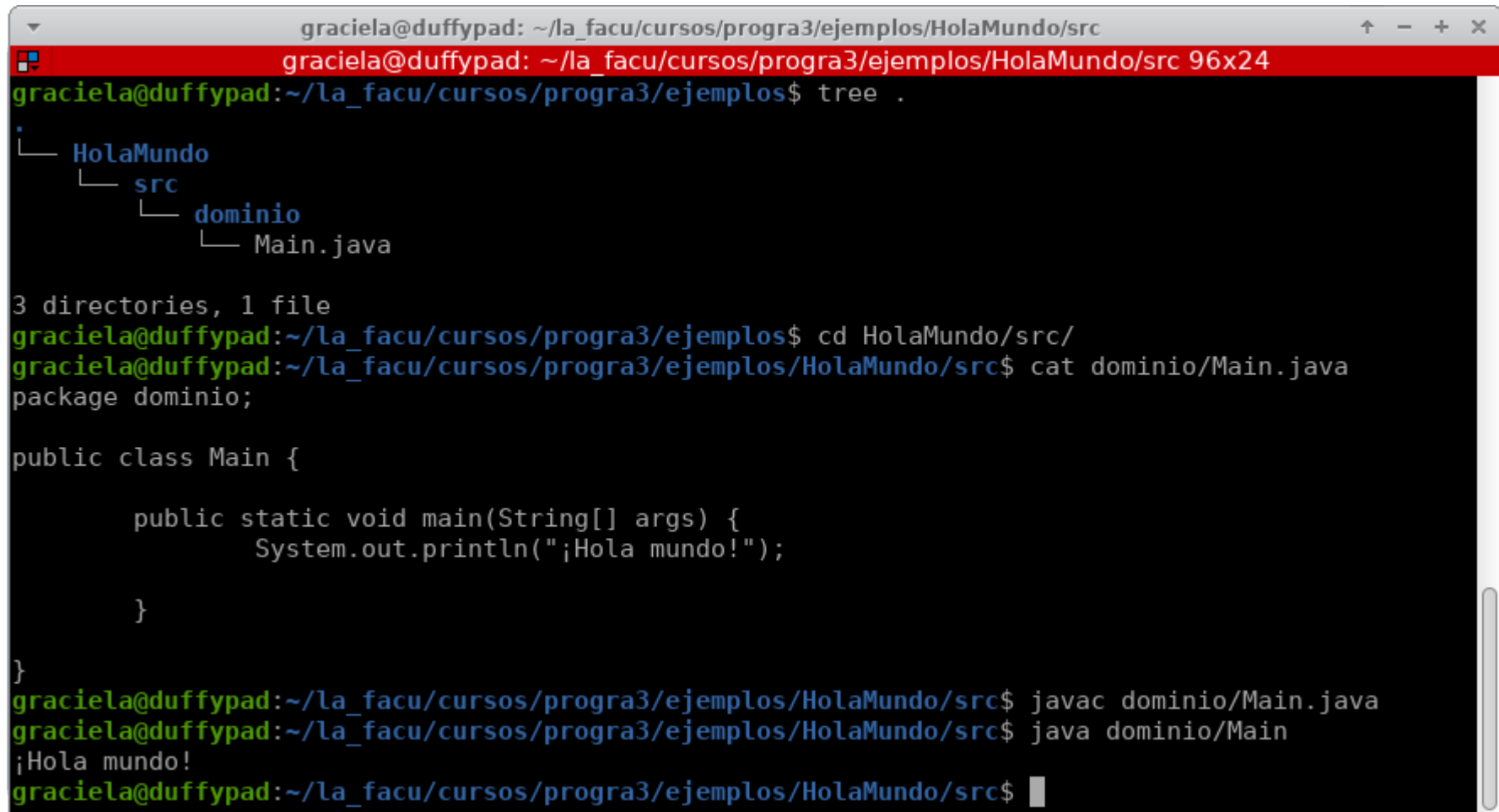


Programación 3

Introducción al lenguaje Java

Empecemos donde dejamos la última vez... el infaltable
¡HolaMundo!



```
graciela@duffypad: ~/la_facu/cursos/progra3/ejemplos/HolaMundo/src
graciela@duffypad: ~/la_facu/cursos/progra3/ejemplos/HolaMundo/src 96x24
graciela@duffypad:~/la_facu/cursos/progra3/ejemplos$ tree .
.
├── HolaMundo
│   └── src
│       ├── dominio
│       └── Main.java
3 directories, 1 file
graciela@duffypad:~/la_facu/cursos/progra3/ejemplos$ cd HolaMundo/src/
graciela@duffypad:~/la_facu/cursos/progra3/ejemplos/HolaMundo/src$ cat dominio/Main.java
package dominio;

public class Main {

    public static void main(String[] args) {
        System.out.println("¡Hola mundo!");
    }

}

graciela@duffypad:~/la_facu/cursos/progra3/ejemplos/HolaMundo/src$ javac dominio/Main.java
graciela@duffypad:~/la_facu/cursos/progra3/ejemplos/HolaMundo/src$ java dominio/Main
¡Hola mundo!
graciela@duffypad:~/la_facu/cursos/progra3/ejemplos/HolaMundo/src$
```

Para poder escribir y ejecutar nuestro *¡Hola mundo!* primero necesitamos conocer aspectos básicos del lenguaje...

Variables

El lenguaje Java define distintos tipos de variables:

- *de clase*, son los atributos estáticos de una clase, de los cuales existen una sola copia para todos los objetos de dicha clase
- *de instancia*, de las que existen una copia por cada objeto instanciado
- *locales*, definidas en el contexto de un método
- *parámetros*, de los métodos definidos en una clase

Dado que Java es fuertemente tipado, al declarar una variable es necesario definir su tipo:

```
tipo_de_dato nombre = valor_inicial;
```

Existen ocho tipos de datos primitivos:

Nombre	Bits	Rango de representación	Val/defecto
byte	8 bits	-128 a 127	0
short	16 bits	-32768 a 32767	0
int	32 bits	-2^{31} a $2^{31}-1$	0
long	64 bits	-2^{63} a $2^{63}-1$	0L
float	32 bits	<i>ver IEEE 754</i>	0.0f
double	64 bits	<i>ver IEEE 754</i>	0.0d
boolean	1 bit	true o false	false
