

# 1.00 Clase 5

## Clases y objetos en Java

### Objetos

- **Los objetos son ‘cosas’.**
  - En el 8.01 esta cuestión se definió como “cosa”.
- **Descomponemos los problemas de programación en un conjunto de objetos que son una representación “natural” de la situación. Ejemplos:**
  - **Un pájaro que se posa en una acera.**
    - Enumere los objetos.
      - Cada uno de estos puede contener otros objetos. Enumérelos.
  - **Paquetes de datos que pasan a través de un *router*.**
    - Enumere los objetos.
      - Cada uno de estos puede contener otros objetos. Enumérelos.

# Objetos

- Los objetos son “cosas”.
  - En 8.01 se definió esta cuestión como “cosa”.
- Descomponemos los problemas de programación en un conjunto de objetos que son una representación “natural” de la situación.  
Ejemplos:
  - Un pájaro que se posa en una acera.
    - Objetos: pájaro, acera, atmósfera (?). Cada uno de estos puede contener otros objetos:
      - Pájaro: posee pico, alas, cerebro (?)...
  - Paquetes de datos que pasan a través de un *router*.  
Objetos: enlaces de redes de entrada y salida, *router*.  
Cada uno de estos puede contener otros objetos:
    - Enlace de red: contiene paquetes de datos.
    - *router*: contiene colas de paquetes, búfers y un procesador.

# Datos y métodos del objeto

- Pájaro:
  - Campos de datos:
    - Enumérelas.
  - Métodos o comportamientos:
    - Enumérelas.
- Acera:
  - Campos de datos:
    - Enumérelas.
  - Métodos:
    - Enumérelas.
- Atmósfera
  - Campos:
    - Enumérelas.
  - Métodos:
    - Enumérelas.

## Datos y métodos del objeto

- **Pájaro:**
  - Campos de datos:
    - Especies, tamaño, peso, lugar.
  - Métodos o comportamientos:
    - Volar, comer, dormir, posarse, emprender el vuelo.
- **Acera:**
  - Campos de datos:
    - Tamaño, lugar, tipo de superficie.
  - Métodos:
    - Existencia ('constructor')
    - Los pájaros (objetos en general) también existen y poseen constructores.
- **Atmósfera:**
  - Campos:
    - Temperatura, velocidad en muchos puntos.
  - Métodos:
    - Corrientes de aire, calentamiento, turbulencias...

## Ejemplo del pájaro (Netscape)

```
public class Head() { ..... }

public class Body() { ..... }

public class Wing() { .....
    public void changeColor() { ..... } }

public class Bird() { .....
    Head birdHead= new Head();
    Wing leftWing= new Wing();
    Wing rightWing= new Wing();
    public Wing getWing() { ..... }
    public void fly() { ..... }
    public void changeStatus() { ..... } }

public class Scene() { .....
    Bird bird= new Bird();
    Wing wing= bird.getWing();
    public void changeWingColor() { wing.changeColor(); }
    public void changeFlyStatus() { bird.changeStatus(); } }
```

## Modelado de objetos

- El modelado de objetos (elección de la representación exacta del problema) es como el modelado científico y de ingeniería en general:
  - Generalmente no existe una única respuesta “correcta” (¡ni siquiera en los boletines de problemas del curso 1.00!).
  - Existen patrones / paradigmas estándar que han probado ser flexibles, correctos y eficientes.
    - En el curso 1.00 le presentaremos los “patrones de software”.
  - Existen muchos objetos estándar en Java.
  - Puede construirse su propia biblioteca de objetos en Java, y la podrá utilizar en futuros programas.

## Clases

- Una clase es un patrón o plantilla a partir de la que se crean los objetos:
  - Es posible que tenga muchos pájaros en una simulación.
    - Una clase Pájaro (o más, si hay más de un tipo de pájaro).
    - Muchos objetos pájaro (ejemplares reales de pájaros).
    - Simulación.
  - Otro ejemplo:
    - *JOptionPane* es una clase del paquete *Swing*. Un programa puede tener muchos cuadros de diálogo, cada uno de los cuales es un objeto de la clase *JOptionPane*.

