el llamado de un método lleva el llamado de otro método que llama al método inicial.

Cree un programa recursivo que calcule el factorial de un numero.

Recursividad- ejercicio

```
import java.util.Scanner;

class Factorial{
        public static long fac(long n){
                if(n>1) return(fac(n-1)*n);
                else return 1;}

        public static long leer(){
                Scanner kyb=new Scanner(System.in);
                System.out.println("Ingrese un valor positivo");
                long valor = kyb.nextInt();
                return valor;}}

public class CalcularFac{

        public static void main (String args []){
                System.out.println("El factorial es :" + Factorial.fac(Factorial.leer()));
        }}
```

Recursividad- ejercicio

```
import java.util.Scanner;

class Factorial{

        public static long fac(long n){
                long res;
                System.out.println("Entra a fac : n = "+ n);
                if(n<=1)
                        res=1;
                else
                res=fac(n-1)*n;
                System.out.println("Sale de a fac : resultado = "+ res);
                return res;}

        public static long leer(){
                Scanner kyb=new Scanner(System.in);
                System.out.println("Ingrese un valor positivo");
                long valor = kyb.nextInt();
                return valor;}}

public class CalcularFac2{

        public static void main (String args []){
                System.out.println("El factorial es :" + Factorial.fac(Factorial.leer()));
        }}
```

Atributos private

La encapsulación consiste en el ocultar los datos miembros de una clase, de modo que solo será posible modificarlos mediante métodos definidos dentro de la misma clase. De esta forma, los detalles de implementación permanecen ”ocultos” a las personas que usan las clases.
La encapsulación es una de las principales ventajas que proporciona la programación orientada a objetos.
Java implementa la encapsulación de datos usando la palabra clave : *private*.
El uso de la encapsulación de datos no es obligatoria pero es recomendada.

Atributos private

Cree una clase Punto con *private int x,int y ;* y con un método mostrar el punto.
Cree una clase circulo con :

```
class Cercle{
        private PuntoTest p;
        private float r;
        public Cercle(int x,int y, float r){
                p=new PuntoTest(x,y);
                this.r=r;
                p.mostrar();} //aqui van los metodos}
```

Cree ademas un método mostrar los parámetros del circulo y otro que desplace los valores del punto central del circulo.

Atributos private - ejercicio

```
class PuntoTest{
        private int x,y;
        public PuntoTest (int abs, int ord){
                this.x=abs;
                this.y=ord;}
        public void mostrar(){
                System.out.println("punto de coordenadas : "+x+" " + y);}
}
```

Atributos private - ejercicio

```
class Cercle{
        private PuntoTest p;
        private float r;
        public Cercle(int x,int y, float r){
                p=new PuntoTest(x,y);
                this.r=r;
                p.mostrar();

        }
        public void mostrar(){
                System.out.println("Punto central del circulo de coordenadas : "+p.x+"
" +p.y);
                System.out.println("Radio del circulo : "+r);}

        public void desplazar(PuntoTest p, int dx, int dy){
                p.x+=dx;
                p.y+=dy;
                System.out.println("Punto central desplazado del circulo : "+p.x+"
" +p.y);}


}
```

Atributos private - ejercicio

```
public class MainCerclePuntoTest{
        public static void main (String args[])
        {PuntoTest p=new PuntoTest(3,2);
                p.mostrar();
                Cercle  c= new Cercle(1,2,5.4f);
                c.mostrar();
                c.desplazar(5,7);}}
```

Atributos private

Dado que los atributos de la clase Punto son privados, estos no son accesibles por la clase circulo. Para evitar esto crearemos la misma clase circulo que tiene un objeto miembro del tipo Punto y a la vez definiremos métodos de acceso *getX* y *getY* de alteración *setX* y *setY*.

```
public int getX(){return x;}
public int setX(int x){return this.x=x;}
```

Cree los métodos *set* y *get* para corregir el ejemplo precedente.

Uso set y get- ejercicio

```
class PuntoTest{
        private int x,y;
        public PuntoTest (int abs, int ord){
                this.x=abs;
                this.y=ord;}
        public void mostrar(){
                System.out.println("punto de coordenadas : "+x+" " + y);}

        public int getX(){return x;}
        public int getY(){return y;}
        public int setX(int x){return this.x=x;}
        public int setY(int y){return this.y=y;}
}
```

Uso set y get- - ejercicio

```
class Cercle{
        private PuntoTest p;
        private float r;
        public Cercle(int x,int y, float r){
                p=new PuntoTest(x,y);
                this.r=r;
        }
        public void mostrar(){
                System.out.println("Punto central del circulo de coordenadas : "+p.getX()+"
" +p.getY());
                System.out.println("Radio del circulo : "+r);}

        public void desplazar(int dx, int dy){
                p.setX(p.getX()+dx);
                p.setY(p.getY()+dy);}


}
```

Uso set y get- - ejercicio

```
public class MainCerclePuntoTest2{
        public static void main (String args[])
        {PuntoTest p=new PuntoTest(3,2);
                p.mostrar();
                Cercle  c= new Cercle(1,2,5.4f);
                c.mostrar();
                c.desplazar(6,7);
                c.mostrar();}}
```

Paquetes

La noción de paquete corresponde a un reagrupamiento logico bajo un identificador común de un conjunto de clases. Los paquetes Java agrupan las clases en librerías (bibliotecas). Los paquetes Java se utilizan de forma similar a como se utilizan las librerías en C++, sólo que en Java se agrupan clases y/o interfaces.
Los paquetes se caracterizan por un nombre que puede ser un simple identificador *MisClases* único o una continuación de identificadores separados por puntos *Utilitarios.Matematica*. Los paquetes proporcionan una forma de ocultar clases, evitando que otros programas o paquetes accedan a clases que son de uso exclusivo de una aplicación determinada.

Paquetes

Paquetes

Los paquetes se declaran utilizando la palabra reservada *package* seguida del nombre del paquete. Esta sentencia debe estar al comienzo del archivo fuente. Concretamente debe ser la primera sentencia ejecutable del código Java, excluyendo, los comentarios y espacios en blanco. Por ejemplo :

package figuras ;
public class Circulo {
. . }.

Paquetes

Paquetes

Para incluir nuevas clases en el paquete se debe colocar la misma sentencia al comienzo de los archivos que contengan la declaración de las clases. Cada uno de los archivos que contengan clases pertenecientes a un mismo paquete, deben incluir la misma sentencia *package*, y solamente puede haber una sentencia *package* por fichero. La sentencia *package* colocada el comienzo de un fichero fuente afectará a todas las clases que se declaren en ese fichero.

Paquetes

Uso de una clase de un paquete

Cuando un programa usa una clase el compilador la busca en el paquete por defecto. Para usar una clase que pertenezca a otro paquete se debe :

- usar el nombre del paquete y de la clase : *MisClases.Punto p = new MisClases.Punto(2,3) ;*
- usar la instrucción *import* para importar una clase : *import MisClases.Punto* o *import MisClases.** para importar el paquete completo. Luego se podrá usar el nombre de la clase sin especificar el nombre del paquete.

Paquetes

Acceso a la clases y paquetes

Cada clase dispone de derecho de acceso, este es definido por la palabra clave *public*.

- con la palabra clave *public* la clase es accesible por todas las otras clases (usando import).
- sin la palabra clave *public* la clase será accesible solo por la clases del mismo paquete.

Mientras se trabaje con el paquete por defecto, la ausencia de la palabra *public* no tiene importancia, pero al definir un paquete la clases no definidas como publicas solo serán accesibles por la clases que pertenecen al paquete.