char	16 bits	'\u0000' (0) a '\uffff' (65,535 inclusive)	'\u0000'

Dado que Java es fuertemente tipado, al declarar una variable es necesario definir su tipo:

```
tipo_de_dato nombre = valor_inicial;
```

Existen ocho tipos de datos primitivos:

Nombre	Bits	Rango de representación	Val/defecto
byte	8 bits	-128 a 127	0
short	16 bits	-32768 a 32767	0
int	32 bits	-2^{31} a $2^{31}-1$	0
long	64 bits	-2^{63} a $2^{63}-1$	
float	32 bits	ver IEEE 754	
double	64 bits	ver IEEE 754	
boolean	1 bit	true o false	
char	16 bits	'\u0000' (0) a	

¡Importante!

Java solo inicializa automáticamente con valores por defecto a variables de instancia y de clase, no así a las locales. Intentar utilizar una variable local sin inicializar generará un error de compilación.

En cuanto a nombres, se puede utilizar cualquier cadena de texto que cumpla con las siguientes reglas:

- pueden comenzar con una letra, el signo \$ o _ (guión bajo), aunque no se recomienda usar ninguna de las dos últimas alternativas
- el resto del nombre puede incluir letras, dígitos, \$ y/o _
- por convención, se utiliza *camel case*: `hora`, `horaDelDia`
- por convención, las constantes se definen sólo con mayúsculas: `PORC_DESC`

Arreglos

Son objetos que contienen valores de un único tipo, compuestos por elementos que son accedidos a través de su índice

Declaración	<code>tipo_elemento[] nombre</code>	<code>int[] numeros;</code>
Creación	<code>variable = new tipo_elemento[cant_elementos]</code>	<code>numeros = new int[5];</code>
Inicialización	<code>nombre[indice] = valor</code>	<code>numeros[0] = 42;</code>
Acceso	<code>variable = nombre[indice]</code>	<code>int nro = numeros[0];</code>

También se los puede declarar, crear e inicializar al mismo tiempo:

```
int[] numeros = {  
    10, 20, 30, 40, 50, 60, 70, 80, 90, 1000  
};
```

En este caso, el tamaño es inferido por la cantidad de inicializadores

Ya vimos a las variables, ahora es el turno de los operadores, para poder hacer algo con ellas...