## Definición de clase

- **Las clases poseen:**
  - **Datos (miembros, campos).**
    - Tipos de datos simples, como *int* o *double* (p. ej., peso del pájaro).
    - Objetos (p. ej., pico del pájaro).
  - **Métodos (funciones, procedimientos).**
    - Acciones que puede ejecutar un objeto (p. ej. el pájaro vuela / se mueve).
- **Las clases proceden de:**
  - Bibliotecas de clases de Java: *JOptionPane*, *Vector*, *Array*, etc. Existen varios miles de clases (*Javadoc*).
  - Bibliotecas de clases procedentes de otras fuentes: video, etc.
  - Clases creadas por uno mismo.
- **Las clases coinciden normalmente con los nombres del enunciado de un problema (p. ej., pájaro).**
- **Los métodos coinciden normalmente con los verbos del enunciado del problema (p. ej., vuela).**

## Cómo crear clases

- **Las clases ocultan al usuario sus detalles de implementación (el programador utiliza una clase anteriormente escrita):**
  - No se puede acceder a sus datos directamente, y los detalles no se encuentran visibles para programas u objetos “externos”.
  - Los datos son casi siempre privados (palabra clave *private*).
- **Los objetos se utilizan al llamar a sus métodos.**
  - El usuario externo conoce qué métodos posee el objeto y los resultados que devuelven. Periodo (normalmente).
    - Los detalles sobre la escritura de sus métodos no están a disposición de elementos externos.
  - Normalmente, los métodos son públicos (palabra clave *public*).
    - Al aislar el resto del programa de los detalles del objeto, resulta más fácil construir programas extensos, y reutilizar los objetos de trabajos anteriores.
    - Esto se conoce como encapsulamiento u ocultación de la información.

# Clases y objetos

- **Las clases son un patrón:**
  - Una clase puede tener varios datos almacenados dentro de ella.
  - Una clase puede tener varios métodos que operan sobre (y pueden modificar) sus datos y devuelven resultados.
- **Los objetos son ejemplares de las clases.**
  - Los objetos tienen sus propios datos, métodos de clase y una identidad (un nombre).
- **No hay diferencia entre las clases de la biblioteca de Java y las que usted creará.**
  - Cuando usted construye una clase, está definiendo un nuevo tipo de datos en Java.
  - Básicamente, está ampliando en lenguaje Java.
- **Las clases también pueden crearse a partir de otras clases.**
  - Una clase construida a partir de otra, la amplía.
  - Esto se conoce como herencia, y lo trataremos más adelante.

## Cómo utilizar una clase existente

- ***BigInteger* es una clase de Java que controla arbitrariamente enteros largos.**
  - Úsela en vez de escribir su propia clase para dirigir la aritmética arbitraria.
- **Para trabajar con objetos:**
  - **Primero, constrúyalos y especifique su estado inicial:**
    - Los constructores son métodos especiales que construyen e inicializan objetos.
    - Pueden tener argumentos (parámetros).
  - **Luego aplíqueles los métodos:**
    - Esto es lo mismo que “enviarles mensajes” para invocar sus comportamientos.
- **Las clases predeterminadas son más abstractas y generales que las clases que usted escribirá.**

## Constructor para el objeto *BigInteger*

- Para construir un nuevo objeto *BigInteger*, se necesitan dos cosas:

- Crear el objeto (utilizando su constructor).

```
new BigInteger("1000000000000");  
// 'new' asigna memoria y llama al constructor
```

- Facilitar un nombre o identidad al objeto:

```
BigInteger a;  
// El nombre del objeto es una referencia al objeto  
// BigInteger es el tipo de datos de a
```

- Combinar estas dos cosas en un único paso:

```
BigInteger a= new BigInteger("1000000000000");
```

- Tenemos ahora un objeto *BigInteger* que contiene un valor de 1,000,000,000,000. En este momento, podemos aplicar métodos al objeto.

## Cómo utilizar los métodos

- Los métodos se invocan utilizando el operador punto(.).

- El método siempre acaba con paréntesis.

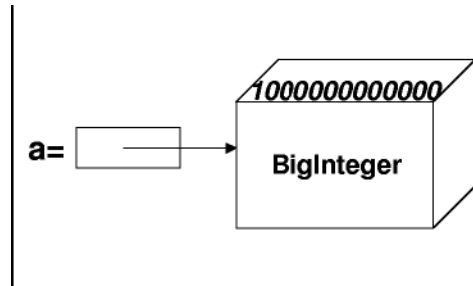
```
BigInteger a= new BigInteger("1000000000000");  
BigInteger z= new BigInteger("23");  
BigInteger c= a.add(z);           // c= a + z  
If (z.isProbablePrime(15))      // ¿es z un número primo?  
    System.out.println("Probablemente z es primo");
```

- Los campos de datos públicos también se invocan con el operador punto.