Operadores

Permiten manipular variables. Existen operadores de distintos tipos:

Asignación	=
Aritméticos	+ - * / %
Unarios	+ - ++ -- !
Igualdad y relacionales	== != < > <= >=
Condicionales	&& ? :
Comparación de tipos	instanceof
A nivel de bit	<< >> >>> & ^ ~

Precedencia

Todos los operadores tienen una precedencia asignada, es decir que, dada una expresión compleja con varios operadores involucrados, éstos serán evaluados en un orden determinado.

Los operadores de mayor precedencia se evalúan antes que los de menor precedencia.

Cuando se tienen operadores de igual precedencia, los operadores binarios son evaluados de izquierda a derecha, excepto la asignación, que se evalúa de derecha a izquierda.

A continuación, la tabla de precedencia de operadores:

Postfix	<code>expr++ expr--</code>
Unarios	<code>++expr --expr +expr -expr ~ !</code>
Multiplicativos	<code>* / %</code>
Aditivos	<code>+ -</code>
Desplazamiento	<code><< >> >>></code>
Relacionales	<code>< > <= >= instanceof</code>
Igualdad	<code>== !=</code>
AND a nivel de bit	<code>&</code>
OR exclusivo a nivel de bit	<code>^</code>
OR inclusivo a nivel de bit	<code> </code>
AND lógico	<code>&&</code>
OR lógico	<code> </code>
Ternario	<code>? :</code>
Asignación	<code>= += -= *= /= %= &= ^= = <<= >>= >>>=</code>

Expresiones, declaraciones y bloques

Una *expresión* es una construcción a partir de variables, operadores e invocaciones a métodos, que se evalúan obteniendo un único valor. Se pueden agrupar utilizando `()`.

Una *declaración* conforma una unidad completa de ejecución. Su terminador es un `;`.

Un *bloque* es un conjunto de cero o más declaraciones entre llaves `{ }`.

Con las variables y los operadores se construyen expresiones, que forman parte de declaraciones. Estas declaraciones pueden estar agrupadas en bloques.

Control de flujo

Las declaraciones incluídas en un programa se ejecutan secuencialmente, de arriba hacia abajo.

Las estructuras de control de flujo alteran el flujo de ejecución a través del uso de condiciones, repeticiones y ramificaciones, permitiendo al programa la ejecución condicional de ciertos bloques de código.

Condicionales

<i>if-then</i>	Es el más simple de todos, donde se evalúa una condición y se plantea qué hacer en caso de que sea verdadera	<pre>if (condicion) { // camino por verdadero }</pre>	<pre>if (nro > 0) { positivo = true; }</pre>
<i>if-then-else</i>	Plantea caminos por verdadero y por falso	<pre>if (condicion) { // camino por verdadero } else { // camino por falso }</pre>	<pre>if (nro > 0) { positivo = true; } else { positivo = false; }</pre>

Condicionales

<i>switch</i>	<p>Evalúa el valor de una variable y plantea por extensión los posibles caminos a seguir. La variable puede ser de tipo <code>byte</code>, <code>short</code>, <code>char</code> e <code>int</code>, de algún tipo de <code>enum</code>, instancia de la clase <code>String</code> e instancia de las clases <code>Character</code>, <code>Byte</code>, <code>Short</code> e <code>Integer</code>.</p>	<pre>switch (var) { case un_valor: // var es igual a un_valor break; case otro_valor: // var es igual a otro_valor break; default: // var es no igual ninguno break; }</pre>	<pre>switch (color) { case "rojo": rgb = "255,0,0"; break; case "verde": rgb = "0,255,0"; break; case "azul": rgb = "0,0,255"; break; default: rgb = ""; break; }</pre>
---------------	--	--	---