- No hay paréntesis después del nombre del campo.
- ```
int j= a.somePublicField;           // Solo ejemplo
```

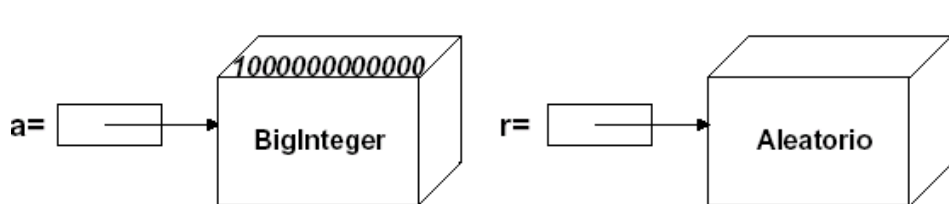
# Objetos y nombres

```
BigInteger a= new BigInteger(1000000000000);
```



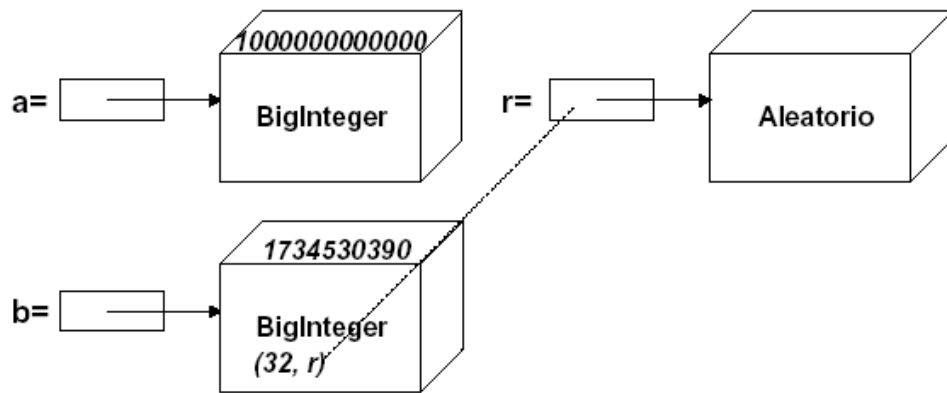
# Objetos y nombres

```
BigInteger a= new BigInteger(1000000000000);  
Random r= new Random();
```



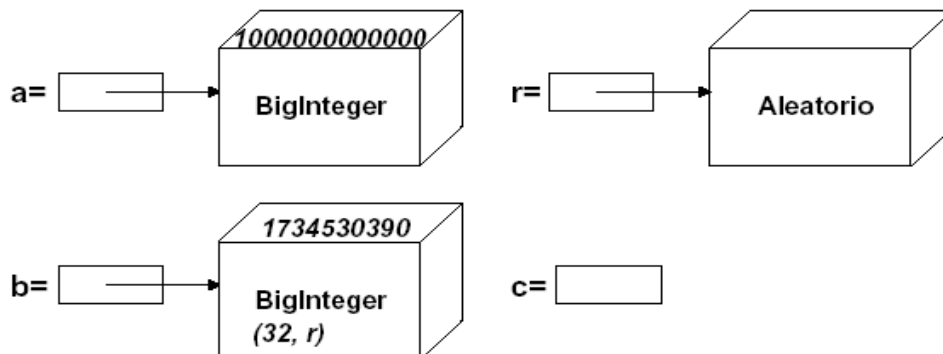
## Objetos y nombres

```
BigInteger a= new BigInteger(1000000000000);  
Random r= new Random();  
BigInteger b= new BigInteger(32, r);
```



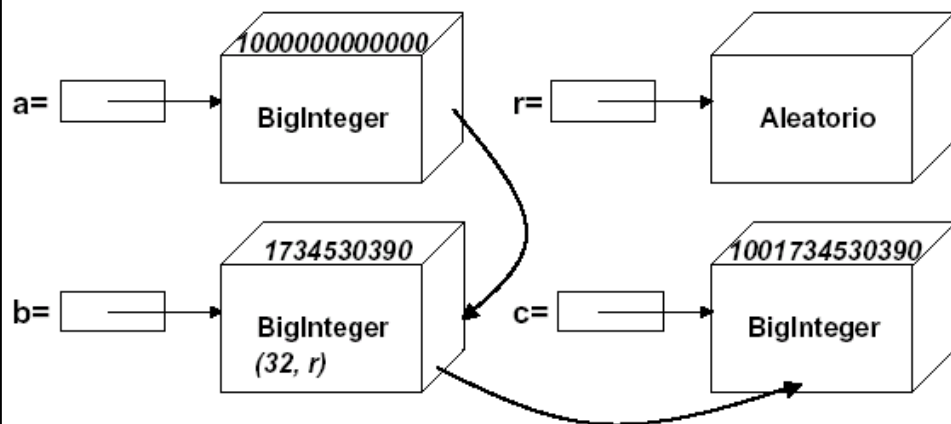
## Objetos y nombres

```
BigInteger a= new BigInteger(1000000000000);  
Random r= new Random();  
BigInteger b= new BigInteger(32, r);  
BigInteger c;
```



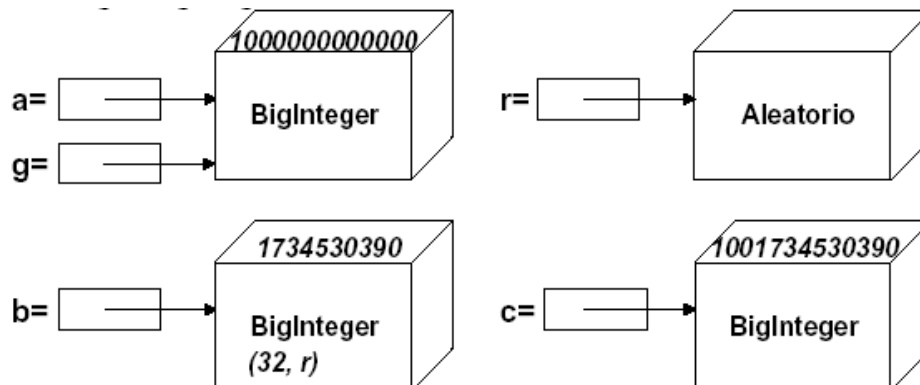
# Objetos y nombres

```
BigInteger a= new BigInteger(1000000000000);  
Random r= new Random();  
BigInteger b= new BigInteger(32, r);  
BigInteger c;  
c= b.add(a);
```



# Objetos y nombres

```
BigInteger a= new BigInteger(1000000000000);  
Random r= new Random();  
BigInteger b= new BigInteger(32, r);  
BigInteger c;  
c= b.add(a);  
BigInteger g= a;
```



## Cómo utilizar la clase *BigInteger*

```
import java.math.*;                // Para BigInteger
import java.util.*;                // Para Random
public class BigIntTest {
    public static void main(String[] args) {
        BigInteger a= new BigInteger("10000000000000");
        Random r= new Random();     // generador de números aleatorios
        BigInteger b= new BigInteger(32, r);           //aleatorio
        BigInteger c;
        c= b.add(a);                // c= b+a
        BigInteger g= a;
        BigInteger d= new BigInteger(32, 10, r);     // Primo
        BigInteger e;
        e= c.divide(d);             // e= c/d
        if (d.isProbablePrime(10))
            System.out.println("Probablemente d es primo");
        else
            System.out.println("Probablemente d no es primo");
        BigInteger f= d.multiply(e);                // f= d*e
    }
}
```

## Ejercicio 1: Clase existente

- Utilice la clase *BigDecimal* (números de coma flotante) para:
  - Construir *BigDecimal*  $a = 13 \times 10^{500}$
  - Construir *BigDecimal* *b* aleatoriamente:
    - Consejo: construya un *BigInteger* aleatorio. Luego, utilice el constructor *BigDecimal* apropiado. Eche un vistazo a Javadoc.
  - Calcule *BigDecimal*  $c = a + b$
  - Calcule *BigDecimal*  $d = c / a$ 
    - Busque un tipo de redondeo en Javadoc.
  - Imprima por pantalla *a*, *b*, *c* y *d* después de calcular cada uno de ellos.
- Escriba el programa en etapas:
  - Construya *a*, imprímalo. Compile y depure.
    - ¡No cuente los ceros!
  - Añada la construcción de *b*, imprímalo. Compile y depure.
  - Haga la suma y la división. Compile y depure.