Repeticiones

<i>while</i>	Primero chequea la condición, luego comienza a iterar. La iteración continúa hasta que la condición se evalúe a falso.	<pre>while (condicion) { // código a ejecutar }</pre>	<pre>while (cont < 10) { cont++; }</pre>
<i>do-while</i>	Este ciclo ejecuta al menos una vez y luego evalúa la condición. Mientras sea verdadera, la iteración continuará.	<pre>do { // código a ejecutar } while(condicion);</pre>	<pre>do { cont++; } while (cont < 10);</pre>

Repeticiones

<i>for</i>	El <i>inicio</i> se ejecuta una única vez. La repetición continúa hasta que <i>fin</i> evalúe a falso. Si es verdadero, se ejecuta el código dentro del ciclo y se calcula el <i>incremento</i> . Las tres partes son opcionales.	<pre>for (ini; fin; inc){ // código a ejecutar }</pre>	<pre>for (int i=0;i<10; i++){ acum += i; } for (; ;){ // ciclo infinito }</pre>
<i>foreach</i>	Se recomienda para iterar sobre conjuntos de elementos, cuando éstos no necesitan ser modificados sino solo recorridos. Se puede iterar sobre arreglos y colecciones.	<pre>for (tipo el: conj){ // código a ejecutar }</pre>	<pre>for (int n: nros){ acum += n; }</pre>

Branching statements

Son utilizados para modificar el flujo normal de ejecución

<i>break</i>	En la construcción <code>switch</code> es utilizado para delimitar el final de un <code>case</code> : e indicar que no se debe seguir evaluando el resto. En los ciclos de repetición, sirve para terminar la iteración.	<pre>for (int i=0;i<10;i++) { // termina cuando llega a 5 if (i == 5) break; System.out.println("i:"+i); }</pre>
<i>continue</i>	En los ciclos de repetición, el <code>continue</code> interrumpe la iteración actual para pasar a la siguiente, es decir, la saltea.	<pre>for (int i=0;i<10;i++) { // Si es par omitir y continuar if (i%2 == 0) continue; // Si es impar, imprimir System.out.print(i + " "); }</pre>
<i>return</i>	El <code>return</code> permite salir del método actual y devolver el control de ejecución a quién invocó dicho método. Puede o no devolver un valor.	<pre>int incrementar (int num, int inc) { return num+inc; }</pre>

Y finalmente, con toda esta base...

Clases

Una clase se declara de la siguiente manera:

```
modif_acceso otros_modif class NombreClase
                                extends NombreSuperclase
                                implements NombreInterfaz {
    // atributos, constructores y métodos
}
```

Tener en cuenta que:

- los modificadores de acceso válidos para clases son `public` y por defecto (es decir, no se escribe ningún modificador)
- por convención, el nombre de las clases siempre comienza con mayúscula, así como también cada palabra que integra su nombre
- una clase puede *extender* a sólo una superclase
- una clase puede *implementar* múltiples interfaces, las cuales se indicarán separadas por coma
- el cuerpo de la clase se delimita con llaves

Atributos

Un atributo se declara de la siguiente manera:

```
modif_acceso otros_modif nombreAtributo =  
                                valor_inicial;
```

Tener en cuenta que:

- los modificadores de acceso aplicables a un atributo son `public`, `protected`, `private` y `default` (sin modificador)
- por convención, el nombre comienza con minúscula (*camel case*)
- pueden ser de un tipo de dato primitivo o ser referencia a algún objeto

Métodos

Un método se declara de la siguiente manera:

```
modif_acceso otros_modif tipo_retorno
    nombreMetodo (parametros) throws excepciones {
    // código a ejecutar
}
```

Tener en cuenta que:

- los modificadores de acceso aplicables a un método son `public`, `protected`, `private` y por defecto (sin modificador)
- si el método no retorna nada, su tipo de retorno es `void`
- por convención, el nombre comienza con minúscula (*camel case*) y usualmente incluye un verbo
- los parámetros se escriben separados por coma, indicando tipo y nombre; en caso de no tener parámetros los paréntesis quedan vacíos
- la *firma* del método incluye sólo su *nombre* y el *listado de parámetros*
- en caso de no lanzar ninguna excepción, el `throws` no aplica
- el cuerpo del método se delimita con llaves

Métodos

Un método se declara de la siguiente manera:

```
modif_acceso otros_modif tipo_retorno
    nombreMetodo (parametros) throws excepciones {
    // código a ejecutar
}
```

Tener en cuenta que:

- los modificadores de acceso aplicables a un método son `public`, `protected`, `private` y por defecto (sin modificador)
- si el método no retorna nada, su tipo de retorno es `void`
- por convención, el nombre comienza con minúscula (*camel case*) y usualmente incluye un verbo

- los parámetros se escriben separados por un punto y coma y un nombre; en caso de no tener parámetros, no se escribe nada
- la firma del método incluye sólo la parte de la declaración
- en caso de no lanzar ninguna excepción, no se escribe nada
- el cuerpo del método se delimita por llaves de cierre

¡Importante!

El paso de parámetros es siempre por valor, ya sea de un dato primitivo o de un objeto.

Si se trata de un objeto, éste podrá ser modificado a través de los métodos que exponga.

Constructores

Un constructor es un método que permite crear objetos a partir de la clase en la que está definido. Se declara de la siguiente manera:

```
modif_acceso NombreClase (parametros) {  
    // código a ejecutar  
}
```

Tener en cuenta que:

- si un constructor se define como `private`, ninguna clase externa podrá instanciar objetos de la clase en la que está definido
- un constructor no tiene valor de retorno
- una clase pueden tener múltiples constructores, cada uno con distintos parámetros o incluso sin parámetros
- en caso de no proveer un constructor, el lenguaje provee el constructor por defecto, es decir, aquel que no recibe ningún parámetro
- desde un constructor, una subclase puede invocar a otro método constructor de su propia clase con la palabra reservada `this` o uno de su superclase con la palabra reservada `super`

Accessors

Un método de acceso permite acceder al estado interno de un objeto, es decir, a sus atributos. Son los llamados *getters* y *setters*. Se declaran de la siguiente manera:

```
public tipo_atributo getNombreAtributo () {  
    return atributo;  
}
```

y

```
public void setNombreAtributo(tipo_atributo nuevo_valor) {  
    atributo = nuevo_valor;  
}
```

Tener en cuenta que:

- los métodos de acceso son por definición públicos
- no es obligatorio que sean incluidos en la definición de la clase
- se utilizan para tener un control del acceso al estado interno de un objeto
- permiten agregar lógica de control a dicho acceso

Creando objetos

Para crear una instancia de una clase es necesario invocar el constructor de dicha clase utilizando el operador `new`:

```
NombreClase objeto_nuevo = new NombreClase(parametros);
```

Tener en cuenta que:

- el operador `new` retorna una referencia a la instancia recién creada
- si la clase tiene múltiples constructores, el lenguaje seleccionará el que se ajuste a los parámetros especificados
- a la izquierda se especifica el tipo de referencia, a la derecha el tipo concreto instanciado
- si la clase no ofrece un constructor, el lenguaje le proveerá el constructor por defecto, que a su vez invocará al constructor por defecto de su superclase. En caso de no tener una, se invocará el constructor por defecto de la clase `Object`.
- a través del operador `.` (punto) se podrá acceder a los miembros del objeto (atributos y métodos), según su nivel de acceso


```

1 package estructura;
2
3 import dominio.Circulo;
4 import dominio.Punto;
5
6 public class Main {
7
8     public static void main(String[] args) {
9
10         // un circulo creado a partir de dos coordenadas (el centro) y un radio
11         Circulo unCirculo = new Circulo(4, 5, 6);
12
13         // un circulo creado a partir de un centro definido por un punto y un radio
14         Punto unCentro = new Punto(9,5);
15         Circulo otroCirculo = new Circulo(unCentro, 7);
16
17         // un circulo creado a partir de un radio
18         Circulo unCirculoMas = new Circulo(8);
19
20         // un circulo definido sin datos de inicio => circulo de centro (0,0) y radio 1
21         Circulo otroCirculoMas = new Circulo();
22
23         System.out.println(unCirculo);
24         System.out.println(otroCirculo);
25         System.out.println(unCirculoMas);
26         System.out.println(otroCirculoMas);
27
28         System.out.println("\nPerimetros:");
29
30         System.out.println("Perímetro de unCirculo= " + unCirculo.calcularPerimetro());
31         System.out.println("Perímetro de otroCirculo= " + otroCirculo.calcularPerimetro());
32         System.out.println("Perímetro de unCirculoMas= " + unCirculoMas.calcularPerimetro());
33         System.out.println("Perímetro de otroCirculoMas= " + otroCirculoMas.calcularPerimetro());
34
35         System.out.println("\nAreas:");
36
37         System.out.println("Area de unCirculo= " + unCirculo.calcularArea());
38         System.out.println("Area de otroCirculo= " + otroCirculo.calcularArea());
39         System.out.println("Area de unCirculoMas= " + unCirculoMas.calcularArea());
40         System.out.println("Area de otroCirculoMas= " + otroCirculoMas.calcularArea());
41
42     }
43
44 }

```

```

1 package estructura;
2
3 import dominio.Circulo;
4 import dominio.Punto;
5
6 public class Main {
7
8     public static void main(String[] args) {
9
10         // un circulo creado a partir de dos coordenadas (el centro) y un radio
11         Circulo unCirculo = new Circulo(4, 5, 6);
12
13         // un circulo creado a partir de un centro definido
14         Punto unCentro = new Punto(9,5);
15         Circulo otroCirculo = new Circulo(unCentro, 7);
16
17         // un circulo creado a partir de un radio
18         Circulo unCirculoMas = new Circulo(8);
19
20         // un circulo definido sin datos de inicio => circulo
21         Circulo otroCirculoMas = new Circulo();
22
23         System.out.println(unCirculo);
24         System.out.println(otroCirculo);
25         System.out.println(unCirculoMas);
26         System.out.println(otroCirculoMas);
27
28         System.out.println("\nPerimetros:");
29
30         System.out.println("Perímetro de unCirculo= " + unCirculo.calcularPerimetro());
31         System.out.println("Perímetro de otroCirculo= " + otroCirculo.calcularPerimetro());
32         System.out.println("Perímetro de unCirculoMas= " + unCirculoMas.calcularPerimetro());
33         System.out.println("Perímetro de otroCirculoMas= " + otroCirculoMas.calcularPerimetro());
34
35         System.out.println("\nAreas:");
36
37         System.out.println("Area de unCirculo= " + unCirculo.calcularArea());
38         System.out.println("Area de otroCirculo= " + otroCirculo.calcularArea());
39         System.out.println("Area de unCirculoMas= " + unCirculoMas.calcularArea());
40         System.out.println("Area de otroCirculoMas= " + otroCirculoMas.calcularArea());
41
42     }
43
44 }

```

Soy un circulo de centro (4,5) y de radio 6.0
 Soy un circulo de centro (9,5) y de radio 7.0
 Soy un circulo de centro (0,0) y de radio 8.0
 Soy un circulo de centro (0,0) y de radio 1.0

Perimetros:
 Perímetro de unCirculo= 37.69911184307752
 Perímetro de otroCirculo= 43.982297150257104
 Perímetro de unCirculoMas= 50.26548245743669
 Perímetro de otroCirculoMas= 6.283185307179586

Areas:
 Area de unCirculo= 113.09733552923255
 Area de otroCirculo= 153.93804002589985
 Area de unCirculoMas= 201.06192982974676
 Area de otroCirculoMas= 3.141592653589793