## Ejercicio 2: Cómo escribir una clase

- En los ejercicios, deberá escribir sus propias clases.
  - Ya ha visto clases en todos nuestros ejemplos, pero éstas no son las que normalmente se utilizan.
    - Sólo poseen un método simple, *main()*
  - La mayoría de las clases no poseen un método *main()*.
- Para construir un programa, escribirá varias clases, de las cuales una, debe tener un método *main()*.

## La clase *Point* (punto)

```
public class SimplePoint {
    private double x, y;           // Miembros de datos
    public SimplePoint() {        // Constructor
        x= 0.0;
        y= 0.0; }
    // Métodos
    public double getX() { return x;}
    public double getY() { return y;}
    public void setX(double xval) { x= xval;}
    public void setY(double yval) { y= yval;}
    public void move(double deltaX, double deltaY) {
        x += deltaX;
        y += deltaY; }
}
// Fin de la clase SimplePoint
// Esto no es un programa porque no posee un método main(), pero
// puede ser utilizado por clases con un método main().
```

## Clase *Point*, *Main()*

```
public class SimplePoint1 {
    public static void main(String[] args) {
        SimplePoint a= new SimplePoint();
        SimplePoint b= new SimplePoint();
        double xa= a.getX();
        double ya= a.getY();
        System.out.println("a= (" + xa + ", " + ya + ")");
        a.move(-9.0, 7.5);

        System.out.println("a= (" + a.getX() + ", " + a.getY() + ")");
    }
}
```

## Ejercicio 2

- Escriba una clase *SimplePoint* diferente que utilice coordenadas polares en vez de coordenadas cartesianas.
  - Implemente los mismos métodos públicos que la clase *SimplePoint* anterior.
  - Utilice *r* y *theta* como campos de datos privados.
  - Recuerde que:
    - $x = r \cos(\theta)$
    - $y = r \sin(\theta)$
    - $r = \sqrt{x^2 + y^2}$
    - $\theta = \tan^{-1}(y/x)$
  - Utilice la clase *Math* de Java (M mayúscula).
    - Utilice *Math.atan2* para el arctan (arcotangente).
- Utilice el mismo método *main()* que antes.

## ¿Por qué hacer esto?

- **Al crear una clase con métodos públicos y datos privados, usted sólo se compromete con una interfaz, no con la implementación:**
  - Si necesita cambiar la implementación, puede hacerlo sin romper cualquier código que dependa de ésta, siempre y cuando la interfaz (conjunto de métodos) permanezca igual.
  - El cambiar los sistemas de coordenadas, los métodos computacionales, etc., es bastante normal, como es el caso de este ejemplo. Esto confiere flexibilidad al software a la hora de ampliarse o cambiarse.

## Clase *Point*, coordenadas polares

```
class SimplePoint {
    private double r, theta;        // Miembros de datos
    public SimplePoint() {          // Constructor
        r= 0.0;
        theta= 0.0; }
    // Métodos (las funciones Trig utilizan radianes)
    public double getX() { return r* Math.cos(theta);}
    public double getY() { return r* Math.sin(theta);}
    public void setX(double xval) {
        double yval= r*Math.sin(theta);
        r= Math.sqrt(xval*xval + yval*yval);
        theta= Math.atan2(yval, xval); }
}
```

## Clase *Point*, coordenadas polares, 2ª parte

```
public void setY(double yval) {
    double xval= r*Math.cos(theta);
    r= Math.sqrt(xval*xval +
yval*yval);
    theta= Math.atan2(yval,
xval); }
public void move(double deltaX, double deltaY) {
    double xval= r*Math.cos(theta);
    double yval= r*Math.sin(theta);
    xval += deltaX;
    yval += deltaY;
    r= Math.sqrt(xval*xval +
yval*yval);
    theta= Math.atan2(yval,
xval);
}
}

// Se puede invocar desde el mismo main() que antes y
// produce los mismos resultados (distintos a los errores
// de redondeo)
```

Todo el código se ha producido con el software de Java™.  
Java™ es una marca de Sun Microsystems Inc.

